MIT/LCS/TM-11

DESCRIPTION AND FLOW CHART OF THE PDP-7/9 COMMUNICATIONS PACKAGE

Philip W. Ward

July 1970

Unnumbered Blank Page

ACKNOWLEDGMENT

Work reported herein was supported in part by Project MAC, an M.I.T. research project sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Nonr-4102(01).

Unnumbered Blank Page

DESCRIPTION AND FLOW CHART OF THE PDP-7/9 COMMUNICATIONS PACKAGE

Technical Memorandum 11

Phillip W. Ward
Electronic Systems Laboratory

July 1970

PROJECT MAC

Massachusetts Institute of Technology

Cambridge Massachusetts 02139

PREFACE

The PDP-7/9 Communications Package was written to provide data transfers between the buffer controller (PDP-7 or PDP-9) of an ESL Display Console and a host computer via a 50-kilobit serial Dataphone link. Initially, only one of the displays (with a PDP-9 buffer controller) was to be operated remotely over a 50-kilobit line, and the only feasible access to the 7094 CTSS host computer was via the PDP-7 buffer controller of the other display, which is directly connected to CTSS channel D. For this connection, the PDP-7 could be looked upon as the "host" for the PDP-9, although it merely served as a message-handling intermediary for the real host, the 7094.

The link between the PDP-9 located at Project MAC (Technology Square) and the PDP-7 located at the M.I.T. Information Processing Center was installed in May, 1969. The communications package described herein was successfully checked out, but integration with the display executive programs of the PDP-7 and PDP-9 to permit remote display operation had not been accomplished when work was terminated in March, 1970.

The work described was performed by the Display Group of the M.I.T. Electronic Systems Laboratory, with the joint support of Project MAC and the U.S. Air Force Materials Laboratory, Wright-Patterson AFB, under Contract F 33615-69-C-1341. The programs were written and debugged by D.E. Thornhill, H. Levin, and M.F. Brescia. This description by P.W. Ward was prepared as a user's guide.

John E. Ward
Electronic Systems Laboratory

CONTENTS

DESCRIPTION OF THE PDP-7/PDP-9 COMMUNICATIONS PACKAGE	page	1
Introduction		1
Overview of Communications Package		1
CONVENTIONS FOR CALLING PDP-7/PDP-9 COMMUNICATIONS PACKAGE		7
Initialization		7
Recieve Set		8
Scheduling Routine		9
Receive Message		11
Send Message		14
Send Message Master/ Slave Modification		17
SUGGESTED PROGRAM MODIFICATIONS		18
Appendix I -- Flow Chart of Communications Package		20
Appendix II -- Communications Package Error Parameters		52

Unnumbered Blank Page

Note: This page is an unnumbered blank used to preserve the printed appearance of the author's original document. It is the back side of the previous numbered page.

DESCRIPTION OF PDP-7/PDP-9 COMMUNICATIONS PACKAGE

Introduction

The program to be described was written specifically for the purpose of providing a message handling facility between a PDP-7 and a PDP-9 computer utilizing a 50 kilobit telephone transmission link (see Figure 1). Each computer is physically connected to the telephone media (typically a Bell 303 Modem) via a DEC 637 Interface*. The 637 conforms (at the modem interface) to the Electronic Industries Association Standard RS-232-B for full duplex operation. At the 637-to-modem level, information is transmitted and received in serial bit synchronous form. (In addition, the Bell Modem "scrambles" and "descrambles" the bit stream to provide uniform spectrum distribution and utilization.) At the Computer-to-637 Interface level, information is transmitted in serial byte synchronous form. In this program implementation, one byte is an 8-bit character; but 6, 7 or 9 bit options are possible with the same 637 Interface unit. The message handling program communicates with the 637 by Input-Output-Transfer (IOT) Commands which provide the status and control information required to effect transmission (and reception) of 8 bit characters from (and to) the PDP-7/9 Accumulator.

Overview of Communications Package

The user of the Communications Package interacts with essentially three subprograms (hereafter called procedures):

---

*The 637 Interface is also referred to by Digital Equipment Corporation as "Bit Synchronous Data Communication System Type 637" or simply "637 Data Communication Channel."

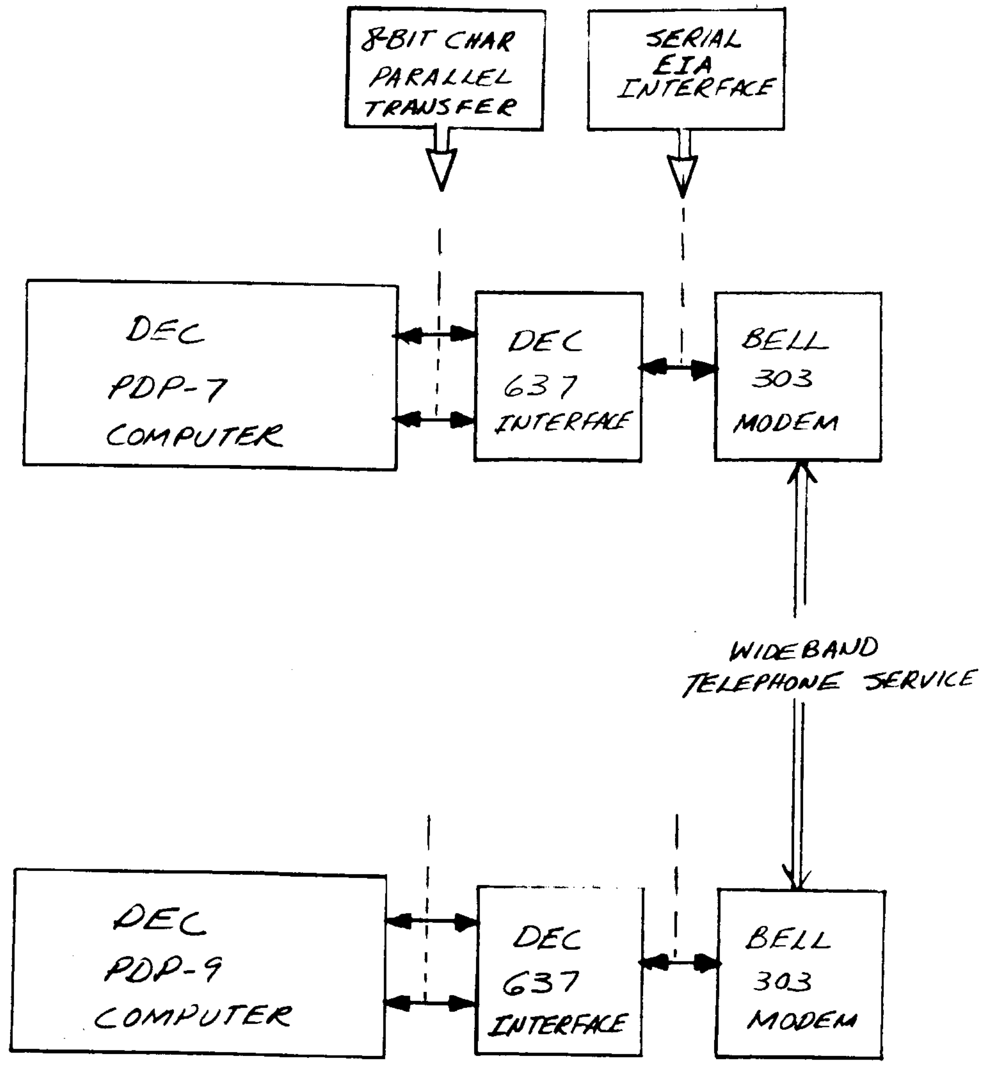


FIGURE 1 - BLOCK DIAGRAM OF PDP-7/9

COMMUNICATIONS LINK

(1) Initialization

(2) Receive Message

(3) Send Message

How the user calls these procedures will be treated individually following a brief overview of what the procedures do.

Figures 2, 3 and 4 illustrate in block form the general flow of operations within a given procedure. For a detailed Flow Chart of the Communications Package refer to Appendix I.

Referring to Figure 2, the Initialization procedure clears the parameter list used by the program, puts the modem into synchronization, waits for an indication that the remote user has initialized, then returns control to the user.

Figure 3 illustrates the Receive Message procedure. Note that the user, in general, interacts with this procedure at three different instances.

(1) User calls Receive Set to provide parameters needed to process an incoming message.

(2) User calls Receive Message when he knows a message is forthcoming.

(3) During the procedure (2), if the entire message is received successfully, the procedure calls the user's scheduling routine. This step not only provides the user with an indication that there were no transmission errors, but also a chance to call procedure (1) again. Thus, an appropriate scheduling routine will prevent overwriting of a message by a subsequent message. After the scheduling routine is complete, control is returned to procedure (2) for completion.

The Send Message procedure is illustrated in Figure 4. In this case, the user invokes the procedure and passes the necessary parameters at the same instance. The procedure attempts to send the message, and, if successful,

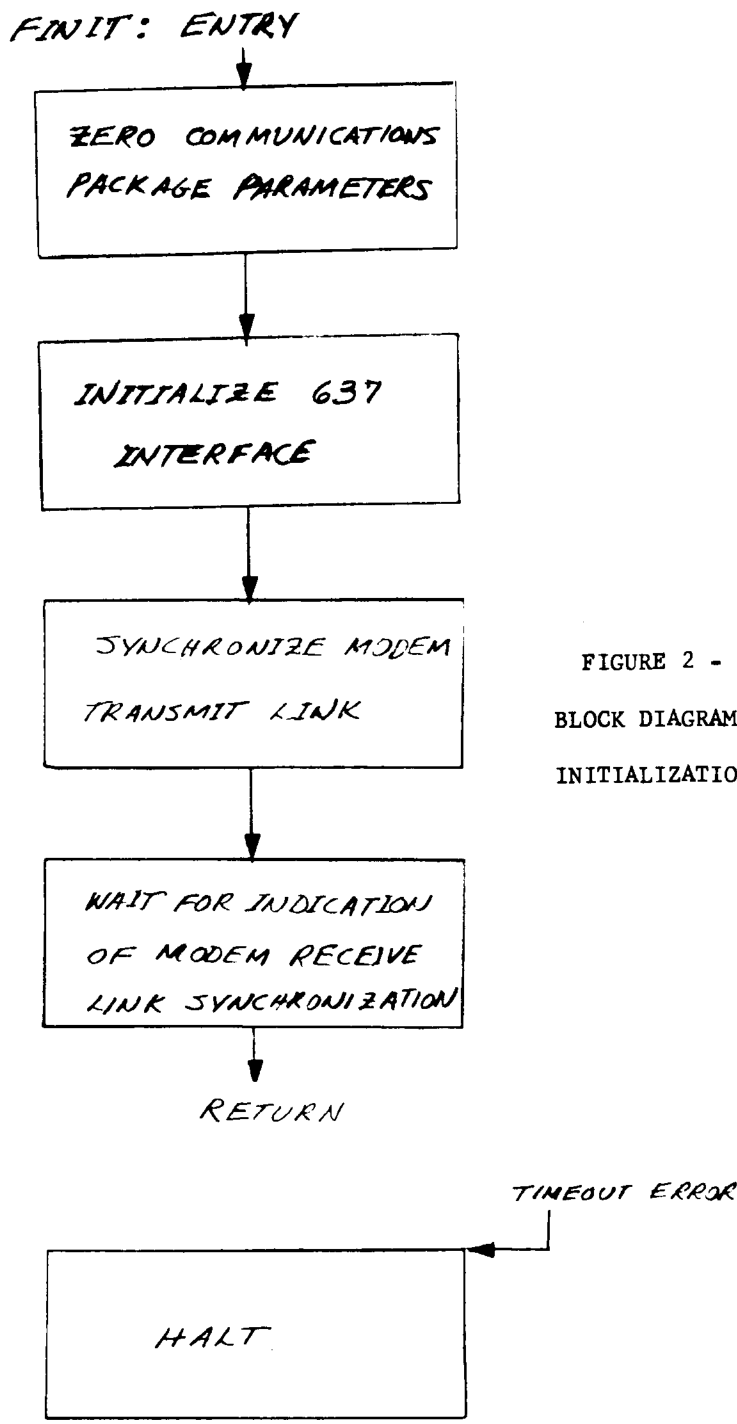


FIGURE 2 -

BLOCK DIAGRAM OF

INITIALIZATION PROCEDURE

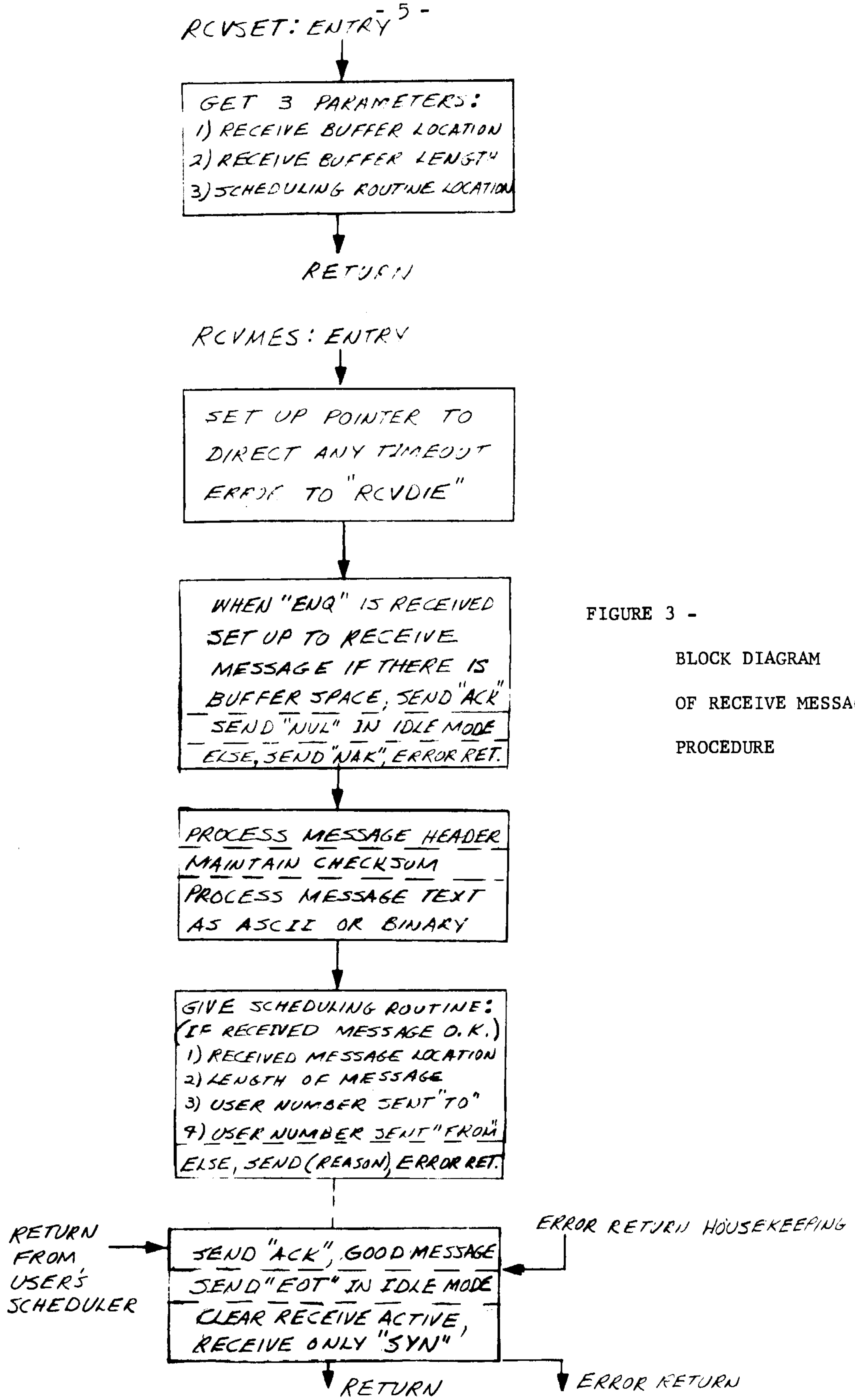


FIGURE 3 -

BLOCK DIAGRAM

OF RECEIVE MESSAGE

PROCEDURE

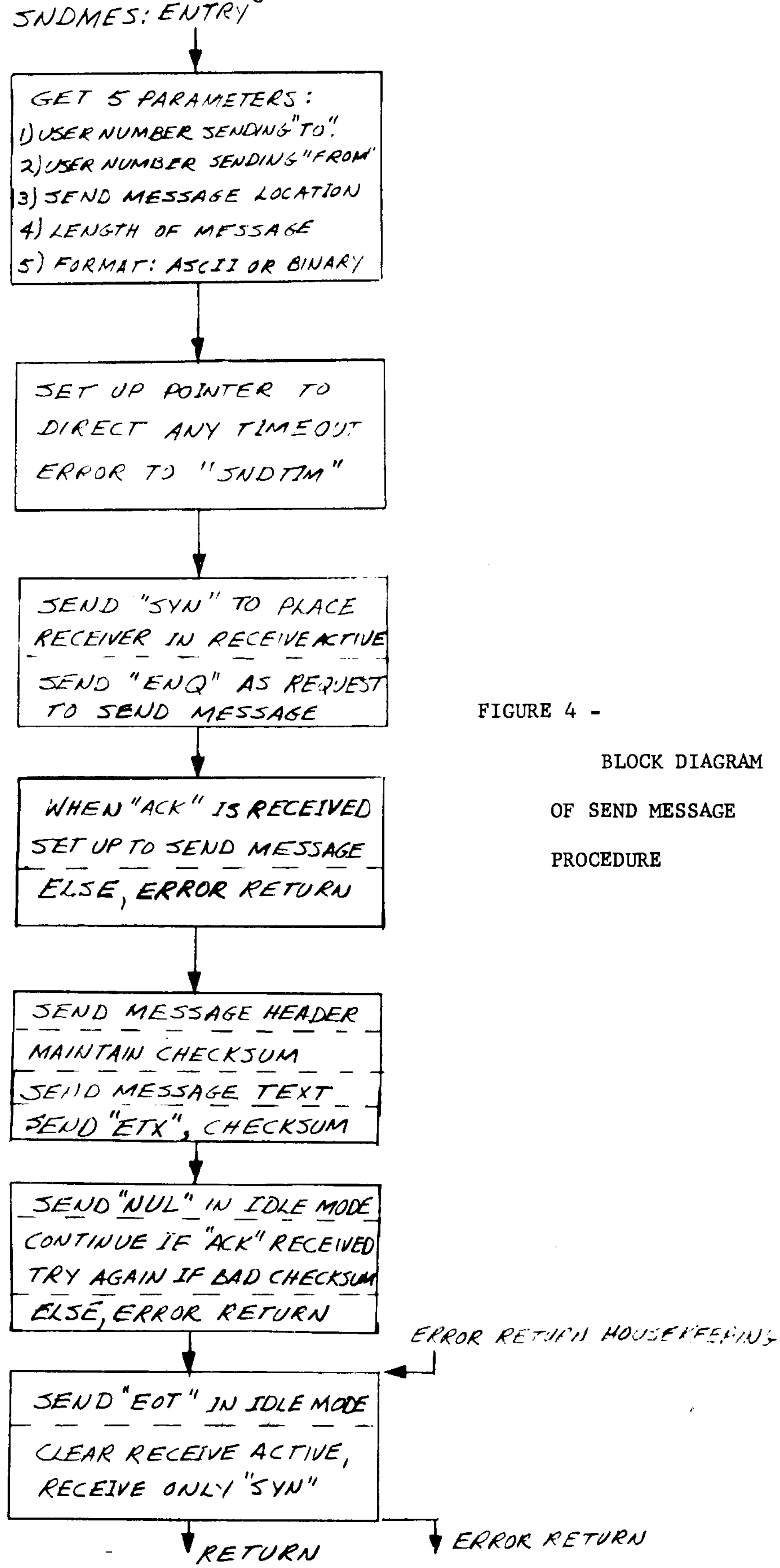


FIGURE 4 - BLOCK DIAGRAM OF SEND MESSAGE PROCEDURE

returns control to the user at his normal return entry point. Otherwise, control is returned to the user's error return.

