

AD-A035 708

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF
THE USE OF MICROCOMPUTERS IN DCS AUTODIN TRIBUTARIES. (U)
DEC 76 G E ANDERSON

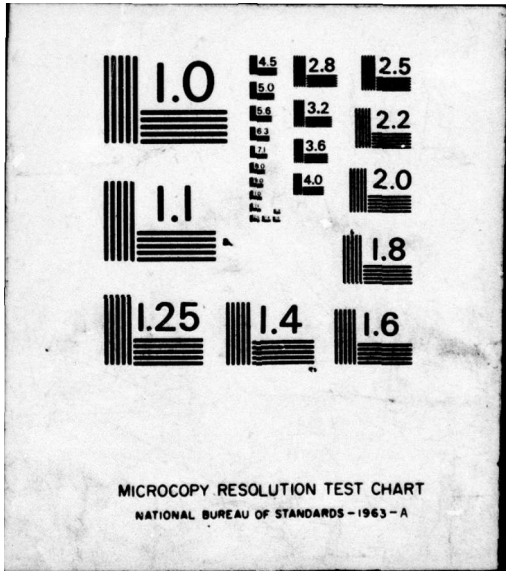
F/G 17/2

UNCLASSIFIED

NL

1 of 2
AD
A035708





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS - 1963 - A

ADA 035708

2
P.S.

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

DDC
DRAFTING
FEB 17 1977
RESISTANCE

THE USE OF MICROCOMPUTERS IN
DCS AUTODIN TRIBUTARIES

by

Gordon Ernest Anderson

December 1976

Thesis Advisor:

V. M. Powers

Approved for public release; distribution unlimited.

COPY AVAILABLE TO DDC DOES NOT
PERMIT FULLY LEGIBLE PRODUCTION

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. REPORT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Use of Microcomputers in DCS AUTODIN Tributaries,		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1976
7. AUTHOR(s) Gordon Ernest Anderson		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1976
		13. NUMBER OF PAGES 118
		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)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Present-day Mode I AUTODIN tributaries utilize large-scale computers such as the IBM 360 series, Burroughs 3500 series, and the Univac DCT 9000. The feasibility of using microcomputers (such as the Intel 8080) for such applications was investigated. It was demonstrated that microcomputers can function as Mode I AUTODIN tributaries at speeds greater than		

251450 ✓

JP

20. Abstract (Cont'd)

2400 baud. This fact could result in the replacement of expensive leased equipment with subsequent cost savings and expanded use of AUTODIN in tactical and mobile situations. In addition, new methods of describing communication protocols were explored.

ACCESSING for		
HTIS	White Section	<input checked="" type="checkbox"/>
DTR	Buff Section	<input type="checkbox"/>
UNANNOUNCED		<input type="checkbox"/>
JUSTIFICATION		
BY		
DISTRIBUTION/AVAILABILITY CODES		
Dist.	AVAIL. and/or	SPECIAL
A		

The Use of Microcomputers in DCS AUTODIN
Tributaries

by

Gordon Ernest Anderson
Captain, United States Marine Corps
B.S., University of Washington, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
December 1976

Author

Gordon E. Anderson

Approved by:

V. Michael Powers

Thesis Advisor

Uno R. Kodres

Second Reader

[Signature]
Chairman, Computer Science Department

David A. Schrey
Dean of Information and Policy Sciences

ABSTRACT

Present-day Mode I AUTODIN tributaries utilize large-scale computers such as the IBM 360 series, Burroughs 3500 series, and the Univac DCT 9000. The feasibility of using microcomputers (such as the Intel 8080) for such applications was investigated. It was demonstrated that microcomputers can function as Mode I AUTODIN tributaries at speeds greater than 2400 baud. This fact could result in the replacement of expensive leased equipment with subsequent cost savings and expanded use of AUTODIN in tactical and mobile situations. In addition, new methods of describing communication protocols were explored.

TABLE OF CONTENTS

I.	INTRODUCTION -----	12
II.	BACKGROUND -----	14
	A. AUTODIN OVERVIEW -----	14
	B. FUNCTIONAL DESCRIPTION OF AUTODIN -----	17
	1. Modes of Operation -----	17
	2. Synchronous Idle Pattern -----	19
	3. Line Block Format -----	20
	4. Control Characters -----	26
	5. An Analysis of Block-by-Block Operation -----	29
	C. REASONS FOR INVESTIGATING MICROCOMPUTERS FOR AUTODIN APPLICATIONS -----	33
III.	MAKING THE AUTODIN PROTOCOL MORE UNDERSTANDABLE -----	36
	A. DIFFICULTIES IN UNDERSTANDING THE AUTODIN PROTOCOL -----	36
	B. THE AUTODIN RECEIVE MACHINE -----	39
	C. THE AUTODIN TRANSMIT MACHINE -----	45
IV.	SOFTWARE DESIGN AND IMPLEMENTATION -----	50
	A. DEFINITION OF THE PROBLEM -----	51
	B. THE HARDWARE ENVIRONMENT -----	53
	C. CHOOSING A PROGRAMMING LANGUAGE -----	58
	D. DESIGNING BY LEVELS -----	58

E.	THE REQUISITE OPERATING SYSTEM -----	60
F.	IMPLEMENTING THE RECEIVER AND TRANSMITTER PROCESSES -----	63
G.	TESTING AND DEBUGGING THE PROGRAM -----	65
V.	FEASIBILITY TESTING -----	68
A.	RESULTS OF THE PERIPHERAL DEVICE TEST -----	68
B.	RESULTS OF THE TIMING TESTS -----	68
VI.	CONCLUSIONS AND RECOMMENDATIONS -----	71
APPENDIX A:	OPERATIONAL SPECIFICATION FOR AUTODIN TEST PROGRAM -----	74
APPENDIX B:	ACTUAL TEST MESSAGE -----	77
APPENDIX C:	TIMING TEST CALCULATIONS -----	78
	THE AUTODIN TEST PROGRAM -----	79
	BIBLIOGRAPHY -----	117
	INITIAL DISTRIBUTION LIST -----	118

LIST OF TABLES

I. AUTODIN RECEIVER STATE TRANSITION
DESCRIPTIONS ----- 42

II. AUTODIN TRANSMITTER STATE TRANSITION
DESCRIPTIONS ----- 47

LIST OF FIGURES

1. LINE BLOCK STRUCTURE OF AN AUTODIN MESSAGE
CONTAINING 277 TEXT CHARACTERS ----- 22
2. BLOCK-BY-BLOCK OPERATION IN ONE DIRECTION WITH
ASC TRANSMITTING AND TRIBUTARY RECEIVING ----- 31
3. THE AUTODIN RECEIVER STATE DIAGRAM ----- 41
4. THE AUTODIN TRANSMITTER STATE DIAGRAM ----- 48

TABLE OF ABBREVIATIONS

ASCII	American Standard Code for Information Interchange
ASC	Automatic Switching Center
AUTODIN	Automatic Digital Network
AUTOSEVOCOM	Automatic Secure Voice Communications
AUTOVON	Automatic Voice Network
baud	(in this thesis) bits per second
DCA	Defense Communications Agency
DCS	Defense Communications System
LMF	Language Media Format
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/Transmitter

AUTODIN CONTROL AND FRAMING CHARACTERS:

ACK1	Acknowledge Number 1
ACK2	Acknowledge Number 2
BP	Block Parity
CAN	Cancel
DEL	Delete
EM	End of Medium
ETB	End of Text Block

ETX	End of Text
INV	Suspected Invalid Message
MC	Mode Change
NAK	Negative Acknowledge
REP	Reply
RM	Reject Your Message
SEC	Security
SEL	Selection Channel Characters
SOH	Start of Header
STX	Start of Text
SYN	Synchronous Idle
WBT	Wait Before Transmitting

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisor Assistant Professor V. Michael Powers for his invaluable assistance and stimulating guidance in the preparation of this thesis. Thanks are due to Lieutenant (jg) Gail Junge and Senior Chief Radioman Frederick Guth of the Naval Telecommunication Center, Monterey, California, for their enthusiastic cooperation and assistance. Also, I must thank Lieutenant Commander Jane Renninger whose pioneering work in reducing the AUTODIN protocol to transition state machines proved indispensable to this thesis.

Finally, I must extend special thanks to my wife, Norma, without whose patience and understanding this thesis could not have been completed.

I. INTRODUCTION

The purpose of this thesis was to investigate the feasibility of using microcomputers (such as the Intel[®] 8080) as Mode I block-by-block AUTODIN tributaries. Before embarking on the feasibility study, the AUTODIN was studied carefully to ensure that the problem was completely understood. Chapter II examines this background information, giving an overview and a functional description of the AUTODIN. In addition, the reasons for investigating microcomputers as potential AUTODIN tributary stations are explored.

Difficulty was encountered in understanding all ramifications of the AUTODIN protocol. As a consequence, the protocol was described in terms of a receive machine and a transmit machine, which are described in Chapter III. A step-by-step description of the software design of an AUTODIN test program is given in Chapter IV. Careful definition of the problem, understanding the hardware environment, and using the top-down, modular approach are the points emphasized.

[®] Intel and Intellec are registered trademarks of the Intel Corporation.

Chapter V describes the results of feasibility testing where the correctness of the AUTODIN test program was verified and timing tests demonstrated that the 8080 CPU could function as an AUTODIN tributary at modulation rates exceeding 2400 baud. Finally, Chapter VI summarizes the conclusions and recommendations of this thesis.

II. BACKGROUND

The Defense Communications System (DCS) consists of three major subsystems: the Automatic Voice Network (AUTOVON), the Automatic Secure Voice Communications Network (AUTOSEVOCOM), and the Automatic Digital Network (AUTODIN). The first two subsystems provide nonsecure and secure voice communications, while the third subsystem, AUTODIN, provides a secure record communication capability. This thesis is concerned with the AUTODIN.

A. AUTODIN OVERVIEW

The AUTODIN functions as a single, integrated, worldwide, high-speed, computer-controlled, general purpose communications network which provides record communications service to the Department of Defense (DOD) and other Federal Government Agencies, such as the Department of State. In addition to providing record communications service via various media (such as printed page, magnetic tape, Hollerith cards, etc.), the AUTODIN is also secure and fully automatic. It was designed, engineered, and programmed to provide responsive and continuous operation, minimal loss of service, and no loss of message traffic.

The AUTODIN is a network which consists of 17 Automatic Switching Centers (ASC's) and numerous tributary stations. Of the 17 ASC's, nine are leased and are located in the continental United States and Hawaii. The remaining eight ASC's are government owned and are located in Europe, the Pacific, and Alaska. Each ASC may have up to 300 tributary stations connected to it. This network of ASC's and tributary stations is able to provide responsive communications by use of a system of four message precedence levels: flash, immediate, priority, and routine. By requiring users of the AUTODIN to curtail their use of the higher precedence levels, and by programming the AUTODIN to handle all message traffic on a precedence basis, it is possible for flash precedence level messages to be switched and transmitted around the world in a matter of a few minutes. This capability for rapid communications greatly enhances the effectiveness of the defense establishment of the United States.

A flash level message interrupts all messages of precedence level immediate or less. Precedence level immediate messages are processed before priority or below level messages; however, the lower precedence level messages are not interrupted. Similarly, priority level messages are processed before routine level messages (without interruption of the routine level messages). By proper selection

of precedence levels, users of the AUTODIN can control the speed at which their messages are propagated through the system, with a lower limit of one to three minutes for flash messages and an upper limit of one to two hours for routine messages.

In addition to the variability of speed of transmission provided by the precedence system, there are two other properties of the AUTODIN which greatly enhance its usefulness. First, there is the capability for multiple addressing. The originator of an AUTODIN message may specify that the message go to one, two or hundreds of addressees. This can be accomplished in two ways: by enumerating the addressees, or (if the addressees are grouped together often) by use of collective addresses. The second additional property of the AUTODIN which enhances its usefulness is the ability to use various media for record communications. For example, the originator of a lengthy supply message might transmit the message from magnetic tape and the message could be received as cards on a card punch at the receiving communication center. Conversely, a small communication center without a card capability could transmit logistical data from paper tape and have it punched as cards at the receiving communication center, thus eliminating the need for keypunching the data at the logistical center.

With such properties as variable speed of transmission, selectable media input and output, and multiple addressing, the AUTODIN has provided flexible, responsive, and reliable record communications for over a decade. In order to understand it more fully, it is necessary to examine it on a more technical and detailed level.

B. FUNCTIONAL DESCRIPTION OF AUTODIN

The AUTODIN is a digital network consisting of ASC's and tributary stations with interconnecting communications channels. Both synchronous and asynchronous operation are employed within the AUTODIN; however, asynchronous operation is permitted only on tributary channels, whereas synchronous operation is permitted on both interswitch trunks and tributary channels. For synchronous operation, the AUTODIN will process information at modulation rates of 75, 150, 300, 600, 1200, 2400, and 4800 baud. For asynchronous operation, modulation rates of 75, 150, and 300 baud are permitted. All synchronous AUTODIN communications channels use the American Standard Code for Information Interchange (ASCII). The basic unit for information transfer in AUTODIN is the line block, several of which are shown in Figure 1.

1. Modes of Operation

There are five modes of operation within the AUTODIN. These are Mode I, which is duplex, synchronous operation with

automatic error and channel controls which allow independent and simultaneous two-way operation; Mode II, which is duplex, asynchronous operation allowing simultaneous two-way operation without automatic error and channel controls; Mode III, which is duplex, synchronous operation with automatic error and channel controls (but with one-way information transfer and the return direction used solely for error control and channel coordination responses); Mode IV, which is a unidirectional synchronous operation which can send only or receive only and does not have automatic error control; and Mode V, which permits duplex asynchronous operation and allows simultaneous and independent two-way transmission but which performs only limited channel coordination and display functions.

From the above descriptions, it should be evident that Mode I AUTODIN is the most efficient and hence most desirable type of AUTODIN. All of the asynchronous modes are limited to modulation rates of 300 baud or less. Thus, for medium or high speed data transfer rates, the synchronous modes (Mode I or Mode III) must be used. Mode III contains an inherent disadvantage in that information transfer (or message transmission) is limited to one direction at a time. Thus, only Mode I AUTODIN offers both high-speed operation and simultaneous and independent two-way transmission of

information. This thesis deals solely with Mode I block-by-block operation. All subsequent discussion of AUTODIN assumes Mode I block-by-block operation. The difference between block-by-block and continuous operation will be discussed in Section II.B.5 of this thesis.

2. Synchronous Idle Pattern

In Mode I AUTODIN operation, whenever information is not being transmitted, synchronous idle pattern must be transmitted at the designated modulation rate. Synchronous idle pattern is an even parity character which is equal to the number 96 hexadecimal (or 10010110 binary). Since synchronous idle is transmitted whenever information is not being sent, the receive side of the AUTODIN logic uses synchronous idle pattern to determine whether or not it is in synchronization. At initialization, the Mode I AUTODIN receiver attempts to detect the synchronous idle character (SYN). After the first SYN is detected, the next three characters are checked for the SYN pattern. If the following three characters are SYN, then the receiver considers itself to be in character frame (or synchronized); otherwise, it repeats the above process, repeatedly attempting to achieve character frame. An AUTODIN transmitter may transmit information only if its receiver is in character frame. Likewise,

an AUTODIN receiver may process incoming information only if it is in character frame.

3. Line Block Format

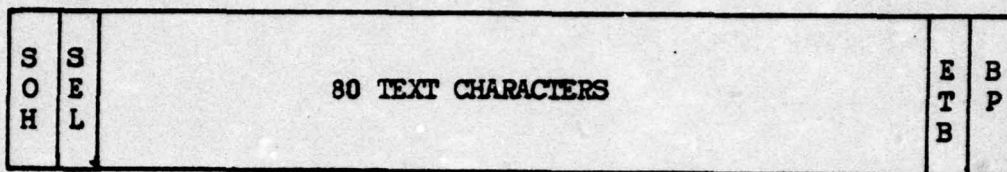
The basic unit for information transfer in AUTODIN is the line block. It may be thought of as a package of information. A typical sequence of events for an ASC transmitting to a tributary station under Mode I operation might be as follows: The ASC sends synchronous idle pattern to the tributary station. The tributary receiver recognizes the synchronous idle pattern and considers itself in character frame. Since Mode I AUTODIN is duplex, the same process takes place (simultaneously and independently) in the opposite direction: the tributary transmitter achieves synchronization with the ASC receiver. Once synchronization has been achieved, it is possible to transmit information in the form of line blocks or "packages" of information. If, for instance, the ASC were transmitting to the tributary, the ASC would send the first line block. If the tributary station received the line block without error, it would reply with an acknowledgement, and the ASC would be free to send a subsequent line block. However, if any error were present in the line block, the tributary would reply with a negative acknowledgement (NAK), and the ASC would retransmit the first line block. In this manner, information is transmitted

in either direction or both directions with channel control and error detection. It should be kept in mind that in Mode I AUTODIN, simultaneous and independent information transfer can occur in both directions. In order to understand more fully the AUTODIN communications protocol, it is necessary to examine the line block structure and associated control characters in detail.

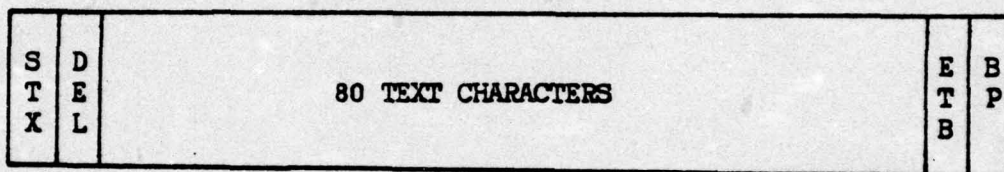
Consider an AUTODIN message which contains 277 text characters or bytes of information. It would be transmitted as four line blocks, the first three of which would contain 80 bytes of information while the fourth would contain 37 bytes of information. Figure 1 shows the line block structure of such a message.

The first character of the first line block of every AUTODIN message is the Start of Heading (SOH) framing character. It is an even parity character which signals the beginning of a new message, and it is always followed by the Select (SEL) framing character. This sequence cannot be split by any other character. The SEL character is an even parity framing character which is always the second framing character of the first line block of every AUTODIN message. Unlike the SOH framing character which is always the same, the SEL character may be one of several alphabetic characters. These alphabetic characters correspond to the

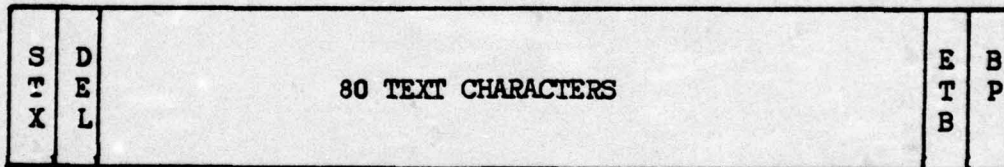
FIRST LINE BLOCK



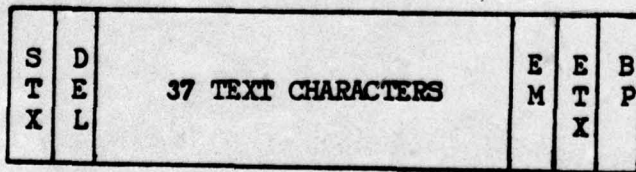
SECOND LINE BLOCK



THIRD LINE BLOCK



FOURTH (LAST) LINE BLOCK



LINE BLOCK STRUCTURE OF AN AUTODIN MESSAGE
CONTAINING 277 TEXT CHARACTERS

FIGURE 1.

various Language Media Format (LMF) indicators but are coded by a different set of characters, according to reference 10. The LMF characters, which appear in the message as it enters and leaves the system, correspond one for one with the SEL characters, which appear in the first line block of the message while it is inside the network. The translation from LMF character to SEL character and back must be accomplished by the network interfaces (tributaries). For example, if an AUTODIN message were narrative in nature, and the originator desired that the addressee of the message receive a printed page version of the message, then the originator would use the LMF indicators "TT." The second "T" would indicate that output on a line printer (or similar device) was desired. This "T" would be translated into the SEL character "H" by the transmitter. Thus, the receiver at an AUTODIN tributary station which received an SOH followed immediately by an even parity "H" would interpret this to mean that the incoming message was to be printed on the line printer. The purpose, therefore, of the SEL character is simply to select the output device at the receiving tributary station. An LMF "C" (meaning card output) would be translated into a "D" SEL character which would cause output on a card punch at the receiving tributary. Reference 2 contains a complete list of SEL and LMF characters.

Following the SOH and SEL framing characters are the first 80 text characters of the AUTODIN message. These characters are transmitted with odd parity. That is, the first seven bits correspond to the American Standard Code for Information Interchange (ASCII) and the eighth bit is either a one or zero such that the total number of ones in the eight-bit byte are odd. The next-to-last character in the first line block of the example in Figure 1 is the End of Transmission Block (ETB) framing character. In fact, ETB is always the third framing character of every line block except the last block of the message. Like all other framing characters, it is an even parity character. The ETB character is immediately followed by the Block Parity (BP) character. No character of any kind may be inserted between ETB and BP. Block Parity is the last framing character of every AUTODIN line block. It may be either odd or even in parity because it is formed by the binary addition without carry (sum modulo 256) of all bytes in the line block. In this way BP serves to check the correctness of received line blocks by detecting single errors.

The second line block of the example message begins with the Start of Text (STX) framing characters. STX is the first framing character of every line block except the first line block which is started with the SOH framing

character. STX is an even parity character which is always followed immediately by a Delete (DEL) framing character, which is also even in parity. The DEL character is the second framing character of every line block except the first one which has an SEL in the second position. The DEL character is used only on links between ASC's and tributary stations. On interswitch trunks between ASC's, the DEL is replaced with a Security (SEC) framing character which is used by the ASC's for the routing of classified and unclassified message traffic.

The remainder of the second line block is the same as the first line block -- 80 text characters followed by the ETB and BP characters. In fact, all subsequent line blocks are the same (STX, DEL, 80 text characters, ETB, and BP) except for the last line block. The last line block begins with STX and DEL framing characters; however, these are followed by 37 text characters and three framing characters. The first of these framing characters is the End of Medium (EM). This even parity character is used to signal the end of an AUTODIN message. It is followed by the End of Text (ETX) framing character (even in parity) and the BP framing character. The BP character is formed as previously described except that it is computed on the 37 text characters and the EM character instead of the 80 text characters as in line blocks one, two, and three.

The line block structure is built and transmitted by the transmit logic of an AUTODIN ASC or tributary. Analogously, the receive logic portion of an AUTODIN ASC or tributary expects to receive information in this line block structure. Now that this structure has been explained, it is possible to discuss the AUTODIN protocol and its associated control characters.

4. Control Characters

In order to provide for channel coordination, control characters are required. Control characters are even parity characters which are always transmitted as contiguous pairs. Six of the most important ones are described below:

(1) Acknowledge Number One (ACK1).

