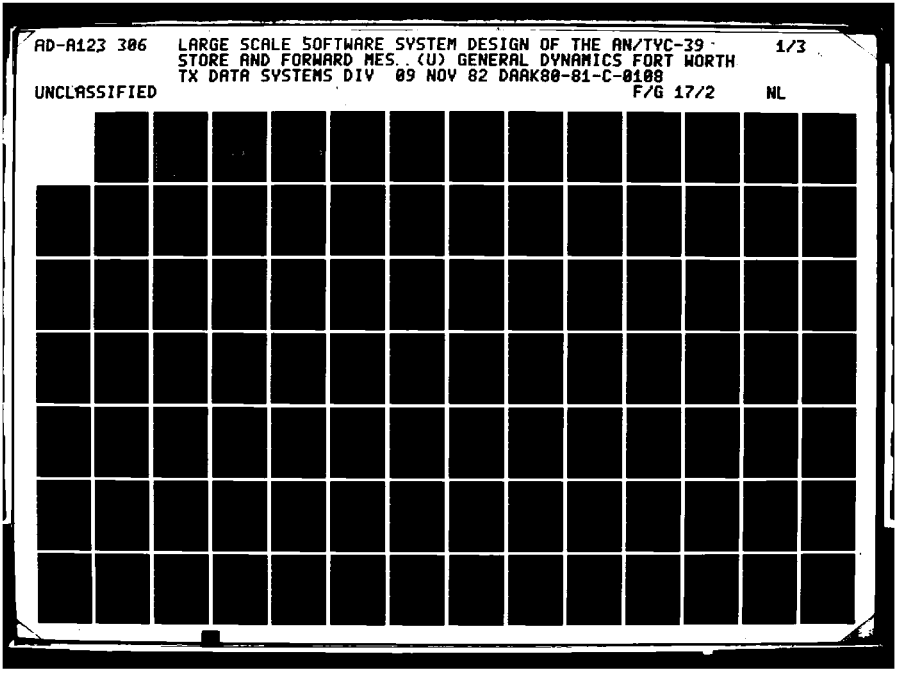
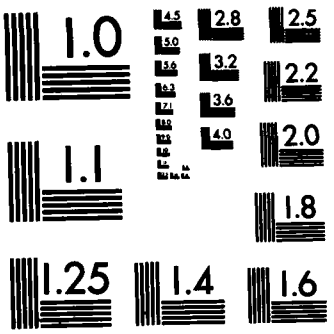
AD-A123 306 LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE AN/TYC-39 STORE AND FORWARD MES..(U) GENERAL DYNAMICS FORT WORTH TX DATA SYSTEMS DIV 09 NOV 82 DAAK80-81-C-0108 1/3

UNCLASSIFIED F/G 17/2 NL





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

AD A123306

CASE STUDY I

FINAL REPORT DEVELOPED FOR LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE AN/TYC-39 STORE AND FORWARD MESSAGE SWITCH USING THE ADA PROGRAMMING LANGUAGE

U. S. ARMY CECOM
CONTRACT NO. DAAK80-81-C-0108

VOLUME III OF IV

DTIC
JAN 12 1983
H

GENERAL DYNAMICS
DATA SYSTEMS DIVISION
CENTRAL CENTER
P. O. BOX 748
FORT WORTH, TX 76101

FILE COPY

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

83 01 12 031

50272-101

REPORT DOCUMENTATION PAGE	1. REPORT NO.	2.	3. Recipient's Accession No. AD-A123306

4. Title and Subtitle

Ada Capability Study: Design of the Message Switching System AN/TYC-39 Using the Ada Programming Language

5. Report Date

November 1982

6.

7. Author(s)

General Dynamics

8. Performing Organization Rept. No.

9. Performing Organization Name and Address

General Dynamics
Data Systems Division
Central Center
P. O. Box 748
Fort Worth, TX 76101

10. Project/Task/Work Unit No.

11. Contract(C) or Grant(G) No.

(C) DAAK80-81-C-0108

(G)

12. Sponsoring Organization Name and Address

USA CECOM
Center for Tactical Computer Systems (CENTACS)
ATTN: DRSEL-TCS-ADA-1
Fort Monmouth, NJ 07703

13. Type of Report & Period Covered

Final

14.

15. Supplementary Notes

16. Abstract (Limit: 200 words)

An Ada oriented framework for the design and documentation of the U. S. Army TYC-39 store and forward message switch (military software) system is presented. This document package contains a Requirements, Design, Ada Integrated Methodology, and Final Report section. A methodology to use Ada in specifying requirements, design, and the implementation of a system was developed. This methodology was used to redesign the TYC-39 message switch system. A selected software module was programmed after the redesign.

DTIC
JAN 12 1983
H

17. Document Analysis a. Descriptors

Ada Programming Language
Software Design with Ada
Designing with Ada

b. Identifiers/Open-Ended Terms

Message Switch
Military Software
Program Design Language

c. COSATI Field/Group

18. Availability Statement:

~~Distribution limited to the United States.~~ Available from National Technical Information Service, Springfield, VA 22161.

19. Security Class (This Report) UNCLASSIFIED

21. No. of Pages 521

20. Security Class (This Page) UNCLASSIFIED

22. Price

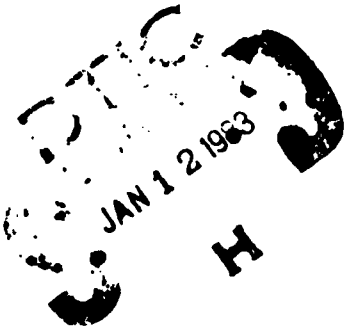
(See ANSI-Z39.18) See instructions on Reverse OPTIONAL FORM

CASE STUDY I

FINAL REPORT DEVELOPED FOR LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE AN/TYC-39 STORE AND FORWARD MESSAGE SWITCH USING THE ADA PROGRAMMING LANGUAGE

U. S. ARMY CECOM
CONTRACT NO. DAAK80-81-C-0108

VOLUME III OF IV



GENERAL DYNAMICS
DATA SYSTEMS DIVISION
CENTRAL CENTER
P. O. BOX 748
FORT WORTH, TX 76101

SCOPE

The system requirements herein described are a subset of the U.S. Army TYC-39 store and forward message switch. In order to understand the terminology used in this document, Chapters 1 and 2 of the Ada Integrated Methodology (developed by General Dynamics under the same contract) must be thoroughly reviewed.

The purpose of this document is to describe an Ada oriented framework for the design and documentation of an existing large scale military software system. It is not a B5 or C5 specification, although it is anticipated that the Functional Decompositional Models and Concurrency Diagrams will form a basis for a B5, while the Ada Functional Requirements should serve as a basis for C5 equivalent specifications. Hardware and software have been specified functionally in Ada form. Thus, the optimum hardware/software partitioning and hardware description is to be a part of the design process (next phase).

Due to the complexity of the switch, certain considerations were assumed as follows:

1. Modems, cryptographic, and line multiplexor equipment are external to the switch.

2. The data adapter equipment is not supported.

3. Although discussed, the operator interface is not described here. A user interactive conversational approach is recommended, but not considered a truly real time aspect of the system.

4. Y-community traffic is considered only to the extent discussed in the non-classified documents used in the project.

5. Although journal and reference storage tapes are generated, the OFF LINE functions, such as message search, are not considered.

6. Mode III, VI, and higher line protocols are not included.

DTIC COPY INSPECTED

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Avail and/or
Dist Special
A

The following documents, in addition to the Ada Integrated Methodology developed by General Dynamics, were used in developing this specification:

1. Performance specification, central office, communications, automatic, TT-B1-1101-0001A.

2. Standards, DCS Autodin Interface and Control Criteria, DCAC 370-D175-1.

3. Communication Instructions, Tape Relay-Procedures, ACP 127(D).

4. Automatic Digital Network (Autodin) Operating Procedures JANAP 128(D).

5. Operator's Aide, Automatic Message Switching Central, AN/TYC-39, July 1978.

TABLE OF CONTENTS

(This page intentionally left blank.)

System Overview

MESSAGE SWITCH SYSTEM DESCRIPTION

The purpose of the TYC-39 message switch is to act as an automatic relay for digital message data in a battlefield functional area (BFA). The equipment is van mounted, which allows the mobility required by ground forces. Although the messages originate at a keyboard terminal, the switch is capable of interfacing to nearly all digital communications equipment in the inventory of the U.S. Army and allied armed forces.

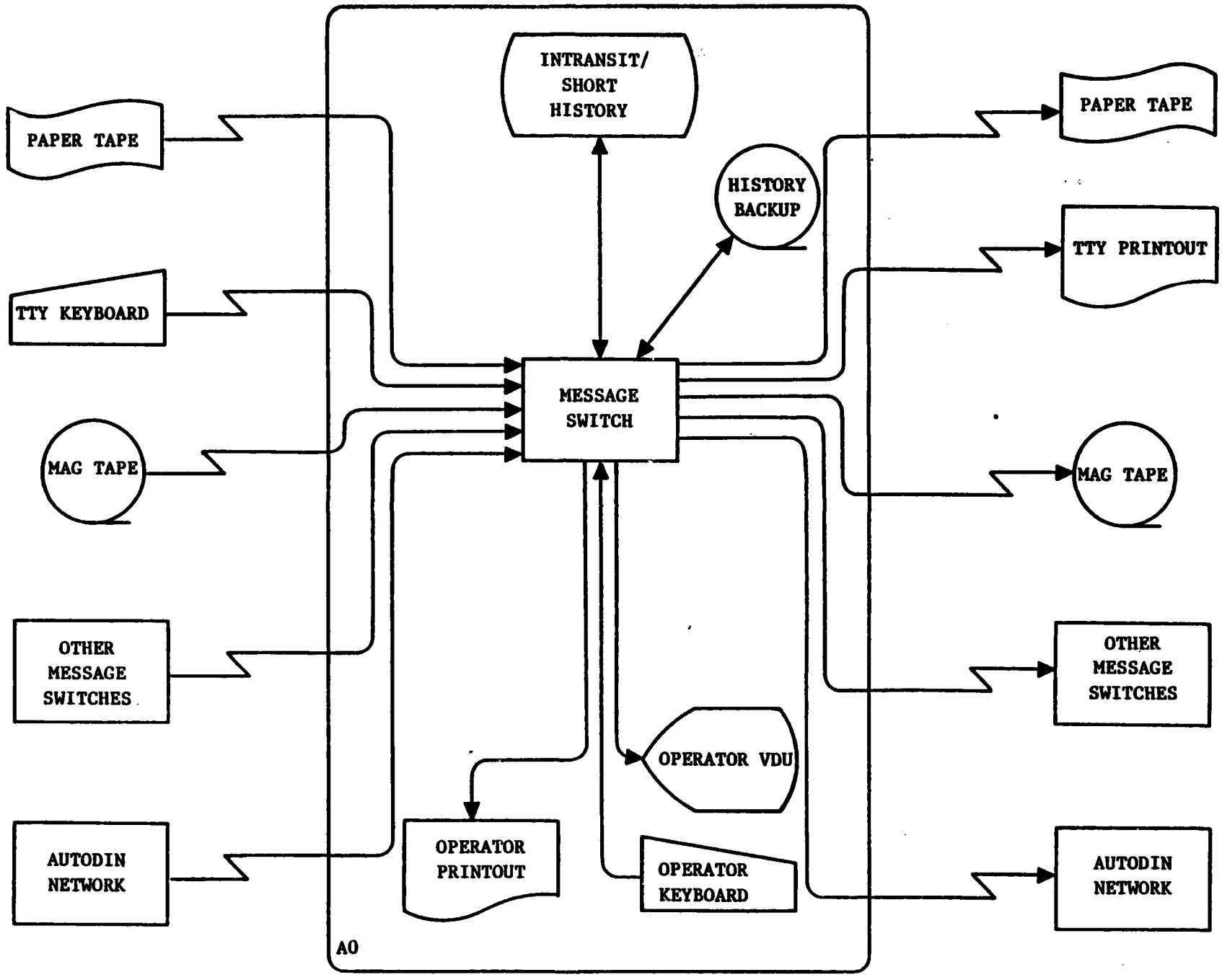
The simplified message switch environment considered in this project is depicted in the Node AO-1 diagram. The devices on the left side, such as paper tape, Autodin, and other message switches are regarded as input, whereas, devices on the right side of the diagram are regarded as outputs from the message switch. This does not preclude the fact that message channels are capable of simultaneous two-way transmission. The Autodin network on the left and right sides of the diagram could be the same network access over a two-way channel. Any combination of up to fifty channels can be connected to the switch. The switch requires the use of non volatile storage for history of message activity and as a backup in case of memory failure. Since this study did not consider cryptographic and modem equipment, none is shown in the diagrams.

Internal message switch processing is depicted by the top level functional decomposition model (Node AO). After switch startup, program load, and database initialization, incoming messages are received, validated for proper format, and forwarded to their final destination or other relay points. The details of this process are explained in the following paragraphs.

System startup (A1) consists of switch initialization and sending the start command to each of the other functions of the switch. Startup is a relatively simple subset of system recovery, which will be discussed later.