The program takes care of all of the overhead and the input/output operations required with the 637 Interface to get a message processed. The user may opt that the message text to be sent consist of ASCII (non-control) characters or 18-bit Binary words. The message header, checksum, send/receive protocol, and the assertion of a program interrupt before sending a message, come under the overhead items which are taken care of by the program. However, the burden is on the user to:

(1) Provide the required interrupt service for receiving a message or, alternatively, if the priority interrupt facility has been disabled, a routine to determine the presence of a message.

(2) Inhibit interrupts when sending a message.

(3) Recover under error return conditions.

CONVENTIONS FOR CALLING PDP-7/PDP-9 COMMUNICATIONS PACKAGE

Now that some insight has been presented concerning what the program does, the specific details of invoking the procedures in the program will be treated in the same order.

Initialization

The first procedure that the user will invoke is initialization. This is accomplished by calling FINIT with no arguments. A typical call follows:

```
        .
        .
        .
    JMS FINIT            /CALL FINIT WITH NO ARGUMENTS
ABLE,   .
        .
        .
```

After initialization is accomplished, control will return to the instruction at ABLE. If initialization cannot be accomplished the program will come to a halt and the operator must take appropriate corrective action, then start over. The initialization routine assumes someone on the other end is also trying to initialize and will wait until he does so before returning control. Initialization not only brings the modem link up to sync, it also zeros the procedure parameter list. The names and descriptions of the parameters used to indicate error conditions are given in Appendix II. The user may wish to use the parameter list to determine his program action in case of error returns from the Receive Message or Send Message procedures.

Receive Set

Before the user invokes the receive message routine the first time, he must call RCVSET with three arguments. A typical call procedure follows:

```
                .
                .
        JMS RCVSET                  /CALL RCVSET WITH 3 ARGUMENTS BELOW
        LAC ARG1R                   /RECEIVE BUFFER STARTING LOCATION
        LAC ARG2R                   /RECEIVE BUFFER LENGTH
        LAW ARG3R                   /RECEIVE SCHEDULING ROUTINE ENTRY POINT
BAKER,          .
                .
                .
```

Control is returned to the instruction at BAKER when RCVSET has accepted the three arguments. If a message should come in before RCVSET has been invoked by the user, it will be refused on the basis that no buffer space is available. If one comes in afterwards, it will be placed in the buffer space last specified. ARG1R and ARG2R are self explanatory, except possibly it should be clarified that ARG2R is the octal number of contiguous locations available in the receive buffer regardless of whether the data type is ASCII or Binary. ARG3R is the entry point to the user's receive scheduling routine.

This example also serves to illustrate how arguments are passed in the PDP-7/9. When the program executes the instruction JMS RCVSET, control is transferred to the instruction in the next location after RCVSET and the address of the instruction LAC ARG1R is placed in location RCVSET. By convention, arguments are passed by writing one instruction for each argument which, when executed, will place the argument in the Accumulator. Thus, LAC ARG1R puts the contents of ARG1R into the Accumulator while LAW ARG3R puts in the address of ARG3R. How these arguments are taken at the invoked procedure is illustrated in a later example. We can assume that after the arguments have been taken, the address contained in RCVSET will be BAKER. Thus, control is returned to BAKER by executing the instruction: JMP I RCVSET. Note that even if the return were to the instruction LAC ARG1R, no problem occurs in the program.

Scheduling Routine

The Receive Message procedure invokes the scheduling routine _if_ a message has been received successfully. It does this _before_ it acknowledges the message to the sender. As explained previously, this is the time to call RCVSET again if the user expects another message before he is finished with the present one. At the other end, the sending procedure will only wait about 1.5 milliseconds real time for a reply, so the scheduline routine cannot be too time consuming.

The scheduling routine can be as simple as the following example:

```
          .
          .
          .
ARG3R,    0                     /ENTRY POINT SCHEDULING ROUTINE
          JMP I ARG3R           /RETURN IMMEDIATELY
          .
          .
          .
```

The above routine simply returns control back to the receive message program and defers any action on the received message until later. However, the user may also wish to store the arguments being passed at this point of the program, namely RCVBUF (same as ARG1R, first location where the message was placed); RCVCNT (the length of buffer space used); TO (user number to whom message is sent); FROM (user number from whom message is sent). Assuming this information is needed, an example of an alternative routine which accepts the above arguments follows:

```
ARG3R,  0                                         /ENTRY POINT SCHEDULING ROUTINE
        XCT I ARG3R    DAC RMSBEG    AOM ARG3R    /PUT RCVBUF INTO RMSBEG
        XCT I ARG3R    DAC RMSLTH    AOM ARG3R    /PUT RCVCNT INTO RMSLTH
        XCT I ARG3R    DAC RMSTO     AOM ARG3R    /PUT TO INTO RMSTO
        XCT I ARG3R    DAC RMSFRM    AOM ARG3R    /PUT FROM INTO RMSFRM
            .
            .                                     /POSSIBLE CHANGE IN RCVSET
            .
        JMP I ARG3R                               /RETURN
```

This routine puts the four arguments into user's locations named RMSBEG, RMSLTH, RMSTO, RMSFRM respectively.

This routine also serves to illustrate how arguments are accepted in the PDP-7/9. The entry point to the routine contains no instruction. When ARG3R is invoked by a JMS instruction, the location of the next instruction is stored at ARG3R and control is given to ARG3R + 1. Following the convention for accepting arguments, the scheduling routine issues an XCT instruction indirected through ARG3R to access the first argument. This places the first argument in the Accumulator. This is followed by an AOM instruction to increment the pointer in ARG3R to the next argument. The process continues until all arguments are taken, leaving the pointer in ARG3R at the return entry point of the invoking procedure. This routine continues with some user defined algorithm that may decide to change the RCVSET parameters for the next message. Ultimately, control is returned by the instruction: JMP I ARG3R.

Receive Message

Assuming that the Receive Set procedure has been furnished with the necessary housekeeping parameters, the Receive Message procedure may be invoked at any time there is an indication of a message being sent. However, the user is almost certain to encounter a timeout error condition if he invokes the Receive Message procedure arbitrarily. The best arrangement is to direct the invocation on an interrupt basis, since this guarantees that the 637 Interface has been activated by a sender and a message is forthcoming. In order to clarify this point, the conventions followed by the Communications Package in this regard are described. The last step of any procedure orders the 637 Interface:

(1) Transmit link to repeatedly transmit an "EOT" control character in order to maintain the Bell Modem synchronization. This leaves the 637 Interface transmit link "idling" and requires no further IOT operations, but a synchronous bit stream is maintained in the modem as required.

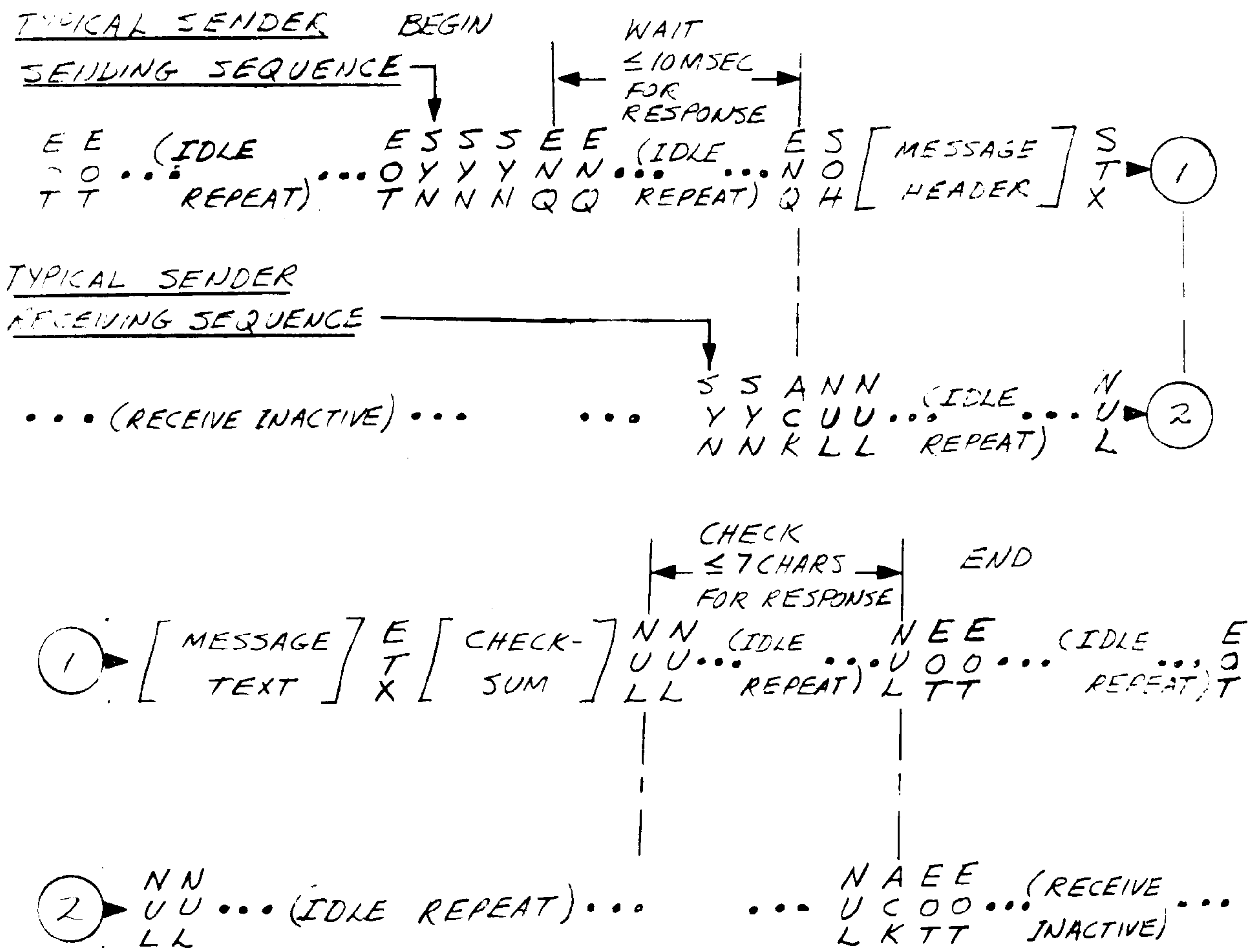
(2) Receive link not to accept any more characters unless the "SYN" control character is detected. Under this status, the 637 receive link continually checks the serial bit stream for any sequence of bits that match the "SYN" character code. When a match is found, the receive link activates, sets its interrupt line active and begins assembling every 8 bits following "SYN" into characters.

In accordance with this convention, the Send Message procedure initially transmits the "SYN" character (three times) to alert the receiver. See Fig. 5.

It is possible to test for a message received status without the use of the interrupt scheme, but this requires IOT commands to the 637 Interface. A typical routine which waits on a message to arrive, then calls RCVMES with no arguments is as follows:

```
        .
        .
        .
        DZM ARG3R                 /CLEAR ENTRY POINT ARG3R
        SNE                       /SKIP IF THERE IS LINE CONTROL
        JMP RPRINT                /GO TO LOST LINE CONTROL PRINT
        SRF                       /SKIP IF 637 IS RECEIVE ACTIVE
        JMP .-3                   /KEEP CHECKING UNTIL ACTIVE
        JMS RCVMES                /CALL RECEIVE MESSAGE
CHARLY, LAC ARG3R                 /ARG3R USED AS MESSAGE FLAG
        SNA                       /IF ARG3R = 0, BAD MESSAGE
        JMP RMSERC                /GO TO ERROR RECOVERY ROUTINE
        JMP RMSPRC                /GO TO MESSAGE PROCESS ROUTINE
        .
        .
        .
```

Control is returned to CHARLY after a message has been processed by RCVMES. It is assumed that subroutines RPRINT, RMSERC and RMSPRC exist in the user's program. If no problems were encountered by RCVMES, ARG3R (the user's scheduling routine) will have been accessed. For this reason ARG3R is used as a flag. If ARG3R is non-zero, the program flow is to RMSPRC where the user processes the message. The other subroutines are determined by the user's application. Typically, RPRINT might be a routine which notifies the operator that the modem has lost its line control.



NOTE:

[MESSAGE HEADER] = [MESSAGE NUMBER MOD 77_8 MESSAGE "TO" MOD 77_8 MESSAGE "FROM" MOD 77_8]

[MESSAGE TEXT] = [FS   ASCII TEXT FORMAT] OR [GS   BINARY TEXT FORMAT]

FIGURE 5 - CHARACTER SEQUENCE

OF TYPICAL MESSAGE

RMSERC could be quite involved in checking the receive message parameter list until the source of error is found and some action taken dictated by the type of error found.

Send Message

If the user desires to send a message he issues a call to SNDMES with five arguments. A typical call procedure follows:

```
          .
          .
          .
      IOF                        /DISABLE INTERRUPT
      JMS SNDMES                 /CALL SEND MESSAGE WITH 5 ARGUMENTS BELOW
      LAC ARG1S                  /TO USER NO.
      LAC ARG2S                  /FROM USER NO.
      LAC ARG3S                  /FIRST LOCATION OF MESSAGE
      LAC ARG4S                  /LENGTH OF MESSAGE BUFFER
      LAC ARG5S                  /0=ASCII, ELSE BINARY
DOG,  JMP SNDERT                 /SNDERT=ENTRY MY ERROR ROUTINE
EASY, ION                        /EASY=MY NORMAL RETURN ENTRY POINT
          .
          .
          .
```

If the message is acknowledged by the receiver, the procedure returns control to the instruction at location EASY, otherwise the return is an error return to DOG which must transfer control to the user's send error routine at SNDERT.

Unless the recipient of the message needs the TO and FROM numbers contained in ARG1S and ARG2S, these arguments can be any arbitrary constant, including 0. In any case the procedure only sends the rightmost six bits of either argument. The next two arguments are self explanatory, since the procedure must know where to get the message and how long it is.

b4	b3	b2	b1	b7 b6 b5: 0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	"	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	'	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	∧	n	¬
1	1	1	1	SI	US	/	?	O	_	o	DEL

Control Characters

Communication Control Characters

Key Characters

Second Category Representing Argument (Includes the Control Character DEL)

Control Character DEL Included in the Argument Set

FIGURE 6 - USASCII CODE TABLE

The last argument offers the user the option of sending the message in 18 bit binary words (if ARG5S is any non-zero constant) or as two (7-bit ASCII) 8 bit characters per word (right justified).

If the user requests the ASCII format, he must have a legal ASCII character in every character position of the send message buffer area indicated to the send routine and he must not use any of the 16 ASCII control characters in the field which includes "ETX". Specifically, no characters with format:

```
b8            b1
X O O X O X X X
```

can be included within the text (in user's send buffer). The X's indicate "don't-care" bit positions. No problem arises in the send routine, but the receive routine must look for control characters and specifically the "ETX" to determine the end of the text. Refer to Figure 6 for ASCII code.

None of the above problem occur in the binary mode since the program sends only 6 bits of the 18 bit word at a time, and the ASCII bit positions "b8 b7" are forced to "O 1" at the send end and masked at the receive end when the message text is being processed. However, the program assumes that the user at the receive end knows whether the message is in ASCII or Binary format, i.e. this parameter is not passed to the receiver of the message. The user should utilize the "TO" and "FROM" parameter information to indicate which to the receiver.

As mentioned earlier (without explanation) the priority interrupt facility should be disabled just before invoking the Send Message procedure. This is because Send Message takes the 637 Interface out of "idle" mode when it gets the go-ahead from the receiving end to send. At this point, every character transmitted has to be a new character furnished by the Send Message procedure on demand by the 637 Interface. Thus, the procedure has to be ready and waiting with the next character when the 637 has finished transmitting the current character. The procedure has one character interval in real time (about 160 microseconds) to perform intermediate fetching and formatting tasks to prepare the next character. This is ample time unless an interrupt is permitted, then the timing is indeterminate. If the response to the 637 Interface's request for the next character is late, loss of line control follows. Initialization will then be required before communications can be re-established.

Send Message Master/Slave Modification

Since it is possible for both parties to request to send at the same time, some provision for Master/Slave priority must be written into the Send Message routine. If the user is the Master, no modification is required. If Slave, change the program in the SNDWAK routine by replacing the instruction:

```
JMP SNDWAK
```

with two instructions:

```
JMS RCVMES
JMP SNDAGN
```

SUGGESTED PROGRAM MODIFICATIONS

In the course of preparing this description of the existing PDP-7/9 communications package, it was noted that certain channel conditions could possibly cause endless looping. Several minor program modifications, described below, should eliminate this danger, but have not been implemented. The suggested modifications are shown in the flow charts, and are marked with an asterisk to indicate a discrepancy between the charts and the actual program listing. One additional change is required if the functions of Master and Slave are reversed. These changes should be made in any future use of the package.

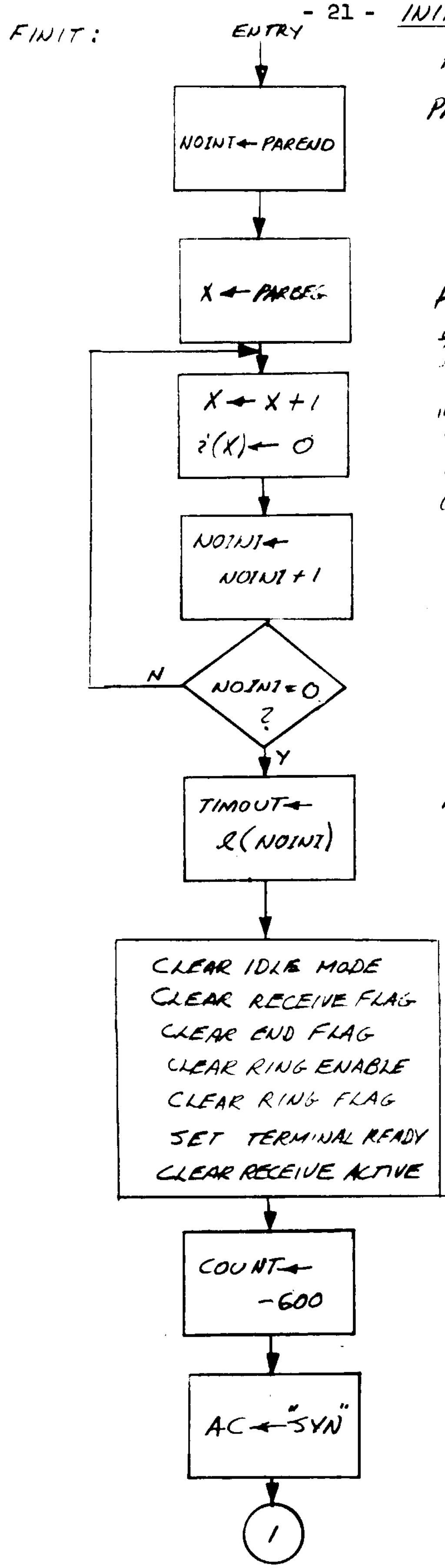
1. RECEIVE MESSAGE (1), near label RCVMES. Change TIMOUT error return from RCVDIE to a new subroutine called RCVTIM which alters the TIMOUT error return before proceeding to RCVDIE routine. This prevents an infinite loop condition in case the transmit routine always times out when called. RCVTIM is shown in RECEIVE MESSAGE (13).
2. RECEIVE MESSAGE (5), near label RCVGO. Add the instruction which increments NOSTX by 1 when "STX" is not found at the start of text.
3. RECEIVE MESSAGE (13), near label RCVDIE. Add the label variable REXIT, the subroutines RCVTIM and RCVEMG as shown to prevent loop condition described in change 1 above.
4. RECEIVE (3), near label RTXERR. Replace instruction JMS RCVDIE with JMS RCVEND.
5. SEND MESSAGE (2), near label SNDWAK. Add the modification per description on page 17 of report "Send Message Master/Slave Modification".