ACK1 is sent by an ASC or tributary to signal the distant transmitter that a line block has been received correctly. ACK1 is the answer to the first line block sent after power-up, or to the first line block received after a message has been cancelled. Thereafter, ACK1 is used alternately with ACK2 to indicate correctly received line blocks.

(2) Acknowledge Number Two (ACK2)

ACK2 is sent as a reply to indicate the correct reception of a line block after a line block has been acknowledged with ACK1. For example, if line block one is received correctly and an ACK1 is sent in reply, then when line block

two is received (correctly), an ACK2 is sent in reply. The sequence of alternate ACK1's and ACK2's is not interrupted between messages; that is, if the answer to last line block of a message was ACK1, then the answer to the first line block of the next message will be ACK2.

(3) Negative Acknowledge (NAK)

Tributaries and ASC's use NAK to signal that a line block has been received with an error in it. NAK is sent after the end of the erroneous line block is received, not at the time the error is detected. Whenever an NAK is received, the transmitting station will retransmit the complete line block to which the NAK applies.

(4) Reject Your Message (RM)

RM's are sent as replies to line blocks. Only an ASC can send an RM, which is sent to the transmitting tributary to signal that there is a defect in the message which cannot be rectified by retransmission of the line block.

(5) Wait Before Transmitting (WBT)

WBT is sent by either an ASC or tributary station in response to a properly framed line block to inform the distant transmitter that the local receiver can no longer accept line blocks. The eventual response to the line block in question may be an ACK1, ACK2, or even NAK; however,

while WBT is being received (and until an ACK or NAK is received), the transmitting station may send only control characters or synchronous idle pattern (SYN).

(6) Reply (REP)

An ASC or tributary station transmitter will use the REP to direct the distant receiver to send its last response or current (updated) response such as ACK1, ACK2, NAK, RM, or WBT. Each transmitter must be equipped with a variable timer hereafter referred to as the answer timer. At the end of each line block transmitted, the answer timer is initialized. When the answer timer expires an REP will be sent if an answer has not been received or if a WBT has been received. Each time an REP is sent the answer timer will be reinitialized. Whenever an ACK1, ACK2, NAK, or RM is received, the answer timer will be stopped. The duration of the answer timer is a function of modulation rate, communication path delays, delays in modems, and receiver response delays. The answer timer duration is determined by adding together all the delays for an expected round trip delay time plus a safety margin. Thus, the answer timer delay is equal to slightly more than the time to receive an expected answer (ACK1, ACK2, NAK, etc.) to a line block or REP. Typical answer timer settings are 3 seconds for 75 to 600 baud circuits, 0.5 seconds for 1200 baud circuits

and 0.25 seconds for 2400 baud circuits. If REP is sent three times in succession without receiving an appropriate reply, an alarm will be sounded.

(7) Cancel (CAN)

CAN is sent by a transmitting station to signal the distant receiver to cancel or discard the current message. The CAN may be initiated manually, automatically by the transmitter upon an incorrectable error condition, or automatically by the receiver whenever an RM is received as the response to a line block.

The aforementioned seven control characters permit channel coordination such that erroneous line blocks are retransmitted, correct line blocks are acknowledged, and, whenever circuit degradation occurs, alarms are activated which bring the requisite human intervention. The next section provides examples which will demonstrate the inter-operative relationship between line blocks, framing characters, and control characters.

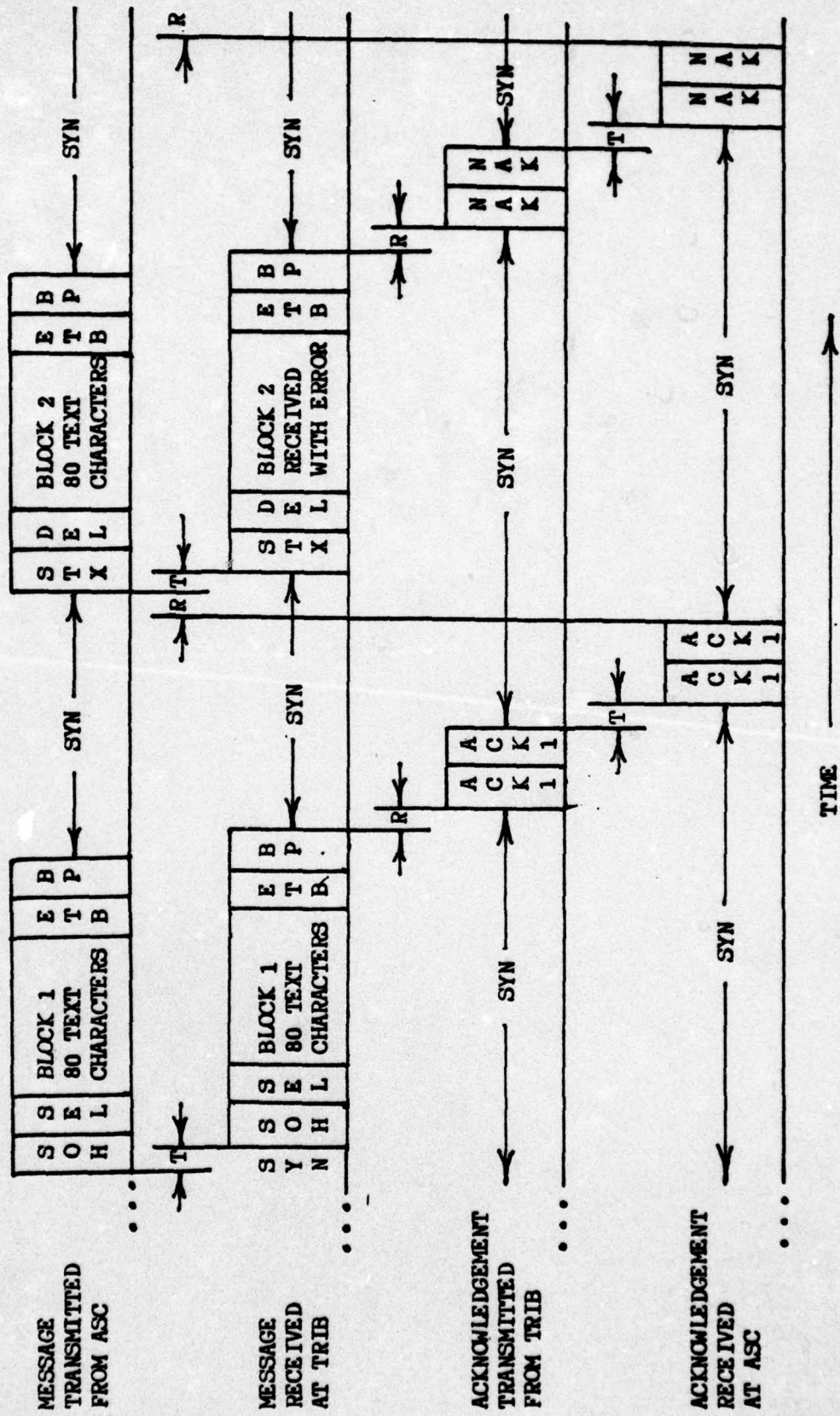
5. An Analysis of Block-By-Block Operation

Within Mode I AUTODIN there are two types of operation: block-by-block and continuous. Under block-by-block operation, a transmitting station sends one line block and does not send a subsequent line block until an ACK1 or ACK2 is received. Under continuous operation, one line block is

sent, then a second one. When continuous operation is working properly, the ACK for the first line block will be received while the second is being transmitted. There is no difference between block-by-block and continuous mode for an AUTODIN receiver and only a trivial change in buffering for an AUTODIN transmitter. This thesis deals only with block-by-block operation.

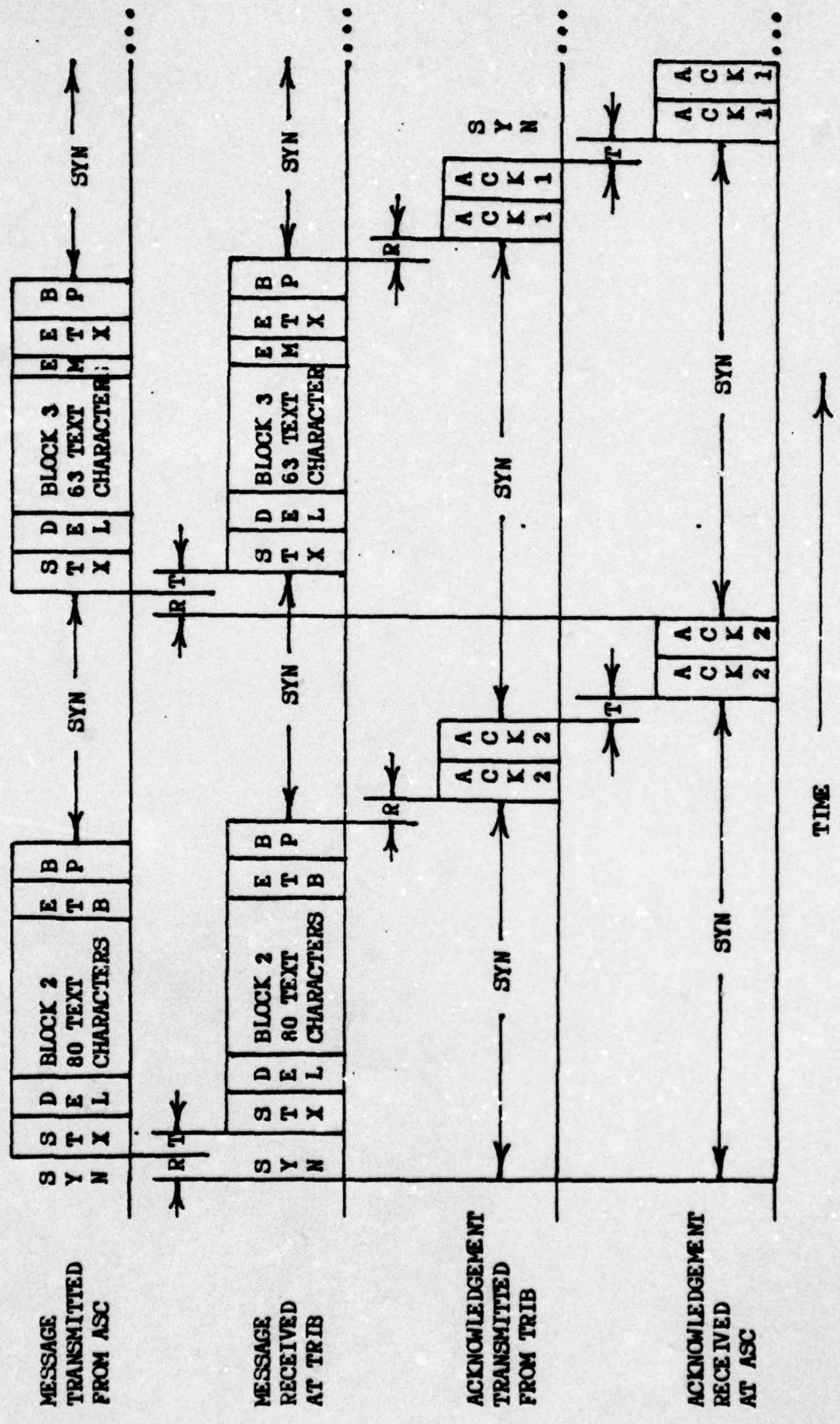
Figure 2 illustrates the AUTODIN protocol: the transmission of data in line block format, the channel coordination obtained from the control characters, the synchronous idle pattern between line blocks and the transmission and response delays involved. The message being transmitted in the example of Figure 2 contains 223 text (or informational) characters. This requires two full-size line blocks of 80 text characters each and a third line block of 63 text characters. In this example, the information transfer is in one direction with the ASC transmitting and the tributary receiving. It will be instructive to trace through Figure 2 from left to right, noting that moving from left to right is analogous to moving forward in time.

Line block one with SOH and SEL for beginning framing characters is transmitted from the ASC and is received at the tributary after a transmission time delay (denoted by "T").



BLOCK-BY-BLOCK OPERATION IN ONE DIRECTION WITH ASC TRANSMITTING AND TRIBUTARY RECEIVING

FIGURE 2a.



BLOCK-BY-BLOCK OPERATION IN ONE DIRECTION WITH ASC TRANSMITTING AND TRIBUTARY RECEIVING
 FIGURE 2b.

After the entire line block is correctly received, and after a response delay time (denoted by "R"), the tributary sends two contiguous ACK1's. These are received back at the ASC a transmission time (T) later. The ASC then transmits the second line block of the message; however, this time there is an error in transmission. Consequently, the tributary sends two contiguous NAK's. When these NAK's are received at the ASC, the second line block of the message is retransmitted. This time it is correctly received by the tributary, which sends the appropriate ACK2. Finally, the last line block of the message, which is a short line block containing 63 text characters (marked at the end of text by an EM character), is transmitted. The tributary acknowledges receipt of the last line block with an ACK1, which illustrates the alternation of ACK1's and ACK2's.

From the above descriptions, the reader should have a general idea of how Mode I block-by-block AUTODIN functions. It is merely the transmission, reception, and acknowledgement of line blocks which contain the information to be communicated.

C. REASONS FOR INVESTIGATING MICROCOMPUTERS FOR AUTODIN APPLICATIONS

Reference 1 states that various computers have been approved and certified by the Defense Communications Agency

(DCA) for use as AUTODIN tributary stations. These computers are:

1. IBM 360 Series.
2. RCA SPECTRA 70 Series.
3. Univac DCT 9000 Series.
4. Control Data Corporation CD1700 Series.
5. SOROBAN DST (Mohawk Data Science Corporation).
6. Honeywell 200 Series.
7. ITT World ADX 9300.
8. Burroughs 3500 Series.

Hundreds of the above machines have been installed around the world to provide AUTODIN service to the far-flung units of the Department of Defense. The Naval Telecommunications Center, Monterey, California, is a typical tributary station. It is a 1200 baud Mode I tributary which consists of a Univac DCT 9000 computer with two magnetic tape drives, a card reader, a card punch, a paper tape reader, a line printer, and a communication interface unit. The annual cost to the government to provide this equipment is \$67,824.00 per year for equipment leasing and \$13,512.00 for on-call maintenance support. Thus, for equipment alone, over \$80,000.00 per year must be spent on this tributary station, and this is not an atypical amount. The Communication Center of the Third Force Service Regiment, Fleet Marine Force Pacific, located on the island of Okinawa, costs a similar amount for the same capability: a 1200 baud, Mode I AUTODIN tributary. In this case the equipment is an IBM 360/20 with equivalent peripheral equipment.

In reviewing the above equipment costs, two questions immediately come into mind: First, are such relatively expensive and powerful computers needed for AUTODIN tributary applications? Second, can inexpensive microcomputers function as AUTODIN tributaries? If microcomputers can be programmed to serve as AUTODIN tributary stations, then it is possible to replace the more expensive, powerful machines presently being used and save millions of dollars each year. In addition, since microcomputers are smaller, lighter, and more rugged than the aforementioned large computers, the potential use of microcomputers as AUTODIN tributaries in tactical and mobile situations could greatly improve the record communication capabilities of deployed combat units. In short, greatly reduced costs and expanded AUTODIN service in tactical situations are two potential benefits to be realized if microcomputers are capable of functioning as AUTODIN tributaries. For this reason, the central question of this thesis is: can a microcomputer function as an AUTODIN tributary? If so, how fast can it process information?

III. MAKING THE AUTODIN PROTOCOL MORE UNDERSTANDABLE

Before designing and writing a computer program which would demonstrate the feasibility of using a microcomputer as an AUTODIN tributary, it was necessary to understand completely all of the details of the AUTODIN protocol for Mode I block-by-block operation.

A. DIFFICULTIES IN UNDERSTANDING THE AUTODIN PROTOCOL

Although reference 2 is a very comprehensive and detailed document, it is difficult to use in gaining a complete and precise understanding of the AUTODIN protocol. The major obstacle which prevented an easy understanding of the protocol is the limitation of short-term human memory: it was impossible for the author to digest reference 2 from cover to cover and then suddenly realize and understand the exceedingly complex AUTODIN protocol. The problem was that reference 2 failed to approach the problem of describing AUTODIN from the top down. In other words, instead of giving an overview of AUTODIN and then explaining it in levels of increasing detail, reference 2 appeared to approach the problem from the inside out, a method which was not suitable for rapid and easy understanding of the protocol. This

contention is reinforced by the following example. In 1975, after over a decade of AUTODIN service, the Univac DCT 9000 computer at the Naval Telecommunication Center, Monterey, California, went into the machine halt condition as the result of a software bug which surfaced while an AUTODIN message was being transmitted. Reference 10 specifies that AUTODIN messages shall be terminated by eight line-feeds followed by four N's. However, the DCT 9000, a computer which is sanctioned for AUTODIN use by DCA, interpreted the presence of four contiguous N's in an encoded weather message as the end of message indicator. No line-feeds were involved. This lack of precision in describing the AUTODIN protocol leads to ambiguities which can cause mistakes in programming. The process of describing a complex, detailed protocol in this manner is analogous to describing a building to a blind man brick by brick without first giving a description of the shape, size, and purpose of the building.

For example, when reading reference 2, the author came across the fact that all control characters are transmitted in contiguous pairs. The question then arose as to what the AUTODIN receive logic must do in the event only one control character is received. Should the receiver ignore the character? Should it act upon the character as though it were a valid two-character control sequence? At first, the

author thought that there was an ambiguity on this point; however, the answer was finally found buried in the details of reference 2: the receiver logic ignores single control characters; it only acts upon contiguous pairs of control characters.

This example and others like it served to illustrate the inadequacies of reference 2. What was needed was an overview of the protocol -- some method of describing the inter-operability of all the facets of the protocol. The flowcharts of reference 2 failed to provide an overview of the protocol and also failed to provide enough precision to cover all contingencies. In other words, a better method of describing the AUTODIN protocol was needed. This better method was first used by Renninger in reference 12.

Renninger described the AUTODIN protocol in terms of two state transition diagrams: a receiver and a transmitter. Indeed, throughout reference 2 there are numerous references made to a receiver and a transmitter, but the reader is never told exactly what they are. From studying the work of Renninger, it became clear to the author what the AUTODIN receiver and transmitter were: they were transition state machines which had starting states and which were driven from state to state. Each incoming or received byte represented a potential state transition for the receive machine;

likewise, each byte to be transmitted represented a potential state transition for the transmit machine. The actual transitions made and actions taken depend upon these inputs to the receiver and transmitter and upon the condition of numerous flags which contain detailed information on the overall state of each machine.

The term transition state machine is used here to denote a machine which is derived from and closely related to a finite state machine. The major difference is that while a finite state machine uses only states to define its logic, a transition state machine uses both flags and states. This somewhat more informal method of describing a logical process has two advantages over the finite state machine model. The first of these is that it permits designers to concentrate on the most important states and the second advantage is that the problem can be reduced to an understandable and readable form. The AUTODIN receive and transmit machines are described in more detail in the sections that follow.

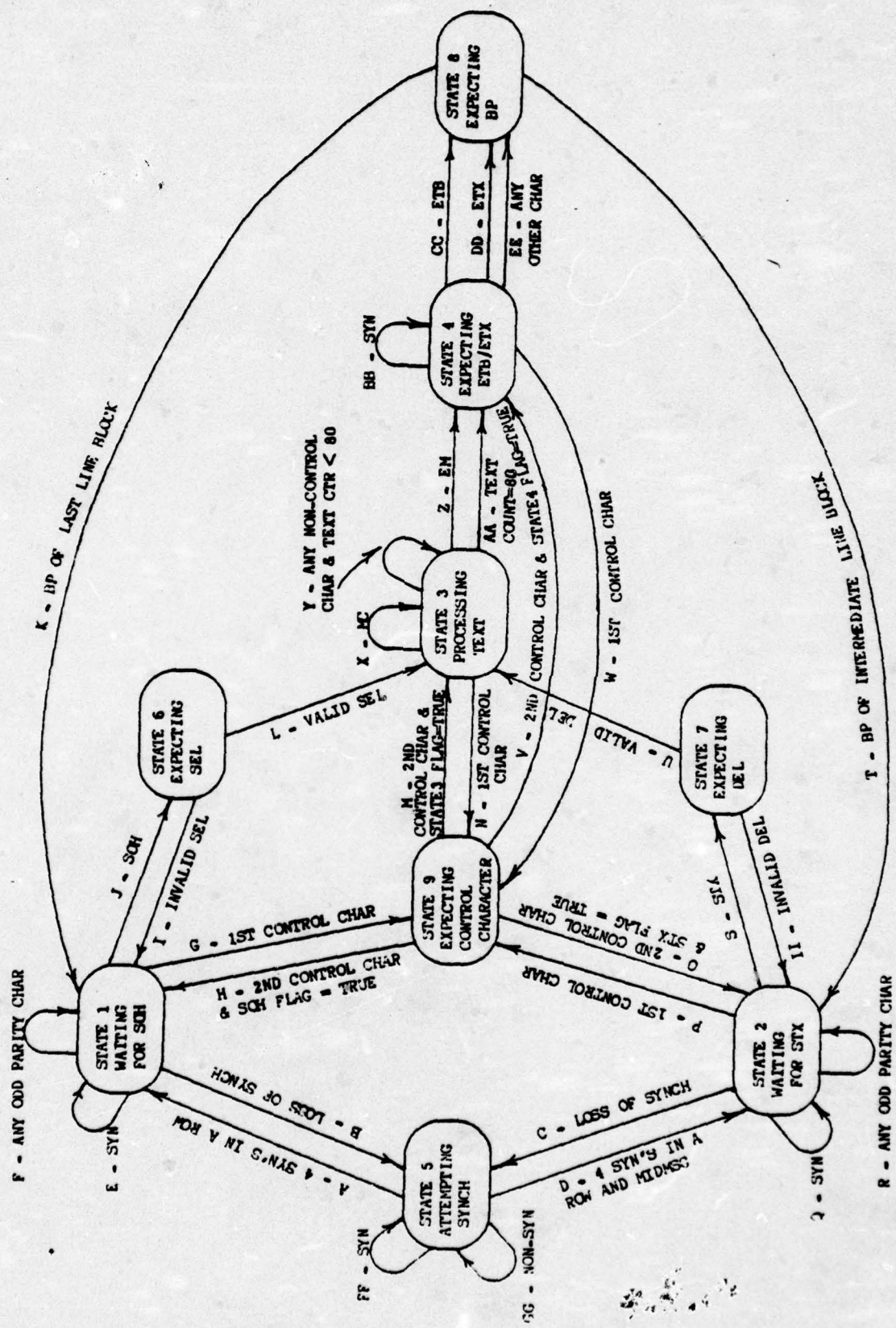
B. THE AUTODIN RECEIVE MACHINE

Renninger described the Mode I AUTODIN receive protocol as a 17-state machine. Here the receive protocol is specified as a nine-state machine. The reason that the protocol can be specified here with eight fewer states is that this

version of the protocol uses more condition flags than Renninger's model, but fewer states. Thus, the two machines are logically equivalent with the following exception: Renninger's machine is for continuous Mode I AUTODIN whereas this machine is for block-by-block Mode I AUTODIN.