The "operator interface" function (A2) is used to communicate with the personnel operating the system. They must be able to enter, verify, and change database information which the switch uses to determine message routing and line or trunk characteristics. Other interface commands obtain information about switch performance, cause printouts of specific messages, trace messages through the system, or exhibit various information about messages passing through the switch. The information required for the printouts comes primarily from the reference and journal files. Other operator commands deal with system shutdown, re-introduction of undeliverable messages, diagnostic capability, etc.

Although the remaining functions of the switch operate simultaneously on up to fifty different messages, for a single message, processing may be viewed as a sequential process. Activity within the message switch is created by receipt of



A0-1 MESSAGE SWITCH SYSTEM

multiple serial bit streams (message data) at the "assemble messages" function (A3, left side). The bit streams are converted to characters, lines, and blocks until a complete message is received. The function must be capable of dealing with several different protocols, ranging from fully asynchronous with no blocking and no acknowledgement or other control to rigidly blocked synchronous traffic with block by block acknowledgement. Received messages must be validated for proper format to eliminate garbled transmissions. High precedence messages may be passed even with some specified errors. The "assemble messages" function keeps a log of its operations on each message in the reference file. It also routes a complete copy of each message as it was received to the reference file, and queues received messages by predecence for "process message".

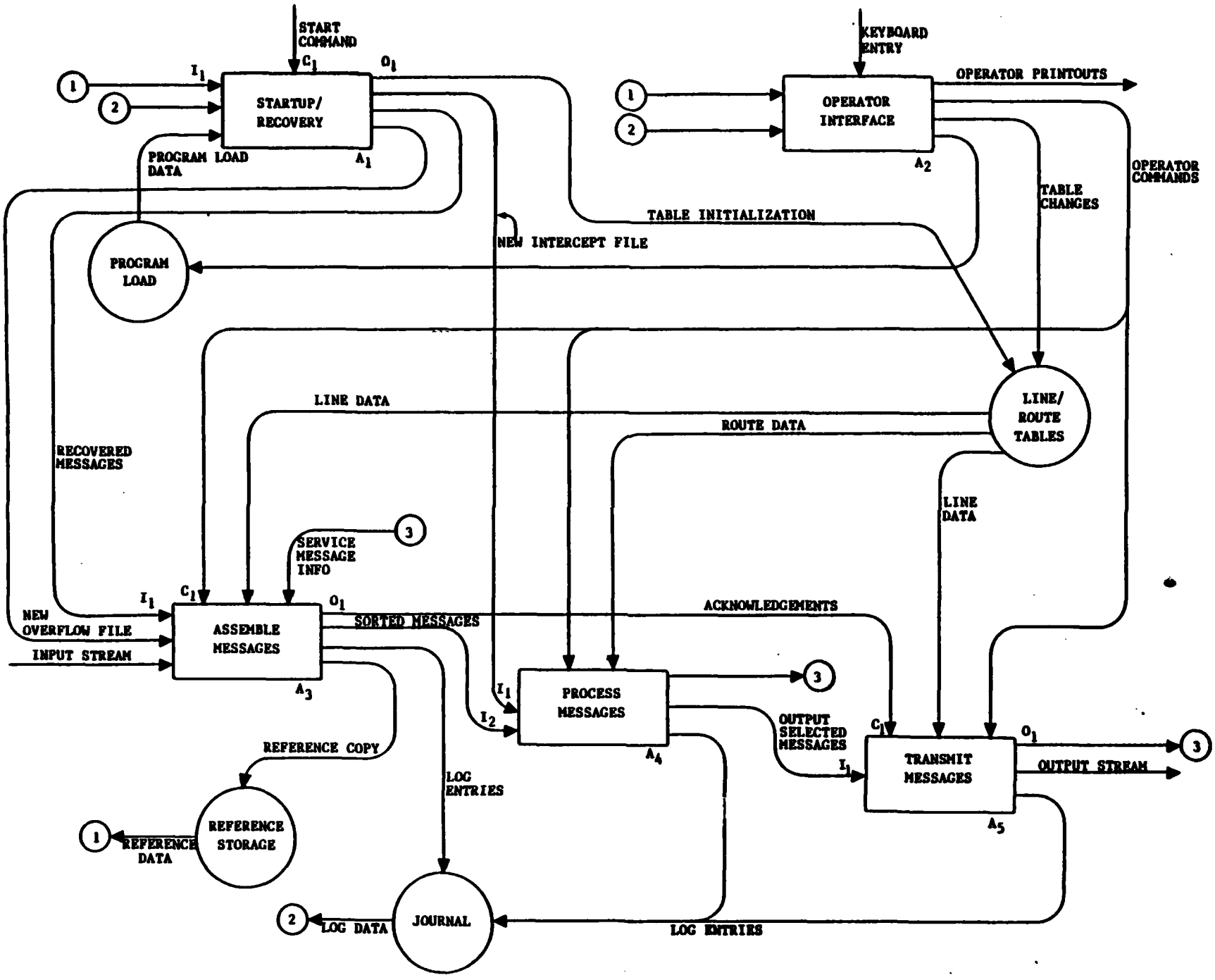
The "process messages" function (A4) obtains received messages from the input queue, determines output routing, performs any necessary translation, and queues for transmission. Once a message has been accepted and queued, a copy is routed to the "in-transit" file (process delineated as a subset of A4). If the "in-transit" file becomes filled beyond a certain threshold, incoming messages are diverted to the "overflow" file, which are automatically reentered into the system as the "in-transit" file lower threshold is reached.

Messages may have more than one destination. When a message must be transmitted over more than one channel, the portion of the message containing the routing information (the routing line) must be edited to remove destinations not applicable to each individual channel. In addition, different destinations may require differing formats (JANAP-128 vs. ACP-127) or different character codes (ASCII vs. ITA#2). If so, these transformations are provided at this point. Routed messages are then sent back to the in-transit file unless the channel for the message is marked as closed, in which case the message is sent to the intercept file. Messages in the intercept file may be re-introduced on operator command.

The "transmit messages" function (A5) converts the routed message into a bit stream and transmits it. This function must be able to handle simultaneous operation of up to fifty lines with several different protocols. Any blocking or control characters must be added at this stage. Final validation, to include a check that the security classification limit for this line is not exceeded, is performed at this time. The transmit function also keeps a log of its activities in the journal file.

The recovery function (A1) must insure that messages being processed at the time of a system failure are not lost. To do this, the journal file is used to perform an audit of the messages received by the switch. Any message which had not completed its final delivery at the time of the failure is re-introduced into the system from the reference file, and labeled as a "suspected duplicate".

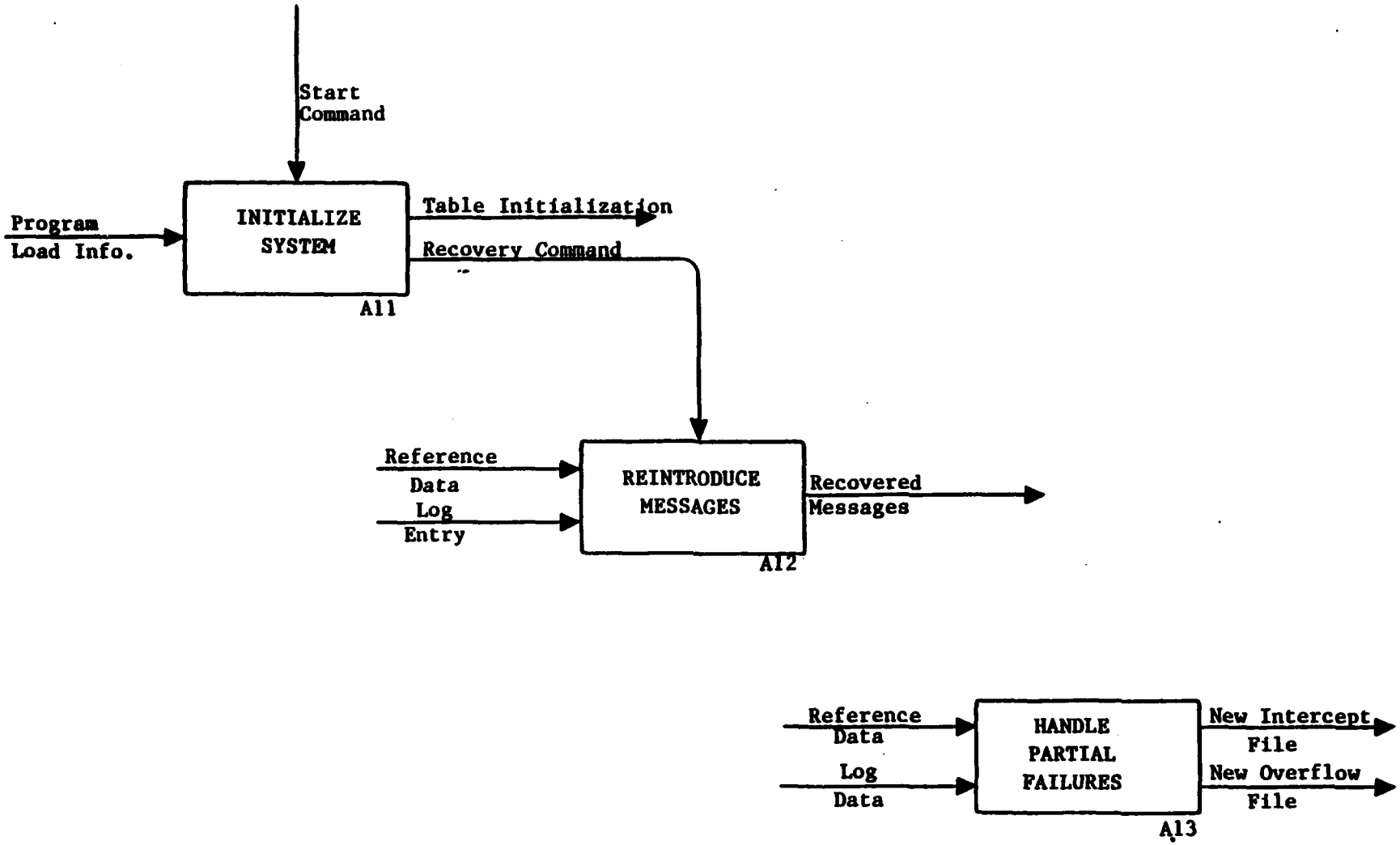
In all operations the switch must maintain high standards of reliability and throughput. The switch must operate on a priority basis, with first-in-first-out operation within precedence.