6. SEND MESSAGE (8), SNDTIM routine. Change TIMOUT error return from SNDTIM to a new subroutine called SNDEMG and put modem into idle mode before calling SNDEND. SNDEMG prevents an infinite loop condition in case the transmit routine always times out when called. SNDEMG is shown adjacent to SNDTIM and SEXIT is a label variable to be added as shown near subroutine SNDEND.

APPENDIX I

FLOW CHART
OF
COMMUNICATIONS
PACKAGE

INITIALIZATION (1)



First the status parameters are zeroed.

PAREND contains (-) the number of locations between, but not including PARBEG and PAREND. These locations act as a named list of status parameters which are used and updated by this program.

PARBEG contains its own location. As used, it effectively points to the first status parameter location.

"X" is location 12_8 which has the property that it automatically increments its contents by one (autoindexing) each time it is accessed indirectly, then the resulting indirect address is accessed.

All status parameters in the list are zeroed when the count in NOINI has reached 0.

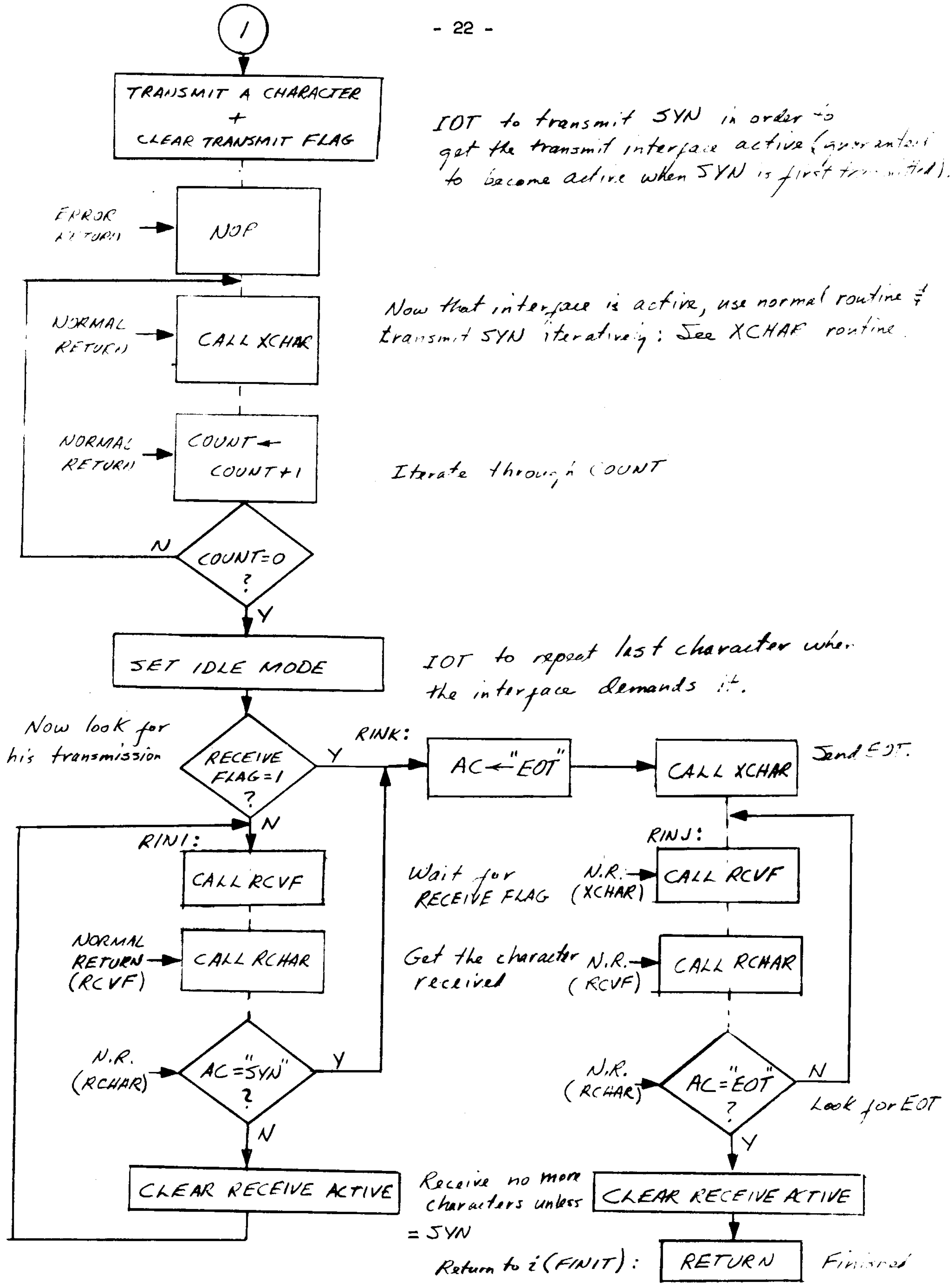
Location NOINI is now the error return in case a timeout occurs during initialization. If a return is made here, the program will halt.

Initialize modem: See IOT Command Descriptions for 637 Interface.

Setting up to send $600_8 = 384_{10}$ synchronizing characters (8 bits/character × 384 = 3074 bits) required by Bell 303 Modem.

SYN = 226_8 the 637 Interface synchronizing signal code for an 8-bit character option.

INITIALIZATION (2)

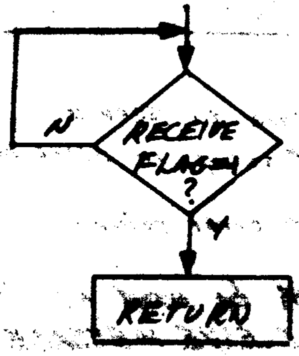


Action Routines Used By: INITIALIZATION (3)

NOINI: ENTRY +1 Location of invoking instruction here

NOINI contains location of part of program which timed out if HALT occurs during initialization.

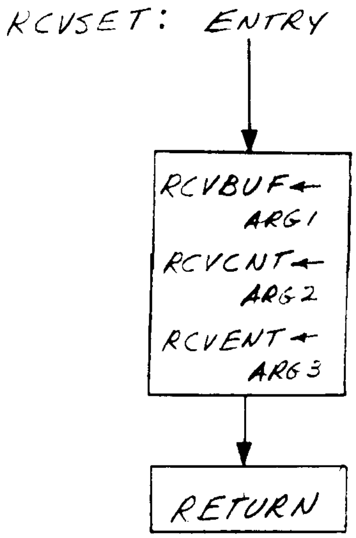
RCVF: ENTRY +1 Location of invoking instruction here



will wait until "SYN" is received before it returns.

Returns to i (RCVF)

RECEIVE MESSAGE (1)



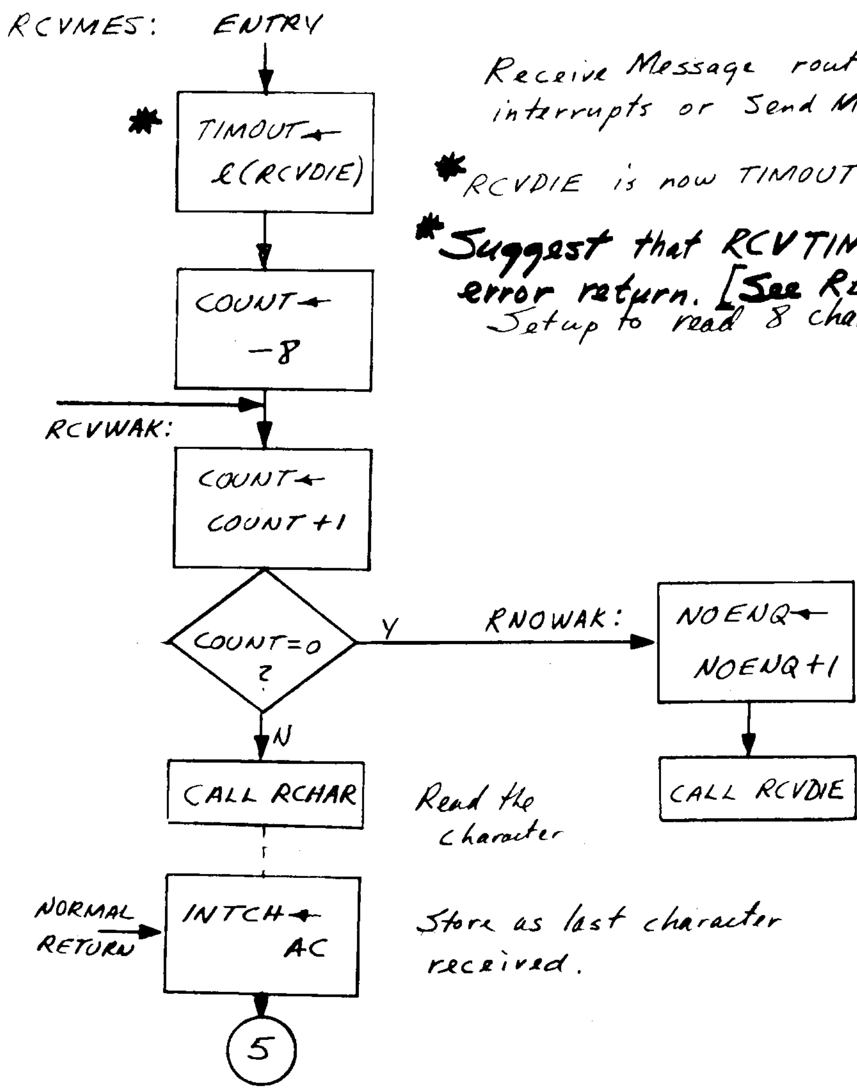
Setup for providing arguments used by RECEIVE MESSAGE

(1) Starting location of buffer where message is to be sent.

(2) Length of buffer space available for message.

(3) Location of user's receive schedule routine.

Return location pointer now points to users normal return.



Receive Message routine to service interrupts or Send Message in slave mode.

*RCVDIE is now TIMOUT error return.

***Suggest that RCVTIM be used as TIMOUT error return. [See RECEIVE MESSAGE (13)]**

Setup to read 8 characters

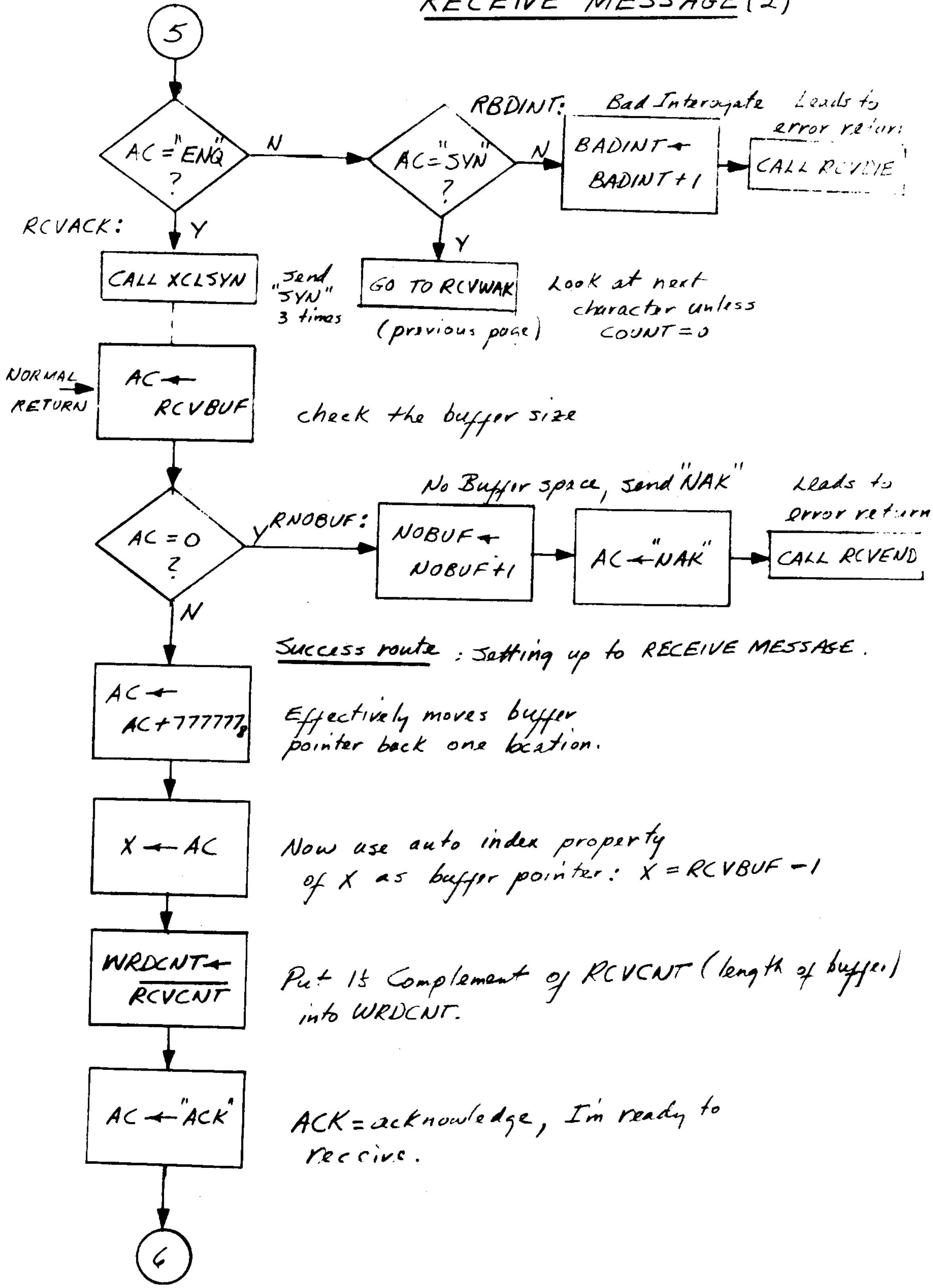
Make a note that no "ENQ" was found within 8 characters.

Error return

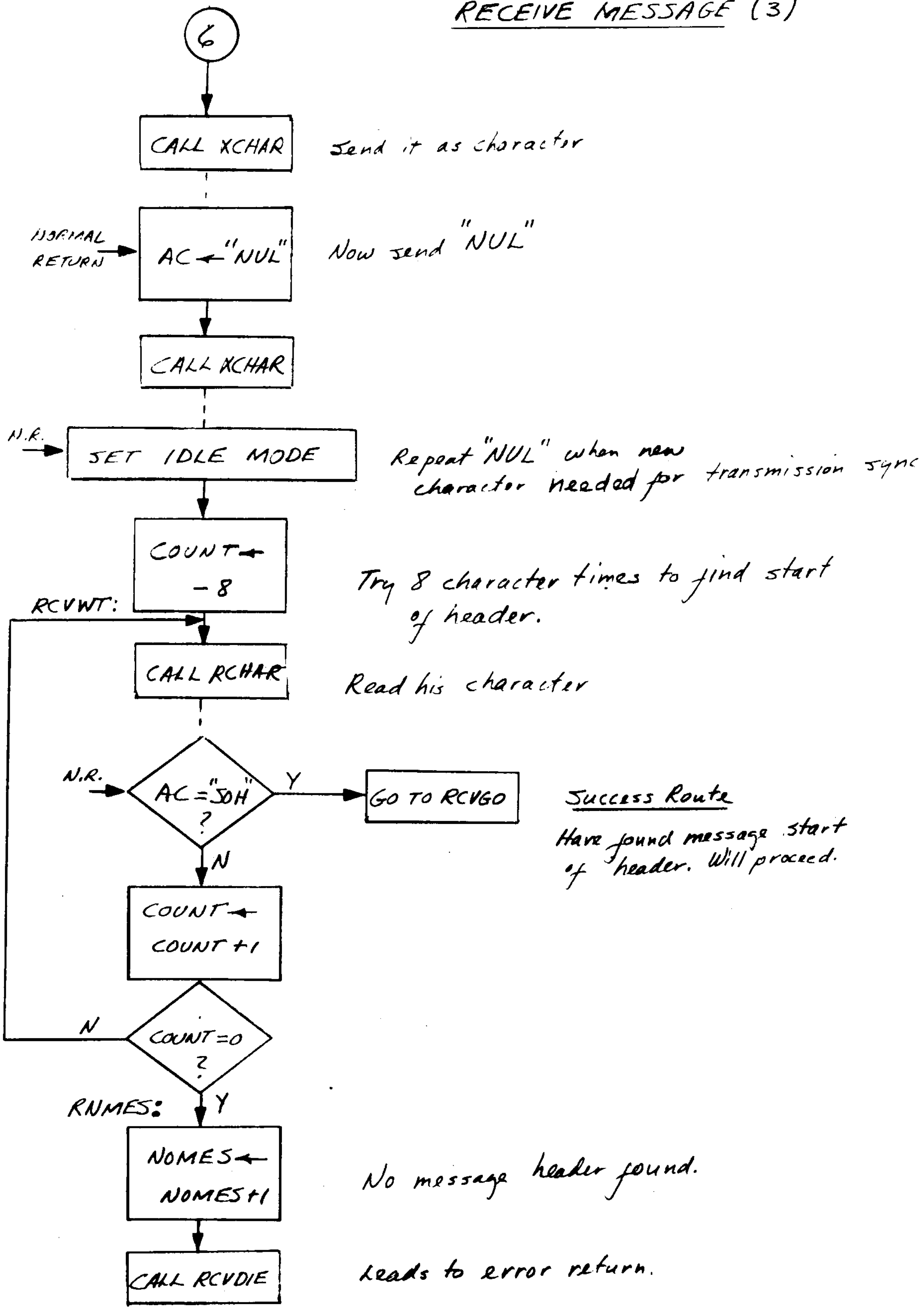
Read the character

Store as last character received.