Figure 3 depicts the AUTODIN receiver in the form of a nine-state transition diagram. The states are numbered, and the transition paths between states are labeled with letters. A description of each state transition is given in Table I. Each incoming byte or message character corresponds to a transition line on the state diagram, with some transitions beginning and ending in the same state.

It is felt that the state diagram of Figure 3 is a superior method of specifying the control logic of the AUTODIN receiver. It is superior to the flow charts and explanatory text of reference 2 because it utilizes the concept of a finite state machine in its graphical representation to completely specify on one page the receiver protocol. Obviously this is better than scores of pages of text and flowcharts.



THE AUTODIN RECEIVER STATE DIAGRAM
Figure 3

TABLE I
AUTODIN RECEIVER STATE TRANSITION DESCRIPTIONS

<u>Transition</u>	<u>Description</u>
A	Synchronization achieved between messages. Four SYN's received in a row and mid-message flag = false.
B	Loss of synchronization between messages. Receiver timer has expired without 4 SYN's being received and mid-message flag = false.
C	Loss of synchronization between line blocks. Receiver timer has expired without 4 SYN's being received and mid-message flag = true.
D	Synchronization achieved between line blocks. Four SYN's received in a row and mid-message flag = true.
E	SYN character received. Increment syn-counter. If syn-counter = 4 then reset receiver timer and set SYN-COUNTER = 0.
F	Any odd parity character received. Set syn-counter = 0. Check to see if receive timer is expired.
G	First character of a two-character control sequence received. Set SOH flag = true.
H	Second character of a two-character control sequence received and SOH flag = true.
I	Invalid SEL character received.
J	SOH received.
K	BP of last line block in message received (last line block because mid-message = false).

TransitionDescription

L	Correct SEL received. Set mid-message flag = true.
M	Second character of a two-character control sequence received and text flag = true.
N	First character of a two-character control sequence received. Set text flag = true.
O	Second character of a two-character control sequence received and STX flag = true.
P	First character of a two-character control sequence received. Set STX flag = true.
Q	SYN character received. Increment syn-counter. If syn-counter = 4 then reset receive timer and set syn-counter = 0.
R	Any odd parity character received. Set syn-counter = 0. Check to see if receive timer is expired.
S	STX received.
T	BP of intermediate line block in message received (intermediate because mid-message flag = true).
U	Correct DEL received. Set mid-message flag = true.
V	Second character of a two-character control sequence and ETB flag = true.
W	First character of a two-character control sequence. Set ETB flag = true.
X	MC received. Set MC flag = true.
Y	Any non-control or non-framing character received and text counter is less than 80. Increment text counter.

TransitionDescription

Z	EM received. Set mid-message flag = false.
AA	Text counter = 80.
BB	SYN received.
CC	ETB received.
DD	ETX received.
EE	Any character other than ETX or ETB received. Set error flag = true.
FF	SYN received. Increment syn-counter.
GG	Any character other than SYN received. Set syn-counter = 0.

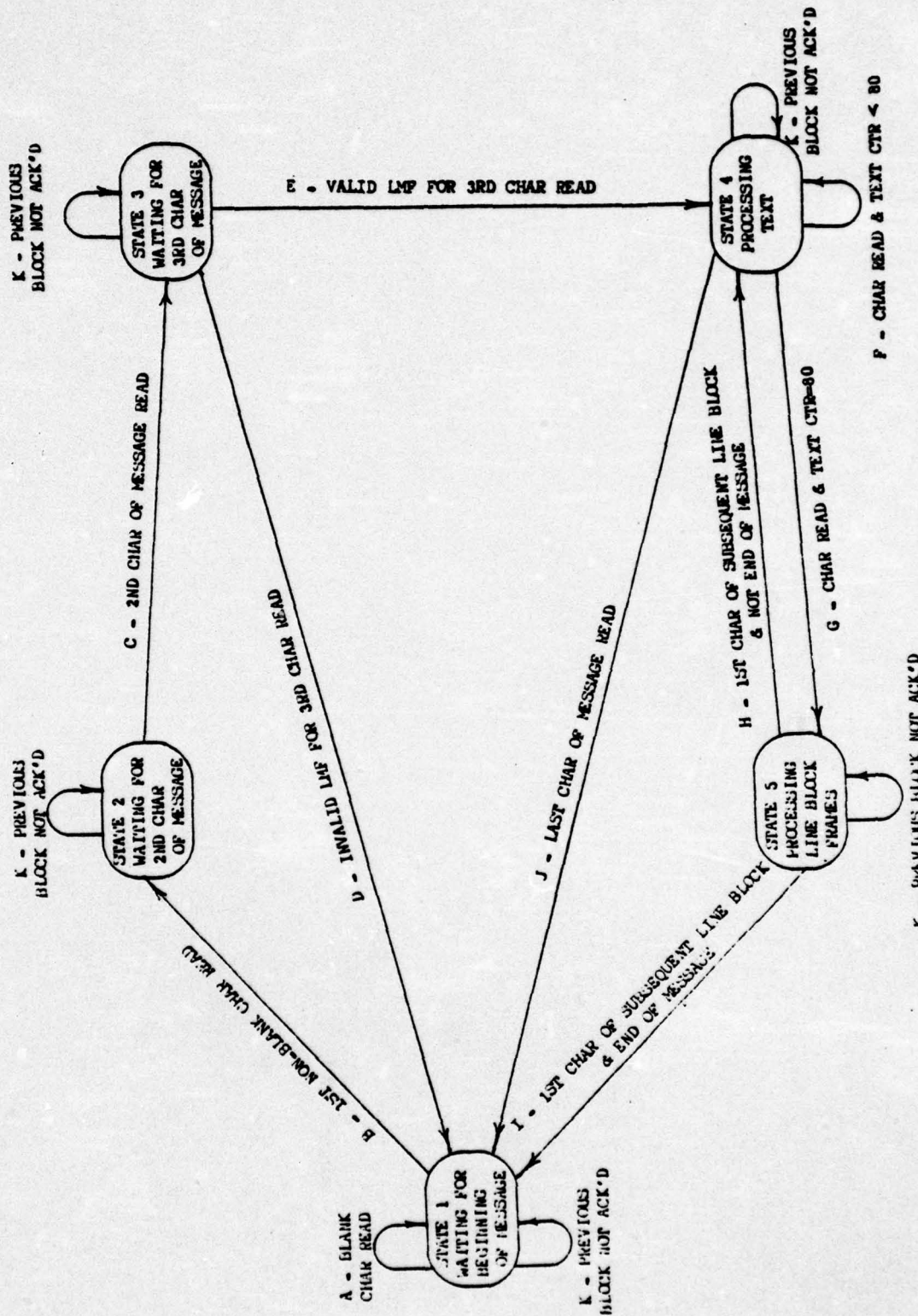
C. THE AUTODIN TRANSMIT MACHINE

The design of the AUTODIN transmit machine is very similar to that of the receive machine except, of course, that the purpose of the transmit machine is to read and transmit characters of text while the purpose of the receive machine is to receive text. Again, Renninger's transmitter consists of nine states whereas the author's machine consists of five states. The reason for fewer states is the same as for the difference in receiver states: fewer states, more condition flags. Finally, this transmit machine and Renninger's differ in that the former is for block-by-block operation and the latter for continuous operation. Otherwise, they are functionally equivalent.

Figure 4 depicts the five-state AUTODIN transmitter as a state transition diagram and Table II gives a description of each of the transitions. Each outgoing byte or text character which is read by the transmit machine corresponds to a line on the state transition diagram. These outgoing bytes can cause transitions from state to state or from a state back to the same state. Again, it is felt that the state diagram method of specifying a communication protocol is far superior to the method used in reference 2.

It should be pointed out that the receive and transmit machines, while specifying the AUTODIN protocol, do not

completely specify all of the details required for implementation on an actual computer. The actual implementation of the receive and transmit machines is the subject of the next section.



THE AUTODIN TRANSMITTER STATE DIAGRAM
Figure 4

TABLE II
AUTODIN TRANSMITTER STATE TRANSITIONS

<u>Transition</u>	<u>Description</u>
A	Blank character read. Blanks are used as leader on paper tape messages and are read and discarded by the transmitter.
B	First non-blank character of a message read. Place character in outgoing line block.
C	Second non-blank character of a message read. Place character in outgoing line block.
D	Third non-blank character of a message read and table lookup indicates invalid LMF character. Cancel the message.
E	Third non-blank character of a message read and table lookup indicates valid LMF character. Place SEL character in outgoing line block and set text counter = 4.
F	Character read and text counter is less than 80. Increment text counter. Place character in outgoing line block.
G	Character read and text counter = 80. Set text counter = 0 and place character in outgoing line block.
H	First character of subsequent line block and end-of-message = false. Set text counter = 2. Place character in outgoing line block.

Transition

Description

- | | |
|---|---|
| I | First character of subsequent line block and end-of-message = true. Place character in outgoing line block. |
| J | Character read and end-of-message = true. Place character in outgoing line block and set text counter = 0. |

IV. SOFTWARE DESIGN AND IMPLEMENTATION

The central issue of this thesis is whether or not a microcomputer can function as a Mode I AUTODIN tributary station and, if it can, at what baud rate can it so function? In order to answer this question it became clear from the very beginning that it would be necessary to develop a computer program which would function as an AUTODIN tributary station. In this way, it would be possible to determine whether or not a microcomputer could perform all of those functions associated with an AUTODIN tributary. If this test proved to be successful, that is, if the microcomputer could perform the necessary tributary functions, then a timing test could be devised which could measure the rate at which the microcomputer could function as an AUTODIN tributary. Consequently, a major portion of the effort expended in this thesis was spent on designing, developing, implementing, testing, and timing a computer program which enabled a microcomputer to function as an AUTODIN tributary station. It should be emphasized, however, that the purpose of the AUTODIN test program is to demonstrate feasibility. It was not designed for actual AUTODIN use.

A. DEFINITION OF THE PROBLEM

The first step in the top-down approach to software development is to define completely the problem to be solved. In this case, the problem was to write a computer program which would enable a microcomputer to function as an AUTODIN tributary station. In addition, the program was to have the property that it could be timed to determine the rate at which it could process AUTODIN messages. This statement defined the problem at its highest level or most general form.

The problem at hand was then taken to the next level of detail. It was determined that the microcomputer, in order to function as an AUTODIN tributary, should be able to interface with the receive side of a communication channel, perform the receive functions of the AUTODIN protocol, and pass the text obtained from the receive channel to a writer device such as a line printer, card punch, or magnetic tape drive. Simultaneously, the microcomputer must also read information from a reader device (such as a paper tape reader, card reader, or magnetic tape drive), put this information into line block format, and interface with the transmit side of a communication channel. In addition, there must be coordination between the transmit and receive functions to provide the full channel coordination and error

detection capability specified and required by the AUTODIN protocol.

The above paragraph represents an important step in the definition of any software problem. That step is specifying the operations which the program must perform. In many circles, this step (or the document which explains it) is called an operational specification. Appendix A is the operational specification for the AUTODIN test program to demonstrate the feasibility of using microcomputers in DCS AUTODIN applications. The reader will note that the operational specification was written in the future tense, since it was developed before the program.

In defining the problem to be solved, two accomplishments served to bring the problem into sharp focus. The first of these was the development of the operational specification of the program. The second was the development of the transmit and receive machine descriptions of the AUTODIN protocol. Indeed, putting the AUTODIN protocol into understandable form was the single most important aspect of defining the problem. The transmit and receive machine descriptions of the AUTODIN protocol appear to be hardware independent; however, many of the points discussed by the operational specification address hardware-dependent problems. For this reason (and in order to achieve an actual

implementation of the AUTODIN protocol) it was necessary to examine the hardware environment in which the program must reside.

B. THE HARDWARE ENVIRONMENT

The Intellec 8/Mod 80 microcomputer development system with its 8080 microprocessor CPU was chosen to develop and test the AUTODIN test program. The first reason for this choice was availability; however, many other reasons also existed. Among these were the wide use of 8080 CPU's (indeed, the 8080 has become an industry standard), the availability of software (such as high-level languages, debuggers, loaders, etc.) for program development and testing, and the ability to address up to 256 peripheral devices. In general, the 8080 is a single-chip, large-scale integrated (LSI) CPU which has 8 and 16-bit registers and can address up to 64K of main memory. References 4 and 5 provide more details on this subject.

The actual microcomputer used for development and testing of the AUTODIN test program was an Intellec 8 microcomputer with 8080 CPU, 16K of main memory, and two input/output (I/O) boards. The first I/O board was configured to permit interfacing with either a teletype or a cathode ray tube (CRT) terminal. The second I/O board was configured to work with

a Universal Asynchronous Receiver/Transmitter (UART). This hardware configuration obviously did not match the normal one found at a Mode I tributary station, which includes magnetic tape drives, card reader, card punch, paper tape reader, and line printer as well as a USART (as opposed to the UART available with the Intellec 8). In addition, in order to assure correctness of the AUTODIN test program, it was decided that tests with actual peripheral devices must be conducted. In order to conduct such tests, an equipment test configuration was developed.

First of all it was decided that the Intellec 8 could be tested back-to-back, with its transmit logic sending to its own receive logic to simulate the full duplex information transfer found on a Mode I communication channel between an ASC and tributary. In fact, as program development progressed, it became obvious that it made no difference whether or not the receiver was receiving information sent by itself (the same computer) or whether it was receiving information from a distant computer. The same was true for the transmitter. Only one minor logical difference became apparent: in using two machines, the receiver, at power-up, would attempt to achieve synchronization before permitting the transmitter to send anything. Obviously, if nothing was ever sent, then SYN characters could never be received. Consequently, for

the back-to-back configuration, a virtual bitstream was programmed into main memory, and part of the initialization of the program would insert SYN characters in this bitstream to achieve initial synchronization. Thereafter, the receive process would fetch bytes from this bitstream just as though it were interfacing with an actual USART. Conversely, the transmit process would insert bytes into this virtual bitstream just as though it were communicating with an actual USART. A side benefit of this method was that it eliminated use of the Intellec 8's USART. The USART was not used for two reasons. First, Mode I AUTODIN calls for synchronous vice asynchronous channel operation. Second, the USART available for testing was configured for seven-bit operation which precluded the use of eight-bit bytes. Eight-bit bytes with odd and even parity are mandatory for the AUTODIN logic. The use of the virtual bitstream concept solved both of these problems and did not cause an adverse effect on the timing considerations since the difference in processing time required to interface with a virtual bitstream and an actual USART is negligible. It is true that with an actual USART, the CPU might have to wait for a byte if the CPU were able to process bytes faster than the USART, or, if the CPU were slower than the USART, then a byte might be missed. However, this contingency was provided for by

conducting worst-case testing (see Section V.B for details). It is interesting to note that an analysis of test results showed that the 8080 CPU must execute an average of 574 instructions per received byte with a virtual bitstream and 573 instructions per received byte for an actual USART. Of these, 572 are identical, demonstrating the negligible difference between the two.

In addition to the virtual bitstream concept, a second aspect of the equipment test configuration had to be carefully thought out prior to programming and testing. This aspect was the matter of peripheral devices. The typical peripheral equipment configuration at a Mode I AUTODIN tributary usually consists of two magnetic tape drives (one for receive, one for transmit), a card reader, a card punch, a paper tape reader, and a line printer. Only two I/O ports, a teletype, and a CRT were available for testing. Since the teletype offered both a print capability for the receive function and a paper tape reader for the transmit function, it was selected over the CRT. The intention was to run tests of the algorithm using the teletype printer and reader simultaneously. This test was needed to check the correctness of the algorithm. However, the program was written so that, on incoming messages, the receiver would examine the SEL character, determine which output

device to select, select the output device, and then write the information on the selected output device (which in this case was always the teletype). In this way, the correctness of the algorithm could be tested without modifying the algorithm which would be used at an actual installation and without modifying the timing considerations. A similar argument holds for the transmit function: the program was made to check I/O ports for ready signals from nonexistent magnetic tape drives and card readers even though any actual input would always take place on the paper tape reader. Incorporating real hardware (such as magnetic tape drives) would require additional device driver routines and additional buffering. These requirements would increase the amount of main memory needed but would have a negligible impact on timing considerations.

By carefully considering all aspects of the hardware configuration prior to writing the program, it was possible to design a program which would be capable of being tested on the existing hardware but which also demonstrated the feasibility of a realistic AUTODIN tributary hardware configuration.

C. CHOOSING A PROGRAMMING LANGUAGE

PL/M, a block-structured, high-level systems language for the 8080 CPU was chosen as the language for developing the AUTODIN test program. There were four major reasons for choosing PL/M. The first reason was that the block structure and other logical constructs (such as if-then-else) facilitated the development of straightforward, efficient algorithms while freeing the programmer from unnecessary details which are often encountered in assembly language programming. The second reason was that, as a systems language, PL/M permits the programmer to control the 8080 just as closely as needed. Third, programs written in high-level languages are much easier to debug and maintain than large assembly language programs. Finally, the use of a high-level language would permit more rapid program development, an important consideration due to time constraints.

D. DESIGNING BY LEVELS

After defining the problem, developing an operational specification, understanding the hardware environment, and choosing a programming language, the next step that was taken was to begin designing the program in levels from the top down to the lowest levels. Much has been written

and spoken about structured programming and the top-down approach; however, in the author's opinion enough cannot be said. The author has used the top-down approach on several medium-size software projects with great success. Applying the approach to the AUTODIN test program also proved to be very successful: the entire project, from conception to successful testing took less than 15 weeks' part-time effort (see Section IV.G for details). It is believed that the reason for this success was due to using the top-down approach and modular, structured programming.

The highest level of the program was designed first, and the most time spent upon it. Correctness was insured at higher levels before proceeding to the design of lower ones. The reader may note that every procedure in the AUTODIN test program was labeled with a design level number. There were five design levels, with level one denoting the highest level and level five the lowest. As the design of the program began at the top level, it was discovered that the receive and transmit machines that were carefully developed in order to understand the AUTODIN protocol did not belong at the highest level of the program but rather at the second level. It became apparent that the actual implementation of these machines would require an operating system at the highest design level of the program to

coordinate and schedule the transmit and receive processes as well as other processes.

E. THE REQUISITE OPERATING SYSTEM

An analysis of the top level program requirements showed that, in addition to the transmit and receive processes, seven other processes were required to implement a functioning AUTODIN tributary. The nine processes are:

1. Receive logic process.
2. Transmit logic process (includes a reader process).
3. Poll peripheral devices process.
4. Poll receive side of USART process.
5. Poll transmit side of USART process.
6. Physical transmit process.
7. Writer process.
8. Operator input process.
9. Operator output process.

The functions of the receive and transmit processes were given in Chapter III of this thesis. The functions of the poll peripheral devices process were to poll the status of the local peripheral devices and to mark the devices as ready or not ready for input or output. Another important process was the poll receive side of USART process whose purpose was to indicate if a newly-received byte were in the USART

and ready for processing. Similarly, the poll transmit side of USART process had as its function to determine if the USART were ready to transmit the next byte. The purpose of the physical transmit process was to actually transfer bytes to the USART for transmission. The purpose of the writer process was to write incoming information onto the selected output device. The operator input process had as its function the input and interpretation of commands from the human operator. Finally, the operator output process had as its function the sending of alarm messages to the operator.

The management of these nine processes was the task of the highest level of the AUTODIN test program. It was necessary for this highest level to schedule the various processes and manage the corresponding peripheral and other devices. This scheduler in algorithmic form is shown below:

DO FOREVER:

```
CALL POLL USART RECEIVER PROCESS;  
IF RECEIVE LOGIC PROCESS IS SCHEDULED OR  
  RECEIVE LOGIC PROCESS DEVICE IS READY  
  THEN CALL RECEIVE LOGIC PROCESS;  
IF WRITER PROCESS IS SCHEDULED AND WRITER  
  PROCESS DEVICE IS READY  
  THEN CALL WRITER PROCESS;  
IF OPERATOR INPUT PROCESS DEVICE IS READY  
  THEN CALL OPERATOR INPUT PROCESS;  
IF OPERATOR OUTPUT PROCESS DEVICE IS READY  
  AND OPERATOR OUTPUT PROCESS IS SCHEDULED  
  THEN CALL OPERATOR OUTPUT PROCESS;  
CALL POLL USART TRANSMIT PROCESS;
```

```
IF TRANSMIT LOGIC PROCESS DEVICE IS READY  
AND SENDING IS TRUE  
THEN CALL TRANSMIT LOGIC PROCESS;  
CALL POLL PERIPHERAL DEVICES;  
END;
```

The above process scheduler was designed to utilize only polling to determine the status of devices. At first, some consideration was given to handling some of the devices (in particular, the receive side of the USART) on an interrupt basis. This could have been achieved since the 8080 CPU possesses an interrupt capability. However, careful analysis of the problem revealed that no advantage whatsoever was to be obtained from interrupt handling some or all of the devices. The main consideration was speed. When an incoming byte reaches the receive side of the USART, it remains there, ready for plucking by some process, for a time equal to eight times the reciprocal of the baud rate for synchronous operation and ten times the reciprocal of the baud rate for asynchronous operation. If the process scheduler can make one loop (performing all required tasks during this loop) and return to pluck the next byte from the receive side of the USART without ever losing a byte, then it will run fast enough to process a given baud rate. The rate at which the process scheduler can cycle through its DO FOREVER loop will be directly proportional to the baud rate it can handle, and this cycle rate is dependent upon the number of instructions

the CPU must perform per cycle. No gain in speed or efficiency can be obtained by interrupt processing in this case.

The actual implementation of the process scheduler may be found at the end of the AUTODIN program listing labeled program level one. It should be pointed out that the AUTODIN test program runs on the 8080 CPU without a resident operating system. In other words, the program contains its own, built-in operating system functions which consist of the process scheduler at level one of the program and the level five procedures which handle the actual input and output of the bytes. Program levels two, three, and four represent the various logic levels of the AUTODIN protocol and its associated processes such as writer, operator input, etc.

F. IMPLEMENTING THE RECEIVER AND TRANSMITTER PROCESSES

The next task to be performed in developing the AUTODIN test program was to implement the receiver and transmitter processes. These processes were well defined in Chapter III. Consequently, the task of implementing them was greatly simplified.

The receiver process (or RECEIVE\$LOGIC, as it was called in the AUTODIN program) was designed to be a nine-state machine and was implemented as a level two procedure which

consisted of a nine-part case statement. Each invocation of the procedure corresponds to waking up of the receive logic process. Based on the input of a newly-received byte and the condition of various flags, the receive logic will perform designated actions and will make a state transition before going back to sleep.

The transmit process (or TRANSMIT\$LOGIC, as it is called in the AUTODIN program) was designed to be a five-state machine and was implemented as a level two procedure which consisted of a five-part case statement. Each state (or case) was implemented as a level three procedure.