A0

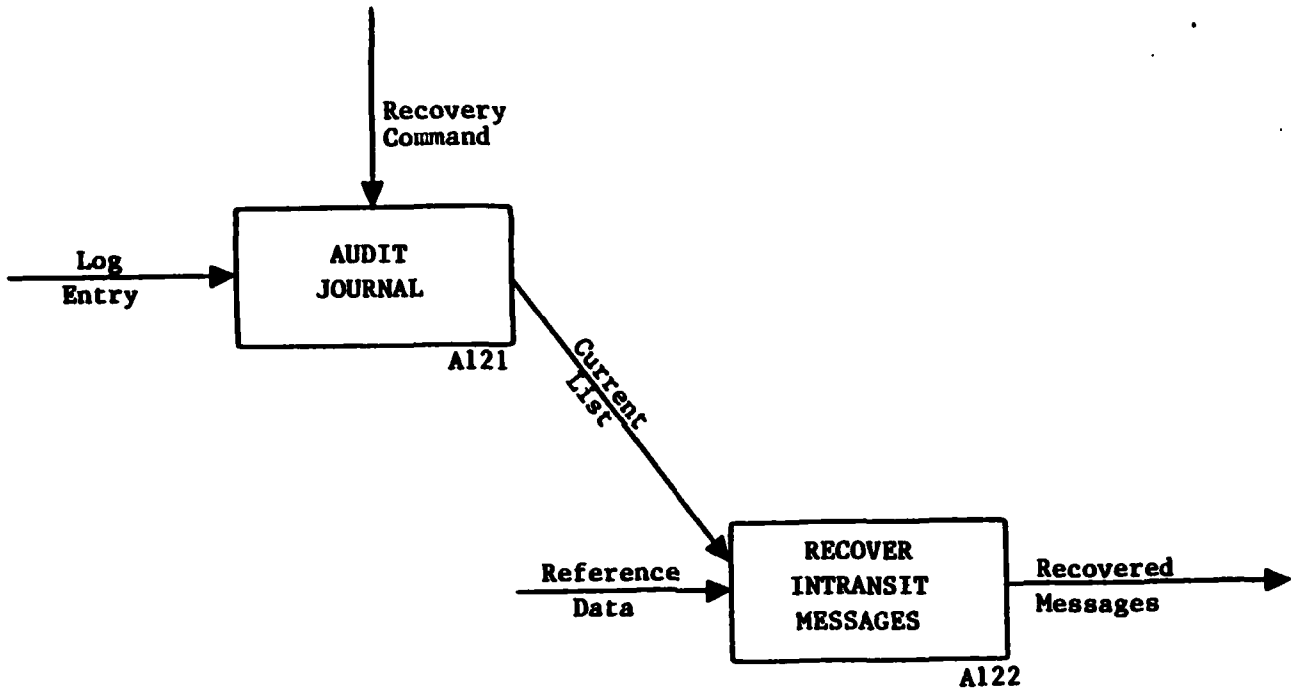
MESSAGE SWITCH

Functional Decomposition Models and Ada Functional Requirements



A1 STARTUP/RECOVERY

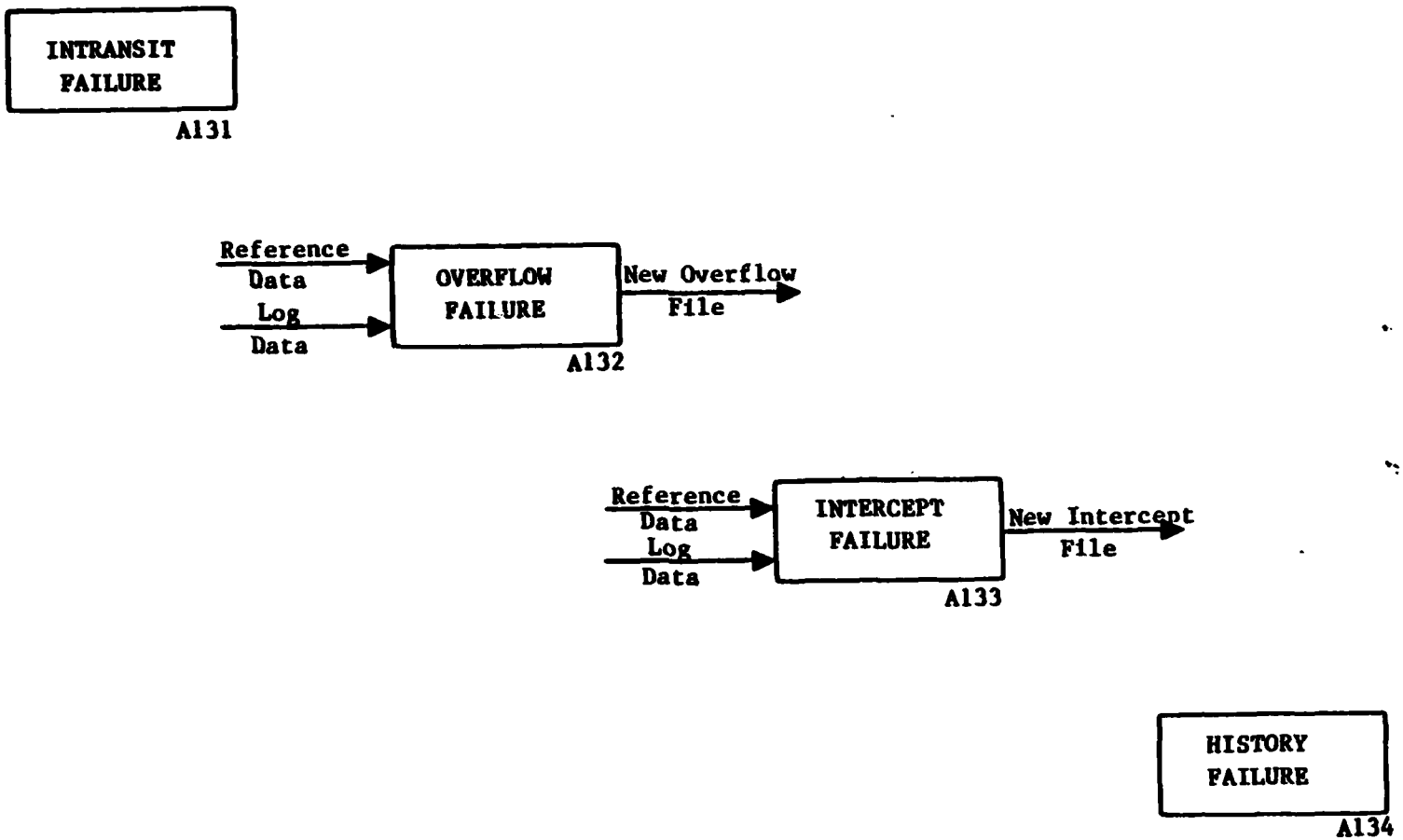
```
        --A11 INITIALIZE SYSTEM
        --REV BAA
        --12/3/81 PD

        procedure INITIALIZE_SYSTEM is
        begin
          --> SET UP SYSTEM TABLES FROM PROGRAM LOAD FILE
              --> (OR OPERATOR CONSOLE)
          --> CLEAR BUFFERS
          --> INITIALIZE LINE HANDLERS
        end INITIALIZE_SYSTEM;

EOT..
```



A12 REINTRODUCE MESSAGES

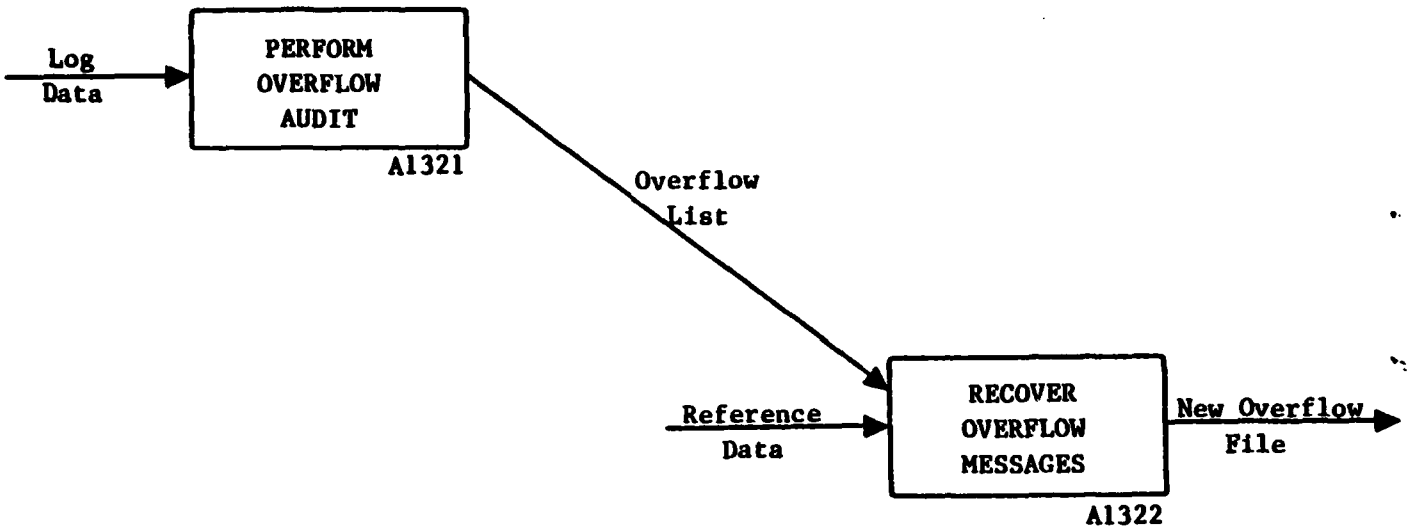
```
--A121 AUDIT JOURNAL
--REV BAAA
--11/24/81 PD

procedure AUDIT_JOURNAL is
begin
  for all LOG_ENTRIES in HISTORY_FILE loop
    case ENTRY_TYPE is
      when EOM_IN =>
        --> ADD MESSAGE_ID TO CURRENT_LIST
      when SVC_GEN =>
        --> ADD MESSAGE_ID TO CURRENT_LIST
      when EOM_OUT =>
        if LAST_COPY then
          -->REMOVE MESSAGE_ID FROM CURRENT_LIST
        end if;
      when OVERFLOW_OUT =>
        --> REMOVE MESSAGE_ID FROM CURRENT_LIST
      when OVERFLOW_IN =>
        --> ADD MESSAGE_ID TO CURRENT_LIST
      when INTERCEPT_OUT =>
        --> REMOVE MESSAGE_ID FROM CURRENT_LIST
      when INTERCEPT_IN =>
        --> ADD MESSAGE_ID TO CURRENT_LIST
      when others =>
        null;
    end case;
    --> COLLECT LAST CSN FOR EACH LINE
  end loop;
end AUDIT_JOURNAL;

EOT..
```

```
      --A122 RECOVER INTRANSIT MESSAGES
      --REV BAAA
      --11/24/81 PD

      procedure RECOVER_INTRANSIT_MESSAGES is
      begin
        for all MESSAGES ON HISTORY_TAPE loop
          if MESSAGE_ID ON CURRENT_LIST then
            --> ADD MESSAGE TO RECOVERED_MESSAGES
            --> REMOVE MESSAGE_ID FROM CURRENT_LIST
          end if;
        end loop;
        if CURRENT_LIST not EMPTY then
          --> NOTIFY OPERATOR
        end if;
      end RECOVER_INTRANSIT_MESSAGES;

EOT..
```



A13 HANDLE PARTIAL FAILURES

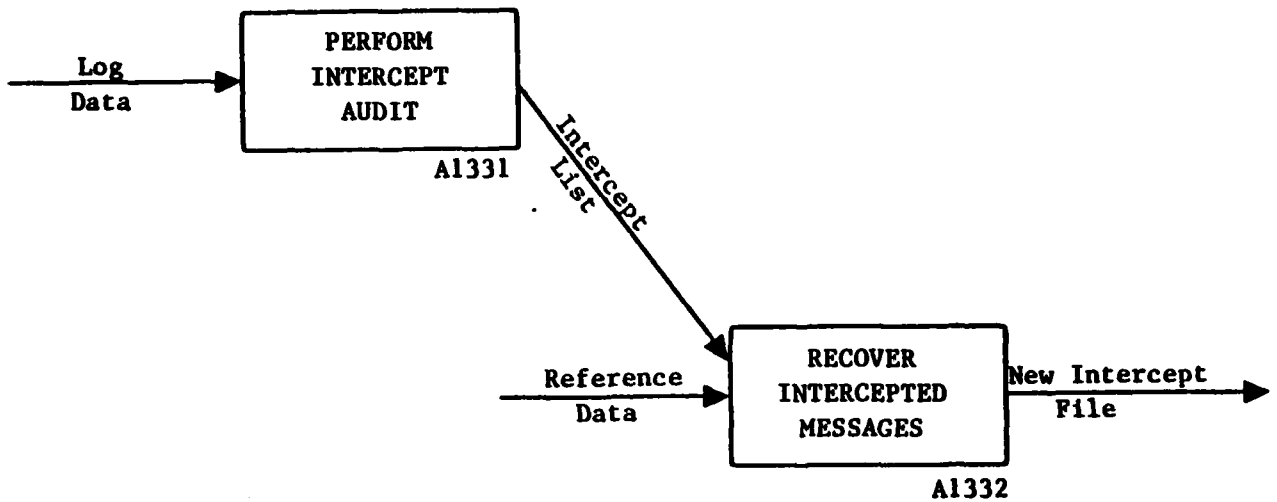
```
      --A131 INTRANSIT FAILURE
      --REV BAAA
      --11/25/81 PD

      procedure INTRANSIT_FAILURE is
      begin
        if FIRST_FAILURE then  --ONE COPY OF INTRANSIT STORAGE REMAINING
          --> NOTIFY OPERATOR
          if FAILED_UNIT = CURRENT_UNIT then
            CURRENT_UNTI := BACKUP_UNIT;
          end if;
        else   -- NO COPIES OF INTRANSIT STORAGE LEFT
          --> NOTIFY OPERATOR
          --> FAIL SYSTEM
        end if;
      end INTRANSIT_FAILURE;

EOT..
```



A132 OVERFLOW FAILURE

```
      --A1321 PERFORM OVERFLOW AUDIT
      --REV BAAAA
      --11/24/81 PD

      procedure PERFORM_OVERFLOW_AUDIT is
      begin
        for all LOG_ENTRIES in HISTORY_FILE loop
          if ENTRY_TYPE = OVERFLOW_OUT then
            --> ADD MESSAGE_ID TO OVERFLOW_LIST
          elsif ENTRY_TYPE = OVERFLOW_IN then
            --> DELETE MESSAGE_ID FROM OVERFLOW_LIST
          else
            null;
          end if;
        end loop;
      end PERFORM_OVERFLOW_AUDIT;

EOT..
```

```
      --A1322 RECOVER OVERFLOW MESSAGES
      --REV BAAAA
      --11/24/81 PD.

      procedure RECOVER_OVERFLOW_MESSAGES is
      begin
        for all MESSAGES ON HISTORY_TAPE loop
          if MESSAGE_ID ON OVERFLOW_LIST then
            --> ADD MESSAGE TO NEW OVERFLOW FILE
            --> REMOVE MESSAGE_ID FROM OVERFLOW_LIST
          end if;
        end loop;
        if OVERFLOW_LIST not EMPTY then
          --> NOTIFY OPERATOR
        end if;
      end RECOVER_OVERFLOW_MESSAGES;

EOT..
```