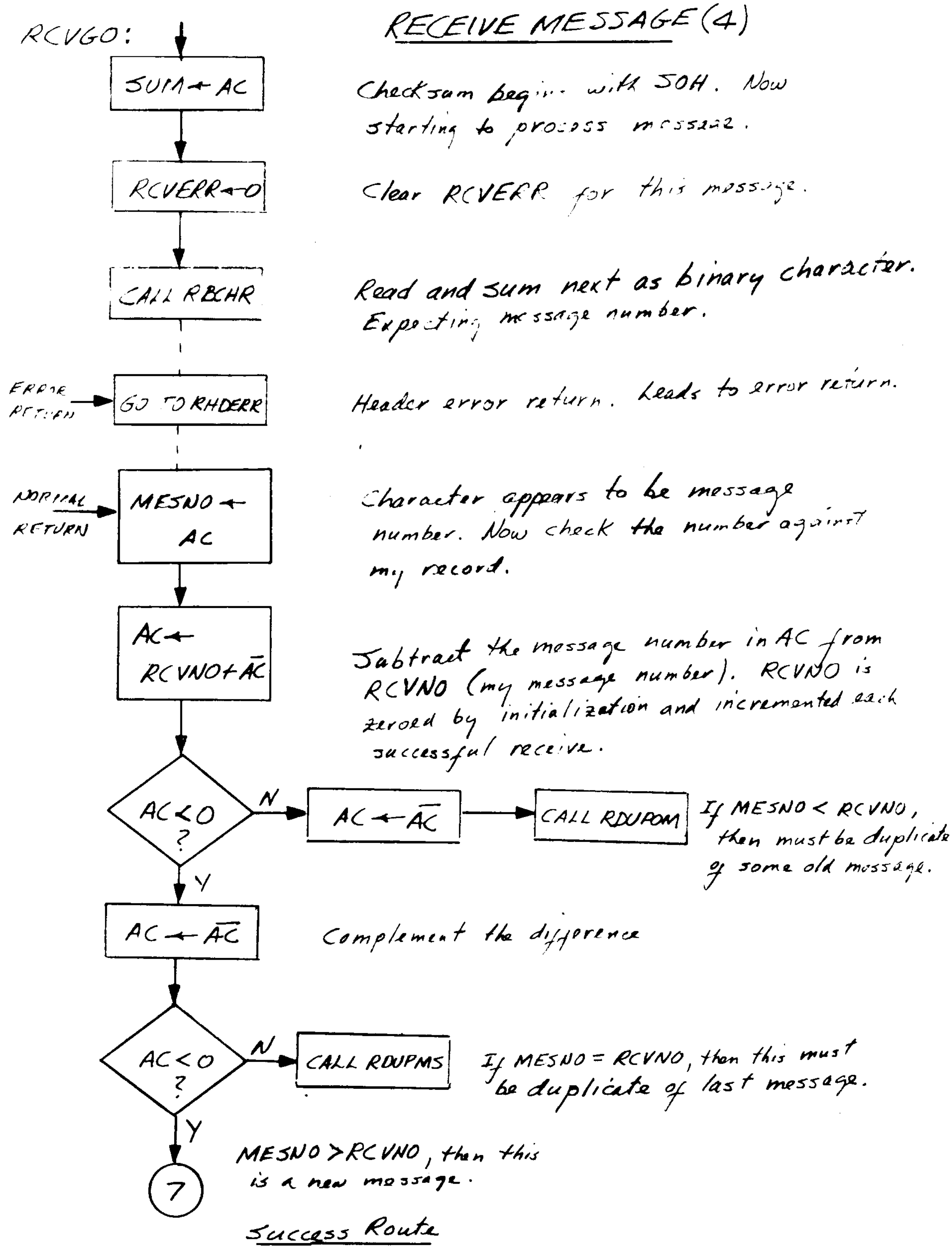
RECEIVE MESSAGE (2)



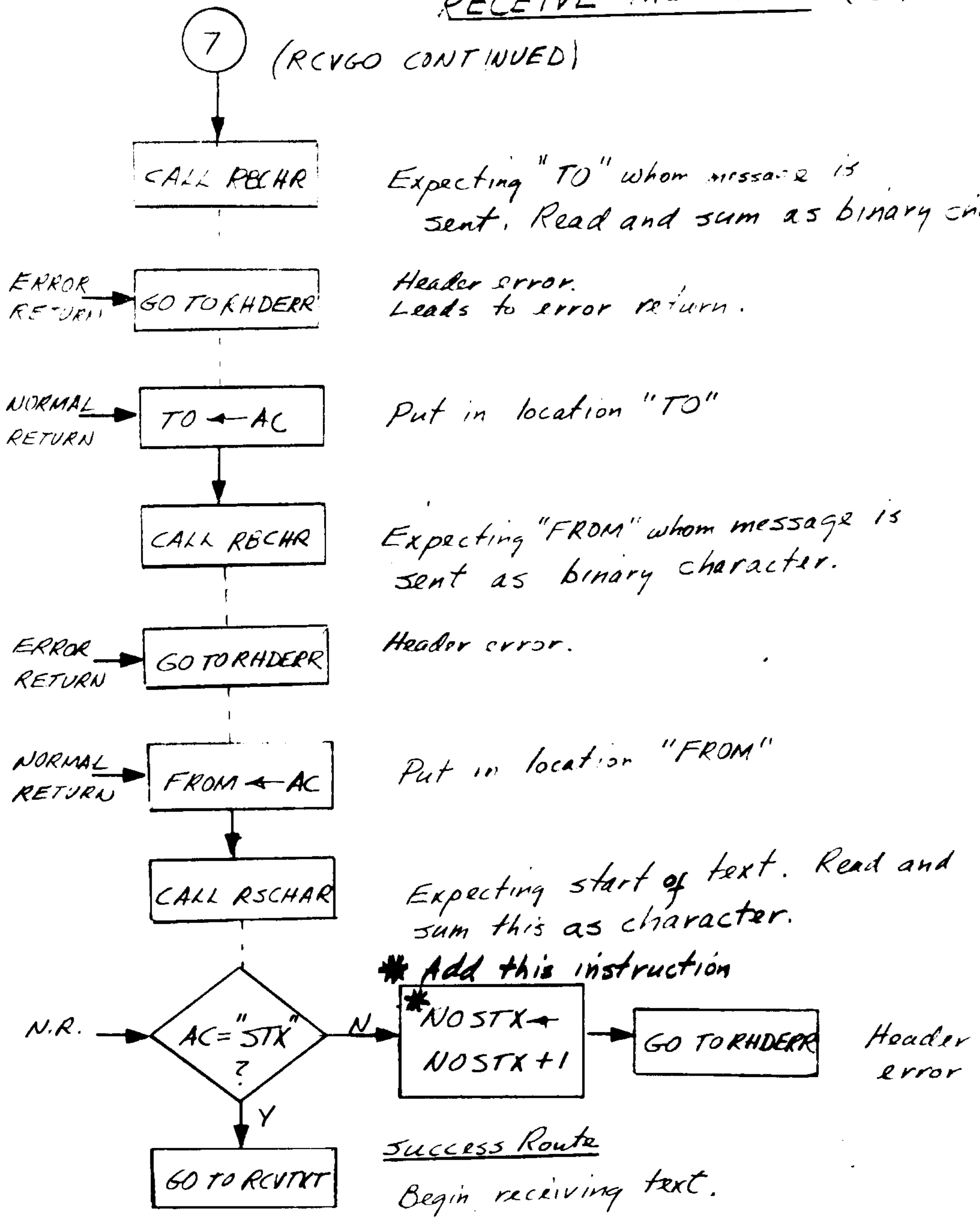
RECEIVE MESSAGE (3)



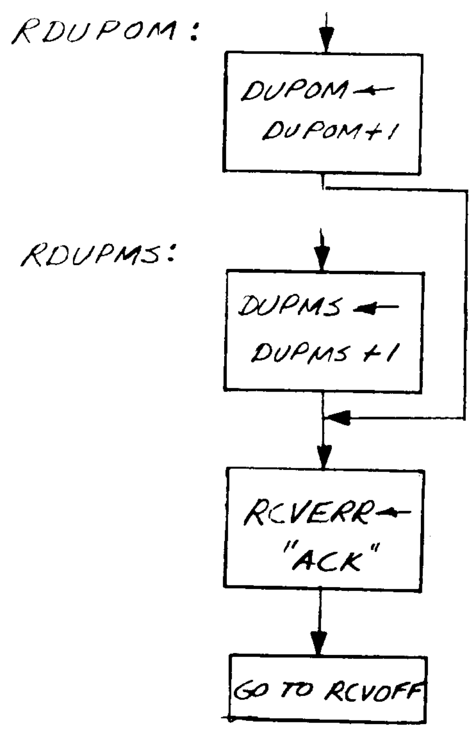
RECEIVE MESSAGE (4)



RECEIVE MESSAGE (5)



RECEIVE MESSAGE (6)

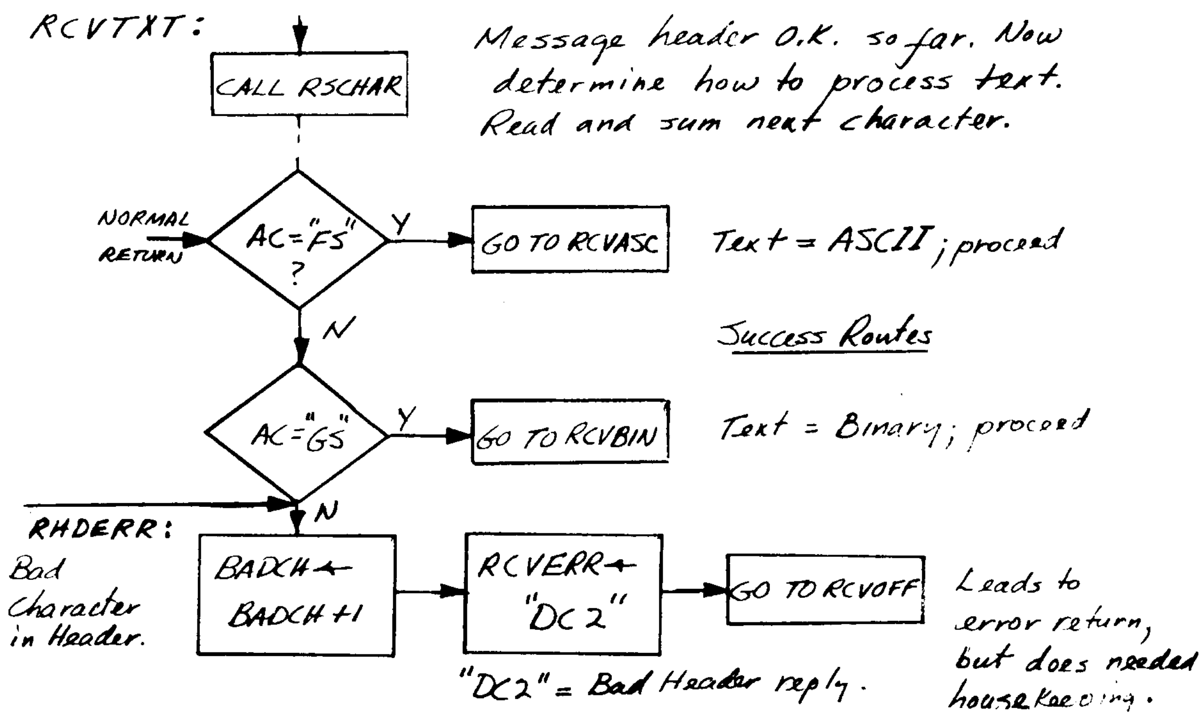


Duplication of old message number indicated.

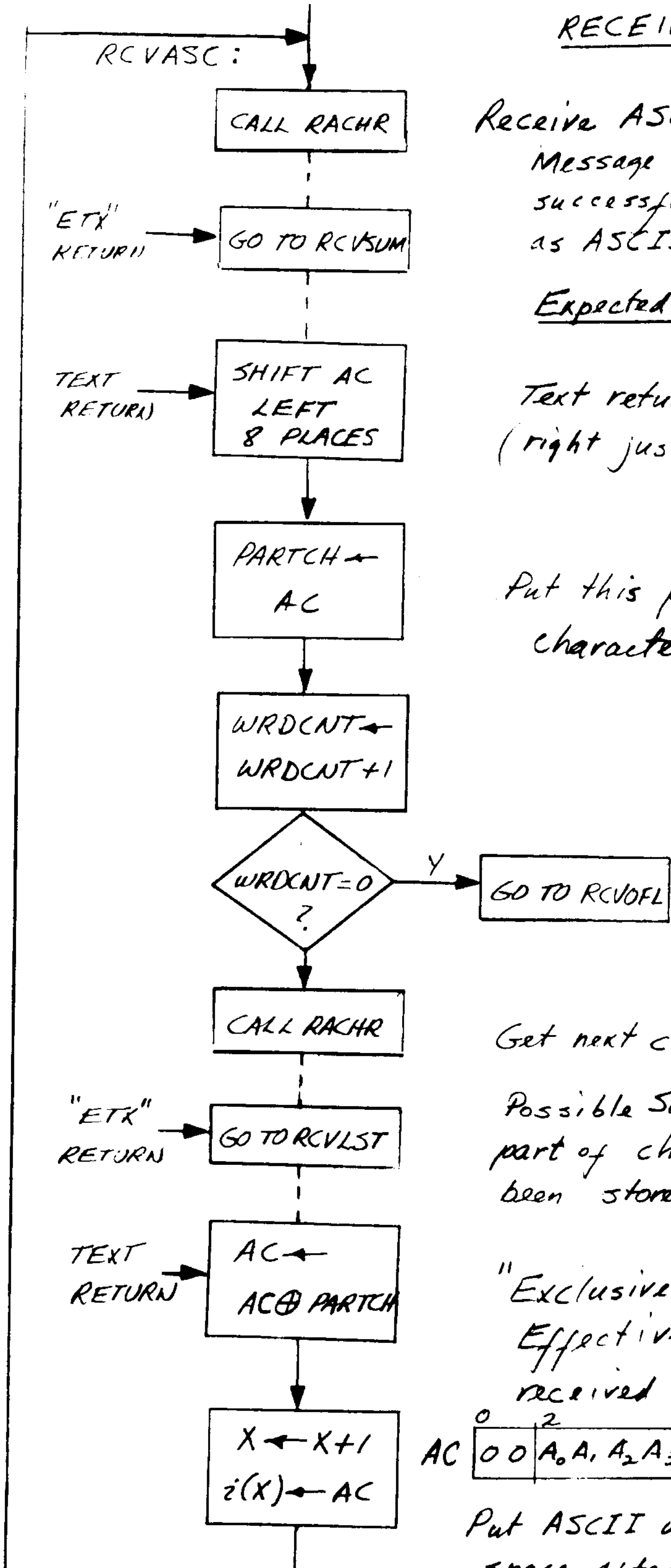
Duplication of last message number indicated.

RCVERR holds error reply if check sum is O.K. Will send acknowledge in this case.

Will process and checksum message since it is already on the way, but the text will not be stored since the message number indicates the message has been received previously.



RECEIVE MESSAGE (7)



Receive ASCII character. The Message Header has been processed successfully. Text will be processed as ASCII. Checksum maintained.

<u>Expected Success Route</u> when "ETX" is found.

Text return has 8-bit ASCII character (right justified) in AC.

Put this part in PARTCH: part of a character.

No more room in user's buffer space.

Get next character.

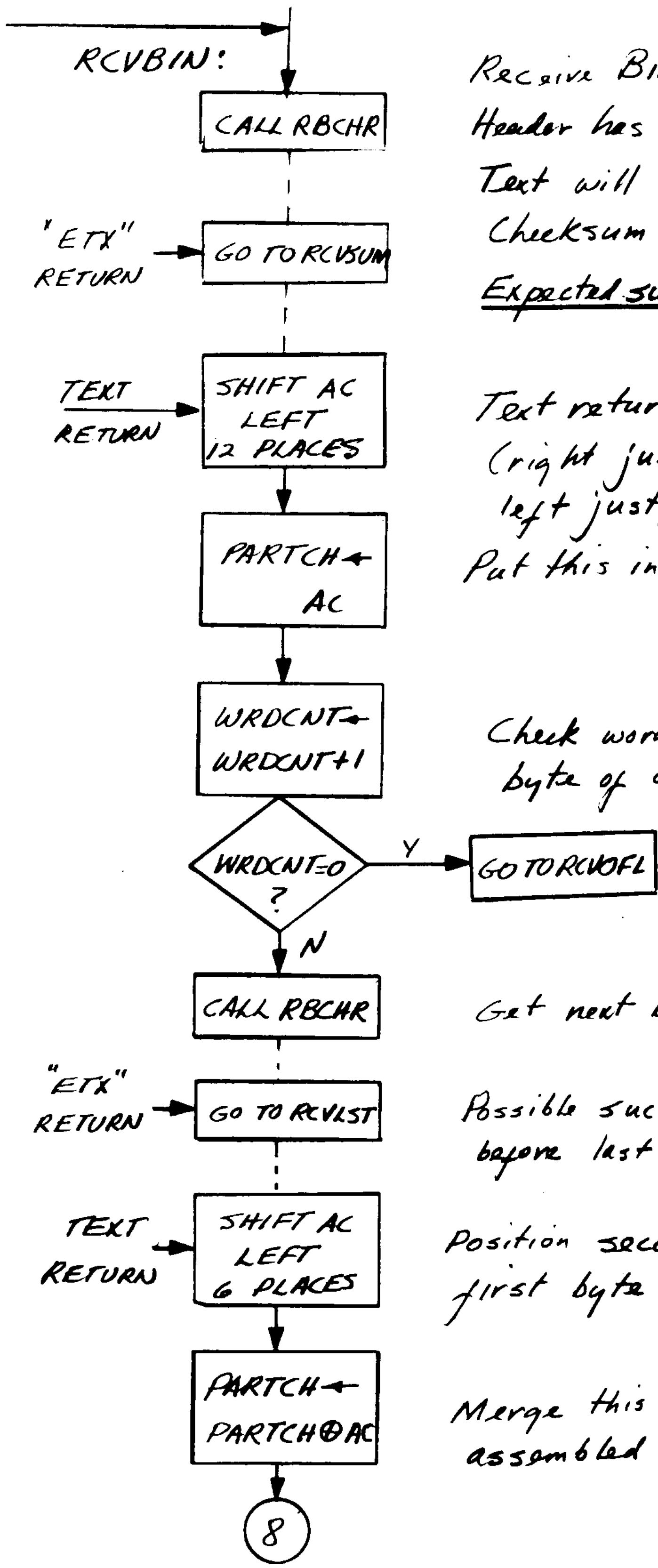
Possible Success Route, but part of character has not yet been stored in user's buffer.

"Exclusive Or" PARTCH with AC. Effectively places first character (A) received adjacent to second (B).

	0		2							9	10							17
AC	0	0	A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	B_0	B_1	B_2	B_3	B_4	B_5	B_6	B_7

Put ASCII word into user's buffer space after indexing buffer pointer.

RECEIVE MESSAGE (8)



Receive Binary character. The Message Header has been processed successfully. Text will be processed as Binary words. Checksum maintained.

<u>Expected success route</u> when "ETX" is found.

Text return has 6-bit binary character (right justified) in AC. Want to left justify first byte.

Put this in PARTCH: part of a character.

Check word count when first byte of a new word is encountered.

Buffer overflow exit.