It is instructive to compare the state transition diagram of Figure 4 with the actual program as given in the listing. The procedure XMT\$STATE\$3 (contained in procedure TRANSMIT\$LOGIC) corresponds to state three of Figure 4. One of two possible transitions will be made from state three. If the byte just read from the selected input device corresponds to a correct LMF character, then the transmit logic will place that character in the third text slot of the outgoing line block, perform a table lookup to find the corresponding SEL character, and place the SEL character in the second framing position of the outgoing line block. Then, the transmit logic will set its new state to four and go to sleep until reawakened by the process scheduler. On

the other hand, if the newly-read byte does not match with a correct LMF character then the transmit logic will send an alarm to the operator, cancel the current message, set its new state to one (the start state), and go to sleep.

The fact that the program was designed in levels is illustrated by pointing out that in this example the job scheduler and device manager are at level one, the transmit logic process is at level two, the actions of transmit state three are at level three, the procedure which checks LMF's for transmit state three is a level four, and the simple procedures which actually input and output bytes are at level five.

G. TESTING AND DEBUGGING THE PROGRAM

The testing and debugging of the AUTODIN test program was performed with relative ease, a fact the author attributes to the top-down approach. Of the 15 weeks spent on the project (from inception to successful testing), seven were spent defining the problem and designing the uppermost levels of the program, five were spent in coding and program development, and three were spent in testing and debugging the program on the Intellec 8. The definition of the problem and design of the upper levels have been discussed previously and consequently will not be discussed here.

Coding and program development were greatly facilitated by the use of an interactive, time-share terminal connected to the IBM 360 system of the Naval Postgraduate School. This terminal provided three invaluable tools for program development: a powerful context editor, a PL/M compiler, and an 8080 simulator (called Interp 80). These tools facilitated rapid program development and permitted the design-by-level approach by allowing testing of program modules at each level of development. Interp 80 was particularly useful in program development. For example, the AUTODIN receiver logic was tested to see if it could correctly recover from error conditions (such as incorrectly received line blocks). Using Interp 80, it was simple to introduce errors in order to test the performance of the receiver logic under various error conditions. Of the five bugs found during program development, four were found using Interp 80 before attempting to test on the Intellec 8. The five program errors discovered were contained in a program of over 1700 lines of source code. This translates into approximately one error per 350 lines of code -- proof that the top-down approach can produce good software.

In addition to the above problem, a timing problem was encountered during testing with the teletype. This was caused by the extremely slow reaction speed of the teletype

as compared to the 8080 CPU. The problem was rectified by inserting delays into the program. Upon completion, the object program was approximately 6100 bytes in size.

V. FEASIBILITY TESTING

As previously mentioned, the AUTODIN test program was designed to be tested in two ways. The first test was performed with actual peripheral devices to test the correctness of the algorithm, and the second test was performed with all devices virtual to obtain timing results on the 8080 CPU. Changing from one type of testing to the other was accomplished by changing one line of source code.

A. RESULTS OF THE PERIPHERAL DEVICE TEST

This program demonstrated its ability to simultaneously input and output information using the teletype printer and paper tape reader. In addition, the program demonstrates its ability to send alarm messages to the operator. Appendix B shows an actual test message which was sent on the Intellec 8.

B. RESULTS OF THE TIMING TESTS

Three timing tests were conducted. In each of them, 180,000 bytes were processed, and the time required for this to be done was recorded. This was actually accomplished by starting the program, using a stopwatch, and having the 8080 go into machine halt after 180,000 received bytes.

Appendix C shows the calculations used to obtain baud rates from these measurements.

The first timing test consisted of running the program with virtual peripheral devices for 180,000 bytes to obtain an average baud rate. During the test, message traffic was always being transmitted and received. Thus, during each cycle of level one, the program was required to receive one byte, write one byte, read one byte, and transmit one byte (in addition to polling all peripheral devices, even though they were not used). The result of this timing test was 3354 baud.

The second timing test was exactly the same as the first one with one difference. In order to make the AUTODIN test program capable of being run with both actual and virtual peripheral devices, it was necessary to make numerous checks throughout the program for the virtual or actual conditions. This required additional time. Consequently, these checks were removed, and the program was again timed. This time, the result was 3723 baud, a slightly faster rate, as expected.

The third timing test took into account worst case conditions as opposed to the average conditions of the first two tests. This test was necessary because the virtual USART was used. When using a virtual USART, no received byte is ever lost. Thus, even though an average baud rate of 3723

was measured, there might be worst case conditions where the receiver logic was going through its worst case (most time-consuming) processing coincident with the transmit logic doing the same, while at the same time, the operator input, operator output, and writer processes all required attention. Analysis of the AUTODIN test program revealed that, for the receiver logic, performing state nine actions (second control character of a two-character control sequence) were most time-consuming. For the transmit logic, state three actions (performing a LMF lookup and LMF-to-SEL conversion) were the most time-consuming. These actions were more time-consuming than error recovery. Under these conditions, an actual USART running at 3723 baud would result in lost received bytes. Therefore, it was necessary to conduct a worst-case test of the AUTODIN test program where these most time-consuming actions were repeatedly performed. The result of this test was 2785 baud. An additional result was that, using an elapsed time of 517 seconds (See Appendix C) and an average instruction time of five microseconds, it was determined that the 8080 CPU executed an average of 574 instructions per received byte.

VI. CONCLUSIONS AND RECOMMENDATIONS

The AUTODIN test program is not an item of software ready for installation in AUTODIN tributary stations around the world. Rather, it was designed to demonstrate the feasibility of using a microcomputer to perform all of the functions required of a Mode I AUTODIN tributary station. In this regard, the AUTODIN test program fulfilled the purpose for which it was designed. By using conservative analysis techniques and taking into account worst-case processing requirements, it was shown that the 8080 CPU can perform all of the functions associated with an AUTODIN tributary station at modulation rates of 2400 baud. Furthermore, the AUTODIN test program required approximately 6000 (8-bit) words of main memory. Part of this memory requirement came from test parameters which need not be present in an actual working program. On the other hand, larger buffer sizes for interfacing with actual magnetic tape drives might be desirable. In addition, more main memory for certain nice-to-have features such as strings containing classification headings would be required. Nevertheless, it is conservatively estimated that 8194 words of main memory would handle the requirements for a fielded, working version of the program.

The result of all this is that AUTODIN communications can join the microcomputer revolution, and the revolution can be joined at a respectable baud rate of 2400. An 8080 CPU costs less than 30 dollars. An 8080 CPU, 8194 words of memory, power supply, cabinet, and I/O boards (with USART) cost less than one thousand dollars. There is absolutely no reason for continuing to lease expensive, large-scale computers at 60-80 thousand dollars per annum. It is true that the cost of peripherals must be added to the low cost of an 8080 based microcomputer system, but even with these costs added, the potential cost savings to the Federal Government are phenomenal. It is recommended that immediate attention be given to the official sanctioning and qualifying of microcomputers as DCS-approved equipment for AUTODIN use.

Another important implication of this thesis is the potential use of AUTODIN tributaries in mobile and tactical applications. Since microcomputers are so small and lightweight, they can be mounted in vehicles and aircraft to provide access to a worldwide digital information network. As defense management and weapon systems become more and more complex, the requirements for information in all forms (printed page, magnetic tape, floppy disk) at lower and lower echelons of command will increase. The use of microcomputers

will make it possible to expand the number of tributaries, giving more commanders at lower levels rapid access to the DCS. The field teletype can be replaced with a microcomputer connected to a lightweight line printer and (perhaps) a floppy disk unit, which will greatly improve the throughput rate and flexibility of communicated information. These are only a few of the potential applications. It is recommended that future development of field record communication systems take into account the use of microcomputers.

Finally, a side-product of this thesis was the state transition method of describing the AUTODIN protocol. This method proved to be vastly superior to the method used by DCA to describe AUTODIN. It is recommended that the state transition model be researched further, for it is felt that, with refinement, it could become a most effective method of describing communication protocols.

APPENDIX A
OPERATIONAL SPECIFICATION FOR AUTODIN
TEST PROGRAM

I. SYSTEM OVERVIEW

The purpose of the test program shall be to investigate the feasibility of using a microcomputer such as the Intel 8080 (or equivalent) as a Mode I block-by-block AUTODIN tributary station. To demonstrate feasibility, it will be necessary to program the microcomputer to perform all of the functions that a tributary normally performs. These functions include the duplex, simultaneous transmission and reception of information via input from magnetic tape, card, or paper tape and output via line printer (or teletype), card, or magnetic tape. In the feasibility demonstration, the aforementioned peripheral devices may be real or virtual. In addition, the simultaneous transmission and reception of information over a full-duplex communication channel via a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) must be accomplished. Furthermore, appropriate messages to the human operator must be sent whenever necessary. Although simultaneous transmission and reception is required, only one input device and one output device (which may be of the same or different type) may be selected

and in use at any point in time. In fact, the input device remains the same for each AUTODIN message transmitted; likewise for received messages and output devices. For example, the system might be simultaneously transmitting information from cards and receiving information which was being printed on a line printer.

In addition to performing the above tasks, the program must be designed such that the correctness of the algorithm may be tested by interfacing with actual peripheral devices. On the other hand, the program must be capable of being easily changed to work with virtual peripheral devices so that timing tests may be conducted. The reason for using virtual peripheral devices for timing tests is so that the central processing unit (CPU) of the microcomputer may run at full speed: the purpose of the program is to determine the speed at which a microcomputer can process AUTODIN messages and not to determine which input/output devices are rapid enough to function at AUTODIN tributaries.

III. PROCESSING REQUIREMENTS

A. RECEIVE PROCESSING

Incoming information arrives at the USART via a communication link and is transferred to the microprocessor, examined for parity correctness, stripped of control and

framing bytes, and transferred to the output device selected according to the SEL character in the incoming message. Acknowledgements (ACK1/ACK2) will be sent for correctly received line blocks; negative acknowledgements will be sent for incorrectly received line blocks. Synchronous idle will be recognizable by the receiver function, and notification of any loss of synchronization will be displayed to the operator.

B. TRANSMIT PROCESSING

When the human operator activates an input peripheral device for transmission by mounting a paper or magnetic tape or by loading a card deck into a card reader, the program must recognize that transmission is to begin. The program must begin transmission by reading the selected input device, building the line blocks for transmission, and must actually transmit the information, byte by byte, to the USART. Included in this operation is the insertion of proper parity and framing characters. In addition, the requisite coordination between the transmit and receive functions must occur so that proper channel coordination takes place according to the AUTODIN protocol.

APPENDIX B

ACTUAL TEST MESSAGE

RTTUZYUW RUWJAGC0000 2122200-UUUU--RUWJAGC.
ZNP UUUUUU
R 302200 JUL 76
FM NTCC MONTEREY CA
TO NTCC MONTEREY CA

BT
UNCLAS//NO0000//
THIS TEST MESSAGE DEMONSTRATES THE ABILITY OF THE 8080
CPU AND INTELLEC 8 MICROCOMPUTER TO PROCESS AUTODIN
MESSAGES. ALTHOUGH THIS PARTICULAR MESSAGE IS PRINTED
ON A TELETYPE, THE 8080 IS CAPABLE OF INTERFACING WITH
OTHER PERIPHERAL DEVICES SUCH AS LINE PRINTERS, MAGNETIC
TAPE DRIVES, AND CARD PUNCHES/READERS. THE POTENTIAL
USE OF MICROCOMPUTERS FOR AUTODIN APPLICATIONS HAS TWO
MAJOR IMPACTS:

- 1) CONSIDERABLE COST SAVINGS MAY RESULT FROM
USING MICROCOMPUTERS IN PLACE OF LARGER,
MORE EXPENSIVE COMPUTERS.
- 2) THE POSSIBLE USE OF LIGHTWEIGHT, RUGGEDIZED
COMMUNICATIONS TRIBUTARIES FOR USE IN
TACTICAL SITUATIONS CAN GREATLY IMPROVE
FIELD RECORD COMMUNICATIONS.

END OF TEST MESSAGE.
BT
#0000

WVV

APPENDIX C

TIMING TEST CALCULATIONS

TEST 1: CONSTANT CHECKING FOR VIRTUAL/ACTUAL DEVICES

ELAPSED TIME = 429.4 seconds

$$\frac{180,000 \text{ bytes}}{429.4 \text{ seconds}} \times 8 \text{ bits/byte} = 3354(\pm 25) \text{ baud}$$

TEST 2: NO CHECKING FOR VIRTUAL/ACTUAL DEVICES

ELAPSED TIME = 386.8 seconds

$$\frac{180,000 \text{ bytes}}{386.8 \text{ seconds}} \times 8 \text{ bits/byte} = 3723(\pm 25) \text{ baud}$$

TEST 3: WORST CASE PROCESSING

ELAPSED TIME = 517.0 seconds

$$\frac{180,000 \text{ bytes}}{517.0 \text{ seconds}} \times 8 \text{ bits/byte} = 2785(\pm 25) \text{ baud}$$

AUT000010
 AUT000020
 AUT000030
 AUT000040
 AUT000050
 AUT000060
 AUT000070
 AUT000080
 AUT000090
 AUT000100
 AUT000110
 AUT000120
 AUT000130
 AUT000140
 AUT000150
 AUT000160
 AUT000170
 AUT000180
 AUT000190
 AUT000200
 AUT000210
 AUT000220
 AUT000230
 AUT000240
 AUT000250
 AUT000260
 AUT000270
 AUT000280
 AUT000290
 AUT000300
 AUT000310
 AUT000320
 AUT000330
 AUT000340
 AUT000350
 AUT000360
 AUT000370
 AUT000380
 AUT000390
 AUT000400
 AUT000410
 AUT000420
 AUT000430
 AUT000440
 AUT000450
 AUT000460
 AUT000470
 AUT000480

100H: /* THE PROGRAM WILL BE LOADED IN THE 8080
 BEGINNING AT 100 HEX */

/* A U T O D I N

PROGRAMMER: GORDON E. ANDERSON.
 SOURCE COMPUTER: IBM 360 (USED FOR COMPILATION).
 OBJECT COMPUTER: INTELLEC 8 (USED FOR EXECUTION).
 CLASSIFICATION: UNCLASSIFIED.
 SOURCE LANGUAGE: PLM.
 DATES: AUGUST - NOVEMBER 1976.

THE PURPOSE OF THIS PROGRAM IS TO DEMONSTRATE THE FEASIBILITY
 OF USING A MICROCOMPUTER, SUCH AS THE INTELLEC 8 WITH 8080 CPU, FOR
 A DEFENSE COMMUNICATION SYSTEM (DCS) MODE I AUTODIN TRIBUTARY STATION
 (BLOCK BY BLOCK MODE OF OPERATION). THE PROGRAM IS DESIGNED TO PERFORM
 ALL OPERATOR INTERFACES, LOCAL PERIPHERAL INTERFACES, COMMUNICATION
 INTERFACES AS WELL AS THE LOGICAL INTERFACE WITH AN THIS
 AUTOMATIC SWITCHING CENTER (ASC). THE PROGRAM WILL PERFORM THIS
 INTERFACING INDEPENDENT OF THE RATE OF INFORMATION TRANSFER
 (BAUD RATE OR BITS PER SECOND). THIS WILL PERMIT TESTING
 OF THE PROGRAM AT DIFFERENT BAUD RATES IN ORDER TO DETERMINE
 THE MAXIMUM SPEED AT WHICH THE MICROCOMPUTER CAN PROCESS
 AUTODIN TRAFFIC.

THIS PROGRAM RUNS ON THE "BARE MACHINES" THAT IS
 IT DOES NOT REQUIRE A RESIDENT OPERATING SYSTEM; RATHER,
 IT CONTAINS ITS OWN OPERATING SYSTEM WHICH PERMITS MULTITASKING
 PROCESSING OF THE INDEPENDENT PROCESSES WHICH MAKE UP THE
 TOTAL AUTODIN PROGRAM. PROCESS SCHEDULING AND DEVICE MANAGE-
 MENT ARE PERFORMED AT THE HIGHEST LEVEL OF THE PROGRAM.
 THE NINE MAJOR PROCESSES WHICH ARE SCHEDULED ARE:

1. POLL\$USAR
2. RECEIVER\$LOGIC
3. WRITER
4. OPERATOR\$INPUT
5. OPERATOR\$OUTPUT
6. TRANSMIT\$LOGIC
7. TRANSMITTER
8. POLL\$DEVICES
9. POLL\$USAT

IT IS IMPORTANT TO NOTE THAT EACH PROCESS IS PERMITTED TO RUN FOR ONE BYTE AT A TIME; CONSEQUENTLY, SIMULTANEOUS TRANSMISSION, RECEPTION, INPUT, AND OUTPUT MAY TAKE PLACE.

WHEN READING THE PROGRAM, BEGIN AT THE HIGHEST LEVEL, LEVEL 1 AND WORK THROUGH THE LEVELS 1, 2, 3, ETC. SINCE THE HIGHER LEVEL PROCEDURES INVOKE THOSE OF LOWER LEVEL (AND LARGER NUMBER). ALL PROCEDURES ARE LABELLED WITH THEIR LEVEL NUMBER. THE HIGHEST LEVEL, LEVEL 1, IS LOCATED AT THE BOTTOM OF THE PROGRAM LISTING.

FINALLY, EXTENSIVE USE OF THE LITERAL SUBSTITUTION CAPABILITY OF PIM HAS BEEN MADE. THIS PERMITS THE PROGRAM TO BE READ IN ALGEBRAIC ENGLISH FORM BY THOSE WHO ARE INTERESTED IN ITS LOGICAL STRUCTURE. THOSE INTERESTED IN BIT MANIPULATIONS AND OTHER DETAILS MAY STUDY THE LITERAL DECLARATIONS. */

/* ***** LITERAL DECLARATIONS ***** */

DECLARE /* THE FOLLOWING LITERALS ARE USED FOR READABILITY */

```
TRUE LITERALLY '1',
FALSE LITERALLY '0',
SPACE LITERALLY '20H',
FORFEVER LITERALLY 'WHILE TRUE',
NCT$READY LITERALLY '0',
REALY LITERALLY '1',
FOOT LITERALLY '0',
SCHEDULE$READY LITERALLY '0',
LP LITERALLY '10',
CR LITERALLY '13',
EVEN LITERALLY ' > 600',
LOW$7$MASK LITERALLY '7PH',
PARTY$MASK LITERALLY '80H',
TTY LITERALLY '0',
MAG$TAPE LITERALLY '2',
CARD$PUNCH LITERALLY '3',
TTY$INFO LITERALLY '0',
TTY$STATUS LITERALLY '1',
CRT$INFO LITERALLY '0',
CRT$STATUS LITERALLY '1',
```

```
/* USED TO REMOVE PARITY BITS
/* USED TO INSERT PARITY BITS
/* USED FOR TELETYPE PORT
/* USED FOR MAG TAPE I/O PORT
/* USED FOR CARD PUNCH I/O PORT
/* USED FOR TTY I/O PORT
/* USED FOR TTY STATUS PORT
/* USED FOR CRT I/O PORT
/* USED FOR CRT STATUS PORT
```

AUT000490
 AUT000500
 AUT000510
 AUT000520
 AUT000530
 AUT000540
 AUT000550
 AUT000560
 AUT000570
 AUT000580
 AUT000590
 AUT000600
 AUT000610
 AUT000620
 AUT000630
 AUT000640
 AUT000650
 AUT000660
 AUT000670
 AUT000680
 AUT000690
 AUT000700
 AUT000710
 AUT000720
 AUT000730
 AUT000740
 AUT000750
 AUT000760
 AUT000770
 AUT000780
 AUT000790
 AUT000800
 AUT000810
 AUT000820
 AUT000830
 AUT000840
 AUT000850
 AUT000860
 AUT000870
 AUT000880
 AUT000890
 AUT000900
 AUT000910
 AUT000920
 AUT000930
 AUT000940
 AUT000950
 AUT000960


```

RCV$$STATE BYTE,
RCV$$REP BYTE,
SYNCH$RCV$BYTE,
RCV$$FLAG BYTE,
RCV$$CTR BYTE,
CHECK$RCV$TIMER BYTE,
RCV$$SYNCH$TIMER ADDRESS,
RCV$$ID$MSG BYTE,
RCV$$ERR$CTR BYTE,
RCV$$RCV$STATE BYTE,
PREV$ACK$1$2 STATE BYTE,
BLOCK$START$STATE BYTE,
RCV$$REP$CTR BYTE,
OUTFUT$DEVICE BYTE,
STILL$WRITING BYTE,

XMT$$STATE BYTE,
XMT$$BP BYTE,
XMT$$XT$CTR BYTE,
XMT$$WAIT$1$2 BYTE,
XMT$$ANS$TIMER ADDRESS,
CHECK$ANS$CTR BYTE,
XMT$$REP$CTR BYTE,
INOUT$DEVICE BYTE,
SENDING$FLAG BYTE,
AWAITING$ACK BYTE,

OUTFUT$STATE BYTE,
ALARM$MSG BYTE,
EOM$STATE BYTE,
LP$CTR BYTE,
N$CTR BYTE,
VIRTUAL BYTE,

RW$EUFFER$1 (81) BYTE,
RW$EUFFER$2 (81) BYTE,
RCV$$EUFFER$ADDR ADDRESS,
(RCV$EUFFER$BASED RCV$EUFFER$ADDR) (81) BYTE,
WRITE$EUFFER$ADDR ADDRESS,

** THE STATE OF THE RECEIVER LOGIC PROCESS
** THE BLOCK PARITY OF THE SYN RECEIVED IN A ROW
** COUNTS THE NUMBER OF SYN RECEIVED IN A ROW
** SAVES THE PREVIOUSLY RECEIVED BYTE O.W.
** TRUE=DATA FORMAT MODE CHANGE; FALSE O.W.
** COUNTS NUMBER OF WBT RECEIVED IN A ROW
** COUNTS NUMBER OF NAK RECEIVED IN A ROW
** * TRUE MEANS CHECK OFF, THE RECEIVE TIMER LOST
** * IF TIMER GOES OFF, SYNCH HAS BEEN LOST
** TRUE WHEN RCVR IS IN THE MIDDLE OF A MSG
** COUNTS IF AN ERROR HAS BEEN FOUND IN BLOCK
** SAVES THE PREVIOUS STATE; 2=ACK2 NEXT
** 1=ACK1 TO BE RCVD OF THE RCVR
** * RCVR STATE AT BEGINNING OF BLOCK
** COUNTS NUMBER OF REP RECEIVED IN A ROW
** RECEIVER SELECTS BASED ON RCVD SEL
** TRUE MEANS WRITER IS STILL WRITING

** THE STATE OF THE TRANSMIT LOGIC PROCESS
** THE BYTE JUST READ IN AND TO BE XMTTR
** KEEPS TRACK OF BLOCK IN PARITY FOR XMTTR
** COUNTS TEXT FOR OUTGOING LINE BLOCKS
** TRUE=XMTTR WAITING FOR ACK'S
** 1= SEND ACK1 NEXT, ANSWER NOT RCVD SOON ENOUGH
** * IF TRUE MEANS TO CHECK THE XMT TIMER
** * KEEPS TRACK OF REPS SENT IN A ROW
** READ THIS DEVICE; TRANSMIT IT'S DATA
** TRUE MEANS WE'RE SENDING A MSG NOW
** TRUE MEANS WE'RE CANCELLING XMTD MSG
** TRUE=WAITING FOR LAST BLOCK TO BE ACKD

** USED TO PERFORM OPR OUTPUTS
** USED TO SEND ONE LTR ALARM TO OPR
** USED TO CHECK FOR END OF MSG 'EOM'
** USED TO COUNT LINE FEEDS (LOOKS FOR 8)
** USED TO COUNT N'S (LOOKS FOR 4 IN A ROW)
** TRUE MEANS READ/WRITE IN CORE; FALSE MEANS
** READ/WRITE ON ACTUAL PERIPHERAL DEVICES

** UTILITY BUFFER FOR RECEIVE LOGIC AND
** WRITER PROCESS
** SAME PURPOSE AS RW$EUFFER$1
** SETS ADDR OF THE CURRENT RCV BUFFER
** (RCV$EUFFER$ADDR) (81) BYTE
** CURRENT BUFFER USED BY RCV LOGIC