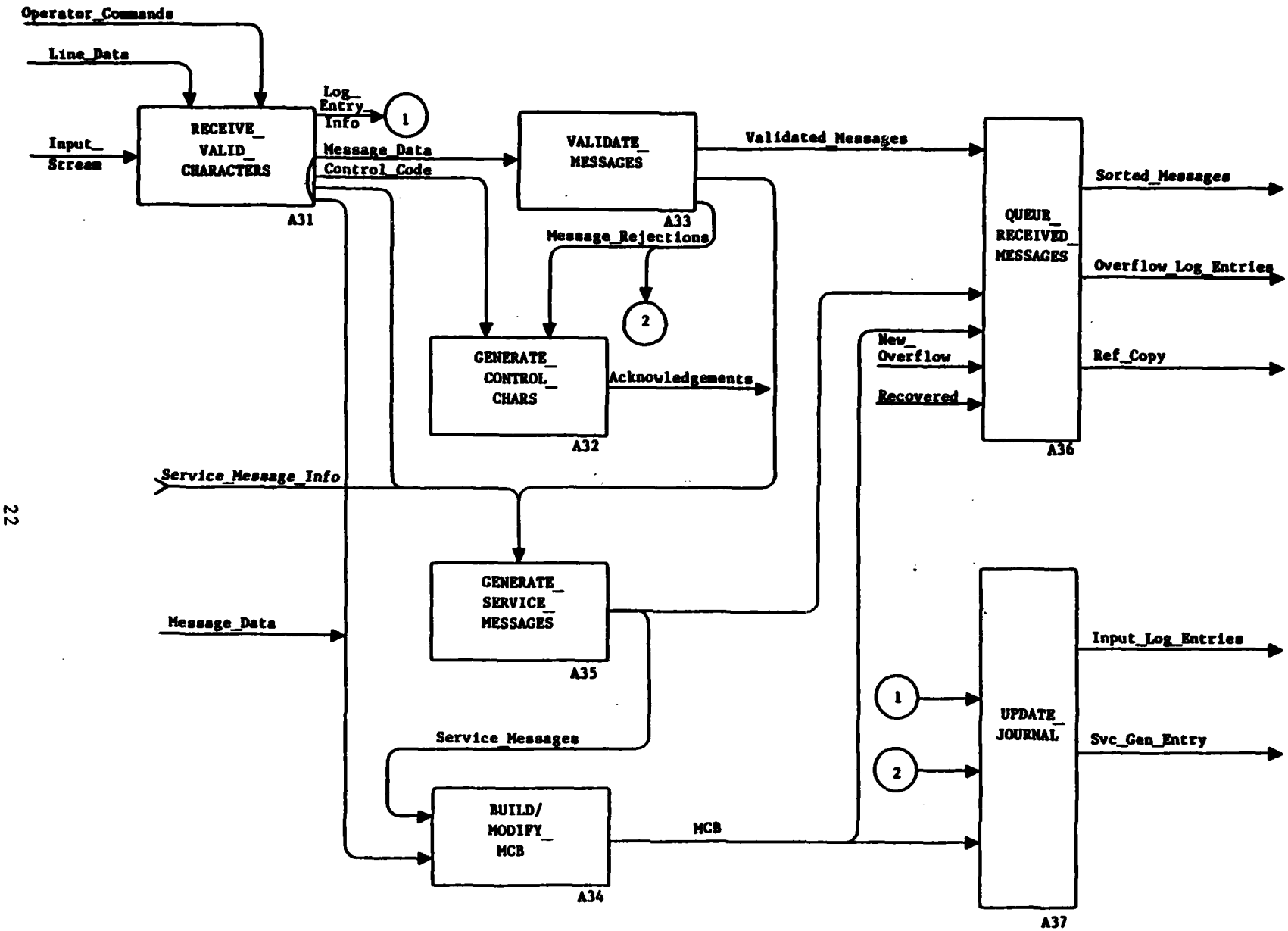


A133 INTERCEPT FAILURE

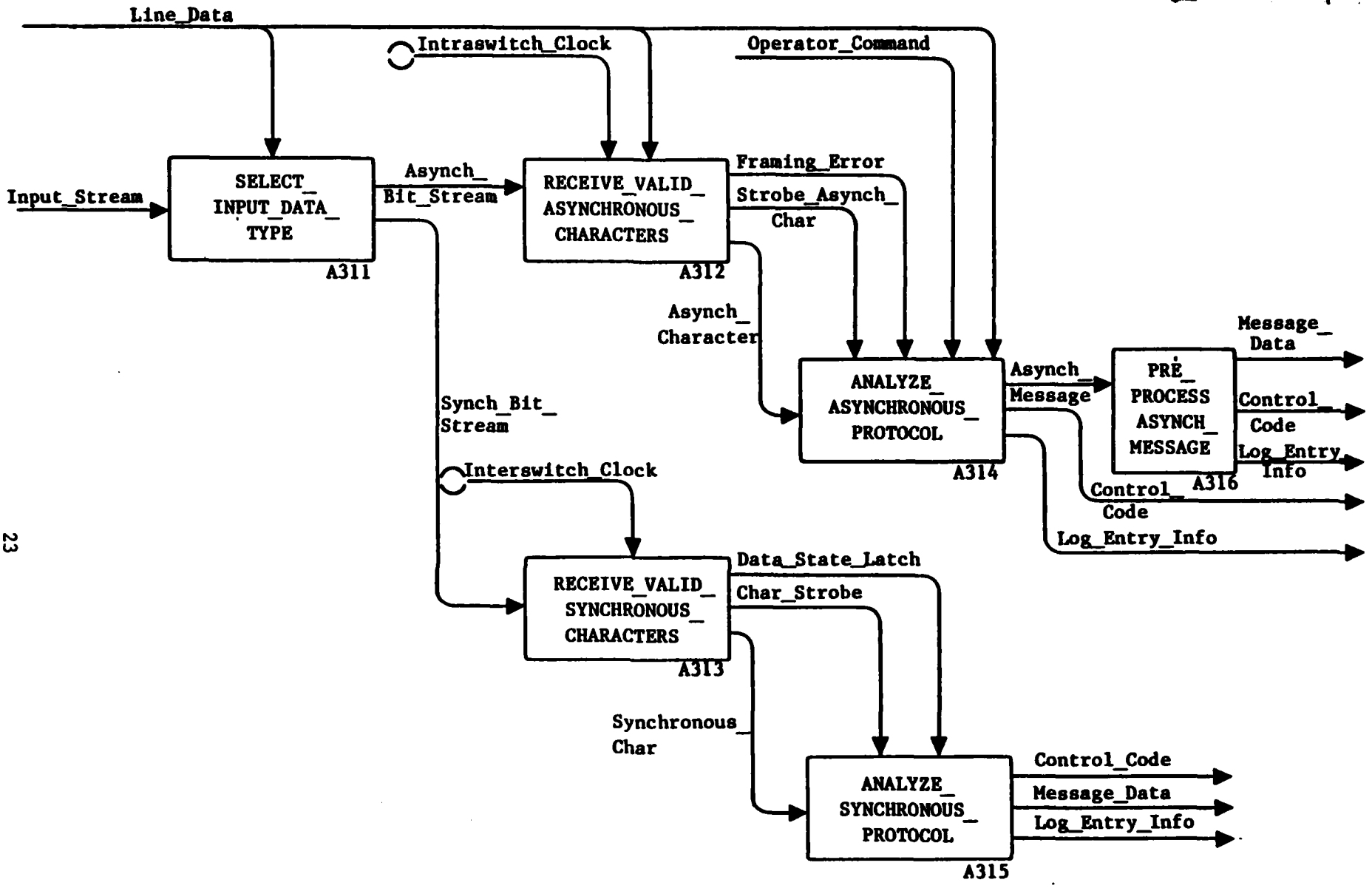
```
      --A1331 PERFORM INTERCEPT AUDIT
      --REV BAAAA
      --11/24/81 PD

      procedure PERFORM_INTERCEPT_AUDIT is
      begin
        for all LOG_ENTRIES in HISTORY_FILE loop
          if ENTRY_TYPE = INTERCEPT_OUT then
            --> ADD MESSAGE_ID TO INTERCEPT_LIST
          elsif ENTRY_TYPE = INTERCEPT_IN then
            --> DELETE MESSAGE_ID FROM INTERCEPT_LIST
          else
            null;
          end if;
        end loop;
      end PERFORM_INTERCEPT_AUDIT;

EOT..
```

```
--A1332 RECOVER INTERCEPT MESSAGES
--REV BAAAA
--11/24/81 PD

procedure RECOVER_INTERCEPT_MESSAGES is
begin
  for all MESSAGES ON HISTORY_TAPE loop
    if MESSAGE_ID ON INTERCEPT_LIST then
      --> ADD MESSAGE TO NEW_INTERCEPT_FILE
      --> REMOVE MESSAGE_ID FROM INTERCEPT_LIST
    end if;
  end loop;
  if INTERCEPT_LIST not EMPTY then
    --> NOTIFY_OPERATOR
  end if;
end RECOVER_INTERCEPT_MESSAGES;

EOT..
```

```
      --A134 HISTORY_FAILURE
      --REV BAAA
      --12/3/81 PD

      procedure HISTORY_FAILURE is
      begin
        if A COPY of HISTORY REMAINS then
          --> NOTIFY OPERATOR
          --> RECOPY REMAINING HISTORY FOR NEW BACKUP
        else
          --> NOTIFY OPERATOR
          --> START NEW HISTORY FILE
        end if;
      end HISTORY_FAILURE;

EOT..
```