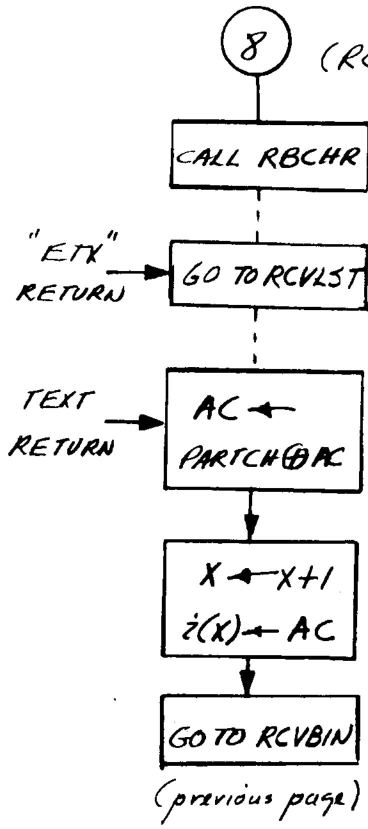
Get next binary character.

Possible success route. ETX found before last binary word completed.

Position second byte adjacent to first byte position before merging.

Merge this with current part of assembled binary word.

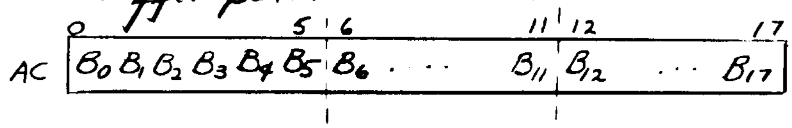
RECEIVE MESSAGE (9)

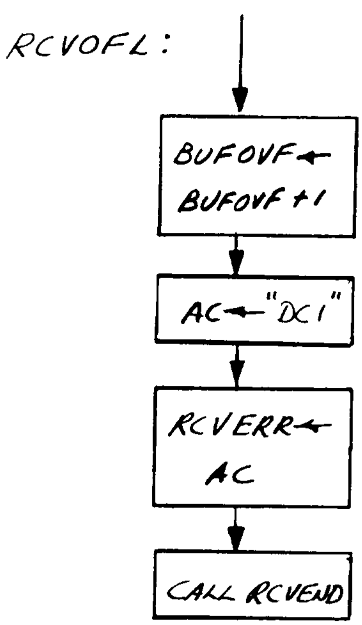


Possible success route. ETX found before last binary word completed.

Merge contents of PARTCH with last 6 bits of the binary word. Format of binary word below.

Put the assembled binary word into user's buffer space after indexing buffer pointer.





Buffer Overflow routine.

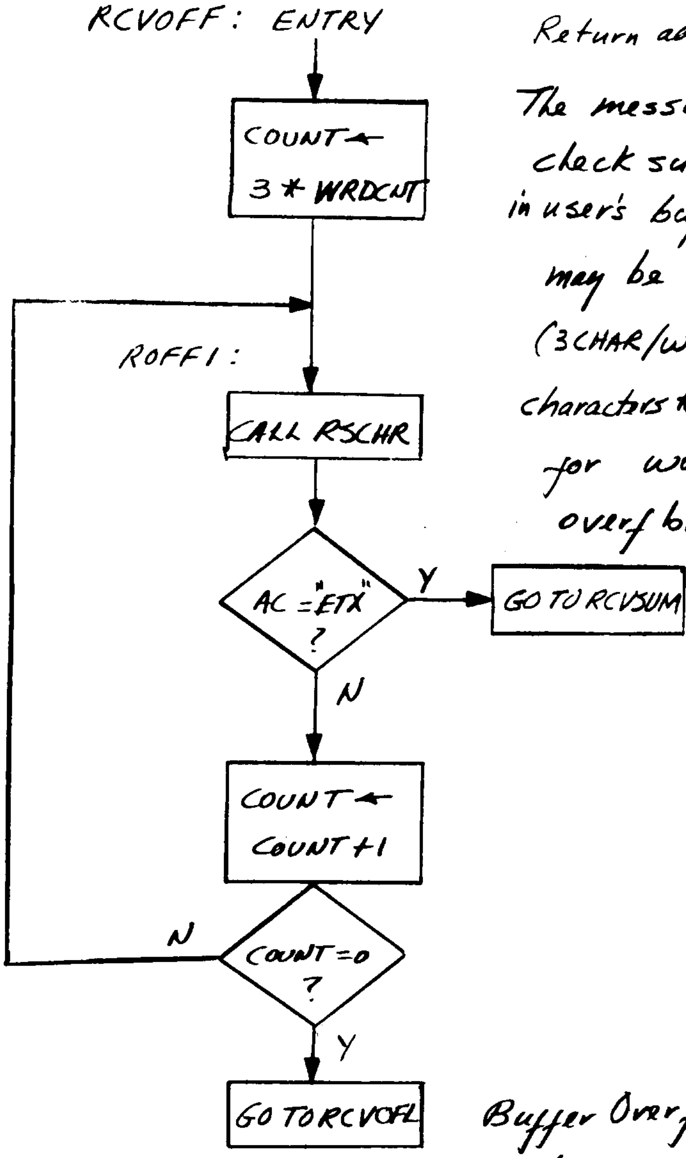
BUFOVF contains number of Buffer Overflow events since initialization.

"DC1" is reserved ASCII character 221_8 = BUFFER OVERFLOW response.

RCVERR contains the current response to receive error.

Leads to error return.

RECEIVE MESSAGE (10)



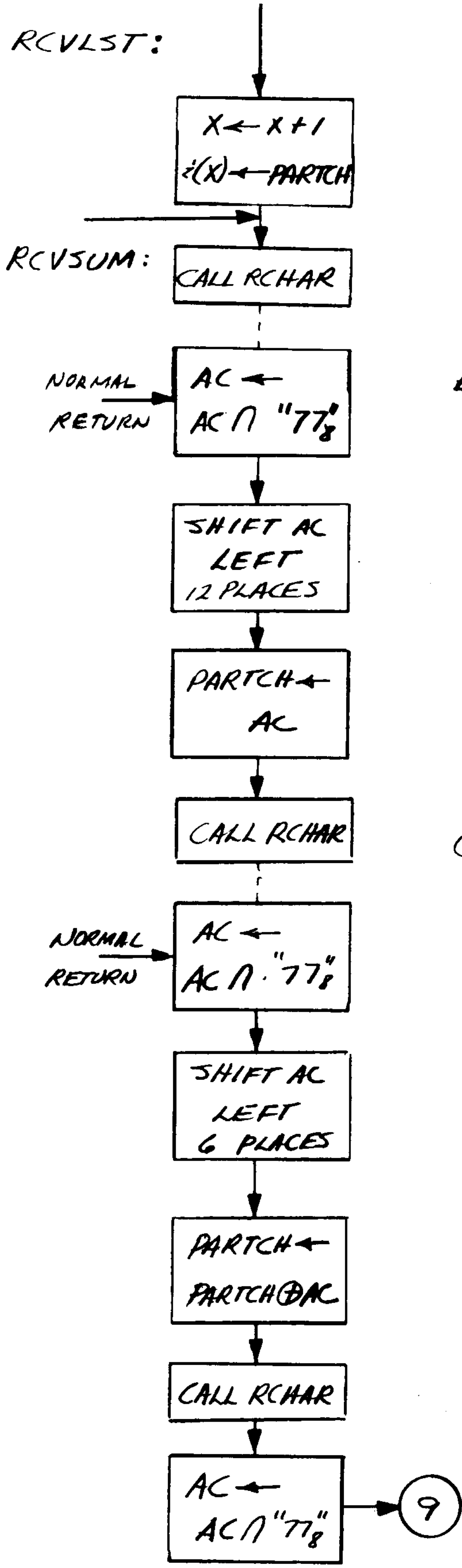
Return address of invoking instruction here.

The message being sent is to be check sumed, but not formatted and placed in user's buffer space. Since this message may be ASCII (2CHAR/WORD) or Binary (3CHAR/WORD), allow 3 times as many characters to be flushed as WRDCNT allows for word space before indicating overflow.

Success Route for flushing this message. Now get checksum.

Buffer Overflow exit even though no buffer space was being used. By convention, no more characters are flushed than worst case word count.

RECEIVE MESSAGE (11)



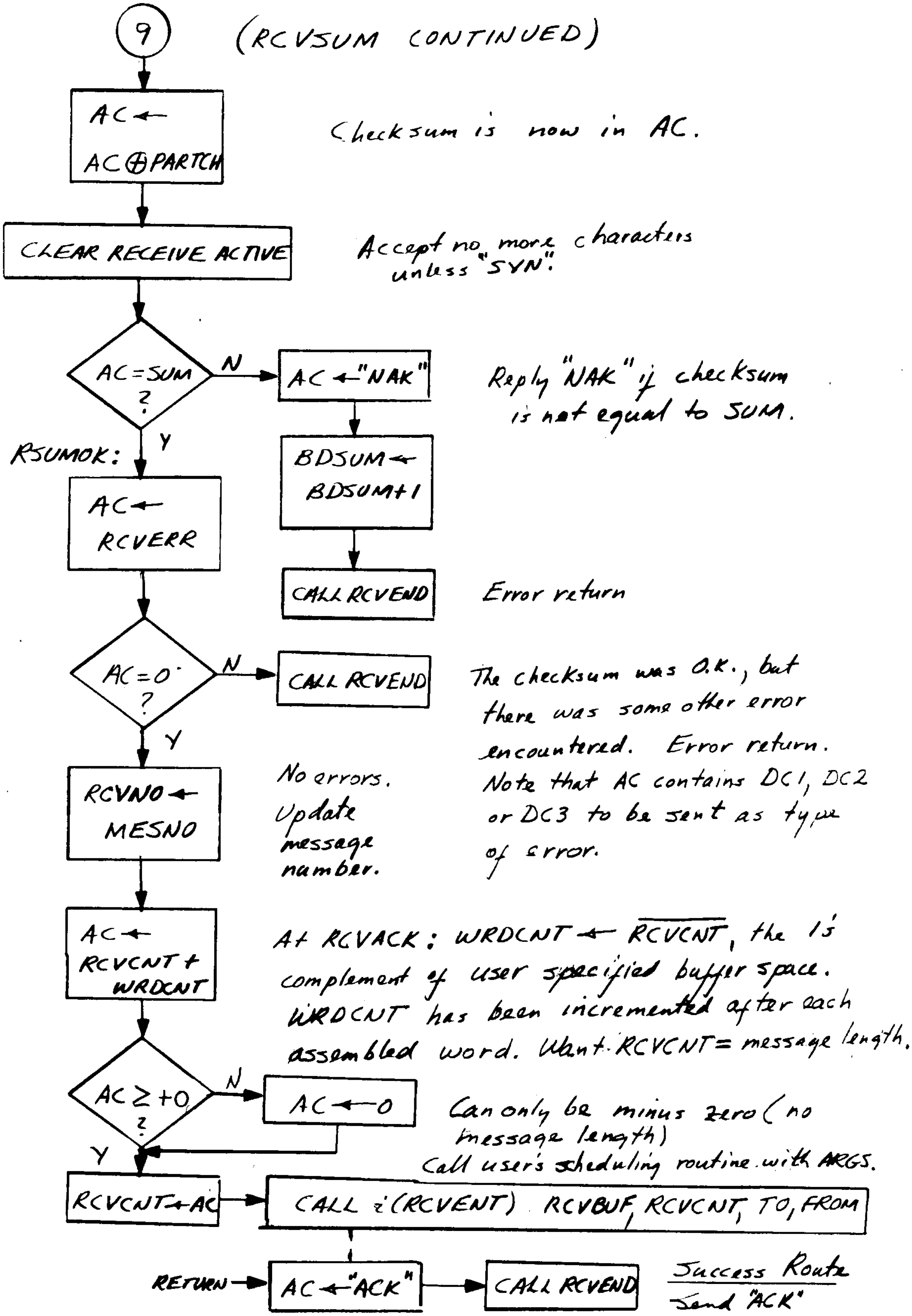
If entry is RCVLST, put the portion of word already assembled into user's buffer area.
<u>Expected entry is RCVSUM.</u>
Get 1st byte of checksum.

Expecting checksum as binary word. Screening out all but rightmost 6 bits.

Assemble binary word for checksum. The checksum characters are <u>not</u> included in checksum.

Get 2nd byte of checksum.

RECEIVE MESSAGE (12)



RECEIVE MESSAGE (13)

RCVEND: ENTRY — Return address of invoking instruction here.

CALL XCHAR — Transmit status character in AC.

NORMAL RETURN → CALL RCVDIE — Go through termination protocol.

RCVDIE: ENTRY — Return address of invoking instruction here.

AC ← "EOT"

CALL XCHAR — Transmit end of transmission.

*REXIT: →

SET IDLE MODE
CLEAR RECEIVE ACTIVE — Retransmit "EOT" on demand. Accept no more characters unless "SYN".

GO TO i(RCVMES) — Return to user's normal return.

*RCVTIM : ENTRY

TIMOUT ← ℓ(RCVEMG) — Change TIMOUT return to emergency exit.