```

AUTO11450
 AUTO11460
 AUTO11470
 AUTO11480
 AUTO11490
 AUTO11500
 AUTO11510
 AUTO11520
 AUTO11530
 AUTO11540
 AUTO11550
 AUTO11560
 AUTO11570
 AUTO11580
 AUTO11590
 AUTO11600
 AUTO11610
 AUTO11620
 AUTO11630
 AUTO11640
 AUTO11650
 AUTO11660
 AUTO11670
 AUTO11680
 AUTO11690
 AUTO11700
 AUTO11710
 AUTO11720
 AUTO11730
 AUTO11740
 AUTO11750
 AUTO11760
 AUTO11770
 AUTO11780
 AUTO11790
 AUTO11800
 AUTO11810
 AUTO11820
 AUTO11830
 AUTO11840
 AUTO11850
 AUTO11860
 AUTO11870
 AUTO11880
 AUTO11890
 AUTO11900
 AUTO11910
 AUTO11920

```

/* ADDRESS OF CURRENT WRITE BUFFER
(WRITE$BUFFER BASED WRITE$BUFFER$ADDR) (81) BYTE
W$BUFFER$PTR BYTE,
/* CURRENT BUFFER FOR WRITE PROCESS
/* POINTER FOR WRITE$BUFFER
*/
AUT01930
AUT01940
AUT01950
AUT01960
AUT01970
AUT01980
AUT01990
AUT02000
AUT02010
AUT02020
AUT02030
AUT02040
AUT02050
AUT02060
AUT02070
AUT02080
AUT02090
AUT02100
AUT02110
AUT02120
AUT02130
AUT02140
AUT02150
AUT02160
AUT02170
AUT02180
AUT02190
AUT02200
AUT02210
AUT02220
AUT02230
AUT02240
AUT02250
AUT02260
AUT02270
AUT02280
AUT02290
AUT02300
AUT02310
AUT02320
AUT02330
AUT02340
AUT02350
AUT02360
AUT02370
AUT02380
AUT02400

/* TWO UTILITY BUFFERS USED FOR XMT
/* AND TRANSMITTER PROCESSES
XMT$LOGIC$BUFFER$ADDR ADDRESS
/* SETS ADDR OF CURRENT XMT LOGIC BUFFER*
(XMT$LOGIC$BUFFER BASED XMT$LOGIC$BUFFER$ADDR) (85) BYTE
/* CURRENT BUFFER USED BY XMT LOGIC
*/
AUT02010
AUT02020
AUT02030
AUT02040
AUT02050
AUT02060
AUT02070
AUT02080
AUT02090
AUT02100
AUT02110
AUT02120
AUT02130
AUT02140
AUT02150
AUT02160
AUT02170
AUT02180
AUT02190
AUT02200
AUT02210
AUT02220
AUT02230
AUT02240
AUT02250
AUT02260
AUT02270
AUT02280
AUT02290
AUT02300
AUT02310
AUT02320
AUT02330
AUT02340
AUT02350
AUT02360
AUT02370
AUT02380
AUT02400

/* SETS ADDR OF CURRENT TRANSMIT BUFFER
/* (85) BYTE
(TRANSMIT$BUFFER BASED TRANSMIT$BUFFER$ADDR) (85) BYTE
/* TRANSMIT PROCESS BUFFER
/* POINTER FOR TRANSMIT BUFFER
/* CURRENT LENGTH OF TRANSMIT BUFFER
*/
AUT02010
AUT02020
AUT02030
AUT02040
AUT02050
AUT02060
AUT02070
AUT02080
AUT02090
AUT02100
AUT02110
AUT02120
AUT02130
AUT02140
AUT02150
AUT02160
AUT02170
AUT02180
AUT02190
AUT02200
AUT02210
AUT02220
AUT02230
AUT02240
AUT02250
AUT02260
AUT02270
AUT02280
AUT02290
AUT02300
AUT02310
AUT02320
AUT02330
AUT02340
AUT02350
AUT02360
AUT02370
AUT02380
AUT02400

/* BUFFER FOR THE CONTROL CHARS TO SEND
/* POINTER FOR CONTROL CHAR BUFFER
/* POINTER FOR CONTROL CHAR BUFFER
*/
AUT02010
AUT02020
AUT02030
AUT02040
AUT02050
AUT02060
AUT02070
AUT02080
AUT02090
AUT02100
AUT02110
AUT02120
AUT02130
AUT02140
AUT02150
AUT02160
AUT02170
AUT02180
AUT02190
AUT02200
AUT02210
AUT02220
AUT02230
AUT02240
AUT02250
AUT02260
AUT02270
AUT02280
AUT02290
AUT02300
AUT02310
AUT02320
AUT02330
AUT02340
AUT02350
AUT02360
AUT02370
AUT02380
AUT02400

/* THE FOLLOWING FOUR VECTORS ARE USED TO STORE RECEIVE
LOGIC TABLE LOOKUP ACTIONS*/
TABLE$1 DATA (0,1,2,2,2,2,3,3,3,3,3,2,2,3,3,3,2,2,3,4,2,3,2,2,2,3,2,3,2)
TABLE$2 DATA (0,2,2,2,2,3,3,3,3,3,2,2,3,4,2,3,2,2,2,3,2,3,2,3,2)
TABLE$3 DATA (0,6,6,6,6,6,3,3,3,3,3,6,6,3,7,8,6,3,6,3,9)
TABLE$4 DATA (0,10,10,11,10,10,3,3,3,3,3,5,5,10,10,5,12,13,3,10,10)
/* THE FOLLOWING TWO VECTORS ARE USED TO STORE THE CORRECT
/* SEL CHARACTERS SO THAT THE TRANSMIT LOGIC CAN INSERT
/* THE CORRECT SEL CHARACTER BASED ON THE LMF CHARACTER
/* OF THE MESSAGE BEING TRANSMITTED */
SEL$LOOKUP$1 DATA ('A','B','D','C','0','H','A','0','B')
SEL$LOOKUP$2 DATA ('A','A','A','C','H')
/* DECLARATIONS FOR TEST AND DEBUG */
DECLARE
/* THE FOLLOWING DECLARATIONS ARE FOR ODD PARITY TEXT */
CHARACTERS WHICH ARE USED IN A TEST MESSAGE
*/

```



```

AUTO03370
AUTO03380
AUTO03390
AUTO03400
AUTO03410
AUTO03420
AUTO03430
AUTO03440
AUTO03450
AUTO03460
AUTO03470
AUTO03480
AUTO03490
AUTO03500
AUTO03510
AUTO03520
AUTO03530
AUTO03540
AUTO03550
AUTO03560
AUTO03570
AUTO03580
AUTO03590
AUTO03600
AUTO03610
AUTO03620
AUTO03630
AUTO03640
AUTO03650
AUTO03660
AUTO03670
AUTO03680
AUTO03690
AUTO03700
AUTO03710
AUTO03720
AUTO03730
AUTO03740
AUTO03750
AUTO03760
AUTO03770
AUTO03780
AUTO03790
AUTO03800
AUTO03810
AUTO03820
AUTO03830
AUTO03840

```

```

OAH OAH,'N','N','N','N').;
DECLARE R$TEST$PTR ADDRESS;

/* PAPER IS A VECTOR WHICH SERVES AS A VIRTUAL PAPER */
DECLARE PAPER(401) BYTE;

/* THE FOLLOWING DECLARATIONS ARE FOR POINTERS WHICH
   REQUIRED FOR THE VIRTUAL BUFFERS PREVIOUSLY DECLARED */
DECLARE S$TEST$PTR ADDRESS;
DECLARE T$TEST$PTR BYTE;
DECLARE PAPER$PTR ADDRESS;
DECLARE EYET$COUNTER BYTE;
DECLARE TEST$COUNTER BYTE;
DECLARE L$DEBUG$FLAG BYTE;

/* END OF TEST/DEBUG DECLARATIONS */

/* ***** GLOBAL UTILITY PROCEDURES ***** */
/* ***** GLOBAL UTILITY PROCEDURES ***** */

CRT$OUT:PROCEDURE(CHARACTER);
/* LEVEL 5 PROCEDURE */

/* SENDS ONE CHARACTER TO THE CONSOLE DEVICE */
DECLARE CHARACTER BYTE;
IF NOT VIRTUAL THEN
  GO;
END;

DO WHILE NOR(INPUT(CRT$$STATUS),2);
END; /* WAIT UNTIL CONSOLE IS READY */
OUTPUT(CRT$INFO)=NOT CHARACTER;

END;

END CRT$OUT;
CRT$IN: PROCEDURE BYTE;
/* LEVEL 5 PROCEDURE */
/* INPUTS ONE BYTE FROM THE CONSOLE DEVICE */
RETURN (NOT INPUT(CRT$INFO)) AND LOW$7$HASK;

```

```

END CRT$IN;

SEND$CC: PROCEDURE(CTRL$CHAR);
/* LEVEL 5 PROCEDURE */
/* PUTS TWO CONTROL CHARACTERS IN THE BUFFER TO BE SENT */
DECLARE CTRL$CHAR BYTE;
CONCHAR$BUFFER(CC$PTR2) CONCHAR$BUFFER(CC$PTR2 + 1) = CTRL$CHAR;
IF (CC$PTR2 = CC$PTR2 + 2) > 7 THEN
  CC$PTR2 = 0;

END SEND$CC;

INITIALIZE$RCVR: PROCEDURE;
/* LEVEL 5 PROCEDURE */
/* INITIALIZES THE RECEIVE LOGIC AND WRITER PROCESSES */
CHECK$RCV$TIMER = TRUE;
MC$FLAG, RCV$MID$MSG, RCV$ERROR, STILL$WRITING = FALSE;
YMT$ACK$1$2, N$BUFFER$PTR, PREV$RCV$STATE, BLOCK$START$STATE = 1;
RCV$BP, SYN$CTR, WBT$CTR, NAK$CTR, RCV$SYNCH$TIMER,
RCV$TYF$CTR, RCV$REP$CTR, RW$BUFFER$1, RW$BUFFER$2 = 0;
CUTEUT$DEVICE = NOT$SELECTED;
RCV$EUFFER$ADDR, WRITE$BUFFER$ADDR = . RW$BUFFER$1;
RCV$STATE = 5;
RECEIVE$LOGIC$PROCESS(SCHEDULED) = TRUE;
WRITER$PROCESS(SCHEDULED) = FALSE;
IF NOT VIRTUAL THEN
  DO;
    RCV$STATE = 1;
    RECEIVE$LOGIC$PROCESS(SCHEDULED) = FALSE;
  END;
END INITIALIZE$RCVR;

```

```

AUT03850
AUT03860
AUT03870
AUT03880
AUT03890
AUT03900
AUT03910
AUT03920
AUT03930
AUT03940
AUT03950
AUT03960
AUT03970
AUT03980
AUT03990
AUT04000
AUT04010
AUT04020
AUT04030
AUT04040
AUT04050
AUT04060
AUT04070
AUT04080
AUT04090
AUT04100
AUT04110
AUT04120
AUT04130
AUT04140
AUT04150
AUT04160
AUT04170
AUT04180
AUT04190
AUT04200
AUT04210
AUT04220
AUT04230
AUT04240
AUT04250
AUT04260
AUT04270
AUT04280
AUT04290
AUT04300
AUT04310
AUT04320

```

AUT04330
 AUT04340
 AUT04350
 AUT04360
 AUT04370
 AUT04380
 AUT04390
 AUT04400
 AUT04410
 AUT04420
 AUT04430
 AUT04440
 AUT04450
 AUT04460
 AUT04470
 AUT04480
 AUT04490
 AUT04500
 AUT04510
 AUT04520
 AUT04530
 AUT04540
 AUT04550
 AUT04560
 AUT04570
 AUT04580
 AUT04590
 AUT04600
 AUT04610
 AUT04620
 AUT04630
 AUT04640
 AUT04650
 AUT04660
 AUT04670
 AUT04680
 AUT04690
 AUT04700
 AUT04710
 AUT04720
 AUT04730
 AUT04740
 AUT04750
 AUT04760
 AUT04770
 AUT04780
 AUT04790
 AUT04800

```

INITIALIZE$XMT: PROCEDURE;
/* LEVEL 5 PROCEDURE */
/* INITIALIZES THE TRANSMIT LOGIC AND TRANSMITTER PROCESSES */
XMT$WAIT, CHECK$ANS$TIMER, CAN$FLAG, SENDING,
AWAITING$ACK=FALSE;
RCV$ACK$1$2, XMT$STATE, T$BUFFER$PTR, EOM$STATE=1;
XMT$BP, XMT$ANS$TIMER, XMT$REP$CTR, T$BUFFER$1,
T$BUFFER$2, LP$CTR, N$CTR=0;
INPUT$DEVICE=NOT$SELECTED;
XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$2;
TRANSMIT$BUFFER$ADDR=.T$BUFFER$1;
TRANSMIT$LOGIC$PROCESS (DEVICE$READY) =FALSE;
END INITIALIZE$XMT;
ALARM: PROCEDURE (CONDITION);
/* LEVEL 5 PROCEDURE */
/* SENDS AN APPROPRIATE ALARM MESSAGE BY SCHEDULING THE
OPERATOR OUTPUT PROCESS. THE ALARM CONDITION IS COMMUNICATED
VIA GLOBAL VARIABLE 'ALARM$MSG'. RE-INITIALIZATION OF
OF THE RECEIVER OR TRANSMITTER IS PERFORMED WHEN NECESSARY. */
LECLARE CONDITION BYTE;
DO CASE CONDITION;
: /* CASE 0 NOT USED */
DO; /* CASE 1 -- 3 NAK'S RECEIVED IN A ROW */
CALL SEND$CC(CAN);
ALARM$MSG='N';
CALL INITIALIZE$XMT;
END;
DO; /* CASE 2 -- 3 WBT'S RCVD IN A ROW */
CALL SEND$CC(CAN);
ALARM$MSG='W';

```

AUT04810
 AUT04820
 AUT04830
 AUT04840
 AUT04850
 AUT04860
 AUT04870
 AUT04880
 AUT04890
 AUT04900
 AUT04910
 AUT04920
 AUT04930
 AUT04940
 AUT04950
 AUT04960
 AUT04970
 AUT04980
 AUT04990
 AUT05000
 AUT05010
 AUT05020
 AUT05030
 AUT05040
 AUT05050
 AUT05060
 AUT05070
 AUT05080
 AUT05090
 AUT05100
 AUT05120
 AUT05130
 AUT05140
 AUT05150
 AUT05160
 AUT05170
 AUT05180
 AUT05190
 AUT05200
 AUT05210
 AUT05220
 AUT05230
 AUT05240
 AUT05250
 AUT05260
 AUT05270
 AUT05280

```

END: CALL INITIALIZE$XMR;
DO: /* CASE 3 -- CAN RCVD FROM THE ASC */
  ALARM$MSG='C';
  CALL INITIALIZE$RCVR;
END:
/* CASE 4 -- REP RECEIVED BETWEEN MSGS */
ALARM$MSG='R';
DO: /* CASE 5 -- RM RECEIVED */
  CALL SEND$CC(CAN);
  ALARM$MSG='H';
  CALL INITIALIZE$RCVR;
END:
/* CASE 6 -- INV RCVD */
ALARM$MSG='I';
DO: /* CASE 7 -- 3 REP'S SENT IN A ROW */
  CALL SEND$CC(CAN);
  ALARM$MSG='3';
  CALL INITIALIZE$XMR;
END:
/* CASE 8 -- INVALID LMF ON OUTGOING MSG */
ALARM$MSG='L';
  CALL INITIALIZE$XMR;
END:
/* CASE 9 FLASH; SEND BELL AND F TO CONSOLE */
ALARM$MSG='P';
/* CASE 10 - LOSS OF SYNCH AND SYNC TIMER EXPD */
ALARM$MSG='S';
/* CASE 11 - USAR DEAD */
ALARM$MSG='D';
END: /* OF CASE CONDITION */
OUTFUT$STATE=1;
OPEFATOR$OUTPUT$PROCESS(SCHEDULED)=TRUE;
END ALARM;
GET$RCVD$BYTE: PROCEDURE BYTE;

```

AUT05290
 AUT05300
 AUT05310
 AUT05320
 AUT05330
 AUT05340
 AUT05350
 AUT05360
 AUT05370
 AUT05380
 AUT05390
 AUT05400
 AUT05410
 AUT05420
 AUT05430
 AUT05440
 AUT05450
 AUT05460
 AUT05470
 AUT05480
 AUT05490
 AUT05500
 AUT05510
 AUT05520
 AUT05530
 AUT05540
 AUT05550
 AUT05560
 AUT05570
 AUT05580
 AUT05590
 AUT05600
 AUT05610
 AUT05620
 AUT05630
 AUT05640
 AUT05650
 AUT05660
 AUT05670
 AUT05680
 AUT05690
 AUT05700
 AUT05710
 AUT05720
 AUT05730
 AUT05740
 AUT05750
 AUT05760

```

/* LEVEL 5 PROCEDURE */
RCV$$SYNCH$$TIMER=RCV$$SYNCH$$TIMER + 1;
IF CHECK$$ANS$$TIMER THEN
  XMT$$ANS$$TIMER=XMT$$ANS$$TIMER + 1;
ELSE XMT$$ANS$$TIMER=0;
/* THESE TWO COUNTERS ARE BUMPED EACH TIME A BYTE IS RECEIVED.
WHEN THEY GET OVER 600 IN VALUE, THE TIMERS HAVE EXPIRED. */
USAR$$CHECK=0; /* JUST GOT A BYTE; SO USAR IS GOOD */

IF VIRTUAL THEN
DO:
EYI$$COUNTER=BYTE$$COUNTER + 1;
IF (R$$TEST$$PTR = R$$TEST$$PTR + 1) > 300 THEN
  R$$TEST$$PTR=0;
  RETURN TEST$$MSG(R$$TEST$$PTR);
END:
ELSE RETURN NOT.(INPUT(USART$$INFO));
END GET$$RCVD$$BYTE;

WRITE$$BYTE: PROCEDURE (CHARACTER);
/* LEVEL 5 PROCEDURE */
DECLARE CHARACTER, BYTE;
IF VIRTUAL THEN
DO:
PAPER(PAPER$$PTR)=CHARACTER;
PAPER$$PTR=PAPER$$PTR + 1;
IF PAPER$$PTR > 400 THEN
  PAPER$$PTR=0;
END:
ELSE DO:
IF OUTPUT$$DEVICE=TTY THEN
DO:
DO WHILE FOR(INPUT(TTY$$STATUS),2);READY */
END:
OUTPUT(TTY)=NOT CHARACTER;
END:
ELSE IF OUTPUT$$DEVICE=MAGTAPE THEN

```

```

AUT05770
AUT05780
AUT05790
AUT05800
AUT05810
AUT05820
AUT05830
AUT05840
AUT05850
AUT05860
AUT05870
AUT05880
AUT05890
AUT05900
AUT05910
AUT05920
AUT05930
AUT05940
AUT05950
AUT05960
AUT05970
AUT05980
AUT05990
AUT06000
AUT06010
AUT06020
AUT06030
AUT06040
AUT06050
AUT06060
AUT06070
AUT06080
AUT06090
AUT06100
AUT06110
AUT06120
AUT06130
AUT06140
AUT06150
AUT06160
AUT06170
AUT06180
AUT06190
AUT06200
AUT06210
AUT06220
AUT06230
AUT06240

```

```

ELSE OUTPUT (MAG$TAPE) = NOT CHARACTER;
ELSE OUTPUT (CARD$PUNCH) = NOT CHARACTER;

```

```
END;
```

```
END WRITE$BYTE;
```

```
SEND$EYE: PROCEDURE (CHARACTER);
```

```
/* LEVEL 5 PROCEDURE */
```

```
DECLARE CHARACTER BYTE;
```

```
IF VIRTUAL THEN
```

```
DO;
```

```
IF (S$TEST$PTR = S$TEST$PTR + 1) > 300 THEN
S$TEST$PTR = 0;
```

```
TEST$MSG(S$TEST$PTR) = CHARACTER;
```

```
END;
```

```
ELSE OUTPUT (USART$INFO) = NOT CHARACTER;
```

```
END SEND$BYTE;
```

```
GET$BYTE$TO$XMT: PROCEDURE BYTE;
```

```
/* LEVEL 5 PROCEDURE */
```

```
IF VIRTUAL THEN
```

```
DO;
```

```
IF (T$TEST$PTR = T$TEST$PTR + 1) > 115 THEN
T$TEST$PTR = 0;
```

```
RETURN TEST$TAPE(T$TEST$PTR);
```

```
END;
```

```
ELSE DO;
```

```
IF INPUT$DEVICE = TTY THEN
```

```
DO;
```

```
IF INPUT (TTY$STATUS) THEN /* STROBE READER */
```

```
DO;
```

```
OUTPUT (TTY$STATUS) = 1;
```

```
OUTPUT (TTY$STATUS) = 0;
```

```
END;
```

```
DO WHILE INPUT (TTY$STATUS);
```

```
END /* WAIT UNTIL TTY IS READY */
```

```
RETURN (NOT INPUT (TTY$INFO) AND LOW$7$MASK);
```

```

ELSE IF INPUT$DEVICE=MAG$TAPE THEN
  RETURN (NOT(INPUT(MAG$TAPE)) AND LOW$7$MASK);
ELSE
  RETURN (NOT(INPUT(CARD$PUNCH)) AND LOW$7$MASK);
END;

END GET$BYTE$TO$YMT;
/* ***** UTILITY PROCEDURES ***** */
/* ***** THE LEVEL 5 UTILITY PROCEDURES ***** */
/* ***** THE 10 MAJOR PROCESSES (LEVEL 2 PROCEDURES) ARE LISTED BELOW ***** */

INITIALIZE: PROCEDURE;
/* LEVEL 2 PROCEDURE */
/* INITIALIZES EVERYTHING AT POWER - UP */
CALL INITIALIZE$RCVR;
CALL INITIALIZE$YMT;
CC$PTR1, CC$PTR2, USAR$CHECK=0;
RECEIVE$LOGIC$PROCESS (DEVICE$READY) =FALSE;
WRITER$PROCESS (DEVICE$READY) =FALSE;
OPERATOR$INPUT$PROCESS (DEVICE$READY) =FALSE;
OPERATOR$OUTPUT$PROCESS (SCHEDULED) =FALSE;
GENERATOR$OUTPUT$PROCESS (DEVICE$READY) =FALSE;
TRANSMITTER$PROCESS (DEVICE$READY) =FALSE;
CUTPUT (TTY$STATUS)=1; /* STROBE TTY READER */
OUTPUT (TTY$STATUS)=0;
/* INITIALIZE TEST/DEBUG BUFFERS AND POINTERS, ETC. */
R$TEST$PTR, BYTE$COUNTER, TEST$COUNTER=0;
/* INITIALIZE PAPER BUFFER */
DO FAER$PTR=0 TO 400;
  PAPER (PAPER$PTR)=0;
END;
PAPER$PTR=0;