A3 ASSEMBLE MESSAGES

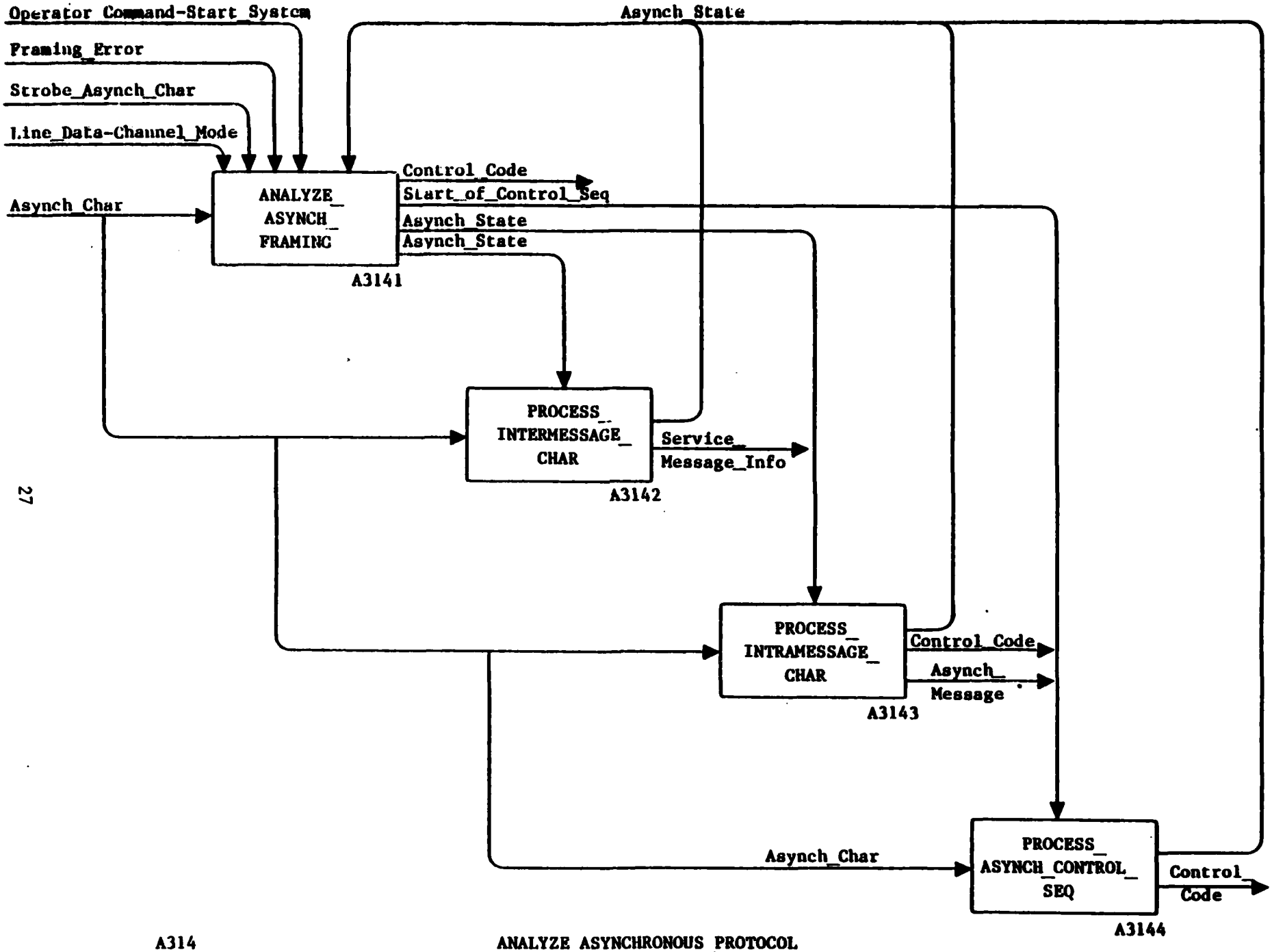


A31 RECEIVE_VALID_CHARACTERS

```
      -- A311 SELECT_INPUT_DATA_TYPE
      -- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
      -- TO SELECT EITHER SYNCHRONOUS OR ASYNCHRONOUS DATA PROCESSING
      -- DEPENDING ON THE MODE OF THE SPECIFIC MESSAGE SWITCH LINE
      -- AS DESCRIBED IN THE FOLLOWING:
      -- MANUAL PARAGRAPH(S)
      -- DCAC 370-D175-1 CHAPTERS 5 AND 11
      -- 01/25/81 HF

      procedure SELECT_INPUT_DATA_TYPE is
      begin
        if CHANNEL_MODE for THIS LINE = [2|4|5] then
          RECEIVE_VALID_ASYNCHRONOUS_CHARACTERS;
        elsif CHANNEL_MODE for THIS LINE = [1|3] then
          RECEIVE_VALID_SYNCHRONOUS_CHARACTERS;
        else
          DATABASE INITIALIZATION ERROR;
        end if;
      end SELECT_INPUT_DATA_TYPE;

EOT..
```

```
--A312 RECEIVE VALID ASYNCHRONOUS CHARACTERS
--REV BAAA
--2/4/82 PD

procedure RECEIVE_VALID_ASYNCHRONOUS_CHARACTERS is
  type PTY is (EVEN,ODD);
  PARITY : PTY;
  type BIT is (LOW,HIGH);
  BIT_SEQ : array(0..7) of BIT;
  pragma PACK(BIT);
  CLOCK_CYCLE : DURATION;
  FRAMING_ERROR : BOOLEAN;
begin
  -- ASSUMES CLOCK RATE IS 16 TIMES BAUD RATE
  -- FOR TIMING PURPOSES ASSUMES THAT INSTRUCTIONS EXECUTE IN
      -- ZERO TIME
  -- FOR IMPLEMENTATION, DELAYS MUST BE ADJUSTED FOR
      -- INSTRUCTION EXECUTION TIMES
  loop
    while ASYNCH_BIT_STREAM /= LOW loop
      delay 1.0 * CLOCK_CYCLE;
    end loop;
    delay 24.0 * CLOCK_CYCLE;
    BIT_SEQ := (0..7 => LOW);
    PARITY := EVEN;
    for I in reverse 8-LEVEL..7 loop
      BIT_SEQ(I) := ASYNCH_BIT_STREAM;
      if BIT_SEQ(I) = HIGH then
        if PARITY = EVEN then
          PARITY := ODD;
        else
          PARITY := EVEN;
        end if;
      end if;
      delay 16.0 * CLOCK_CYCLE;
    end loop;
    FRAMING_ERROR := FALSE;
    I := 1;
    loop
      if ASYNCH_BIT_STREAM = LOW then
        FRAMING_ERROR := TRUE;
      end if;
      I := I + 1;
      delay 8.0 * CLOCK_CYCLE;
      exit when I > NO_STOP_BITS;
      delay 8.0 * CLOCK_CYCLE;
    end loop;
    ASYNCH_CHAR := UNCHECKED_CONVERSION(BIT_SEQ(1..7));
    STROBE_ASYNCH_CHAR;
  end loop;
end RECEIVE_VALID_ASYNCHRONOUS_CHARACTERS;

EOT..
```

```
-- A313 RECEIVE_VALID_SYNCHRONOUS_CHARACTERS
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- TO SYNCHRONIZE AND CONVERT A SERIAL BIT STREAM TO A CHARACTER
-- ORIENTED FORMAT, AND CHECK FOR PARITY ERRORS AS DESCRIBED IN
-- THE FOLLOWING:
--              MANUAL                    PARAGRAPH(S)
-- DCAC 370-D175-1                      CHAPTER 4 AND 5
-- 02/08/82 HF
   procedure RECEIVE_VALID_SYNCHRONOUS_CHARACTERS is
   begin
   -- THIS REQUIREMENT IS SIMILAR IN NATURE TO A312, FOR WHICH THERE
   -- IS AN EXAMPLE. IT IS EXPECTED THAT THIS FUNCTION WILL BE
   -- IMPLEMENTED BY HARDWARE.
   end RECEIVE_VALID_SYNCHRONOUS_CHARACTERS;
EOT..
```



A314 ANALYZE_ASYNCHRONOUS_PROTOCOL

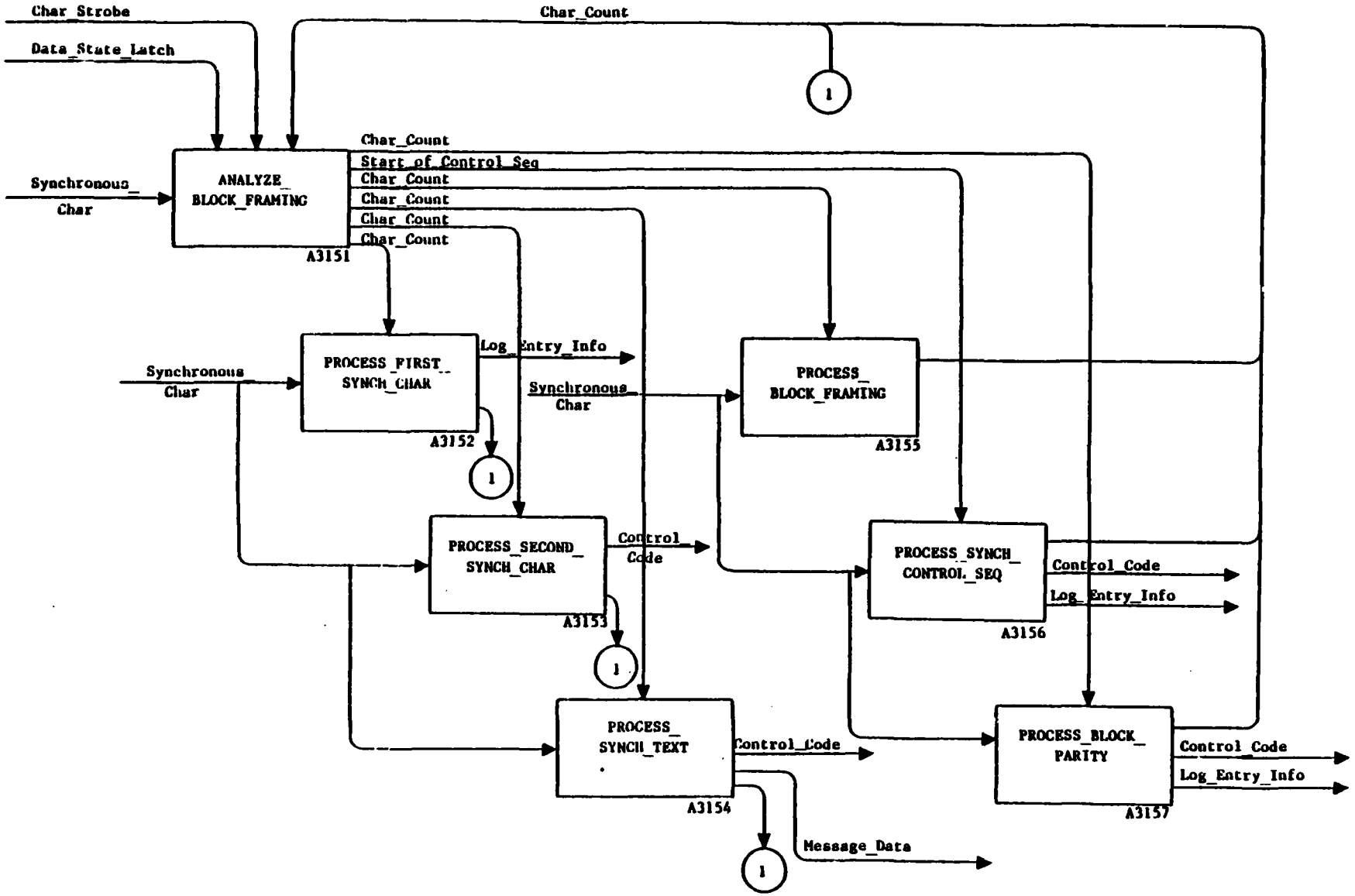
```
-- A3141 ANALYZE_ASYNCH_FRAMING
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR ANALYSIS OF THE FRAMING OF AN ASYNCHRONOUS MESSAGE
-- RECEIVED FROM A MESSAGE SWITCH TERMINAL.
-- AS DESCRIBED IN THE FOLLOWING:
--                MANUAL                          PARAGRAPH(S)
-- TT-B1-1101-0001A AND DCAC 370-D175-1, PAGE 109 AND CH. 11
-- 01/31/82 HF

procedure ANALYZE_ASYNCH_FRAMING is
begin
  if START_SYSTEM_COMMAND or FRAMING_ERROR then
    GENERATE_CONTROL_CHARS;
    -- USE RT_RCVD CODE (RT)
    START_OF_CONTROL_SEQUENCE:=FALSE;
    ASYNCH_STATE:=INTERMESSAGE;
  elsif STROBE_ASYNCH_CHAR then
    if CHANNEL_MODE = 5 then
      if EXPIRED_PAUSE_TIMER then
        if PROTOCOL = ITA_#2 or (ASCII and CHARACTER_PARITY =
            EVEN) then
          if RECEIVED CHARACTER = [ASYNCH_RECEIVE_CONTROL_CHAR
              | CANCEL_CHAR | REPLY_CHAR | START_CHAR] then
            START_OF_CONTROL_SEQUENCE:=TRUE;
            -- CTL SEQ IS EFFECTED IF NEXT CONSECUTIVE
            -- CHARACTER IS A REPEAT OF THIS CHARACTER.
            START_PAUSE_TIMER;
            -- IF THE PAUSE TIMER EXPIRES BEFORE THE NEXT
            -- CHARACTER ARRIVES, THIS SEQUENCE STARTS OVER.
            --> ACCEPT CONTROL CHARACTER
            -- NOT TO BE INTERMIXED WITH MESSAGE CHARACTERS.
          else
            ASYNCH_STATE:=INTERMESSAGE;
            -- IMPLIES THAT CHARACTER IS TO BE IGNORED.
          end if;
        else
          ASYNCH_STATE:=INTERMESSAGE;
        end if;
      else
        if START_OF_CONTROL_SEQUENCE = TRUE then
          PROCESS_ASYNCH_CONTROL_SEQ;
        elsif ASYNCH_STATE = INTERMESSAGE then
          PROCESS_INTERMESSAGE_CHAR;
          START_PAUSE_TIMER;
        elsif ASYNCH_STATE = INTRAMESSAGE then
          PROCESS_INTRAMESSAGE_CHAR;
          if CHAR_BUFFER = UPPER_THRESHOLD then
            GENERATE_CONTROL_CHARS;
            -- USE 'SEND_STOP' REQUEST CODE
          end if;
        end if;
      end if;
    else
      if CHANNEL_MODE = 2 or 4 then
        if ASYNCH_STATE = INTERMESSAGE then
```

```
                PROCESS_INTERMESSAGE_CHARACTER;
              elsif ASYNCH_STATE = INTRAMESSAGE then
                PROCESS_INTRAMESSAGE_CHAR;
              end if;
            end if;
          end if;
        elsif NO CHARACTERS RECEIVED for LAST 30 MINUTES then
          GEN_SVC_MSG;
          -- SVC_MESSAGE_TYPE = TRAFFIC_CHECK;
        end if;
      end ANALYZE_ASYNCH_FRAMING;

EOT..
```

```
      --A3142 PROCESS_INTERMESSAGE_CHAR
      --REQUIRED BY:JANAP-128 PAGE 3-24 OR ACP-127 AND
      --TT-B1-1101-0001A,PAR. 3.2.1.2.10.2.10 PAGE 109
      -- 01/31/82 HF

      procedure PROCESS_INTERMESSAGE_CHAR is
      begin
        if PROTOCOL = ITA_#2 or (ASCII and CHARACTER_PARITY = EVEN)
            then
          --> VALIDATE START OF MESSAGE SEQUENCE 'ZCZC'
          if SOM_SEQ = VALID then
            ASYNCH_STATE:=INTRAMESSAGE;
            UPDATE_JOURNAL;
            -- SEND SOM_IN LOG ENTRY INFORMATION CODE
          end if;
          -- IMPLIES THAT ANY OTHER TEXT CHARACTERS WILL BE IGNORED.
        end if;
      end PROCESS_INTERMESSAGE_CHAR;

EOT..
```

```
-- A3143 PROCESS_INTRAMESSAGE_CHAR
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING THE TEXT PORTION OF AN ASYNCHRONOUS MESSAGE
-- FROM A TERMINAL CONNECTED TO THE MESSAGE SWITCH.
-- AS DESCRIBED IN THE FOLLOWING:
--                  MANUAL                          PARAGRAPH(S)
-- TT-B1-1101-0001A AND DCAC 370-D175-1,PAGE 109 AND CHAPTER 11
-- 01/31/82 HF

procedure PROCESS_INTRAMESSAGE_CHAR is
begin
  if PROTOCOL = ITA_#2 then
    if CHARACTER = LETTERS_CHAR then
      CHARACTER_CASE:=LETTERS;
      --> ACCEPT CHARACTER
    elsif CHARACTER = FIGS_CHAR then
      CHARACTER_CASE:=FIGS;
      --> ACCEPT CHARACTER
    else
      --> ACCEPT CHARACTER
    end if;
  elsif PROTOCOL = ASCII then
    if CHARACTER_PARITY = ODD then
      --> ACCEPT CHARACTER
    end if;
  end if;
  if CHARACTER ACCEPTED then
    --> CHECK FOR VALID 'EOM' SEQUENCE (LINE FEED + NNNN)
    if VALID EOM_SEQUENCE DETECTED then
      GENERATE_CONTROL_CHARS;
      -- USE EOMS_RCVD_CODE
      --> ACCEPT MESSAGE
      -- CAN BE OVERRIDDEN BY VALIDATION AND REF STORAGE.
      UPDATE_JOURNAL;
      -- SEND 'EOM_IN' LOG ENTRY INFORMATION CODE
    end if;
    if CHANNEL_MODE = 5 then
      START_PAUSE_TIMER;
    end if;
  else
    ASYNCH_STATE:=INTERMESSAGE;
    GENERATE_CONTROL_CHARS;
    -- USE SEND_RT MESSAGE CODE. THIS WILL INITIALLY CAUSE
    -- STOP TO BE SENT OUT FROM THE XMITTER, CAUSING AN
    -- EVENTUAL REPLY TO BE RECEIVED. THEN RETRANSMIT MESSAGE
    -- (RT) WILL BE SENT OUT.
    UPDATE JOURNAL;
    -- USE LOG_ENTRY_INFO = REJECT_ENTRY
  end if;
end PROCESS_INTRAMESSAGE_CHAR;

EOT..
```

```
-- A3144 PROCESS_ASYNCH_CONTROL_SEQ
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING A CONTROL CHARACTER SEQUENCE FOR A MODE 5
-- ASYNCHRONOUS TERMINAL CONNECTED TO THE MESSAGE SWITCH
-- AS DESCRIBED IN THE FOLLOWING:
--      MANUAL    PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 11
-- 01/31/82 HF

procedure PROCESS_ASYNCH_CONTROL_SEQ is
begin
  if PROTOCOL = ITA_#2 or (ASCII and CHARACTER_PARITY = EVEN)
      then
    if CHARACTER /= PREVIOUS CONTROL_CHAR then
      ASYNCH_STATE:=INTERMESSAGE;
      -- IMPLIES THAT PAUSE TIMER WILL EXPIRE AND SEQ MUST
      -- RESTART.
    elsif CHARACTER = [STOP_CHAR ¦ ACK_CHAR ¦ RETRANSMIT_CHAR]
        then
      GENERATE_CONTROL_CHARS;
      -- USE CODE REQUIRED BY THE CONTROL SEQUENCE, SUCH AS
      -- STOP_RCVD, ACK_1_RCVD, ACK_2_RCVD, OR RT_RCVD.
      START_PAUSE_TIMER;
    elsif CHARACTER = CANCEL_CHAR then
      GENERATE_CONTROL_CHARS;
      -- USE CANCEL_RCVD CODE. THIS ALSO IMPLIES THAT ACK
      -- ALTERNATION SEQUENCE IS SET TO TRANSMIT AN
      -- ACK_2 OUT OF THE TRANSMITTER.
      -- IN ADDITION, IF A STOP SEQUENCE IS IN PROGRESS, IT
      -- IS TO BE RESET.
      UPDATE_JOURNAL;
      -- USE LOG_ENTRY_INFO = CANCEL_REC_ENTRY
      START_PAUSE_TIMER;
    elsif CHARACTER = REPLY_CHAR then
      GENERATE_CONTROL_CHARS;
      -- USE REPLY_RCVD CODE (MESSAGE STATUS ASSUMED TO
      -- BE RT UNLESS APPROPRIATELY ACKED.
      START_PAUSE_TIMER;
    elsif CHARACTER /= START_CHAR then
      ASYNCH_STATE:=INTERMESSAGE;
      -- IMPLIES THAT CHARACTER SEQUENCE IS TO BE IGNORED.
    end if;
  else
    ASYNCH_STATE:=INTERMESSAGE;
  end if;
  START_OF_CONTROL_SEQUENCE:=FALSE;
end PROCESS_ASYNCH_CONTROL_SEQ;

EOT..
```