RTMOUT ← RTMOUT+1 — Note TIMOUT error in Receive Mode

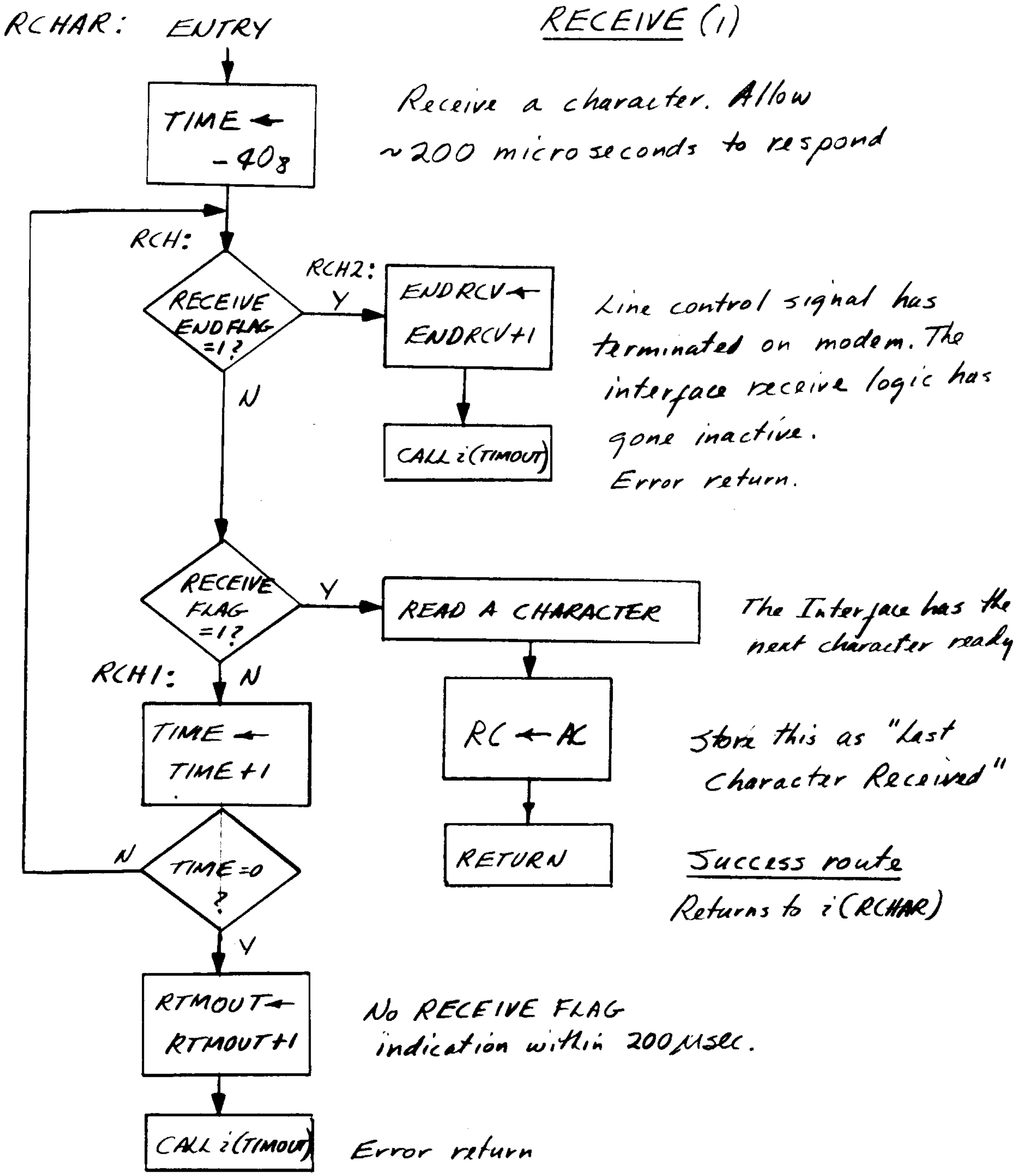
GO TO RCVDIE

*RCVEMG: ENTRY

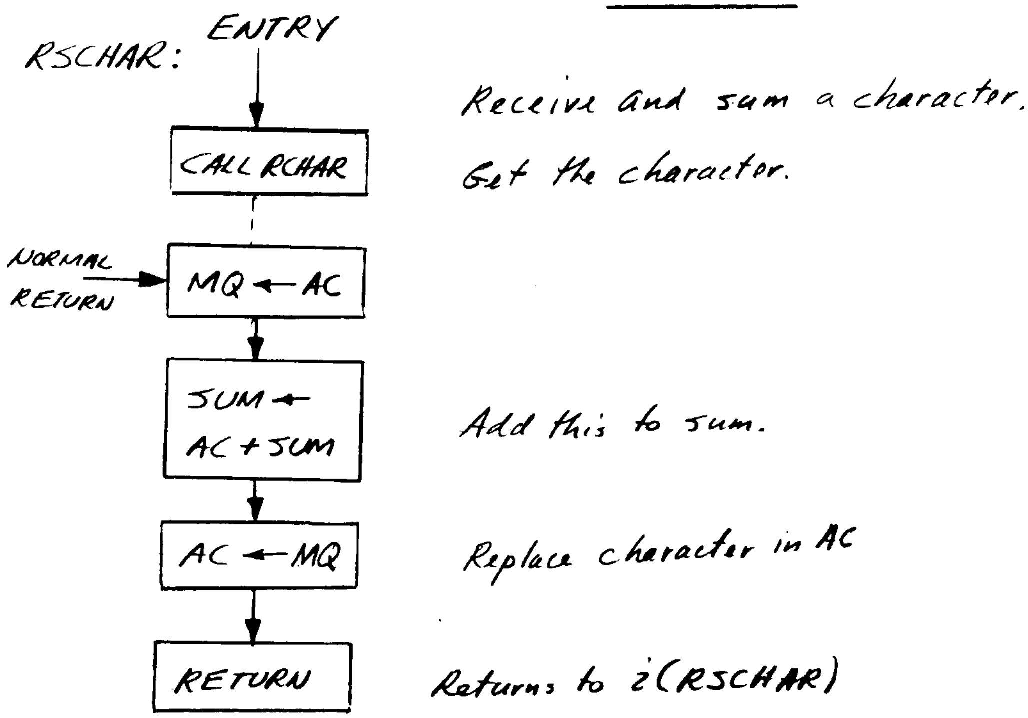
GO TO REXIT

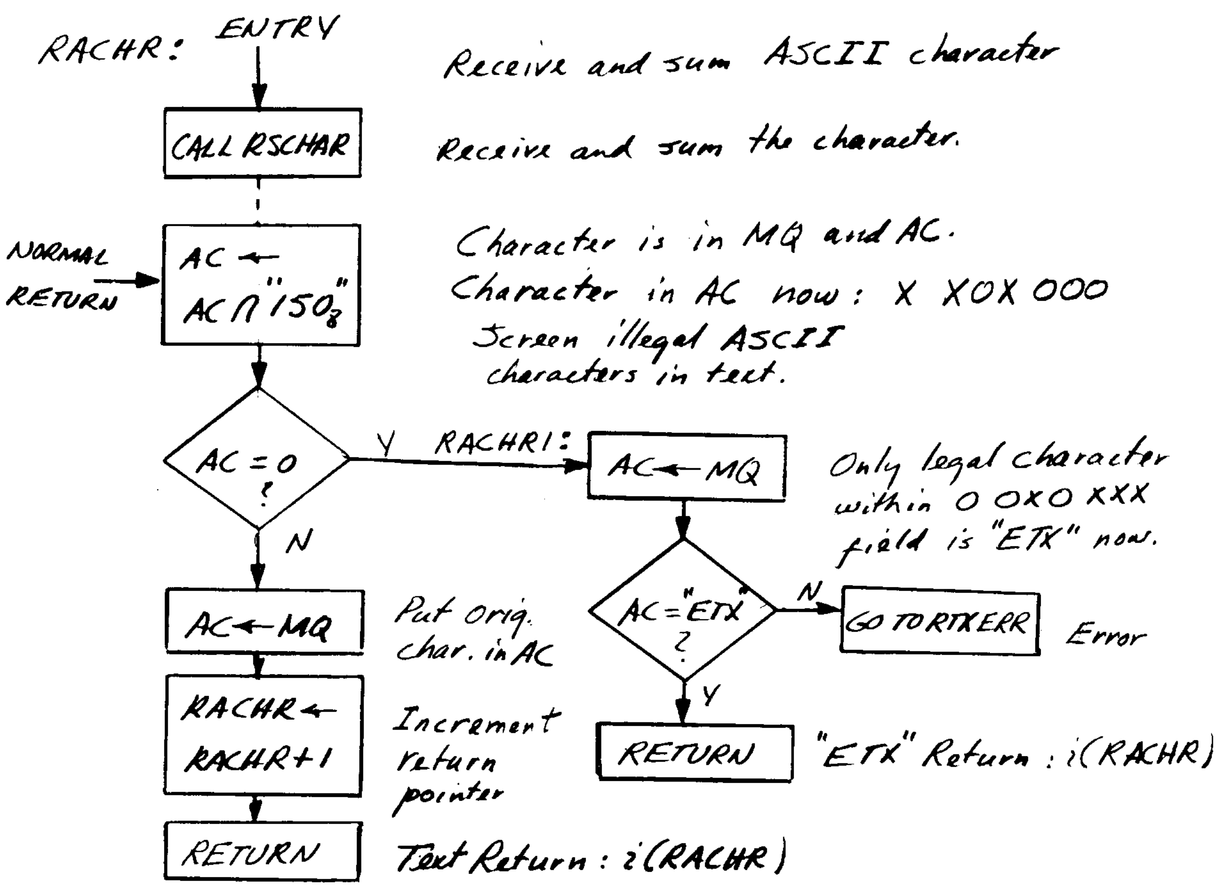
* See suggested program modifications

RECEIVE (1)

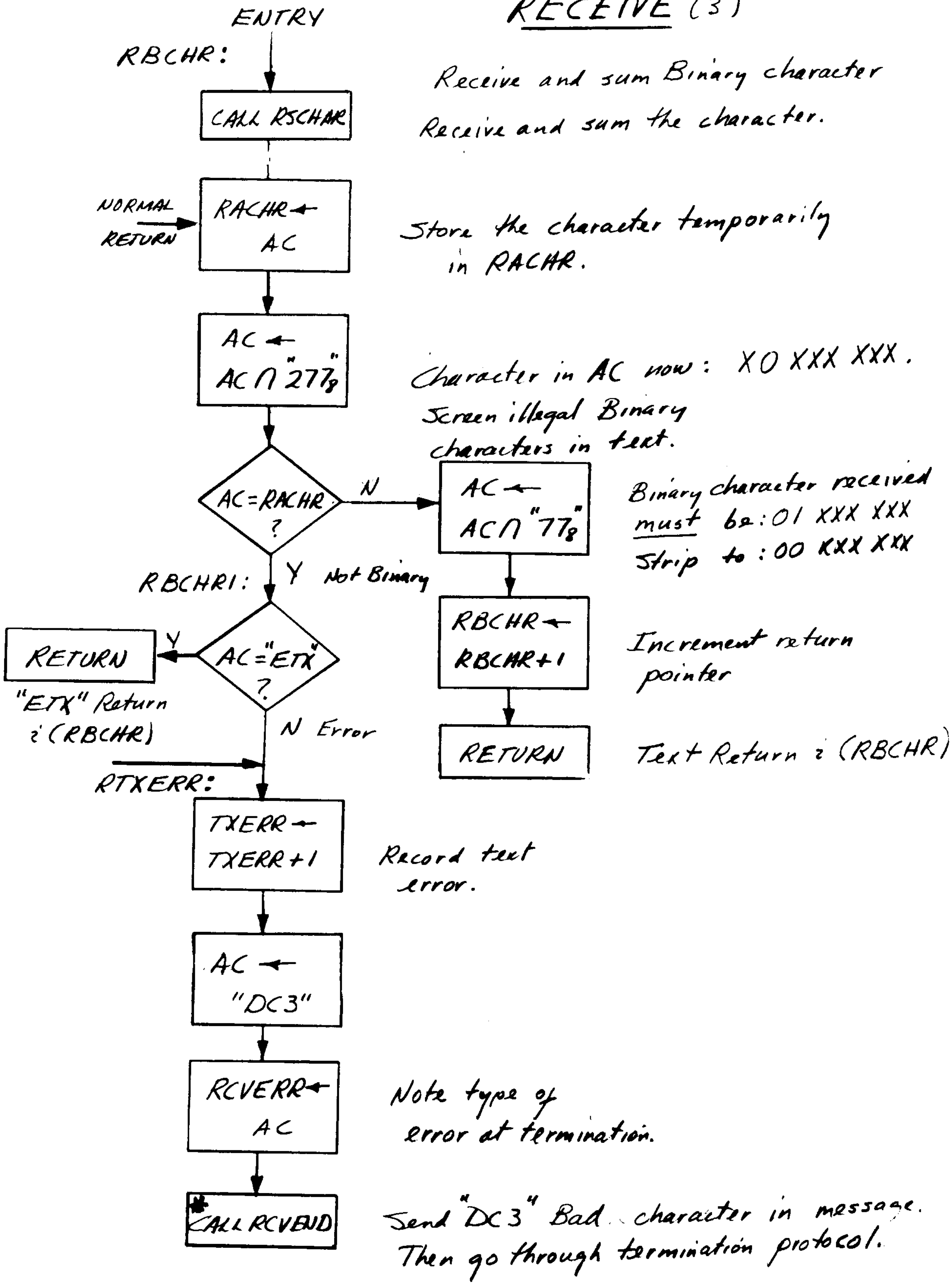

RCHAR: ENTRY
TIME ← -40_8
Receive a character. Allow ~200 microseconds to respond
RCH:
RECEIVE ENDFLAG =1?
Y
N
RCH2:
ENDRCV ← ENDRCV+1
CALL i(TIMOUT)
Line control signal has terminated on modem. The interface receive logic has gone inactive.
Error return.
RECEIVE FLAG =1?
Y
READ A CHARACTER
The Interface has the next character ready.
RCH1:
N
TIME ← TIME+1
RC ← AC
Store this as "Last Character Received"
TIME=0 ?
N
Y
RETURN
Success route
Returns to i(RCHAR)
RTMOUT ← RTMOUT+1
No RECEIVE FLAG indication within 200 µsec.
CALL i(TIMOUT)
Error return

RECEIVE (2)


RSCHAR:
ENTRY
Receive and sum a character.
CALL RCHAR
Get the character.
NORMAL RETURN
MQ ← AC
SUM ← AC + SUM
Add this to sum.
AC ← MQ
Replace character in AC
RETURN
Returns to i(RSCHAR)

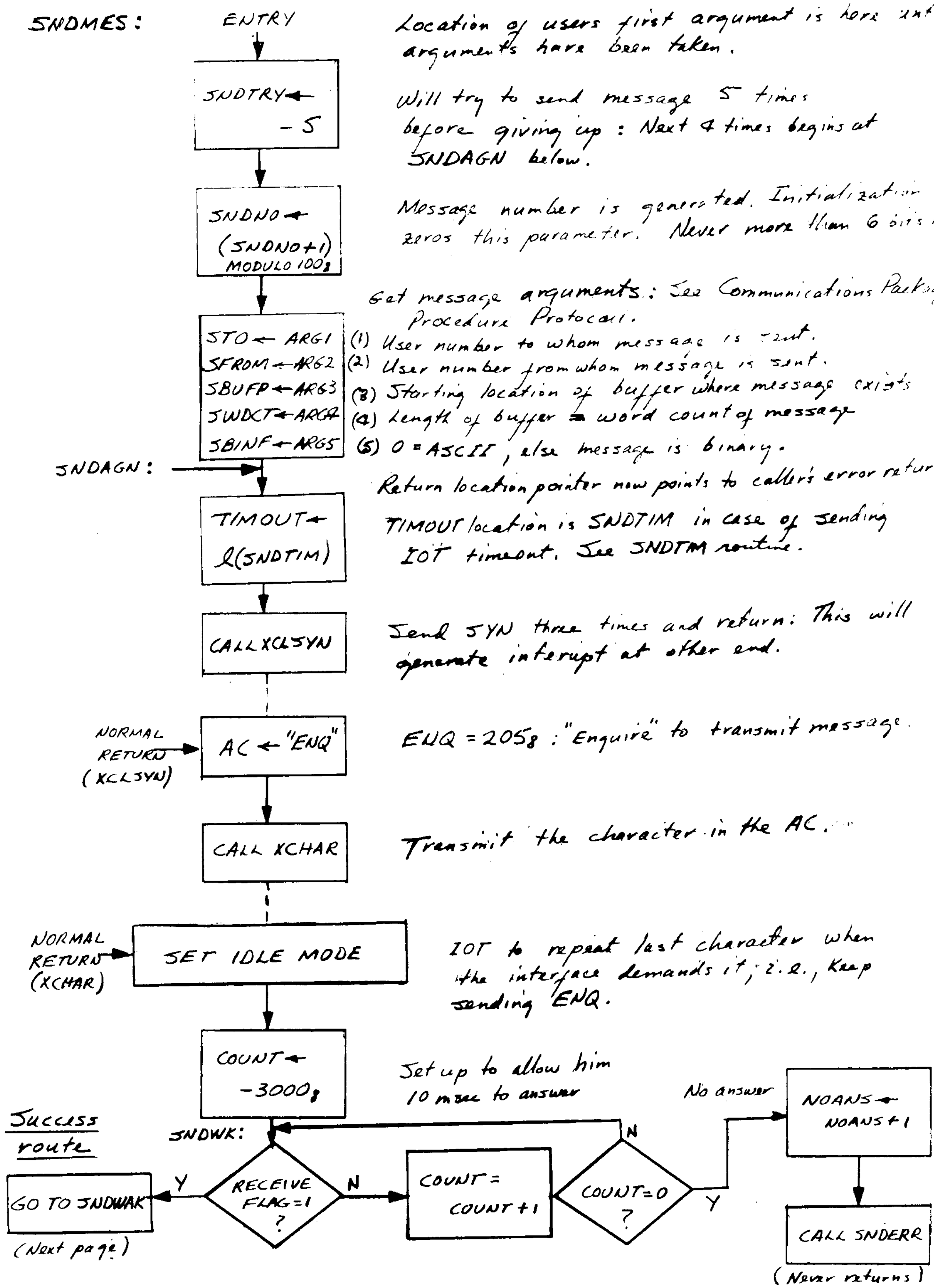

RACHR:
ENTRY
Receive and sum ASCII character
CALL RSCHAR
Receive and sum the character.
NORMAL RETURN
AC ← AC ∩ "150₈"
Character is in MQ and AC.
Character in AC now: X XOX 000
Screen illegal ASCII characters in text.
AC = 0 ?
Y
N
RACHR1:
AC ← MQ
Only legal character within 0 0X0 XXX field is "ETX" now.
AC = "ETX" ?
N
GO TO RTXERR
Error
Y
RETURN
"ETX" Return : i(RACHR)
AC ← MQ
Put orig. char. in AC
RACHR ← RACHR+1
Increment return pointer
RETURN
Text Return : i(RACHR)

RECEIVE (3)



* This instruction replaces CALL RCVDIE.

SEND MESSAGE (1)



Location of users first argument is here until arguments have been taken.

Will try to send message 5 times before giving up: Next 4 times begins at SNDAGN below.

Message number is generated. Initialization zeros this parameter. Never more than 6 bits long.

Get message arguments: See Communications Package Procedure Protocall.
(1) User number to whom message is sent.
(2) User number from whom message is sent.
(3) Starting location of buffer where message exists
(4) Length of buffer = word count of message
(5) 0 = ASCII, else message is binary.

Return location pointer now points to caller's error return.

TIMOUT location is SNDTIM in case of sending IOT timeout. See SNDTM routine.

Send SYN three times and return: This will generate interupt at other end.

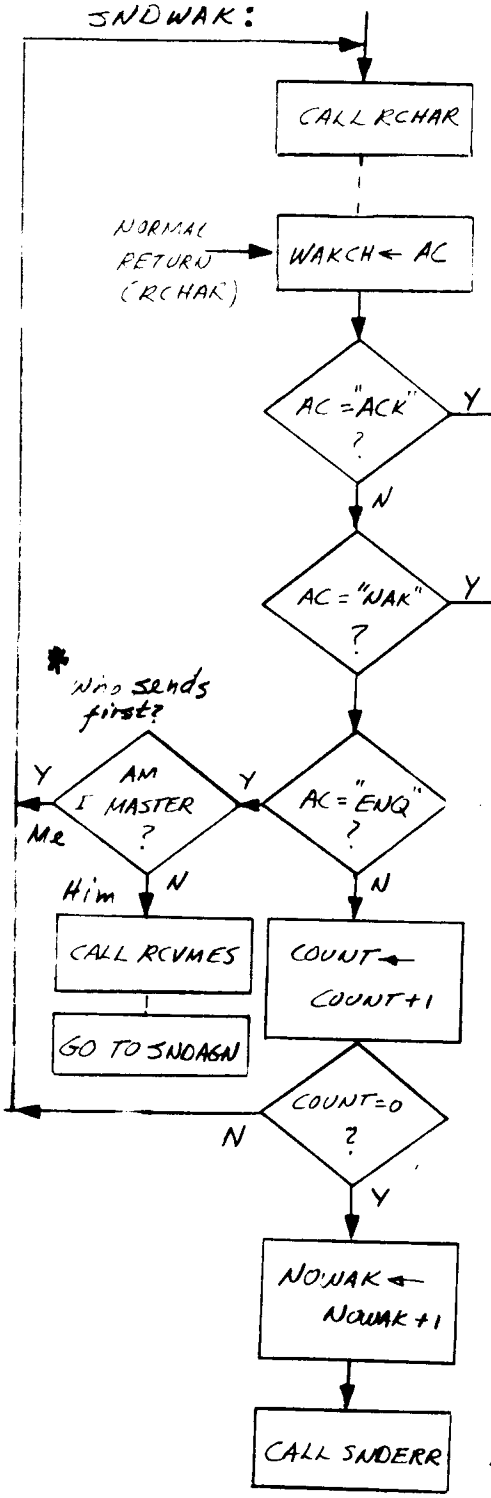
ENQ = 205₈ : "Enquire" to transmit message.

Transmit the character in the AC.

IOT to repeat last character when the interface demands it, i.e., keep sending ENQ.

Set up to allow him 10 msec to answer

SEND MESSAGE (2)



Get his answer from read character routine into AC, then see what it is.

Put it in "wake character" for the record.

Success Route
He has acknowledged to receive my transmission. Time to send it.

He has no buffer area available.

Error return.

He wants to send a message, too?

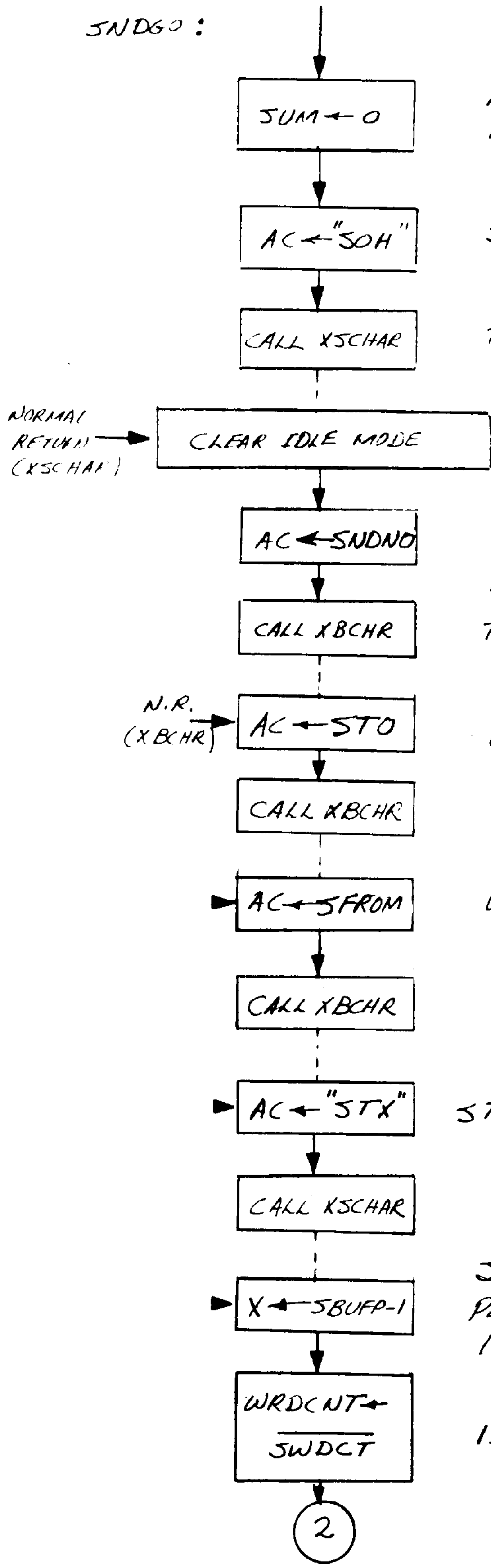
*** The program must be modified to establish the SLAVE status.**

His message none of these. Get next character if time permits.

Couldn't wake him up in time.

Never returns, but return location of calling instruction is stored there for reference.

SEND MESSAGE (3)



Now attempting to send message. Begin by clearing checksum.

Put on message header.
SOH = 201_8 : Start of header sent first

Transmit and sum character in AC.

Want to send a new character every interval now. Would rather modem loose sync than duplicate previous character if I can't supply the next one on time.

Message number follows "SOH".