```

```

AUT06250
AUT06260
AUT06270
AUT06280
AUT06290
AUT06300
AUT06310
AUT06320
AUT06330
AUT06340
AUT06350
AUT06360
AUT06370
AUT06380
AUT06390
AUT06400
AUT06410
AUT06420
AUT06430
AUT06440
AUT06450
AUT06460
AUT06470
AUT06480
AUT06490
AUT06500
AUT06510
AUT06520
AUT06530
AUT06540
AUT06550
AUT06560
AUT06570
AUT06580
AUT06590
AUT06600
AUT06610
AUT06620
AUT06630
AUT06640
AUT06650
AUT06660
AUT06670
AUT06680
AUT06690
AUT06700
AUT06710
AUT06720

```

AUT06730
 AUT06740
 AUT06750
 AUT06760
 AUT06770
 AUT06780
 AUT06790
 AUT06800
 AUT06810
 AUT06820
 AUT06830
 AUT06840
 AUT06850
 AUT06860
 AUT06870
 AUT06880
 AUT06890
 AUT06900
 AUT06910
 AUT06920
 AUT06930
 AUT06940
 AUT06950
 AUT06960
 AUT06970
 AUT06980
 AUT06990
 AUT07000
 AUT07010
 AUT07020
 AUT07030
 AUT07040
 AUT07050
 AUT07060
 AUT07070
 AUT07080
 AUT07090
 AUT07100
 AUT07110
 AUT07120
 AUT07130
 AUT07140
 AUT07150
 AUT07160
 AUT07170
 AUT07180
 AUT07190
 AUT07200

```

SSTEST$PTR=7:
T$TEST$PTR=0:
SENDING=TRUE:

/* END OF TEST DEBUG INITIALIZATION */
END INITIALIZE;

POLL$USAR: PROCEDURE;

/* LEVEL 2 PROCEDURE */
/* FOLLS THE RECEIVE SIDE OF THE USART TO SEE IF A RECEIVED
   BYTE IS READY FOR PROCESSING. */
IF INEUT (USART$STATUS) THEN /* USAR NOT READY */
  RECEIVE$LOGIC$PROCESS (DEVICE$READY) = FALSE;
ELSE /* DEVICE IS READY */
  RECEIVE$LOGIC$PROCESS (DEVICE$READY) = TRUE;

IF VIRTUAL THEN
  RECEIVE$LOGIC$PROCESS (DEVICE$READY) = TRUE;
END ECLI$USAR;

RECEIVE$LOGIC: PROCEDURE;

/* LEVEL 2 PROCEDURE */
/* EXAMINES INCOMING (RECEIVED) BYTES ONE AT A TIME.
   DETERMINES LOSS OR GAIN OF SYNCHRONOUS IDLE CHECKS
   FOR ODD/EVEN PARITY, RECOGNIZES AND ACTS UPON CONTROL
   CHARACTERS, SCHEDULES THE TRANSMITTER PROCESS WHEN
   NECESSARY, PLACES TEST BYTES IN BUFFERS, SCHEDULES
   THE WRITER, PROCESSES THESE IN BUFFER (OUTPUT) READY FOR
   BY EXAMINING SELECTS THE CORRECT WRITER (OUTDEV) AND
   LOGICAL FLAGS AND VARIABLES BASED UPON RECEIVED
   INFORMATION, SCHEDULES OPERATOR CONSOLE PROCESS
   WHENEVER ALARM CONDITIONS OCCUR AND SCHEDULES ITSELF
   WHENEVER SYNCHRONIZATION MUST BE ACHIEVED.

THE RECEIVE LOGIC PROCEDURE IS A 9-STATE AUTOMATON
WHICH HAS THE FOLLOWING 9 STATES:

1. WAITING FOR SOH.
  
```

```

AUT07210
AUT07220
AUT07230
AUT07240
AUT07250
AUT07260
AUT07270
AUT07280
AUT07290
AUT07300
AUT07310
AUT07320
AUT07330
AUT07340
AUT07350
AUT07360
AUT07370
AUT07380
AUT07390
AUT07400
AUT07410
AUT07420
AUT07430
AUT07440
AUT07450
AUT07460
AUT07470
AUT07480
AUT07490
AUT07500
AUT07510
AUT07520
AUT07530
AUT07540
AUT07550
AUT07560
AUT07570
AUT07580
AUT07590
AUT07600
AUT07610
AUT07620
AUT07630
AUT07640
AUT07650
AUT07660
AUT07670
AUT07680

```

```

2: WAITING FOR STX.
3: PROCESSING TEXT.
4: WAITING FOR ETB.
5: ATTEMPTING TO ACHIEVE SYNCHRONIZATION.
6: ATTEMPTING SEL.
7: EXPECTING DEL.
8: EXPECTING BLOCK PARITY.
9: EXPECTING SECOND CHARACTER OF 2-CHARACTER CONTROL
SEQUENCE.*/

DECLARE
  LCKUP BYTE; /* USED TO INDEX ACTION LCKUPS */
  ACTION BYTE; /* THE ACTION 1-13 PERFORMED */
PERFORM$ACTION: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* PERFORMS ONE OF 13 ACTIONS FOR RECEIVER STATES 1-4 */

DO CASE ACTION;
  /* CASE 0 NOT USED */
DO; /* CASE 1: EXPECTED SOH, GOT SOH */
  RCV$STATE=6;
  RCV$ERROR, CHECK$RCV$TIMER=FALSE;
  BLOCK$START$STATE=1;
END;
  /* CASE 2: DIDN'T GET EXPECTED CHARACTER */
  SYN$CTR=0;
DO; /* CASE 3: FIRST OF 2-CHARACTER CONTROL SEQUENCE */
  SYN$CTR=0;
  PREV$RCV$STATE=RCV$STATE; /* SAVE PREVIOUS STATE */
  PREV$RCV$BYTE=RCV$BYTE; /* SAVE PREVIOUS BYTE */
  RCV$STATE=9;
END;
DO; /* CASE 4: SYN RECEIVED BETWEEN LINE BLOCKS */
  SYN$CTR=SYN$CTR + 1;
  IF SYN$CTR > 3 THEN
    SYN$CTR, RCV$SYNCH$TIMER=0;
END;
DO; /* CASE 5: EXPECTED STX, GOT STX */

```

```

CHECK$RCV$TIMER=FALSE;
BLOCK$START$STATE=2;
RCV$STATE=7;
END;
DO; /* CASE 6: PROCESSING TEXT, INVALID CHAR RECEIVED */
RCV$TXT$CTR=RCV$TXT$CTR + 1;
RCV$ERROR=TRUE;
IF RCV$TXT$CTR > 80 THEN
RCV$STATE=4;
END;
DO; /* CASE 7: PROCESSING TEXT, RECEIVE EN */
RCV$BP=RCV$BP + RCV$BYTE;
RCV$STATE=4;
END;
DO; /* CASE 8: PROCESSING TEXT, MC RECEIVED */
MC$FLAG=TRUE;
RCV$BP=RCV$BP + RCV$BYTE;
END;
DO; /* CASE 9: PROCESSING TEXT, GET TEXT CHARACTER */
RCV$BP=RCV$BP + RCV$BYTE;
RCV$TXT$CTR=RCV$TXT$CTR + 1;
RCV$BUFFER(RCV$TXT$CTR)=RCV$BYTE AND LOW$7$MASK;
IF RCV$TXT$CTR=80 THEN RCV$STATE=4;
/* MASK REMOVES LEFTMOST PARITY BIT */
END;
DO; /* CASE 10: EXPECTING ETB/ETX, GET ERRONEOUS CHAR */
RCV$ERROR=TRUE;
RCV$STATE=8;
END;
DO; /* CASE 11: EXPECTING ETB, GET EIX */
RCV$MID$MSG, MC$FLAG=FALSE;
RCV$BP=RCV$BP + RCV$BYTE;
RCV$STATE=8;
END;
; /* CASE 12: GET SYN AT END OF BLOCK - DO NOTHING */
DO; /* CASE 13: EXPECTING ETB/ETX, GET ETB */
RCV$BP=RCV$BP + RCV$BYTE;
RCV$STATE=8;
END;

```

```

AUT07690
AUT07700
AUT07710
AUT07720
AUT07730
AUT07740
AUT07750
AUT07760
AUT07770
AUT07780
AUT07790
AUT07800
AUT07810
AUT07820
AUT07830
AUT07840
AUT07850
AUT07860
AUT07870
AUT07880
AUT07890
AUT07900
AUT07910
AUT07920
AUT07930
AUT07940
AUT07950
AUT07960
AUT07970
AUT07980
AUT07990
AUT08000
AUT08010
AUT08020
AUT08030
AUT08040
AUT08050
AUT08060
AUT08070
AUT08080
AUT08090
AUT08100
AUT08110
AUT08120
AUT08130
AUT08140
AUT08150
AUT08160

```

AD-A035 708

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF
THE USE OF MICROCOMPUTERS IN DCS AUTODIN TRIBUTARIES.(U)
DEC 76 G E ANDERSON

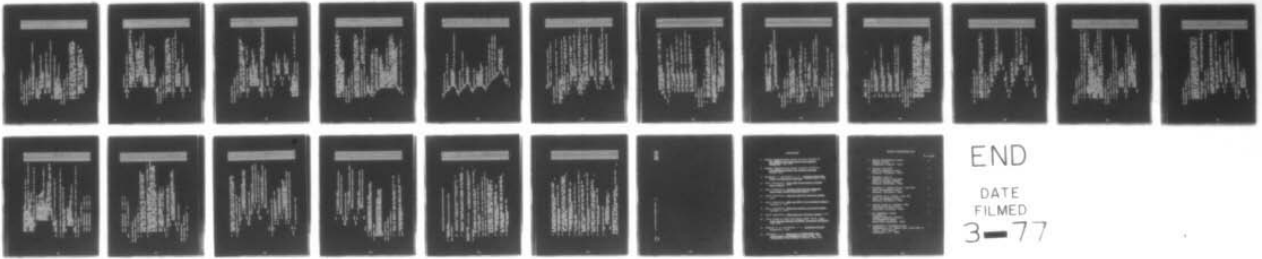
F/G 17/2

UNCLASSIFIED

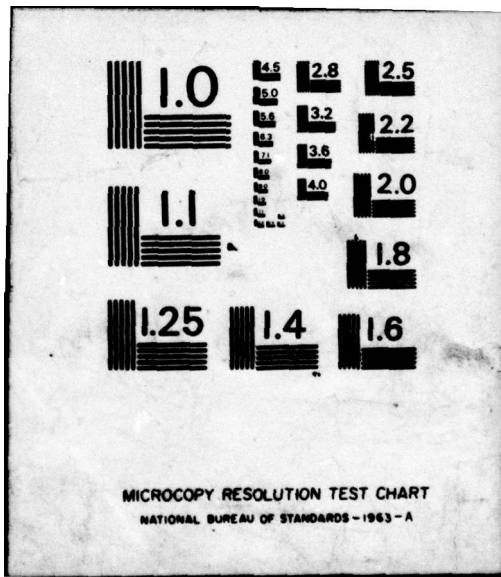
NL

2 OF 2

AD
A035708



END
DATE
FILMED
3-77



```

END: /* OF CASE ACTION */
END PERFORM$ACTION;
RCV$STATE$5: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* WHEN THE RECEIVER IS IN STATE 5 IT IS ATTEMPTING
TO ACHIEVE SYNCHRONIZATION */
RCV$$SYNCH$TIMER=0;
SYN$CTR=0;
DO WHILE SYN$CTR < 4;
  DO WHILE NOT (RECEIVE$LOGIC$PROCESS
    (DEVICE$READY));
    CALL POLL$USAR;
  END;
  IF (RCV$BYTE=-GET$RCVD$BYTE)=SYN THEN
    SYN$CTR=SYN$CTR + 1;
  ELSE SYN$CTR = 0;
  IF RCV$$SYNCH$TIMER EXPIRED THEN CALL ALARM(10);
END;
RECEIVE$LOGIC$PROCESS(SCHEDULED)=FALSE;
RCV$$SYNCH$TIMER=0;
CHECK$RCV$TIMER=TRUE;
IF RCV$HID$MSG THEN RCV$STATE=2;
ELSE RCV$STATE=1;
END RCV$STATE$5;
RCV$STATE$6: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* WHEN THE RECEIVER IS IN STATE 6 IT IS EXPECTING
TO GET AN 'SEL' CHARACTER. IF THE SEL CHARACTER IS
CORRECT (BASED ON A TABLE LOOKUP) THE RECEIVER GOES
TO STATE 3 (TO PROCESS TEXT); OTHERWISE, IT RETURNS
TO STATE 1 (WAITING FOR SOH). */
IF ((RCV$BYTE=RCV$BYTE AND LOW$7$MASK) < 'A' OR
RCV$BYTE > 'D') AND RCV$BYTE <> 'H' AND
RCV$BYTE <> 'P' AND RCV$BYTE <> 'S' THEN
/* WE HAVE A BAD 'SEL' CHARACTER */
DO;
  CHECK$RCV$TIMER=TRUE;
  RCV$$SYNCH$TIMER=0;

```

```

AUT08170
AUT08180
AUT08190
AUT08200
AUT08210
AUT08220
AUT08230
AUT08240
AUT08250
AUT08260
AUT08270
AUT08280
AUT08290
AUT08300
AUT08310
AUT08320
AUT08330
AUT08340
AUT08350
AUT08360
AUT08370
AUT08380
AUT08390
AUT08400
AUT08410
AUT08420
AUT08430
AUT08440
AUT08450
AUT08460
AUT08470
AUT08480
AUT08490
AUT08500
AUT08510
AUT08520
AUT08530
AUT08540
AUT08550
AUT08560
AUT08570
AUT08580
AUT08590
AUT08600
AUT08610
AUT08620
AUT08630
AUT08640

```

```

RCV$$STATE=1;
END; /* GOOD 'SEL' CHARACTER RECEIVED */
ELSE DO;
  RCV$$BP=(RCV$$BP=0) + RCV$$BYTE; /* INITIALIZE BP */
  IF RCV$$BYTE='A' OR RCV$$BYTE='H' OR RCV$$BYTE='S' THEN
  /* WE WANT TO OUTPUT ON THE TELETYPE */
  ELSE
  IF RCV$$BYTE='D' OR RCV$$BYTE='F' THEN
  /* THEN IT'S A CARD MESSAGE */
  OUTPUT$DEVICE=CARD$PUNCH;
  ELSE /* IT MUST BE A MAG TAPE MESSAGE */
  OUTPUT$DEVICE=MAG$TAPE;
  RCV$$TX$CTR=0; /* INITIALIZE FOR NEW LINE BLOCK */
  RCV$$MID$MSG=TRUE;
  RCV$$ERROR=FALSE;
  RCV$$STATE=3;
  /* CHECK FOR FLASH MESSAGE - RING BELL
  FOR OPERATOR IF FLASH */
  IF RCV$$BYTE='P' OR RCV$$BYTE='S' THEN
  CALL ALARM(9);
END;
END RCV$$STATE$6;
RCV$$STATE$7: PROCEDURE;
  /* LEVEL 3 PROCEDURE */
  /* WHEN THE RECEIVER IS IN STATE 7 IT IS EXPECTING A
  'DEL' CHARACTER. IF DEL RECEIVED THEN GOES INTO
  STATE 3 (PROCESSING TEXT) OTHERWISE, IT GOES INTO
  STATE 2 (WAITING FOR STX CHARACTER). */
  IF RCV$$BYTE=DEL THEN /* GO TO TEXT PROCESSING STATE */
  DO;
  RCV$$ERROR=FALSE;
  RCV$$TX$CTR=RCV$$BP=0;
  RCV$$BP=RCV$$BP + RCV$$BYTE;
  RCV$$STATE=3;
  ELSE /* BYTE IS NOT A DEL - GO BACK TO STATE 2 */
  DO;
  RCV$$SYNCH$TIMER=0;
  CHECK$RCV$TIMER=TRUE;
  RCV$$STATE=2;
END;
END;

```

```

AUT08650
AUT08660
AUT08670
AUT08680
AUT08690
AUT08700
AUT08710
AUT08720
AUT08730
AUT08740
AUT08750
AUT08760
AUT08770
AUT08780
AUT08790
AUT08800
AUT08810
AUT08820
AUT08830
AUT08840
AUT08850
AUT08860
AUT08870
AUT08880
AUT08890
AUT08900
AUT08910
AUT08920
AUT08930
AUT08940
AUT08950
AUT08960
AUT08970
AUT08980
AUT08990
AUT09000
AUT09010
AUT09020
AUT09030
AUT09040
AUT09050
AUT09060
AUT09070
AUT09080
AUT09090
AUT09100
AUT09110
AUT09120

```

AUT09130
 AUT09140
 AUT09150
 AUT09160
 AUT09170
 AUT09180
 AUT09190
 AUT09200
 AUT09210
 AUT09220
 AUT09230
 AUT09240
 AUT09250
 AUT09260
 AUT09270
 AUT09280
 AUT09290
 AUT09300
 AUT09310
 AUT09320
 AUT09330
 AUT09340
 AUT09350
 AUT09360
 AUT09370
 AUT09380
 AUT09390
 AUT09400
 AUT09410
 AUT09420
 AUT09430
 AUT09440
 AUT09450
 AUT09460
 AUT09470
 AUT09480
 AUT09490
 AUT09500
 AUT09510
 AUT09520
 AUT09530
 AUT09540
 AUT09550
 AUT09560
 AUT09570
 AUT09580
 AUT09590
 AUT09600

```

END RCV$STATE$7;
RCV$STATE$8: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* IN STATE 8 THE RECEIVER IS LOOKING FOR VALID BLOCK
PARITY. IF BP IS VALID, RECEIVER FINISHES THE
BLOCK AND ACKS FOR IT, OTHERWISE IT SENDS A
NAK (NEGATIVE ACKNOWLEDGEMENT) */
IF RCV$BP <> RCV$BYTE THEN /* BAD PARITY */
DO;
  CALL SEND$CC (NAK);
  NAK$CTR=NAK$CTR + 1;
  IF NAK$CTR > 3 THEN CALL ALARM (1);
  RCV$STATE=BLOCK$STATE;
  RCV$SYNCH$TIMER=0;
  CHECK$RCV$TIMER=TRUE;
  RETURN;
END;
IF $BUFFER$PTR > 1 THEN STILL$WRITING=TRUE;
IF STILL$WRITING AND RCV$MID$MSG THEN /* WRITER PROCESS IS NOT
FINISHED WITH LAST BLOCK */
DO;
  CALL SEND$CC (WBT);
  NAK$CTR=0;
END;
ELSE DO;
  IF XMT$ACK$1$2=1 THEN
  DO;
    CALL SEND$CC (ACK 1);
    NAK$CTR=0;
    XMT$ACK$1$2=2;
  END;
  ELSE DO;
    CALL SEND$CC (ACK 2);
    NAK$CTR=0;
    XMT$ACK$1$2=1;
  END;
  RCV$SYNCH$TIMER=0;
  CHECK$RCV$TIMER=TRUE;
  IF RCV$MID$MSG THEN RCV$STATE=2; /* SET NEXT RCV STATE */
  ELSE RCV$STATE=1;
  WRITER$PROCESS (SCHEDULED)=TRUE;

```

```

AUT096110
AUT09620
AUT09630
AUT09640
AUT09650
AUT09660
AUT09670
AUT09680
AUT09690
AUT09700
AUT09710
AUT09720
AUT09730
AUT09740
AUT09750
AUT09760
AUT09770
AUT09780
AUT09790
AUT09800
AUT09810
AUT09820
AUT09830
AUT09840
AUT09850
AUT09860
AUT09870
AUT09880
AUT09890
AUT09900
AUT09910
AUT09920
AUT09930
AUT09940
AUT09950
AUT09960
AUT09970
AUT09980
AUT09990
AUT10000
AUT10010
AUT10020
AUT10030
AUT10040
AUT10050
AUT10060
AUT10070
AUT10080

```

```

/* THIS SCHEDULES THE WRITER PROCESS TO DUMP THE BUFFER */
RCV$BUFFER=RCV$TXTR; /* INSERT BUFFER LENGTH IN SLOT */
/* NOW SWITCH BUFFERS FOR THE RECEIVER LOGIC PROCESS */
IF RCV$BUFFER$ADDR=.RW$BUFFER$1 THEN
  RCV$BUFFER$ADDR=.RW$BUFFER$2;
ELSE RCV$BUFFER$ADDR=.RW$BUFFER$1;

END RCV$STATE$8;

RCV$STATE$9: PROCEDURE;

  /* LEVEL 3 PROCEDURE */

  /* IN STATE 9 THE RECEIVER IS EXPECTING THE SECOND OF A
  TWO-CHARACTER CONTROL SEQUENCE. IF THIS IS RECEIVED, THE
  CONTROL CHARACTERS ARE ACTED UPON, OTHERWISE AN ERROR
  CONDITION IS FLAGGED (RCV$ERROR). */

  RCV$STATE=PREV$RCV$STATE; /* NORMAL RETURN IS TO
  IF PREV$RCV$BYTE <> RCV$BYTE THEN /* NOT A VALID CONTROL SEQ */
    DO
      RCV$ERROR=TRUE;
    END;
  IF RCV$BYTE=ACK1 OR RCV$BYTE=ACK2 THEN
    /* WE HAVE A VALID CONTROL CHAR SEQUENCE */
    DO
      IF (RCV$BYTE=ACK1 AND RCV$ACK$1$2=1) OR
        (RCV$BYTE=ACK2 AND RCV$ACK$1$2=2) THEN
        DO
          AWAITING$ACK,XMT$WAIT=FALSE;
          WBT$CTR,NAK$CTR,XMT$REP$CTR=0;
          CHECK$ANS$TIMER=FALSE;
          XMT$ANS$TIMER=0;
          /* RESET TOGGLE */
          IF RCV$ACK$1$2=1 THEN RCV$ACK$1$2=2;
          ELSE RCV$ACK$1$2=1;
          /* RESET POINTER FOR REXMT */
          T$BUFFER$PTR=0; /* MARKS BUFFER AS ACK'D */
          TRANSMIT$PTR=0; /* HARKS BUFFER AS ACK'D */
          /* NOW SWITCH BUFFERS FOR TRANSMITTER PROCESS */
          IF TRANSMIT$BUFFER$ADDR=.T$BUFFER$1 THEN
            TRANSMIT$BUFFER$ADDR=.T$BUFFER$2;
          ELSE TRANSMIT$BUFFER$ADDR=.T$BUFFER$1;
          END;
        ELSE
          CALL SEND$CC(INV); /* ACK1/2 SEQUENCE WRONG */
        END;
      END;
    END;
  END;