A315 ANALYZE_SYNCHRONOUS_PROTOCOL

```
-- A3151 ANALYZE_BLOCK_FRAMING
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- TO ANALYZE SYNCHRONOUS PROTOCOL AFTER BIT AND CHARACTER
-- SYNCHRONIZATION HAS BEEN PREVIOUSLY ESTABLISHED.
-- AS DESCRIBED IN THE FOLLOWING:
--      MANUAL      PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5
-- 01/19/82 HF

procedure ANALYZE_BLOCK_FRAMING is
begin
  if CHAR_STROBE then
    if DATA_SCATE_LATCH = TRUE and DATA_STATE=FALSE then
      DATA_STATE:=TRUE;
      CHAR_COUNT:=1;
      START_OF_CONTROL_SEQUENCE:=FALSE;
    elsif DATA_STATE_LATCH = FALSE then
      if not PROCESS_BLOCK_FRAMING_STATE and
          SYNCH_DATA_MODE:=BLOCK_BY_BLOCK then
        DATA_STATE:=FALSE;
        exit;
      end if;
      exit;
    end if;
    if START_OF_CONTROL_SEQUENCE = TRUE then
      PROCESS_SYNCH_CONTROL_SEQ;
      exit;
    end if;
    case CHAR_COUNT is
      when 1 =>
        PROCESS_FIRST_SYNCH_CHAR;
      when 2 =>
        PROCESS_SECOND_SYNCH_CHAR;
      when 3 .... 82 =>
        PROCESS_SYNCH_TEXT;
      when 83 or END_OF_MEDIUM_SEQUENCE:=TRUE =>
        PROCESS_BLOCK_FRAMING;
      when 84 =>
        PROCESS_BLOCK_PARITY;
    end case;
  else
    if NO CHARACTERS RECEIVED for LAST 30 MINUTES then
      GEN_SVC_MSG;
      -- SVC_MESSAGE_TYPE:=TRAFFIC_CHECK;
      -- IMPLIES RECEIVER IS EITHER OUT OF SYNC OR
      -- RECEIVING ONLY SYNC CHARACTERS.
    end if;
  end if;
end ANALYZE_BLOCK_FRAMING;

EOT..
```

```
-- A3152 PROCESS_FIRST_SYNCH_CHAR
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING THE FIRST CHARACTER OF A SYNCHRONOUS INPUT
-- TRUNK TO THE MESSAGE SWITCH AS DESCRIBED IN THE FOLLOWING:
--     MANUAL     PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5
-- 01/17/82 HF

procedure PROCESS_FIRST_SYNCH_CHAR is
begin
  if CHARACTER_PARITY = ODD then
    exit;
  end if;
  if CONTROL_CHARACTER = [REQUEST_FOR_ANSWER_CHAR |
      RECEIVE_CONTROL_CHAR | INVALID_CHAR] then
    START_OF_CONTROL_SEQUENCE:=TRUE;
    -- THE CONTROL SEQUENCE IS EFFECTED IF THE NEXT CONSECUTIVE
    -- CHARACTER IS A REPEAT OF THIS CONTROL CHARACTER.
    --> ACCEPT CHARACTER
    exit;
  end if;
  if SYNCH_DATA_MODE = BLOCK_BY_BLOCK then
    VERIFY THAT WE HAVE ANSWERED for THE LAST BLOCK RECEIVED;
    if WE HAVE not ANSWERED then
      exit;
    end if;
  else
    if SYNCH_DATA_MODE = CONTINUOUS then
      if FIRST CHARACTER AFTER BLOCK PARITY of BLOCK in ERROR
          then
        exit;
      end if;
    end if;
  end if;
  if CONTROL_CHARACTER = START_OF_HEADER then
    if ETX_BP or CANCEL not LAST PROCESSED then
      exit;
    end if;
  elsif CONTROL_CHARACTER = START_OF_TEXT then
    if ETB_BP not LAST CHARACTER PROCESSED then
      exit;
    end if;
  else
    exit;
  end if;
  --> ACCEPT FIRST_CHARACTER
  UPDATE_JOURNAL;
    -- USE SOM_IN ENTRY.
  CHAR_COUNT:=CHAR_COUNT+1;
end PROCESS_FIRST_SYNCH_CHAR;

EOT..
```

```
-- A3153 PROCESS_SECOND_SYNCH_CHAR
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING THE SECOND CHARACTER OF A SYNCHRONOUS INPUT
-- TRUNK TO THE MESSAGE SWITCH AS DESCRIBED IN THE FOLLOWING:
--    MANUAL      PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5  .
-- 01/17/82 HF

procedure PROCESS_SECOND_SYNCH_CHAR is
begin
  if CHARACTER_PARITY = ODD then
    CHAR_COUNT:=1;
    exit;
  end if;
  if FIRST_CHARACTER = START_OF_HEADER_CHAR then
    if CONTROL_CHARACTER = SELECT_CHAR then
      --> ACCEPT SECOND_CHARACTER
    else
      GENERATE_CONTROL_CHARS;
      -- USE SEND_RM CODE
    end if;
  else
    if FIRST_CHARACTER = START_OF_TEXT_CHAR then
      if AUTODIN_INTERSWITCH_TRUNK then
        if VALID_SECURITY_CHARACTER then
          --> ACCEPT SECOND_CHARACTER
        else
          GENERATE_CONTROL_CHARS;
          -- USE SEND_RM CODE
        end if;
      elsif DELETE_CHAR then
        --> ACCEPT SECOND_CHARACTER
      else
        GENERATE_CONTROL_CHARS;
        -- USE SEND_RM CODE
      end if;
    end if;
  end if;
  if SECOND_CHARACTER ACCEPTED then
    CHAR_COUNT:=CHAR_COUNT+1;
    BP_CALC:=SECOND_CHARACTER;
  end if;
end PROCESS_SECOND_SYNCH_CHAR;

EOT..
```