Transmit word in AC as 18-bit binary number (send as 3 8-Bit characters – see XBCHR) and add to checksum.

User number who is to receive it.

Send as binary.

User number who is sending it.

Send as binary.

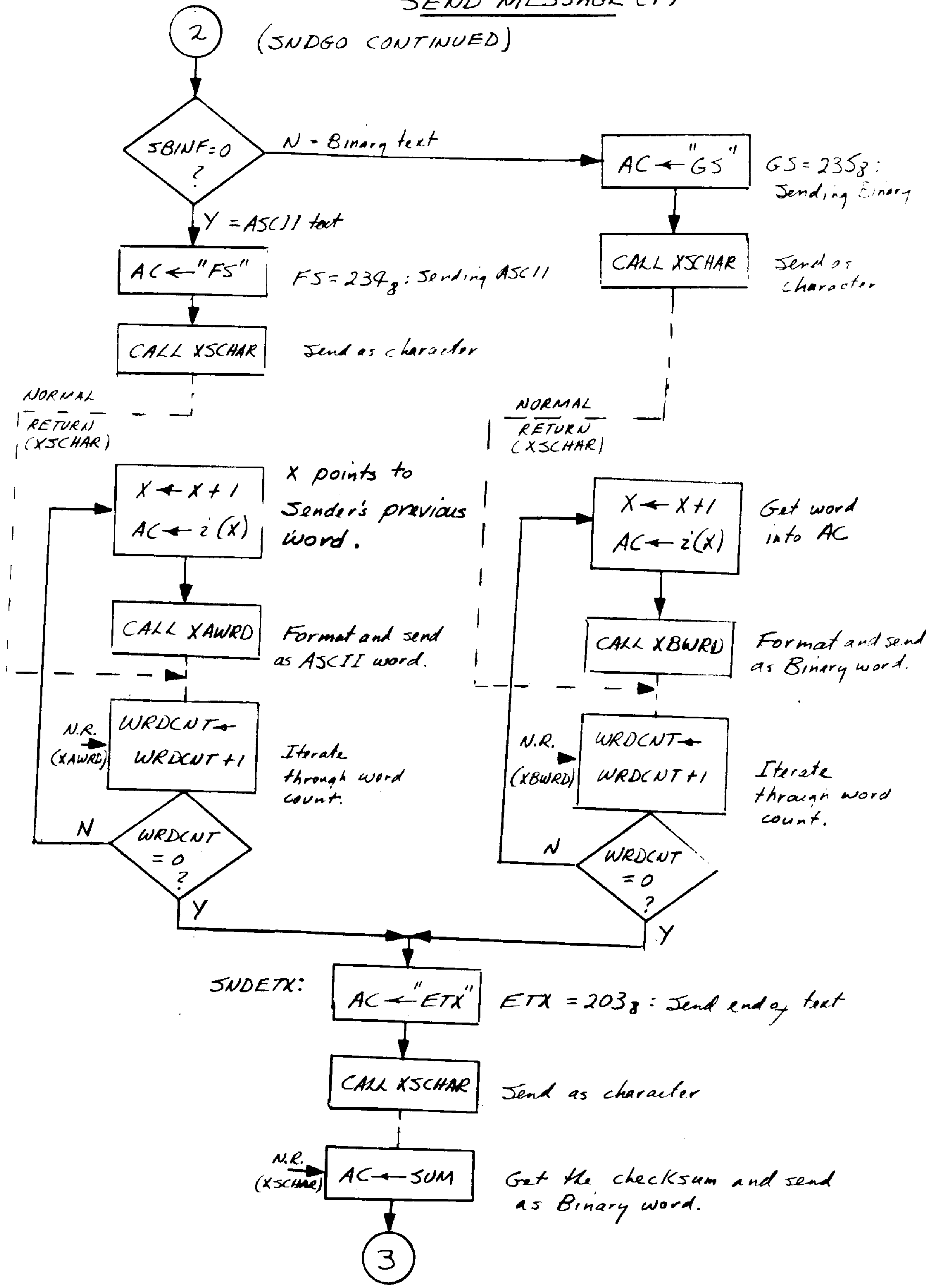
STX = 202_8 : Start of text

Send as character.

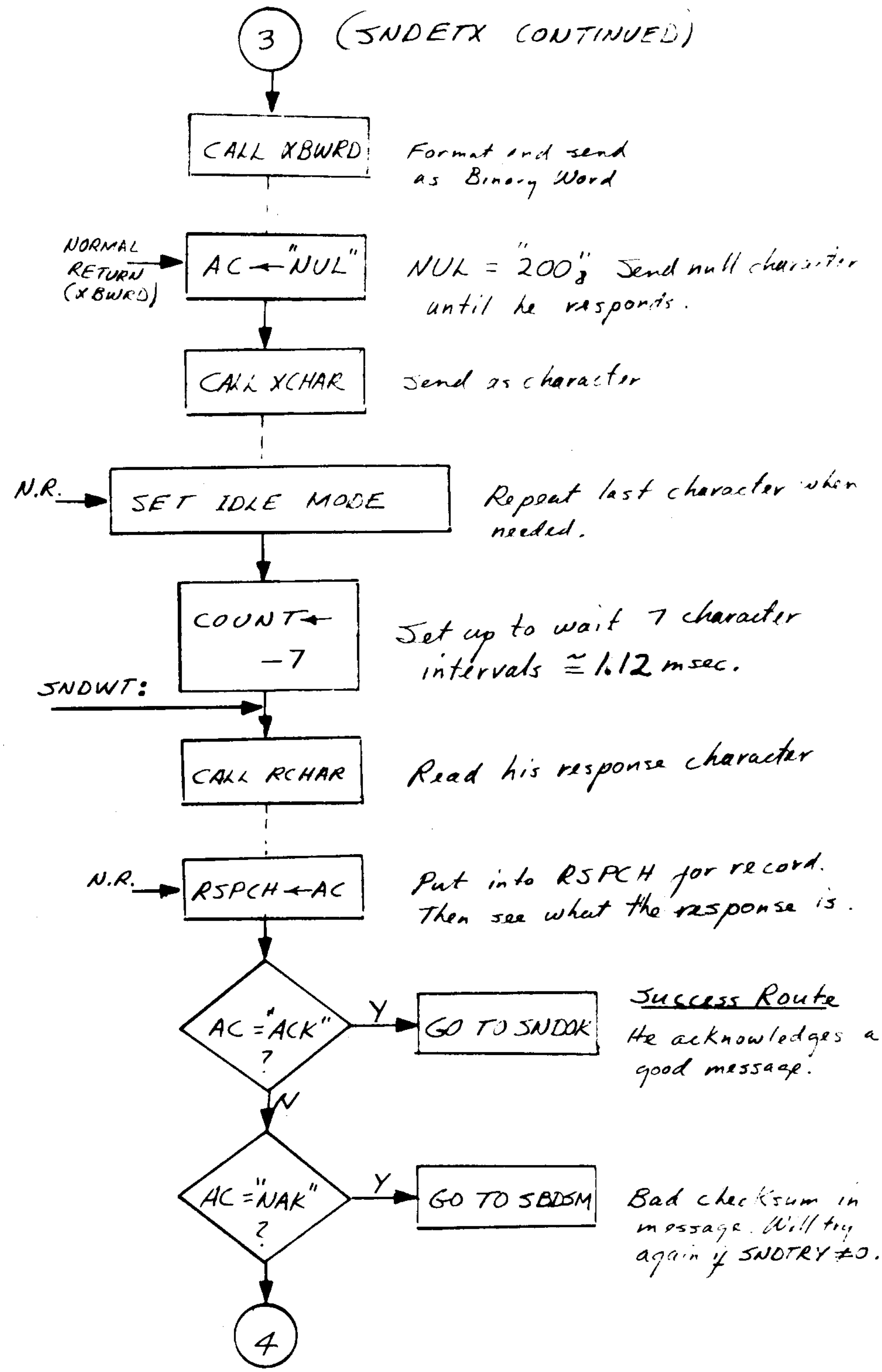
Set up for sending text.
Put buffer pointer (less 1) into auto-indexing location. X.

1's complement of SWDCT (length of buffer) into WRDCNT.

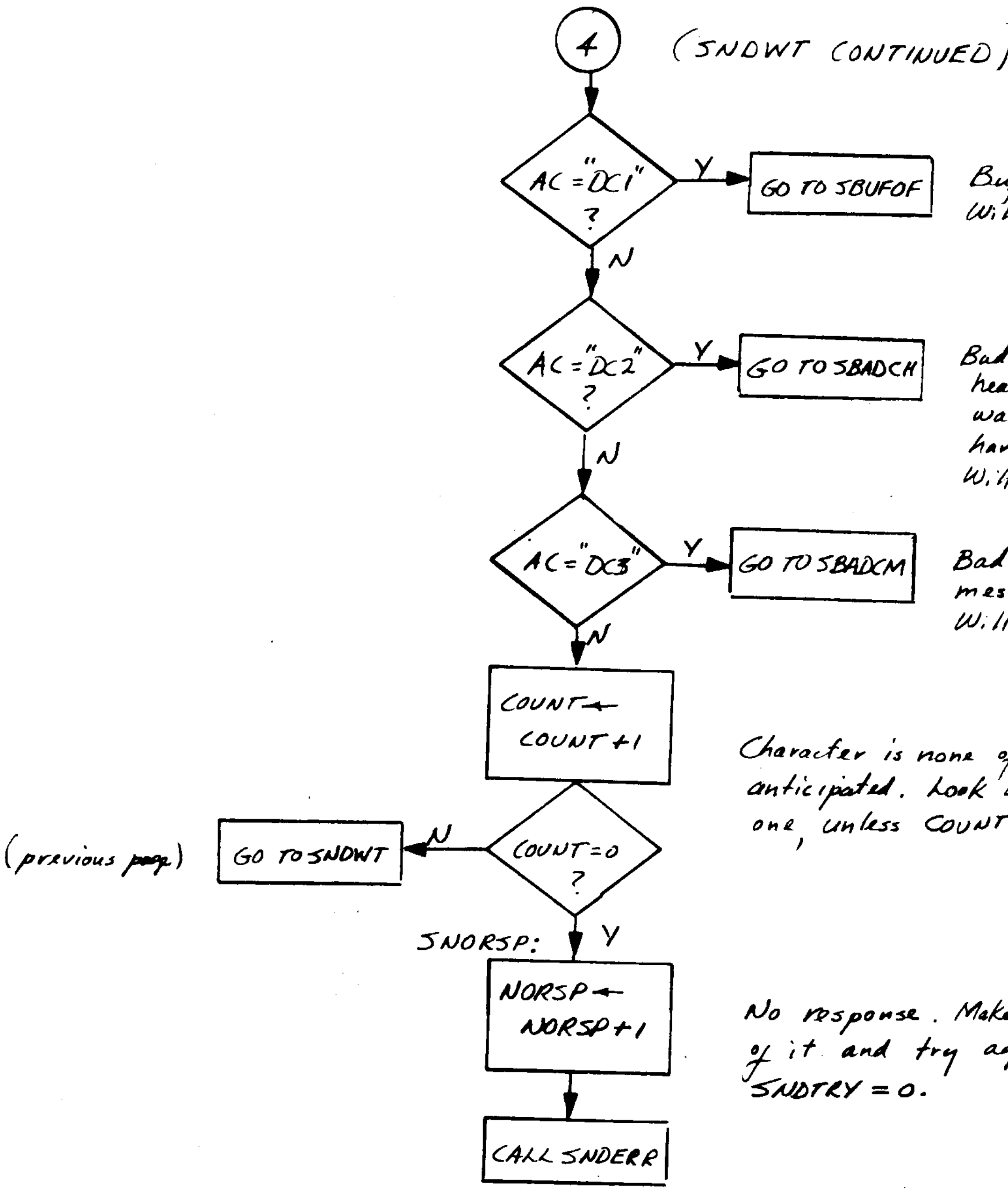
SEND MESSAGE (4)



SEND MESSAGE (5)



SEND MESSAGE (c)



Buffer overflow. Will not try again.

Bad character in header. Since checksum was O.K., user must have made the error. Will not try again

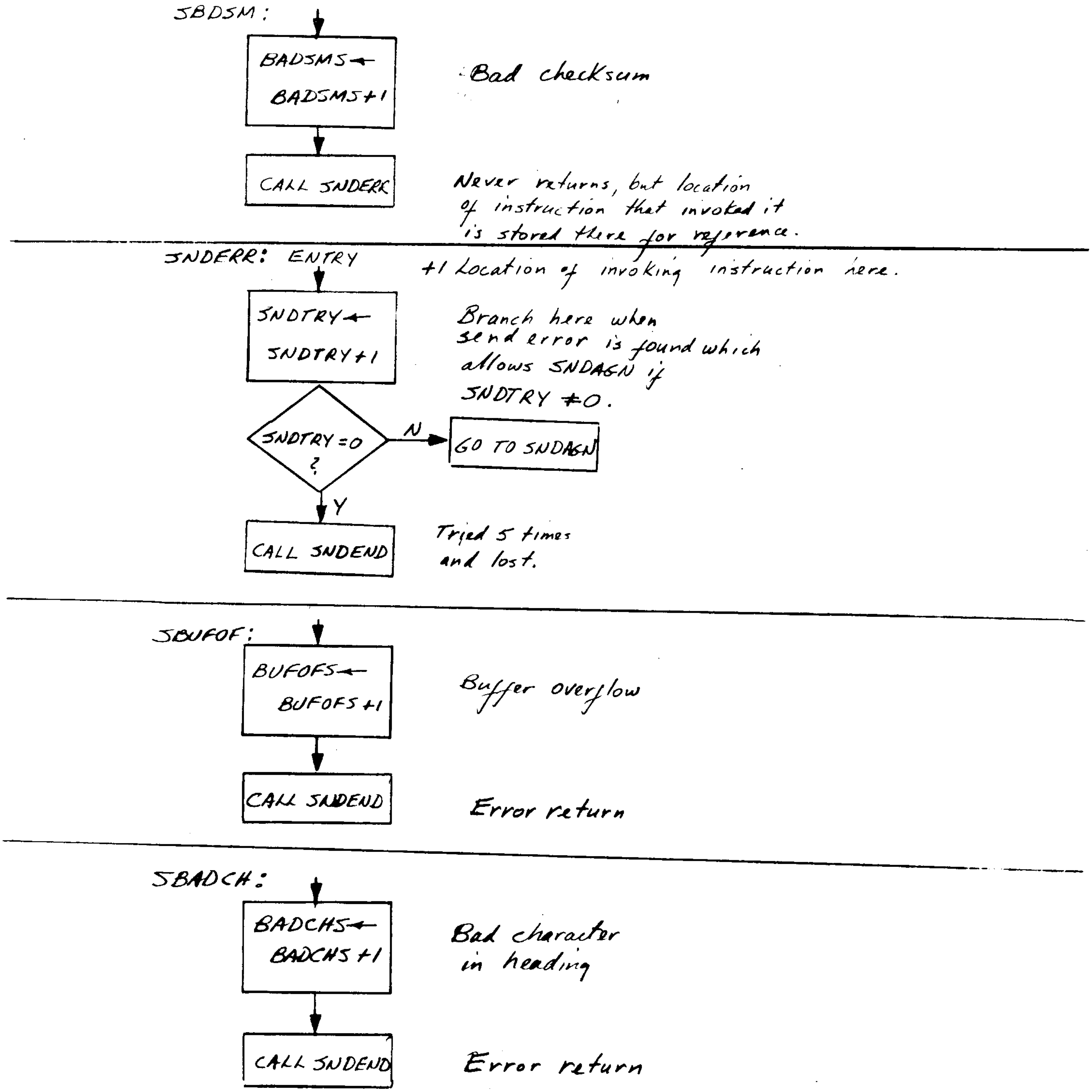
Bad character in message. User error. Will not try again.

Character is none of those anticipated. Look at next one, unless COUNT = 0.

No response. Make a note of it and try again, unless SNDTRY = 0.

SEND MESSAGE (7)

Action Routines Used By:



Action Routines Used By: SEND MESSAGE (8)

SBADCM:

BADCMS ← BADCMS+1 — Bad character in message.

CALL SNDEND — Error return

---

* SNDTIM: ENTRY — +1 Location of invoking instruction here.

* TIMOUT ← 2(SNDEMG) — Change TIMOUT return to emergency exit.

STMOUT ← STMOUT+1 — Time Ran Out in transmission attempt.

* SET IDLE MODE

CALL SNDEND — Error return

*SNDEMG: ENTRY

GO TO SEXIT

* Suggested modifications

SNDOK:

SNDMES ← SNDMES+1

SENTOK ← SENTOK+1

CALL SNDEND

Success route

Return location is at SNDMES. Pointer has been at users error return location. Since success route has been found in send message routine, increment this location by one so as to point at users normal return location.

Normal return

SNDEND: ENTRY — +1 Location of invoking instruction here.