```

```

AUT110090
AUT110100
AUT110110
AUT110120
AUT110130
AUT110140
AUT110150
AUT110160
AUT110170
AUT110180
AUT110190
AUT110200
AUT110210
AUT110220
AUT110230
AUT110240
AUT110250
AUT110260
AUT110270
AUT110280
AUT110290
AUT110300
AUT110310
AUT110320
AUT110330
AUT110340
AUT110350
AUT110360
AUT110370
AUT110380
AUT110390
AUT110400
AUT110410
AUT110420
AUT110430
AUT110440
AUT110450
AUT110460
AUT110470
AUT110480
AUT110490
AUT110500
AUT110510
AUT110520
AUT110530
AUT110540
AUT110550
AUT110560

```

```

ELSE
IF RCV$BYTE=NAK THEN
DO
IF NAK$CTR > 3 THEN CALL ALARM(1);
ELSE
DO
XMT$ANS$TIMER=0;
NAK$CTR=NAK$CTR + 1; /* RESETS FOR RETRANSMISSION */
T$BUFFER$PTR=1;
END;
ELSE
IF RCV$BYTE=HBT THEN
DO
IF HBT$CTR > 3 THEN CALL ALARM(2);
ELSE
DO
WBT$CTR=WBT$CTR + 1;
XMT$ANS$TIMER=0;
END;
ELSE
IF RCV$BYTE=CAN THEN
DO
CALL ALARM(3);
RCV$STATE=1;
XMT$ACK$1$2=1;
CHECK$RCV$TIMER=TRUE;
RCV$SYNCH$TIMER=0;
END;
ELSE
IF RCV$BYTE=REP THEN
DO
IF RCV$MID$MSG OR STILL$WRITING THEN
DO
IF STILL$WRITING THEN CALL SEND$CC(WBT);
ELSE
IF NAK$CTR > 0 THEN CALL SEND$CC(NAK);
ELSE
DO
IF XMT$ACK$1$2=1 THEN
CALL SEND$CC(ACK2);
ELSE CALL SEND$CC(ACK1);
END;
END;
ELSE
CALL ALARM(4);
END;

```

```

ELSE RCV$BYTE=RM THEN CALL ALARM(5);
ELSE
CALL ALARM(6); /* INV WAS RECEIVED */

END RCV$STATE$9;

/* ***** RECEIVER LOGIC BEGINS EXECUTION HERE ***** */
RCV$BYTE=GET$RCVD$BYTE;
IF RCV$STATE < 5 THEN
DC; /* FIRST DETERMINE ACTION BASED ON RCVD BYTE AND
RCV STATE */
RCV$BYTE=RCV$BYTE AND OFFH; /* PERFORM AND TO INSURE
IF PARITY EVEN THEN
DO;
IF (LOOKUP:=(RCV$BYTE AND LOW$7$MASK)) > 7 THEN
IF (LOOKUP:=LOOKUP-9) > 21 THEN
LOOKUP=20;
/* THIS HAS HAPPED THE CONTROL CHARACTERS INTO
THE INTEGERS 1-21 */
END;
LOOKUP=22; /* BYTE IS NOT CONTROL CHARACTER */
/* PERFORM TABLE LOOKUP BASED ON RCV STATE AND CHARACTER */
DO CASE RCV$STATE;
/* CASE 0 NOT USED */
/* CASE 1 - CHECK FOR LOSS OF SYNCH. IF SYNCH
OK THEN DO LOOKUP FOR RCV STATE 1 */
IF CHECK$RCV$TIMER AND RCV$SYNCH$TIMER EXPIRED THEN
DO;
RCV$STATE=5;
RECEIVE$LOGIC$PROCESS (SCHEDULED)=TRUE;
RETURN;
END;
ACTION=TABLE$1 (LOOKUP);
END;
/* CASE 2 - CHECK FOR SYNCH LOSS. IF SYNCH
OK THEN DO LOOKUP FOR RCV STATE 2 */
IF CHECK$RCV$TIMER AND RCV$SYNCH$TIMER EXPIRED THEN
DO;
RCV$STATE=5;
RECEIVE$LOGIC$PROCESS (SCHEDULED)=TRUE;
RETURN;
END;
ACTION=TABLE$2 (LOOKUP);
END;

```

AUT110570
AUT110580
AUT110590
AUT110600
AUT110610
AUT110620
AUT110630
AUT110640
AUT110650
AUT110660
AUT110670
AUT110680
AUT110690
AUT110700
AUT110710
AUT110720
AUT110730
AUT110740
AUT110750
AUT110760
AUT110770
AUT110780
AUT110790
AUT110800
AUT110810
AUT110820
AUT110830
AUT110840
AUT110850
AUT110860
AUT110870
AUT110880
AUT110890
AUT110900
AUT110910
AUT110920
AUT110930
AUT110940
AUT110950
AUT110960
AUT110970
AUT110980
AUT110990
AUT111000
AUT111010
AUT111020
AUT111030
AUT111040

```

AUT111050
AUT111060
AUT111070
AUT111080
AUT111090
AUT111100
AUT111110
AUT111120
AUT111130
AUT111140
AUT111150
AUT111160
AUT111170
AUT111180
AUT111190
AUT111200
AUT111210
AUT111220
AUT111230
AUT111240
AUT111250
AUT111260
AUT111270
AUT111280
AUT111290
AUT111300
AUT111310
AUT111320
AUT111330
AUT111340
AUT111350
AUT111360
AUT111370
AUT111380
AUT111390
AUT111400
AUT111410
AUT111420
AUT111430
AUT111440
AUT111450
AUT111460
AUT111470
AUT111480
AUT111490
AUT111500
AUT111510
AUT111520

```

```

ACTION=TABLE$3(LOOKUP); /* CASE 3 - RCV STATE 3 */
ACTION=TABLE$4(LOOKUP); /* CASE 4 - RCV STATE 4 */
END; /* OF CASE RCVSTATE */
CALL PERFORM$ACTION;

END;

ELSE DO CASE (ACTION:=RCV$STATE - 5);
/* RECEIVER STATE IS BETWEEN 5 AND 9 */
/* CASE 0 - RCV$STATE = 5 - ATTEMPTING TO
ACHIEVE SYNCHRONIZATION */
CALL RCV$STATE$5;
/* CASE 1, RCV$STATE = 6 - EXPECTING 'SEL' BYTE */
CALL RCV$STATE$6;
/* CASE 2, RCV$STATE=7 - EXPECTING 'DEL' BYTE */
CALL RCV$STATE$7;
/* CASE 3, RCV$STATE=8 - EXPECTING ELOCK PARITY BYTE */
CALL RCV$STATE$8;
/* CASE 4, RCV$STATE=9 - PROCESSING THE SECOND CHAR
IN A TWO-CHARACTER CONTROL SEQUENCE */
CALL RCV$STATE$9;

END; /* OF CASE ACTION */

END RECEIVE$LOGIC;
WRITER: PROCEDURE;
/* LEVEL 2 PROCEDURE */
/* WRITES BYTES (ONE AT A TIME) ON THE SELECTED
OUTPUT DEVICE AND RESCHEDULES ITSELF WHENEVER
MORE BYTES ARE TO BE DUMPED FROM A WRITE BUFFER. */
CALL WRITE$BYTE(WRITE$BUFFER(W$BUFFER$PTR));
W$BUFFER$PTR=W$BUFFER$PTR+1;
IF W$BUFFER$PTR > WRITE$BUFFER THEN /* DONE WRITING CURRENT BUFF */
DO;
WRITE$BUFFER=0;
/* SET LENGTH OF BUFFER- IT'S ALREADY WRITTEN.
THEN FLIP BUFFER */
IF WRITE$BUFFER$ADDR=.RW$BUFFER$1 THEN
WRITE$BUFFER$ADDR=.RW$BUFFER$2;

```

AUT111530
 AUT111540
 AUT111550
 AUT111560
 AUT111570
 AUT111580
 AUT111590
 AUT111600
 AUT111620
 AUT111630
 AUT111640
 AUT111650
 AUT111660
 AUT111670
 AUT111680
 AUT111690
 AUT111700
 AUT111710
 AUT111720
 AUT111730
 AUT111740
 AUT111750
 AUT111760
 AUT111770
 AUT111780
 AUT111790
 AUT111800
 AUT111810
 AUT111820
 AUT111830
 AUT111840
 AUT111850
 AUT111860
 AUT111870
 AUT111880
 AUT111890
 AUT111900
 AUT111910
 AUT111920
 AUT111930
 AUT111940
 AUT111950
 AUT111960
 AUT111970
 AUT111980
 AUT111990
 AUT112000

```

ELSE WRITESBUFFER$ADDR=.RW$BUFFER$1;
W$BUFFER$PTR=1;
STILL$WRITING=FALSE;
IF WRITESBUFFER=0 THEN /* GO TO SLEEP - NO
                        RESCHEDULE NEEDED */
DO: IF NOT RCV$MID$MSG THEN OUTPUT$DEVICE=NOT$SELECTED;
WRITER$PROCESS(SCHEDULED)=FALSE;
END;

END;

END WRITER;

OPERATOR$INPUT: PROCEDURE;
/* LEVEL 2 PROCEDURE */
/* INTERPRETS COMMANDS FROM THE OPERATOR CONSOLE -
T= TRANSMIT NEW MESSAGE.
C= CANCEL CURRENTLY TRANSMITTED MSG.
B= REBOOT (RE-INITIALIZE). */
DECLARE CHARACTER BYTE;
RETURN /* INSERTED FOR TEST/DEBUG ONLY */
IF (CHARACTER=:CRT$IN) = 'T' THEN /* TRANSMIT NEW MSG */
SENDING=TRUE;
ELSE IF CHARACTER='C' THEN /* CANCEL THIS MSG */
DO:
CALL SEND$CC(CAN);
CALL INITIALIZE$PTR;
END;
ELSE IF CHARACTER='B' THEN /* REBOOT THE SYSTEM */
GO TO RESTART;
/* ELSE NOTHING - ONLY T,C, AND B VALID COMMANDS */

END OPERATOR$INPUT;

OPERATOR$OUTPUT: PROCEDURE;
/* LEVEL 2 PROCEDURE */
/* OUTPUTS MESSAGES TO THE CRT ONE EYTE AT A TIME:
STATE 1: SEND BELL
STATE 2: SEND ONE LETTER DIAGNOSTIC MSG.
```

```

STATE 3: SEND CR.
STATE 4: SEND LF
DO CASE OUTPUT$STATE;
; /* CASE 0 NOT USED */
DO: /* CASE 1 OUTPUT BELL ALARM */
CALL CRT$OUT(BELL);
OUTPUT$STATE=2;
END;
DO: /* CASE 2 - OUTPUT DIAGNOSTIC CHARACTER */
CALL CRT$OUT(ALARM$MSG);
OUTPUT$STATE=3;
END;
DO: /* CASE 3 - OUTPUT <CR> */
CALL CRT$OUT(CR);
OUTPUT$STATE=4;
END;
DO: /* CASE 4 - OUTPUT <LF> AND GO TO SLEEP */
CALL CRT$OUT(LF);
OPERATOR$OUTPUT$PROCESS (SCHEDULED) =FALSE;
END;
END; /* OF CASE OUTPUT STATE */
END OPERATOR$OUTPUT;
TRANSMIT$LOGIC: PROCEDURE;
/* LEVEL 2 PROCEDURE */
/*
BUILDS BLOCKS FOR OUTGOING MESSAGE TRANSMISSION (ONE STX,
BYE AT A TIME). INSERTS NEEDED FRAMING BYTES (SOH, STX,
SEL, DEL, EM, ETB, ETX AND BP) CALCULATES ODD PARITY
FOR TEXT BYTES (SCHEDULES TRANSMISSION WHEN LINE
BLOCKS BECOME AVAILABLE FOR TRANSMISSION AND SCHEDULES
THE OPERATOR OUTPUT PROCESS WHEN ALARM CONDITIONS OCCUR.
THE TRANSMIT LOGIC PROCESS IS A 5 STATE AUTOMATON WITH
THE FOLLOWING STATES:
1. PROCESSING BEGINNING OF A NEW MESSAGE.
2. WAITING FOR SECOND CHARACTER OF A MESSAGE.
3. WAITING FOR THIRD CHARACTER OF A MESSAGE.
4. PROCESSING THE MIDDLE OF A MESSAGE.
5. PROCESSING THE BEGINNING OF AN INTERMEDIATE LINE BLOCK.
*/

```

AUT112490
 AUT112500
 AUT112510
 AUT112520
 AUT112530
 AUT112540
 AUT112550
 AUT112560
 AUT112570
 AUT112580
 AUT112590
 AUT112600
 AUT112610
 AUT112620
 AUT112630
 AUT112640
 AUT112650
 AUT112660
 AUT112670
 AUT112680
 AUT112690
 AUT112700
 AUT112710
 AUT112720
 AUT112730
 AUT112740
 AUT112750
 AUT112760
 AUT112770
 AUT112780
 AUT112790
 AUT112800
 AUT112810
 AUT112820
 AUT112830
 AUT112840
 AUT112850
 AUT112860
 AUT112870
 AUT112880
 AUT112890
 AUT112900
 AUT112910
 AUT112920
 AUT112930
 AUT112940
 AUT112950
 AUT112960

```

  /*
  EOM: PROCEDURE (CHARACTER) BYTE;
  /* LEVEL 3 PROCEDURE */
  /* CHECKS FOR 8 LINE FEEDS FOLLOWED BY 4 N'S IN A ROW.
  RETURNS TRUE ONLY IF THIS HAS OCCURED (THIS IS THE END OF A
  MESSAGE IN AUTODIN). OTHERWISE, IT RETURNS FALSE.
  DECLARE CHARACTER BYTE;
  IF EOM$STATE=1 THEN
  DO;
  IF CHARACTER=08AH THEN
  /* 08AH=ODD PARITY LP */
  DO;
  LF$CTR=LF$CTR + 1;
  IF LF$CTR=8 THEN /* WE HAVE 8 LF'S IN A ROW
  GO TO EOM$STATE 2 AND COUNT N'S */
  DO;
  LF$CTR=0;
  EOM$STATE=2;
  END;
  ELSE LF$CTR=0;
  END;
  ELSE DO;
  IF CHARACTER=0CEH THEN
  /* 0CEH=ODD PARITY 'N' */
  DO;
  N$CTR=N$CTR + 1;
  IF N$CTR=4 THEN /* WE'RE AT END OF MSG */
  DO;
  N$CTR=0;
  EOM$STATE=1; /* RESET NEXT TIME */
  RETURN TRUE;
  END;
  ELSE /* NOT AN EOM - GO BACK TO STATE 1 */
  DO;
  N$CTR=0;
  EOM$STATE=1;
  END;
  RETURN FALSE;
  END;
  */
  
```

```

END EOM;
XMT$STATE$1: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* STATE 1 IS WAITING TO START A NEW HSG FOR TRANSMISSION */
IF (XMT$BYTE:=GET$BYTE$TO$XMT)=SPACE THEN /* GOBBLE SPACES
AT THE BEGINNING OF TAPE */
RETURN;
/* OTHERWISE GET A NEW BUFFER AND PUT AN 'SOH' IN SLOT 1 */
IF XMT$LOGIC$B$BUFFER$ADDR=.T$BUFFER$1 THEN
XMT$LOGIC$B$BUFFER$ADDR=.T$BUFFER$2;
ELSE XMT$LOGIC$B$BUFFER$ADDR=.T$BUFFER$1;
XMT$LOGIC$B$BUFFER(1)=SOH;
XMT$BYTE=XMT$BYTE AND OFFH; /* ENSURE PARITY SET */
IF PARITY EVEN THEN
XMT$BYTE=XMT$BYTE OR PARITY$MASK; /* INSERT ODD PARITY */
XMT$LOGIC$B$BUFFER(3)=XMT$BYTE;
XMT$BP=(XMT$BP:=0) + XMT$BYTE; /* INITIALIZE AND COMPUTE BP */
XMT$STATE=2; /* SET NEXT STATE */
END XMT$STATE$1;
XMT$STATE$2: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* PROCESSES THE SECOND BYTE OF A TRANSMITTED MESSAGE */
IF CAN$FLAG THEN /* CANCEL OUTGOING MESSAGE */
DO;
CAN$FLAG=FALSE;
XMT$STATE=1;
CALL SEND$CC(CAN);
SENDING=FALSE;
ELSE /* PROCESS SECOND BYTE OF MESSAGE */
DO;
XMT$BYTE=(XMT$BYTE:=GET$BYTE$TO$XMT) AND OFFH);
IF PARITY EVEN THEN
XMT$BYTE=XMT$BYTE OR PARITY$MASK;
XMT$LOGIC$B$BUFFER(4)=XMT$BYTE;
XMT$BP=XMT$BP + XMT$BYTE;
XMT$STATE=3; /* SET NEXT STATE */
END;
END;

```

AUT12970
AUT12980
AUT12990
AUT13000
AUT13010
AUT13020
AUT13030
AUT13040
AUT13050
AUT13060
AUT13070
AUT13080
AUT13090
AUT13100
AUT13110
AUT13120
AUT13130
AUT13140
AUT13150
AUT13160
AUT13170
AUT13180
AUT13190
AUT13200
AUT13210
AUT13220
AUT13230
AUT13240
AUT13250
AUT13260
AUT13270
AUT13280
AUT13290
AUT13300
AUT13310
AUT13320
AUT13330
AUT13340
AUT13350
AUT13360
AUT13370
AUT13380
AUT13390
AUT13400
AUT13410
AUT13420
AUT13430
AUT13440

```

END XMT$STATE$2;
XMT$STATE$3: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* CHECKS THE THIRD CHARACTER OF THE OUTGOING MESSAGE TO
SEE IF IT IS A VALID LMF CHARACTER. IF IT IS
CONVERTED TO THE CORRESPONDING SEL CHARACTER. THE LMF
CHARACTER IS INSERTED IN THE THIRD TEXT SLOT OF THE
LINE BLOCK. THE SEL CHARACTER IS INSERTED IN THE SECOND
FRAMING SLOT OF THE LINE BLOCK. */
DECLARE SEL BYTE;
CHECK BYTE;
/* USED TO LOOK UP THE SEL CHAR */
/* USED TO CHECK CORRECTNESS OF SEL */
CHECK$LMF: PROCEDURE BYTE;
/* LEVEL 4 PROCEDURE */
/* RETURNS TRUE AND SETS THE SEL CORRECTLY IF THE LMF
CHARACTER (THIRD BYTE OF THE OUTGOING MSG) WAS CORRECT.
RETURNS FALSE OTHERWISE. */
IF XMT$BYTE > 40H AND XMT$BYTE < 4AH AND XMT$BYTE <> 'E'
AND XMT$BYTE <> 'H' THEN
DO;
SEL=SEL$LOOKUP$1(XMT$BYTE - 41H);
RETURN TRUE;
END;
ELSE IF XMT$BYTE > 50H AND XMT$BYTE < 55H THEN
DO;
SEL=SEL$LOOKUP$2(XMT$BYTE - 51H);
RETURN TRUE;
END;
ELSE /* HAD LMF CHARACTER - NO CORRESPONDING SEL */
RETURN FALSE;
END CHECK$LMF;
IF CAN$FLAG THEN /* CANCEL MESSAGE */
DO;
CAN$FLAG=FALSE;
XMT$STATE=1;
CALL SEND$CC(CAN);
SENDING=FALSE;
END;
ELSE

```

```

AUT113450
AUT113460
AUT113470
AUT113480
AUT113490
AUT113500
AUT113510
AUT113520
AUT113530
AUT113540
AUT113550
AUT113560
AUT113570
AUT113580
AUT113590
AUT113600
AUT113610
AUT113620
AUT113630
AUT113640
AUT113650
AUT113660
AUT113670
AUT113680
AUT113690
AUT113700
AUT113710
AUT113720
AUT113730
AUT113740
AUT113750
AUT113760
AUT113770
AUT113780
AUT113790
AUT113800
AUT113810
AUT113820
AUT113830
AUT113840
AUT113850
AUT113860
AUT113870
AUT113880
AUT113890
AUT113900
AUT113910
AUT113920

```

```

DO:
XMT$BYTE=GET$BYTE$TO$XMT;
IF (CHECK:=CHECK$LMP) THEN /* GOOD LMF CHARACTER */
DO:
SEL=SEL AND OFFP; /* ENSURE PARITY SET */
IF NOT (PARITY EVEN) THEN
SEL=SEL OR PARITY$MASK;
XMT$BP=XMT$BP + SEL;
XMT$LOGIC$BUFFER(2)=SEL;
XMT$BYTE=XMT$BYTE AND OFFP;
IF PARITY EVEN THEN
XMT$BYTE=XMT$BYTE OR PARITY$MASK;
XMT$BP=XMT$BP + XMT$EYTE;
XMT$LOGIC$BUFFER(5)=XMT$BYTE;
XMT$TXTR=6;
XMT$STATE=4;
ELSE CALL ALARM(8);
END;
END XMT$STATE$3;
XMT$STATE$4: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* PROCESSES TEXT OF AN OUTGOING MESSAGE. BUILDING THE
INFORMATIONAL PORTION OF THE LINE BLOCKS. */
DECLARE CHECK BYTE;
IF CAN$FLAG THEN /* CANCEL OUTGOING MESSAGE */
DO:
CAN$FLAG=FALSE;
XMT$STATE=1;
CALL SEND$C(CAN);
SENDING=FALSE;
ELSE DO:
XMT$BYTE=(XMT$BYTE:=GET$BYTE$TO$XMT) AND OFFP;
IF PARITY EVEN THEN
XMT$BYTE=XMT$BYTE OR PARITY$MASK;
XMT$LOGIC$BUFFER(XMT$TXTR)=XMT$BYTE;
XMT$BP=XMT$BP + XMT$EYTE;
XMT$TXTR=XMT$TXTR + 1;
IF (XMT$TXTR > 82) OR (CHECK:=EOM(XMT$BYTE)) THEN
DO:

```

```

AUT113930
AUT113940
AUT113950
AUT113960
AUT113970
AUT113980
AUT113990
AUT114000
AUT114010
AUT114020
AUT114030
AUT114040
AUT114050
AUT114060
AUT114070
AUT114080
AUT114090
AUT114100
AUT114110
AUT114120
AUT114130
AUT114140
AUT114150
AUT114160
AUT114170
AUT114180
AUT114190
AUT114200
AUT114210
AUT114220
AUT114230
AUT114240
AUT114250
AUT114260
AUT114270
AUT114280
AUT114290
AUT114300
AUT114310
AUT114320
AUT114330
AUT114340
AUT114350
AUT114360
AUT114370
AUT114380
AUT114390
AUT114400

```

```

IF CHECK THEN /* WE'RE AT THE END OF MSG */
DO: IF XMT$TYT$CTR < 82 THEN
DO:
XMT$LOGIC$BUFFER(XMT$TYT$CTR)=EN;
XMT$BP=XMT$BP + EN;
SENDING=PALSE;
XMT$TYT$CTR=XMT$TYT$CTR + 1;
XMT$DEVICE=NOT$SELECTED;
IF VIRTUAL THEN SENDING=TRUE;
END;
XMT$STATE=1; /* SET STATE FOR NEW MSG */
XMT$LOGIC$BUFFER(XMT$TYT$CTR)=ETY;
XMT$BP=XMT$BP + ETY;
XMT$TYT$CTR=XMT$TYT$CTR + 1;
END;
ELSE DO:
XMT$STATE=5;
XMT$LOGIC$BUFFER(XMT$TYT$CTR)=ETB;
XMT$BP=XMT$BP + ETB;
XMT$TYT$CTR=XMT$TYT$CTR + 1;
END;
XMT$ANS$TIMER=0;
CHECK$ANS$TIMER=TRUE;
XMT$LOGIC$BUFFER(XMT$TYT$CTR)=XMT$BP;
XMT$LOGIC$BUFFER=XMT$TYT$CTR; /* INSERTS
LENGTH AT BEGINNING OF BUFFER */
IF AWAITING$ACK THEN
XMT$WAIT=TRUE;
END;
END XMT$STATE$4;
XMT$STATE$5: PROCEDURE;
/* LEVEL 3 PROCEDURE */
/* PROCESSING THE FRAMING CHARACTERS REQUIRED AT THE
BEGINNING OF IMMEDIATE LINE BLOCKS OF OUTGOING MSG */
DECLARE CHECK BYTE; /* USED TO CHECK FOR END OF MSG */
/* GET NEW BUFFER */
IF XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$1 THEN
XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$2;
ELSE XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$1;
XMT$LOGIC$BUFFER(1)=STY; /* INSERT 'STY' */

```

```

AUT114410
AUT114420
AUT114430
AUT114440
AUT114450
AUT114460
AUT114470
AUT114480
AUT114490
AUT114500
AUT114510
AUT114520
AUT114530
AUT114540
AUT114550
AUT114560
AUT114570
AUT114580
AUT114590
AUT114600
AUT114610
AUT114620
AUT114630
AUT114640
AUT114650
AUT114660
AUT114670
AUT114680
AUT114690
AUT114700
AUT114710
AUT114720
AUT114730
AUT114740
AUT114750
AUT114760
AUT114770
AUT114780
AUT114790
AUT114800
AUT114810
AUT114820
AUT114830
AUT114840
AUT114850
AUT114860
AUT114870
AUT114880

```

AUT114890
 AUT114900
 AUT114910
 AUT114920
 AUT114930
 AUT114940
 AUT114950
 AUT114960
 AUT114970
 AUT114980
 AUT114990
 AUT115000
 AUT115010
 AUT115020
 AUT115030
 AUT115040
 AUT115050
 AUT115060
 AUT115070
 AUT115080
 AUT115090
 AUT115100
 AUT115110
 AUT115120
 AUT115130
 AUT115140
 AUT115150
 AUT115160
 AUT115170
 AUT115180
 AUT115190
 AUT115200
 AUT115210
 AUT115220
 AUT115230
 AUT115240
 AUT115250
 AUT115260
 AUT115270
 AUT115280
 AUT115290
 AUT115300
 AUT115310
 AUT115320
 AUT115330
 AUT115340
 AUT115350
 AUT115360

```

XMT$LOGIC$BUFFER(2) = DEL;
XMT$BYTE = (XMT$BYTE = GET$BYTE$TO$XMT) AND OFFH);
IF PARITY EVEN THEN
  XMT$BYTE = XMT$BYTE OR PARITY$MASK; /* INSERT ODD PARITY */
XMT$LOGIC$BUFFER(3) = XMT$BYTE;
XMT$BP = (XMT$BP = 0) + DEL + XMT$BYTE;
IF (CHECK = EOH(XMT$BYTE)) THEN /* WE'RE AT END OF A MSG */
  DO;
    XMT$LOGIC$BUFFER(4) = EM;
    SENDING = FALSE;
    XMT$LOGIC$BUFFER(5) = ETX;
    XMT$BP = XMT$BP + EM + ETX;
    XMT$LOGIC$BUFFER(6) = XMT$BP;
    XMT$ANS$TIMER = 0; /* STORE LENGTH OF BUFFER */
    CHECK$ANS$TIMER = 0;
    IF AWAITING$ACK AND XMT$ANS$TIMER = TRUE; XMT$WAIT = TRUE;
    /* TRANSMIT LOGIC PROCESS HAS 2 BLOCKS PENDING
    IT MUST WAIT UNTIL 1 OF THEM IS ACK'D. */
    XMT$STATE = 1;
  END;
ELSE XMT$STATE = 4; /* CONTINUE PROCESSING TEXT */
END XMT$STATE$5;

/* ***** TRANSMIT LOGIC PROCESS BEGINS EXECUTION HERE ***** */
IF AWAITING$ACK AND XMT$ANS$TIMER EXPIRED AND
CHECK$ANS$TIMER THEN
  DO;
    CALL SEND$CC(REP);
    XMT$ANS$TIMER = 0;
    XMT$REP$CTR = XMT$REP$CTR + 1;
    IF XMT$REP$CTR > 3 THEN CALL ALARM(7);
  RETURN;
  END;
IF XMT$WAIT THEN RETURN; /* MUST SLEEP UNTIL A BLOCK IS ACK'D */
DO CASE XMT$STATE;
  ; /* CASE 0 NOT USED */
  CALL XMT$STATE$1; /* CASE 1 */
  CALL XMT$STATE$2; /* CASE 2 */
  CALL XMT$STATE$3; /* CASE 3 */

```

```

CALL XMT$STATE$4; /* CASE 4 */
CALL XMT$STATE$5; /* CASE 5 */
END; /* OF CASE XMT$STATE */

END TRANSMIT$LOGIC;
TRANSMITTER: PROCEDURE;
/* LEVEL 2 PROCEDURE */
/* TRANSMITS BYTES ONE AT A TIME ACCORDING TO THE PRIORITY:
1) CONTROL CHARACTER TO SEND, 2) LINE BLOCK DATA TO
SEND, OR 3) SYN TO SEND. IS AUTOMATICALLY SCHEDULED
EVERY TIME THE USAT IS READY. */
IF (CC$PTR1 <> CC$PTR2) AND (T$BUFFER$PTR <> 2 OR T$BUFFER$PTR <>
(T$BUFFER$LENGTH:=TRANSMIT$BUFFER) THEN ARE CONTROL CHARACTERS
/* THE ABOVE LINES CHECK TO SEE IF THERE ARE CONTROL CHARACTERS
TO SEND, CHECKS TO SEE THAT WE'RE NOT FRAMING A BLOCK AND
ASSIGNS BUFFER LENGTH THE VALUE STORED IN THE FIRST ELEMENT
OF THE TRANSMIT BUFFER (WHERE TRANSMIT LOGIC PROCESS STORED
THE LENGTH OF THE BUFFER TO TRANSMIT. */
DO; /* SEND A CONTROL CHARACTER */
CALL SEND$BYTE(CONCHAR$BUFFER(CC$PTR1));
IF (CC$PTR1:=CC$PTR1 + 1) > 7 THEN
CC$PTR1=0; /* RESET THE DEQUE */
END;
ELSE IF T$BUFFER$PTR <= TRANSMIT$BUFFER THEN
/* SEND ONE BYTE OF THE LINE BLOCK */
DO;
IF T$BUFFER$PTR=1 THEN AWAITING$ACK = TRUE;
CALL SEND$BYTE(TRANSMIT$BUFFER(T$BUFFER$PTR));
T$BUFFER$PTR=T$BUFFER$PTR + 1;
END;
ELSE CALL SEND$BYTE(SYN);
END TRANSMITTER;
POLL$DEVICES: PROCEDURE;
/* LEVEL 2 PROCEDURE */

```

```

AUT15370
AUT15380
AUT15390
AUT15400
AUT15410
AUT15420
AUT15430
AUT15440
AUT15450
AUT15460
AUT15470
AUT15480
AUT15490
AUT15500
AUT15510
AUT15520
AUT15530
AUT15540
AUT15550
AUT15560
AUT15570
AUT15580
AUT15590
AUT15600
AUT15610
AUT15620
AUT15630
AUT15640
AUT15650
AUT15660
AUT15670
AUT15680
AUT15690
AUT15700
AUT15710
AUT15720
AUT15730
AUT15740
AUT15750
AUT15760
AUT15770
AUT15780
AUT15790
AUT15800
AUT15810
AUT15820
AUT15830
AUT15840

```

AUT15850
 AUT15860
 AUT15870
 AUT15880
 AUT15890
 AUT15900
 AUT15910
 AUT15920
 AUT15930
 AUT15940
 AUT15950
 AUT15960
 AUT15970
 AUT15980
 AUT15990
 AUT16000
 AUT16010
 AUT16020
 AUT16030
 AUT16040
 AUT16050
 AUT16060
 AUT16070
 AUT16080
 AUT16090
 AUT16100
 AUT16110
 AUT16120
 AUT16130
 AUT16140
 AUT16150
 AUT16160
 AUT16170
 AUT16180
 AUT16190
 AUT16200
 AUT16210
 AUT16220
 AUT16230
 AUT16240
 AUT16250
 AUT16260
 AUT16270
 AUT16280
 AUT16290
 AUT16300
 AUT16310
 AUT16320

```

/* PERFORMED ONCE EVERY LOOP OF THE EXECUTIVE PROCESS
  SCHEDULER. CHECKS TO SEE IF LOCAL I/O DEVICES ARE
  READY AND MARKS THEM READY FOR THE SCHEDULER. */
IF OUTPUT$DEVICE <> NOT$SELECTED THEN
  CO;
  IF OUTPUT$DEVICE=TTY THEN
    DO;
    IF NOT ROR(INPUT(1),2) THEN
      WRITER$PROCESS(DEVICE$READY)=TRUE;
    ELSE WRITER$PROCESS(DEVICE$READY)=FALSE;
  END;
  ELSE
  IF OUTPUT$DEVICE=MAG$TAPE THEN
    DO;
    IF NOT ROR(INPUT(3),2) THEN
      WRITER$PROCESS(DEVICE$READY)=TRUE;
    ELSE WRITER$PROCESS(DEVICE$READY)=FALSE;
  END;
  ELSE /* CHECK THE CARD PUNCH FOR STATUS */
    DO;
    IF NOT ROR(INPUT(4),2) THEN
      WRITER$PROCESS(DEVICE$READY)=TRUE;
    ELSE WRITER$PROCESS(DEVICE$READY)=FALSE;
  END;
  END;
  IF INPUT$DEVICE=NOT$SELECTED THEN
    DO;
    IF NOT(INPUT(TTY$STATUS)) THEN
      INPUT$DEVICE=TTY;
    ELSE
      IF NOT(INPUT(MAG$TAPE)) THEN
        INPUT$DEVICE=MAG$TAPE;
      ELSE
        IF NOT(INPUT(CARD$PUNCH)) THEN
          INPUT$DEVICE=CARD$PUNCH;
        IF INPUT$DEVICE=NOT$SELECTED THEN
          TRANSMIT$LOGIC$PROCESS(DEVICE$READY)=FALSE;
        ELSE
          TRANSMIT$LOGIC$PROCESS(DEVICE$READY)=TRUE;
        END;
      ELSE DO;
        IF INPUT$DEVICE=TTY THEN

```

AUT16330
 AUT16340
 AUT16350
 AUT16360
 AUT16370
 AUT16380
 AUT16390
 AUT16400
 AUT16410
 AUT16420
 AUT16430
 AUT16440
 AUT16450
 AUT16460
 AUT16470
 AUT16480
 AUT16490
 AUT16500
 AUT16510
 AUT16520
 AUT16530
 AUT16540
 AUT16550
 AUT16560
 AUT16570
 AUT16580
 AUT16590
 AUT16600
 AUT16610
 AUT16620
 AUT16630
 AUT16640
 AUT16650
 AUT16660
 AUT16670
 AUT16680
 AUT16690
 AUT16700
 AUT16710
 AUT16720
 AUT16730
 AUT16740
 AUT16750
 AUT16760
 AUT16770
 AUT16780
 AUT16790
 AUT16800

```

DO:
  IF NOT INPUT(1) THEN
    WRITER$PROCESS(DEVICE$READY) = TRUE;
  ELSE
    WRITER$PROCESS(DEVICE$READY) = FALSE;
  END;
ELSE INPUT$DEVICE=MAGTAPE THEN
DO:
  IF NOT INPUT(3) THEN
    WRITER$PROCESS(DEVICE$READY) = TRUE;
  ELSE
    WRITER$PROCESS(DEVICE$READY) = FALSE;
  END;
ELSE DO:
  IF NOT INPUT(4) THEN
    WRITER$PROCESS(DEVICE$READY) = TRUE;
  ELSE
    WRITER$PROCESS(DEVICE$READY) = FALSE;
  END;
END;
IF NOT OR(INPUT(CRT$STATUS), 2) THEN
  OPERATOR$OUTPUT$PROCESS(DEVICE$READY) = TRUE;
ELSE OPERATOR$OUTPUT$PROCESS(DEVICE$READY) = FALSE;
END;
IF VIRTUAL THEN
DO:
  WRITER$PROCESS(DEVICE$READY) = TRUE;
  TRANSMIT$LOGIC$PROCESS(DEVICE$READY) = TRUE;
  OPERATOR$INPUT$PROCESS(DEVICE$READY) = TRUE;
  OPERATOR$OUTPUT$PROCESS(DEVICE$READY) = TRUE;
  END;
END ECLI$DEVICES;
POLL$USAT: PROCEDURE;
/* LEVEL 2 PROCEDURE */
/* FOLLS THE TRANSMIT SIDE OF THE USART TO SEE IF THE
  USART IS READY TO SEND THE NEXT BYTE */
IF OR(INPUT(USART$STATUS), 2) THEN /* USAT NOT READY */
  TRANSMIT$PROCESS(DEVICE$READY) = FALSE;
ELSE /* USAT IS READY */

```

```

TRANSMITTER$PROCESS (DEVICE$READY) =TRUE;

IF VIRTUAL THEN
TRANSMITTER$PROCESS (DEVICE$READY) =TRUE;

END PCLISAT;

/* ***** PROGRAM LEVEL 1 ***** */
/* ***** EXECUTIVE SCHEDULER AND DEVICE MANAGER (NON-INTERRUPT
VERSION). FIRST EXECUTABLE STATEMENT IN THE PROGRAM. */

VIRTUAL=TRUE;
RESTART: CALL INITIALIZE; /* INITIALIZE ALL PROGRAM VARIABLES
AND RETURN HERE FOR RESTART */

DO FOREVER;
/* THE PROGRAM WILL LOOP AS LONG AS THE CPU IS FUNCTIONING
AND ATTEMPT TO PROCESS AUTODIN MESSAGES */
IF NOT VIRTUAL THEN /* SLOW DOWN FOR THE TTY */
CALL TIME(250);

CALL POLL$DEVICES; /* CHECK THE PERIPHERAL DEVICES */
CALL POLL$USAR; /* CHECK THE RECEIVE SIDE OF THE USART */

/* IF SYNCH HAS NOT BEEN ACHIEVED THEN THE RECEIVE
LOGIC PROCESS IS SCHEDULED TO ACHIEVE SYNCH
IF IT'S DEVICE IS READY THEN IT MUST PROCESS AN
INCOMING BYTE NO MATTER WHAT IT'S STATUS IS */

IF RECEIVE$LOGIC$PROCESS (SCHEDULED) OR
RECEIVE$LOGIC$PROCESS (DEVICE$READY) THEN
CALL RECEIVE$LOGIC;

/* IF INCOMING MESSAGE TRAFFIC NEEDS TO BE WRITTEN ON
THE SELECTED OUTPUT DEVICE, THEN THE WRITER PROCESS
WILL BE SCHEDULED

IF WRITER$PROCESS (SCHEDULED) THEN
CALL WRITER;

```

```

AUT16810
AUT16820
AUT16830
AUT16840
AUT16850
AUT16860
AUT16870
AUT16880
AUT16890
AUT16900
AUT16910
AUT16920
AUT16930
AUT16940
AUT16950
AUT16960
AUT16970
AUT16980
AUT16990
AUT17000
AUT17010
AUT17020
AUT17030
AUT17040
AUT17050
AUT17060
AUT17070
AUT17080
AUT17090
AUT17100
AUT17110
AUT17120
AUT17130
AUT17140
AUT17150
AUT17160
AUT17170
AUT17180
AUT17190
AUT17200
AUT17210
AUT17220
AUT17230
AUT17240
AUT17250
AUT17260
AUT17270
AUT17280

```

AUT117290
 AUT117300
 AUT117310
 AUT117320
 AUT117330
 AUT117340
 AUT117350
 AUT117360
 AUT117370
 AUT117380
 AUT117390
 AUT117400
 AUT117410
 AUT117420
 AUT117430
 AUT117440
 AUT117450
 AUT117460
 AUT117470
 AUT117480
 AUT117490
 AUT117500
 AUT117510
 AUT117520
 AUT117530
 AUT117540
 AUT117550
 AUT117560
 AUT117570
 AUT117580
 AUT117590
 AUT117600
 AUT117610
 AUT117620
 AUT117630
 AUT117640
 AUT117650
 AUT117660
 AUT117670
 AUT117680
 AUT117690
 AUT117700
 AUT117710
 AUT117720
 AUT117730
 AUT117740
 AUT117750
 AUT117760

```

/* IF THE OPERATOR INPUT PROCESS DEVICE IS READY
IT MEANS THE HUMAN OPERATOR DESIRES TO INPUT
SOME INFORMATION TO THE PROGRAM */
IF CPERATOR$INPUT$PROCESS (DEVICE$READY) THEN
  CALL OPERATOR$INPUT;
/* IF AN ALARM NEEDS TO BE SENT TO THE HUMAN OPERATOR
THEN THE OPERATOR OUTPUT PROCESS WILL BE SCHEDULED.
IF THE DEVICE IS ALSO READY, THEN THE PROCESS
IS PERMITTED TO RUN. */
IF OPERATOR$OUTPUT$PROCESS (SCHEDULED) AND
OPERATOR$OUTPUT$PROCESS (DEVICE$READY) THEN
  CALL OPERATOR$OUTPUT;
IF NOT VIRTUAL THEN /* SLOW DOWN FOR THE TTY */
  CALL TIME(250);
CALL POLL$USAT; /* CHECK THE TRANSMIT SIDE OF THE USART */
/* IF TRANSMIT LOGIC'S DEVICE IS READY AND WE ARE
SENDING A MESSAGE, THEN WE MUST PROCESS AN
CUTGOING BYTE. */
IF TRANSMIT$LOGIC$PROCESS (DEVICE$READY) AND SENDING THEN
  CALL TRANSMIT$LOGIC;
/* IF TRANSMIT PROCESS'S DEVICE IS READY THEN WE
MUST SEND SOMETHING TO THE TRANSMIT SIDE OF THE
USART, EVEN IF IT IS ONLY SIM PATTERN */
IF TRANSMITTER$PROCESS (DEVICE$READY) THEN
  CALL TRANSMITTER;
IF EYTE$COUNTER=60000 THEN
  DO;
  BYTE$COUNTER=0;
  TEST$COUNTER=TEST$COUNTER + 1;
  IF TEST$COUNTER=3 THEN HALT;
  END;
USAR$CHECK=USAR$CHECK + 1;
IF USAR$CHECK > 250 THEN CALL ALARM(11);
/* WE'VE LOOPED 250 TIMES WITHOUT GETTING A RECEIVED BYTE -
SOMETHING IS OBVIOUSLY WRONG - CALL ALARM. */

```

AUT17770
AUT17780
AUT17790

END: /* OF THE DO FOREVER LOOP */

EOF

BIBLIOGRAPHY

1. Defense Communications Agency Circular 310-D70-30, DCS AUTODIN Switching Center and Tributary Operations, June 1970.
2. Defense Communications Agency Circular 370-D175-1, DCS AUTODIN Interface and Control Criteria, October 1970.
3. Hartmanis, J. and Stearns, R. E., Algebraic Structure Theory of Sequential Machines, Prentice-Hall, 1966.
4. Intel Corporation, Intel 8080 Microcomputer Systems User's Manual, 1976.
5. Intel Corporation, Intellec 8/Mod 80 Microcomputer Development System Reference Manual, 1974.
6. Intel Corporation, Intellec 8/Mod 80 Operators Manual, 1974.
7. Intel Corporation, 8008 and 8080 PL/M Programming Manual, revision A, 1975.
8. Intel Corporation, 8080 PL/M Compiler Operators Manual, revision A, 1975.
9. Intel Corporation, 8080 Simulator Software Package, 1974.
10. Joint Chiefs of Staff Publication JANAP 128(E), Automatic Digital Network (AUTODIN) Operating Procedures, June 1973.
11. Madnick, S. E. and Donovan, J. J., Operating Systems, McGraw-Hill, 1974.
12. Renninger, J. F., Reduction of AUTODIN/NIDN Line Disciplines to Programmed Control Logic, Naval Postgraduate School Master's Thesis, June 1975.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3. Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93940	1
4. Professor V. Michael Powers, Code 52Pw Naval Postgraduate School Monterey, California 93940	2
5. Professor Uno R. Kodres, Code 52Kr Naval Postgraduate School Monterey, California 93940	1
6. Captain Gordon E. Anderson, USMC 2125 Mountain Vista Drive Encinitas, California 92024	1
7. Mr. Kenneth R. Ausich Building 110 Stanford Research Institute 333 Ravenswood Avenue Menlo Park, California 94025	1
8. Commandant of the Marine Corps Headquarters, U.S. Marine Corps (Code CCPP-11) (Attn: Major Beeler) Washington, D. C. 20380	1