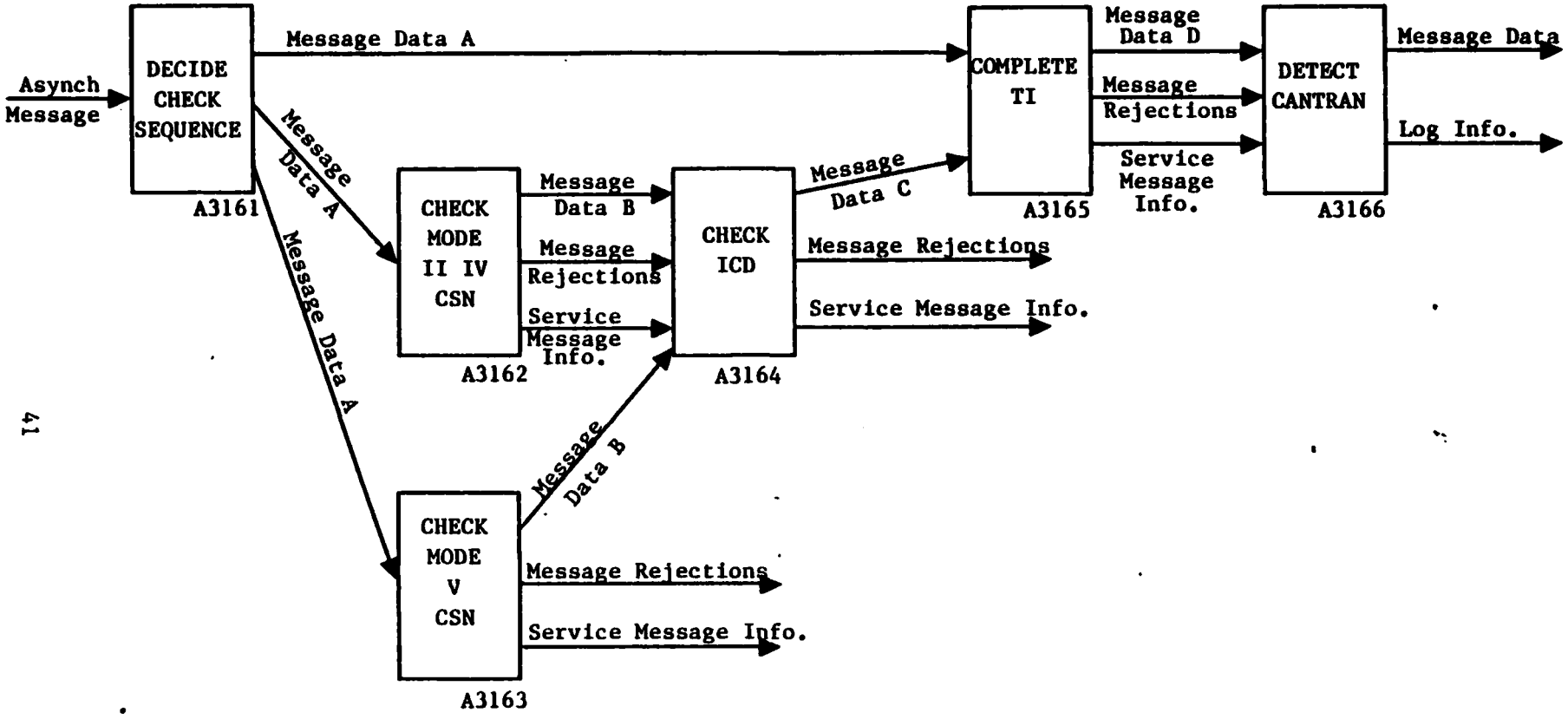
```
-- A3154 PROCESS_SYNCH_TEXT
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING THE TEXT CHARACTERS OF A SYNCHRONOUS INPUT
-- TRUNK TO THE MESSAGE SWITCH AS DESCRIBED IN THE FOLLOWING:
--     MANUAL     PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5
-- 01/18/82 HF

procedure PROCESS_SYNCH_TEXT is
begin
  if CHARACTER_PARITY = EVEN then
    if CONTROL_CHARACTER = MODE_CHANGE_CHAR or
        END_OF_MEDIUM_CHAR then
      --> ACCEPT CHARACTER
      CHAR_COUNT:=CHAR_COUNT+1;
      BLOCK_PARITY_CALC:=BLOCK_PARITY_CALC xor CONTROL_CHAR;
      if CONTROL_CHARACTER = END_OF_MEDIUM then
        END_OF_MEDIUM_SEQ:=TRUE;
      end if;
    elsif CONTROL_CHARACTER = [RECEIVE_CONTROL_CHAR |
        INVALID_CHAR] then
      START_OF_CONTROL_SEQ:=TRUE;
      -- THE CONTROL SEQUENCE IS EFFECTED IF THE NEXT
      -- CONSECUTIVE CHARACTER IS IDENTICAL TO THIS ONE.
      --> ACCEPT CHARACTER
    else
      CHAR_COUNT:=1;
      GENERATE_CONTROL_CHARS;
      -- USE SEND_NAK CODE (TO BE OUTPUT ONLY AFTER BLOCK
      -- FRAMING, REPLY, OR CANCEL)
    end if;
  elsif TEXT_CHARACTER then
    --> ACCEPT CHARACTER
    CHAR_COUNT:=CHAR_COUNT+1;
    BP_CALC:=BP_CALC xor TEXT_CHARACTER;
  end if;
end PROCESS_SYNCH_TEXT;

EOT..
```

```
      -- A3155 PROCESS_BLOCK_FRAMING
      -- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
      -- FOR PROCESSING THE BLOCK FRAMING CHARACTER POSITION OF A
      -- SYNCHRONOUS INPUT TRUNK TO THE MESSAGE SWITCH
      -- AS DESCRIBED IN THE FOLLOWING:
      --      MANUAL     PARAGRAPH(S)
      -- DCAC 370-D175-1 CHAPTER 5
      -- 01/19/82 HF

      procedure PROCESS_BLOCK_FRAMING is
      begin
        if CHARACTER_PARITY = EVEN then
          if CONTROL_CHARACTER = ETX_CHAR or ETB_CHAR then
            if PREVIOUSLY RECEIVED BLOCK HAS BEEN ACKNOWLEDGED then
              --> ACCEPT CHARACTER
              BP_CALC:=BP_CALC xor CONTROL_CHAR;
              if ETX_CHAR then
                END_OF_MESSAGE_SEQ:=TRUE;
              end if;
            else
              CHAR_COUNT:=1;
              -- IMPLIES THAT TRANSMITTER IS NOT KEEPING UP.
              -- GOOD AREA TO APPLY AUTOMATIC ERROR DETECTION.
            end if;
          else
            CHAR_COUNT:=1;
            -- IMPLIES THAT CHARACTER WAS NOT ACCEPTED; WAIT FOR
            -- REPLY OR CANCEL.
          end if;
        else
          CHAR_COUNT:=1;
          -- IMPLIES THAT CHARACTER WAS NOT ACCEPTED; WAIT FOR REPLY
          -- OR CANCEL.
        end if;
        END_OF_MEDIUM_SEQ:=FALSE;
      end PROCESS_BLOCK_FRAMING;

EOT..
```

```
-- A3156 PROCESS_SYNCH_CONTROL_SEQ
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR PROCESSING TWO CONSECUTIVE CONTROL CHARACTERS RECEIVED
-- OVER A SYNCHRONOUS INPUT TRUNK TO THE MESSAGE SWITCH
-- AS DESCRIBED IN THE FOLLOWING:
--     MANUAL    PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5
-- 01/19/82 HF

procedure PROCESS_SYNCH_CONTROL_SEQ is
begin
  if CONTROL_CHARACTER /= PREVIOUS CONTROL_CHARACTER then
    if CHAR_COUNT > 1 then
      GENERATE_CONTROL_CHARS;
      -- USE NAK CODE
      CHAR_COUNT:=1;
    end if;
  elsif CONTROL_CHARACTER = INVALID_CHAR then
    GENERATE_CONTROL_CHARS;
    -- USE INVALID_RCVD CODE
  elsif CONTROL_CHARACTER = RECEIVE_CONTROL_CHAR then
    GENERATE_CONTROL_CHARS;
    -- USE CODE REQUIRED BY CONTROL CHAR, SUCH AS WBT_RCVD,
    -- NAK_RCVD, RM_RCVD, ACK_1_RCVD OR ACK_2_RCVD.
  elsif PROCESSING_SYNCH_TEXT_STATE then
    GENERATE_CONTROL_CHARS;
    -- USE INVALID_CHAR CODE
    CHAR_COUNT:=1;
    -- CHANGES STATE TO PROCESS_FIRST_SYNCH_CHAR
  elsif CONTROL_CHARACTER = REQUEST_FOR_ANSWER_CHAR then
    GENERATE_CONTROL_CHARS;
    -- USE CODE AS REQUIRED BY CONTROL CHARACTER, SUCH AS
    -- [REPLY_RECEIVED | CANCEL RECEIVED | ENQUIRY RECEIVED]
    -- AND REPORT THAT AN ANSWER IS REQUIRED.
    if CONTROL_CHARACTER = CANCEL_CHAR then
      UPDATE_JOURNAL;
      -- USE CANCEL_RECEIVED CODE
    end if;
  end if;
  START_OF_CONTROL_SEQUENCE:=FALSE;
end PROCESS_SYNCH_CONTROL_SEQ;

EOT..
```

```
-- A3157 PROCESS_BLOCK_PARITY
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- TO CHECK THE BLOCK PARITY CHARACTER OF A SYNCHRONOUS
-- MESSAGE AND TAKE APPROPRIATE ACTION BASED ON THE RESULT
-- AS DESCRIBED IN THE FOLLOWING:
--      MANUAL      PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 5
-- 01/21/82 HF

procedure PROCESS_BLOCK_PARITY is
begin
  if BP_CALC = CONTROL_CHAR then
    if END_OF_MESSAGE_SEQ = TRUE then
      --> ACCEPT CHARACTER
      --> ACCEPT MESSAGE
      -- COMMENT FOR ACCEPT BLOCK ALSO APPLIES HERE.
      END_OF_MESSAGE_SEQ:=FALSE;
    else
      --> ACCEPT CHARACTER
      --> ACCEPT BLOCK
      -- CAN BE OVERRIDDEN BY MESSAGE VALIDATION ROUTINES
      -- AND REFERENCE STORAGE. TIME CONSTRAINTS EXIST FOR
      -- A REPLY, AS WELL AS SYNCHRONIZATION OF THE PROCESSES.
    end if;
    CHAR_COUNT:=1;
    GENERATE_CONTROL_CHARS;
    -- USE EOMS_RCVD CODE
    UPDATE_JOURNAL;
    -- USE EOM_IN CODE
  else
    CHAR_COUNT:=1;
    GENERATE_CONTROL_CHARS;
    -- USE SEND_RM CODE
    -- WAIT FOR REPLY OR CANCEL.
  end if;
end PROCESS_BLOCK_PARITY;
```

EOT..



A316 PREPROCESS ASYNCH MESSAGE

```
      --A3161 DECIDE CHECK SEQUENCE
      --REV BAAAA
      --2/2/82 PD

      procedure DECIDE_CHECK_SEQUENCE is
      begin
        if PRECEDENCE = [W|Y|Z] then
          if CHANNEL_MODE = [2|4] then
            ECSN := ECSN + 1;
          end if;
          COMPLETE_TI;
        elsif CHANNEL_MODE = [2|4] then
          CHECK_MODE_II_IV_CSN;
        elsif CHANNEL_MODE = 5 then
          CHECK_MODE_V_CSN;
        end if;
      end DECIDE_CHECK_SEQUENCE;

EOT..
```

```
        --A3162 CHECK MODE II IV CSN
        --PARA 3.2.1.2.10.2.10 B (2)
        --REV BAAAA
        --2/3/82 PD

        procedure CHECK_MODE_II_IV_CSN is
        begin
          if ICSN(1..3) not in digits then
            GENERATE_CONTROL_CHARATER(REJECT_MESSAGE,MESSAGE_ID);
            GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
                (INCORRECT_CSN,MESSAGE_ID));
            ECSN := ECSN + 1;
          elsif ICSN /= ECSN then
            GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
            GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                (INCORRECT_CSN,MESSAGE_ID));
            ECSN := ICSN + 1;
          else
            if LAST_GOOD_CSN + 1 /= ICSN then
              GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                  (OPEN_CSN,MESSAGE_ID,, LAST_GOOD_CSN,ICSN));
            end if;
            LAST_GOOD_CSN := ICSN;
            ECSN := ICSN + 1;
          end if;
        end CHECK_MODE_II_IV_CSN;

    EOT..
```

```
      --A3163 CHECK MODE V CSN
      --PARA 3.2.1.2.10.2.10 B (2)
      --REV BAAAA
      --2/2/82 PD

      procedure CHECK_MODE_V_CSN is
      begin
        if ICSN(1..3) not in digits then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE, MESSAGE_ID);
          GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
              (INCORRECT_CSN, MESSAGE_ID));
          LAST_CSN_REJ := FALSE;
        elsif ICSN /= ECSN and ((LAST_CSN_REJ and ICSN /= IRCSN) or
            not LAST_CSN_REJ) then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
          GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
              (INCORRECT_CSN,MESSAGE_ID));
          LAST_CSN_REJ := TRUE;
          IRCSN := ICSN;
        else
          if LAST_GOOD_CSN +1 /= ICSN then
            GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                (OPEN_CSN,MESSAGE_ID,, LAST_GOOD_CSN,ICSN));
          end if;
          LAST_CSN_REJ := FALSE;
          LAST_GOOD_CSN := ICSN;
        end if;
      end CHECK_MODE_V_CSN;

EOT..
```

```
      --A3164 CHECK ICD
      --PARA 3.2.1.2.10.2.10 A
      --REV BAAAA
      --2/3/82 PD

      procedure CHECK_ICD is
      begin
        if ICD /= CHNL_DES then
          GENERATE_CONTROL_CHARACTER( REJECT_MESSAGE,MESSAGE_ID);
          GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
              INVALID_HEADER_REJ,MESSAGE_ID));
        end if;
      end CHECK_ICD;

EOT..
```

```
      --A3165 COMPLETE TI
      --PARA 3.2.1.2.10.2.10 C
      --REV BAAAA
      --2/2/82 PD

      procedure COMPLETE_TI is
      begin
        --> TI LINE MUST NOT EXCEED 79 CHARACTERS
        --> STARTING WITH ICD AND ENDING WITH FIRST LF
        --> TI LINE MUST END IN LOWER CASE IF ITA
        -->MINIMUM END OF LINE IS ONE LF
        if ANY of ABOVE not MET then
          GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
              (INVALID_HEADER_REJ,MESSAGE_ID) );
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
          --> TERMINATE PROCESSING ON THIS MESSAGE
        end if;
      end COMPLETE_TI;

EOT..
```