AC ← "EOT" — Send end of transmission

CALL KCHAR — Send as character

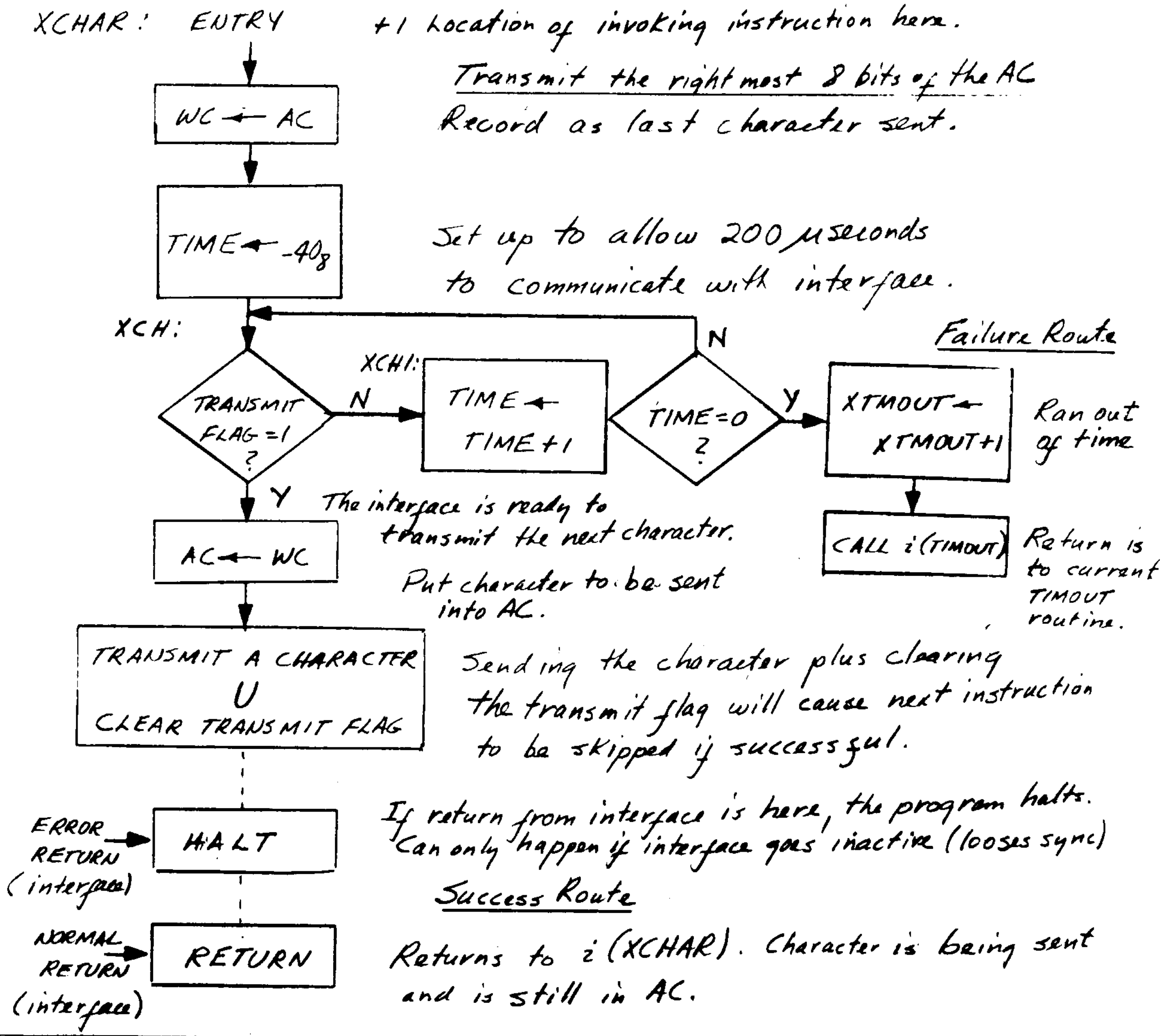
*SEXIT:

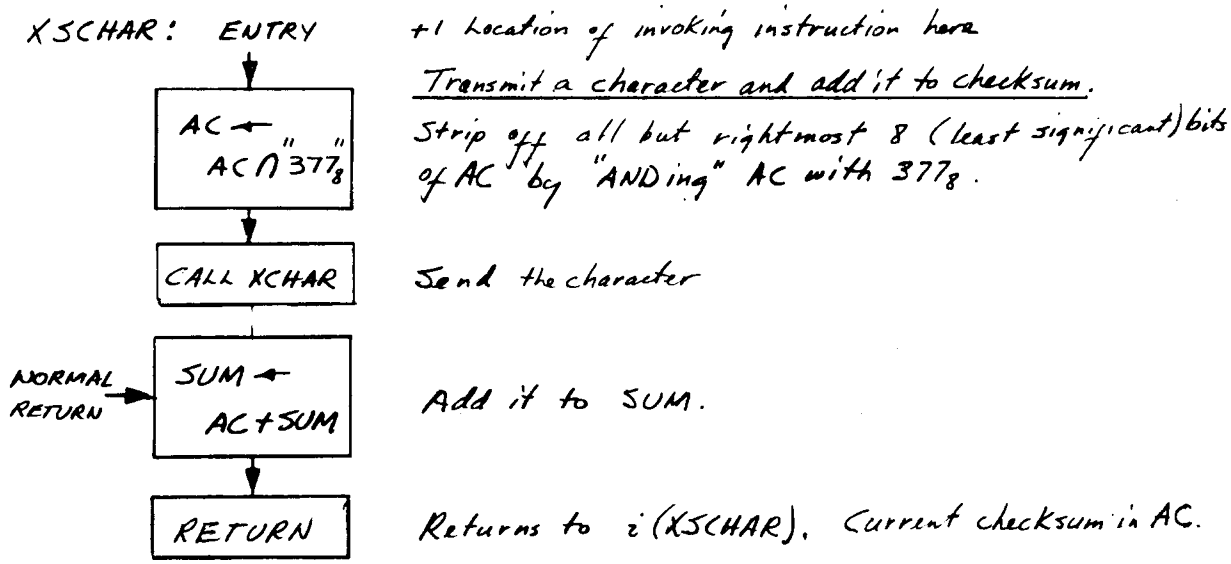
Normal Return (KCHAR)

CLEAR RECEIVE ACTIVE — Accept no more characters unless = SYN.

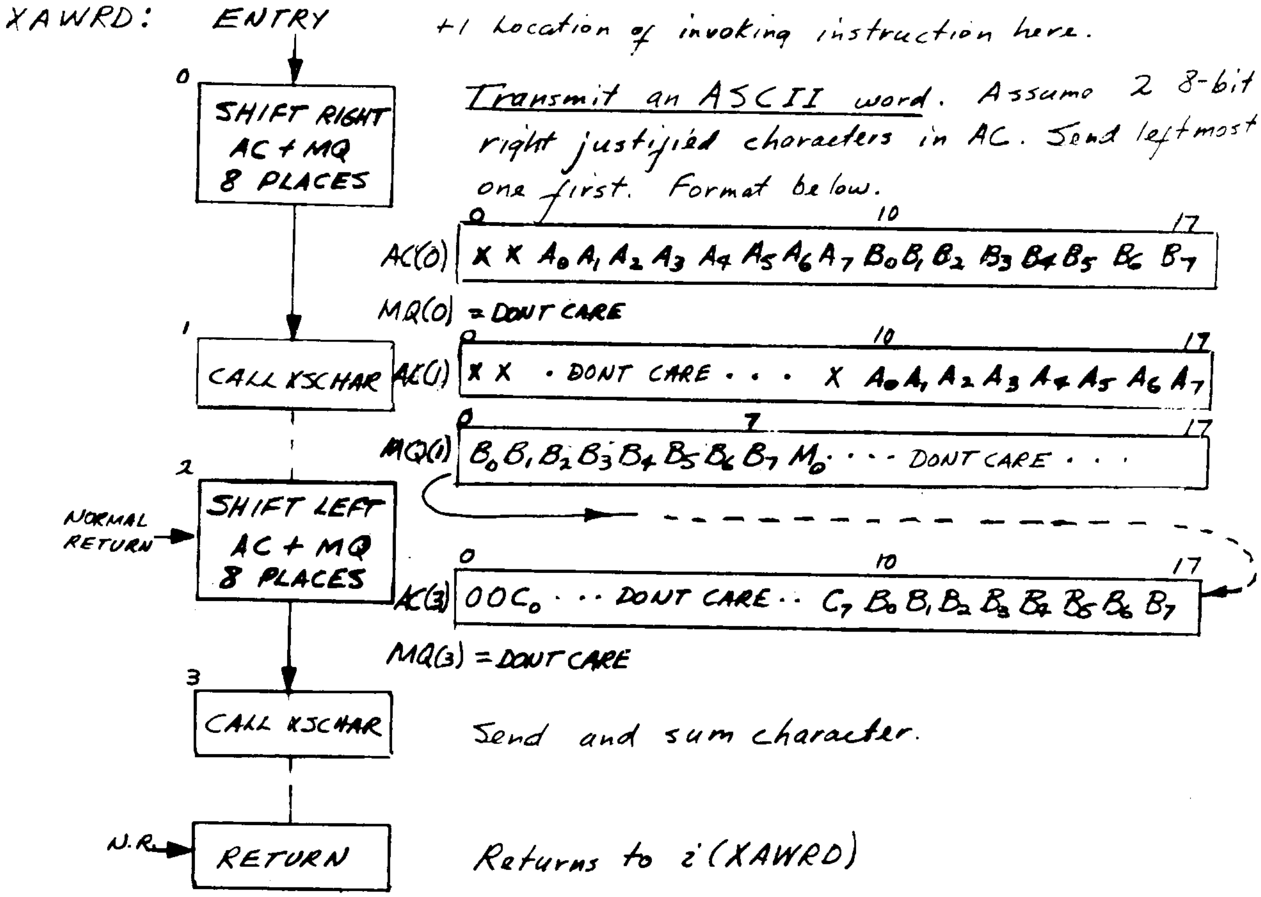
RETURN — Return to 2(SNDMES); either points to users error return or normal return.

TRANSMIT (1)


XCHAR: ENTRY
+1 Location of invoking instruction here.
Transmit the rightmost 8 bits of the AC
WC ← AC
Record as last character sent.
TIME ← -40₈
Set up to allow 200 μseconds to communicate with interface.
XCH:
TRANSMIT FLAG = 1 ?
N
XCH1:
TIME ← TIME+1
TIME=0 ?
N
Y
Failure Route
XTMOUT ← XTMOUT+1
Ran out of time
CALL i(TIMOUT)
Return is to current TIMOUT routine.
Y
The interface is ready to transmit the next character.
AC ← WC
Put character to be sent into AC.
TRANSMIT A CHARACTER U CLEAR TRANSMIT FLAG
Sending the character plus clearing the transmit flag will cause next instruction to be skipped if successful.
ERROR RETURN (interface)
HALT
If return from interface is here, the program halts. Can only happen if interface goes inactive (looses sync)
Success Route
NORMAL RETURN (interface)
RETURN
Returns to i(XCHAR). Character is being sent and is still in AC.


XSCHAR: ENTRY
+1 Location of invoking instruction here
Transmit a character and add it to checksum.
AC ← AC ∩ 377₈
Strip off all but rightmost 8 (least significant) bits of AC by "ANDing" AC with 377₈.
CALL XCHAR
Send the character
NORMAL RETURN
SUM ← AC+SUM
Add it to SUM.
RETURN
Returns to i(XSCHAR). Current checksum in AC.

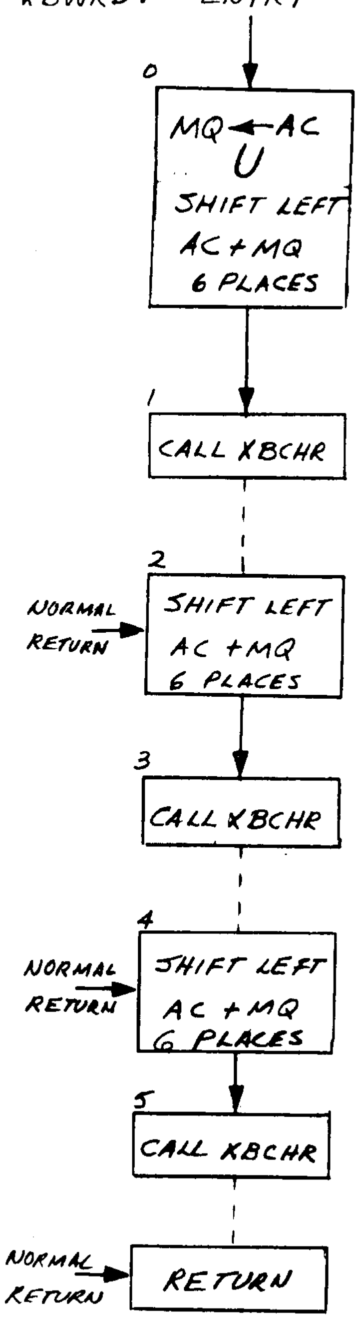
TRANSMIT (2)


XAWRD:
ENTRY
+1 location of invoking instruction here.
0
SHIFT RIGHT AC + MQ 8 PLACES
Transmit an ASCII word. Assume 2 8-bit right justified characters in AC. Send leftmost one first. Format below.
AC(0) X X $A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7 B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7$
MQ(0) = DONT CARE
1
CALL XSCHAR
AC(1) X X · DONT CARE · · · X $A_0 A_1 A_2 A_3 A_4 A_5 A_6 A_7$
MQ(1) $B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7 M_0$ · · · DONT CARE · · ·
2
NORMAL RETURN
SHIFT LEFT AC + MQ 8 PLACES
AC(3) 0 0 C_0 · · · DONT CARE · · $C_7 B_0 B_1 B_2 B_3 B_4 B_5 B_6 B_7$
MQ(3) = DONT CARE
3
CALL XSCHAR
Send and sum character.
N.R.
RETURN
Returns to i(XAWRD)

TRANSMIT (3)

XBWRD: ENTRY

+1 Location of invoking instruction here



Transmit a Binary word. Send as 3 8-Bit characters using 6 Bits of AC in each character. Send leftmost 6 bits of AC first. Format below.

AC(0): bits 0–17: $B_0\ B_1 \cdots \cdots B_{16}\ B_{17}$

MQ(0) = DONT CARE

AC(1): bits 0–11: X ⋯ DONT CARE ⋯ X X; bits 12–17: $B_0\ B_1\ B_2\ B_3\ B_4\ B_5$

MQ(1): bits 0–11: $B_6\ B_7 \cdots \cdots B_{16}\ B_{17}$; bits 12–17: X ⋯ ⋯ X

Sending $B_0 - B_5$

AC(3): bits 0–11: X ⋯ DON'T CARE ⋯ X X X; bits 12–17: $B_6\ B_7\ B_8\ B_9\ B_{10}\ B_{11}$ — All that is used.

Format as Binary Character and send.

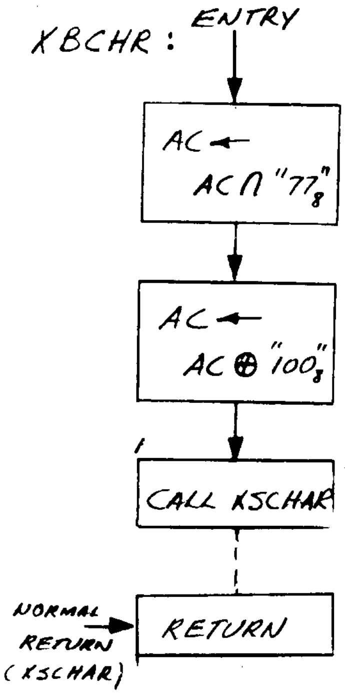
Sending $B_6 - B_{11}$

AC(5): bits 0–11: X ⋯ DON'T CARE ⋯ X; bits 12–17: $B_{12}\ B_{13}\ B_{14}\ B_{15}\ B_{16}\ B_{17}$

Sending $B_{12} - B_{17}$

Returns to i (XBWRD).

TRANSMIT (4)

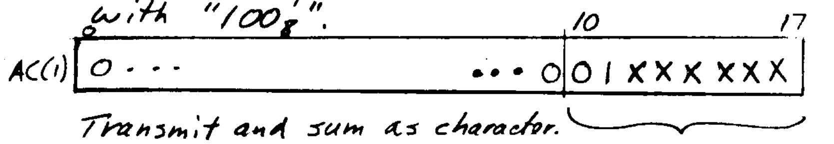


+1 Location of invoking instruction here

Transmit a Binary character

Strip off all but rightmost 6 (least significant) bits of AC by "ANDing" AC with "77_8"

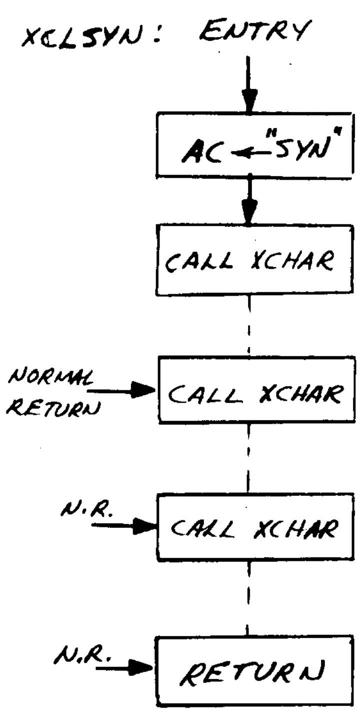
Map this into ASCII non-control type characters by "EXCLUSIVE OR ing" AC with "100_8".



Transmit and sum as charactor.

Binary Character

Returns to i (XBCHR)



+1 Location of invoking instruction here

Clear line and sync

SYN = 226_8 : Convention to send "SYN" 3 times before message transmission.

Send as character.

"SYN" is still in AC.

Returns to i (XCLSYN)

APPENDIX II

COMMUNICATIONS

COMMUNICATIONS PACKAGE ERROR PARAMETERS

Receive Errors in Message

Each time an error occurs within the Receive Message Procedure, the source of error is indicated by adding "1" to one of the following parameters.

BADCH Bad character in header; produced in header processing, if an "ETX" was found during, or "STX" was not found after, the header was processed. Note that other types of header errors are possible.

BADCM Bad character in message; not used (see TXERR).

BADINT Bad character after"SYN"; produced when a character other than "ENQ" followed the "SYN" sequence at beginning of received message.

BADSUM Bad checksum; produced when checksum maintained by Receive Message Procedure did not match checksum received after "ETX" of message.

BUFOVF Buffer overflow; produced when message text received exceeded size of buffer space allocated. The part of the message that did not fit was lost.

DUPMS Duplication of last message; produced when message number contained in header was same as last message number. Message numbering is maintained by the Communications Package.

DUPOMS Duplication of old message; produced when message number contained in header was smaller than last message number.

INTCH The last character received before an error return due to NOENQ or BADINT. Otherwise, INTCH contains the "ENQ" character.

NOBUF No receive buffer; produced when user did not provide RCVSET with a buffer, after response to "ENQ", request to send a message.

NOENQ No "ENQ"; produced when "ENQ" does not follow "SYN" within 8 characters of the "SYN" sequence at beginning of received message.

NOMES No message header found; produced when no "SOH" was received within 8 characters after sender's "ENQ" was received and acknowledged at beginning of received message.

NOSTX No "STX" received after message header was processed.

TXERR Text error; produced while receiving text if a control character other than "ETX" was received.

Send Errors in Message

Each time an error occurs within the Send Message Procedure, the source of error is indicated by adding "1" to one of the following parameters.

BADCMS Bad character in message; produced if received response to message after it was sent was "DC3" indicating receiver found bad character in message.

BADCHS Bad character in header; produced if received response to message after it was sent was"DC2" indicating receiver found bad character in header.

BADSMS Bad sum; produced if received response to message after it was sent was "NAK" indicating receiver's checksum did not match SUM sent as binary word.

BUFOFS Buffer overflow; produced if received response to message after it was sent was "DCI" indicating receiver's buffer overflowed.

NOANS No answer; produced before message was sent if no "SYN" sequence was received within 10 milliseconds of sending an "ENQ", request to send message.

NOBUFS No buffer at receive end; produced before message was sent if "NAK" was received in response to "ENQ", request to send message.

NORSP No response; produced after message was sent if no legal response character was received within 7 character intervals after "NUL" send sequence.

NOWAK No acknowledge after "SYN"; produced before message was sent if received response was "SYN" sequence, but not followed by legal "ACK," "NAK" or "ENQ" replies within 10 millisecond response time allowed.

RSPCH The response character received after a message was sent. If the message was sent successfully this character will be "ACK," otherwise, the type of error indicated should identify RSPCH.

STMOUT Timeout occurred while in Send Message Procedure. The timeout may have been due to attempt to transmit or receive. See Error Parameters Common to Send or Receive Message Procedures.

WAKCH The wake character received before a message was sent. It is the character received in response to "ENQ," request to send message unless NOANS was indicated.

Error Parameters Common to Send or Receive Message Procedures

ENDRCV Receive End Flag came on; produced when the Modem has lost line control (usually due to lost synchronization) in RCHAR routine attempting to read next character.

RTMOUT No Receive Flag indication before timeout occurred; produced when Interface did not signal that it has received a character within 200 microseconds after entry into RCHAR routine (usually due to Interface in Receive Inactive state and no "SYN" sequence arrives to activate it).

RC The last character successfully received by the RCHAR routine.

WC The last character sent to XCHAR routine to be transmitted.

XTMOUT No Transmit Flag indication before timeout occurred; produced when Interface did not signal that it was ready to transmit a character within 200 microseconds after entry into XCHAR routine (usually due to loss of sync).