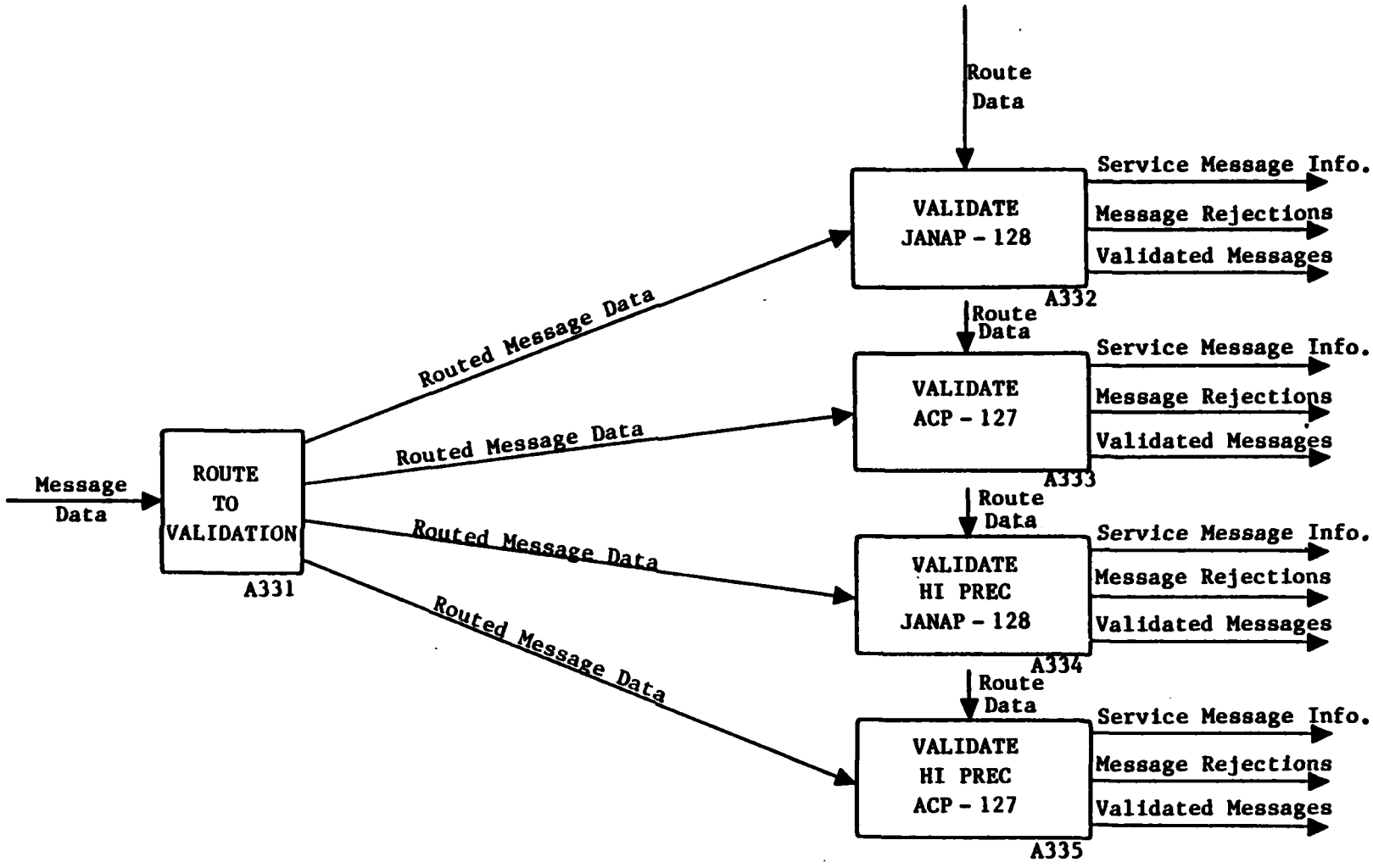
```
      --A3166 DETECT_CANTRAN
      --PARA 3.2.1.2.12
      --REV BAAAA
      --2/2/82 PD

      procedure DETECT_CANTRAN is
      begin
        if ACP_127 or (JANAP_128 and FIRST_LMF_CHARACTER /= [S|C|B|D
            |I] then
          START := INDEX of EOM SEQUENCE;    --LF NNNN
          FINI := START - 23;
          for I in reverse FINI..START loop
            exit when MESSAGE(I) = '#';
            if MESSAGE(I..I+3) = "E E AR" then
              --> MAKE CANTRAN_REC LOG ENTRY
              --> DISCARD MESSAGE
              --> NOTIFY OPERATOR
              --> TERMINATE PROCESSING ON THIS MESSAGE
              exit;
            end if;
          end loop;
        end if;
      end DETECT_CANTRAN;

EOT..
```

```
-- A32 GENERATE_ASYNCH_CTL_CHARS
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR INTERFACING MESSAGE CONTROL SEQUENCES FROM THE RECEIVERS,
-- VALIDATOR, AND REFERENCE STORAGE TO THE TRANSMITTER AS
-- DESCRIBED IN THE FOLLOWING:
-- MANUAL PARAGRAPH(S)
-- DCAC 370-D175-1 CHAPTER 11
-- 02/04/82 HF

procedure GENERATE_ASYNCH_CTL_CHARS is
begin
  if REPLY_TIMER (2-4 SECONDS) = EXPIRED then
    if MESSAGE_STATUS = CANCEL_SEQ then
      CONTROL_CODE:=CANCEL_SENT;
    elsif MESSAGE_STATUS:=REPLY_SEQ then
      CONTROL_CODE:=REPLY_SENT;
    else
      raise exception;
    end if;
  end if;
  if STOP_TIMER (2-8 SECONDS) = EXPIRED then
    CONTROL_CODE:=SEND_STOP;
  end if;
  case CONTROL_CODE is
    when SEND_RT =>
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- TRANSMIT STOP_CODE
      START_STOP_TIMER;
      -- EXPECT REPLY SEQUENCE
      MESSAGE_STATUS:=RT;
    when SEND_STOP =>
      if CHAR_BUFFER < LOWER_THRESHOLD then
        if MESSAGE_STATUS = RT then
          SEND_ACKNOWLEDGEMENT_RESPONSE;
          -- TRANSMIT RT_CODE
          MESSAGE_STATUS:=NOT_STOP_SEQ;
        elsif MESSAGE_STATUS = OK then
          SEND_ACKNOWLEDGEMENT_RESPONSE;
          -- TRANSMIT ACK_SEQ_REPLY (ACK_1 OR ACK_2)
          -- ALTERNATE ACK_SEQ_REPLY
        elsif MESSAGE_STATUS = INCOMPLETE then
          SEND_ACKNOWLEDGEMENT_RESPONSE;
          -- TRANSMIT STOP_CODE
          MESSAGE_STATUS:=STOP_SEQ;
          --> CHECK FOR THIRD STOP TRANSMITTED AND NOTIFY
              -- OPERATOR IF SO.
        else
          raise exception;
        end if;
      else
        SEND_ACKNOWLEDGEMENT_RESPONSE;
        -- TRANSMIT STOP_CODE
        MESSAGE_STATUS:=STOP_SEQ;
        if THIRD_STOP_TRANSMITTED then
          --> NOTIFY OPERATOR
```

```
    end if;
  end if;
when ACK_SEND =>
  SEND_ACKNOWLEDGEMENT_RESPONSE;
  -- TRANSMIT ACK_SEQ_REPLY (TWO ACK_1 OR ACK_2 CHARACTERS
      -- ALTERNATING
  -- BETWEEN MESSAGE TRANSMISSIONS.
  ACK_SEQ_REPLY:=ALTERNATE of ACK_1 or ACK_2;
when CANCEL_RCVD =>
  ACK_SEQ_XPTD:=ACK_2;
  SEND_ACKNOWLEDGEMENT_RESPONSE;
  -- STOP MESSAGE TRANSMISSION (IF ANY)
  RESET_STOP_TIMER (if SET);
  --> SEND PARTIALLY RECEIVED MESSAGE TO REFERENCE STORAGE
  --> AND CLEAN UP FOR NEXT MESSAGE.
when REPLY_RCVD =>
  if MESSAGE_STATUS = STOPPED then
    SEND_ACKNOWLEDGEMENT_RESPONSE;
    -- TRANSMIT STOP_CODE
    if THIRD STOP TRANSMITTED then
      --> NOTIFY OPERATOR
    elsif MESSAGE_STATUS = RT then
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- TRANSMIT RT_CODE
    elsif MESSAGE_STATUS = OK then
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- TRANSMIT ACK_SEQ_REPLY
      ACK_SEQ_REPLY:=ALTERNATE of ACK_SEQ_REPLY;
    end if;
  end if;
when RT_RCVD =>
  SEND_ACKNOWLEDGEMENT_RESPONSE;
  -- STOP MESSAGE TRANSMISSION
  ACK_SEQ_XPTD:=ACK_2;
  --> PREPARE MESSAGE FOR RETRANSMISSION AND WAIT
  MESSAGE_STATUS:=EOMS_NOT_OUTSTANDING and NOT_REPLY_SEQ;
when STOP_RCVD =>
  if MESSAGE_STATUS = CANCEL_SEQ then
    CONTROL_CODE:=CANCEL_SENT;
  end if;
  SEND_ACKNOWLEDGEMENT_RESPONSE;
  -- STOP MESSAGE TRANSMISSION
  START STOP_TIMER;
when EOMS_CODE_SENT =>
  START REPLY_TIMER;
  REPEAT_COUNTER:=REPEAT_COUNTER+1;
  MESSAGE_STATUS:=EOMS_OUTSTANDING;
when EOMS_CODE_RCVD =>
  if CHARACTER_BUFFER > UPPER THRESHOLD then
    SEND_ACKNOWLEDGEMENT_RESPONSE;
    -- TRANSMIT STOP_CODE
    STOP_SEQ:=TRUE;
  else
    if MESSAGE_STATUS = COMPLETE then
      if MESSAGE_STATUS = RT then
```

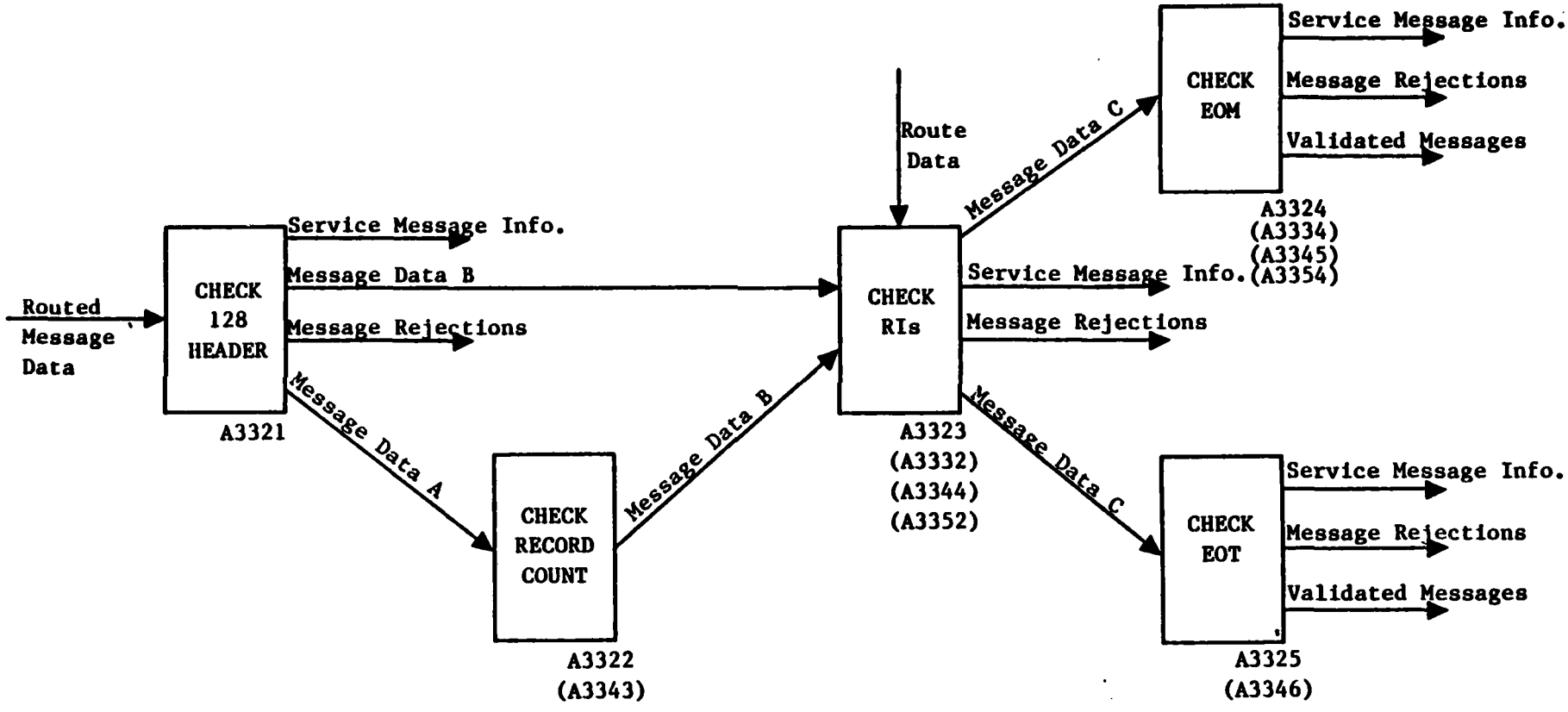
```
          SEND_ACKNOWLEDGEMENT_RESPONSE;
          -- TRANSMIT RT_CODE
        elsif MESSAGE_STATUS = OK then
          SEND_ACKNOWLEDGEMENT_RESPONSE;
          -- TRANSMIT ACK_SEQ_REPLY (ACK_1 OR ACK_2)
        else
          raise exception;
        end if;
        MESSAGE_STATUS:=EOMS_RCVD;
      end if;
    end if;
  when REPLY_SENT =>
    START REPLY_TIMER;
    REPEAT_COUNTER:=REPEAT_COUNTER+1;
    SEND_ACKNOWLEDGEMENT_RESPONSE;
    -- TRANSMIT REPLY_CODE
    MESSAGE_STATUS:=REPLY_SEQ;
  when ACK_1_RCVD =>
    REPEAT_COUNTER:=0;
    RESET REPLY_TIMER;
    if ACK_SEQ_XPTD:= ACK_1_RCVD then
      --> CONTINUE MESSAGE TRANSMISSION
    else
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- CANCEL MESSAGE
      --> PREPARE TO RETRANSMIT
    end if;
    ACK_SEQ_XPTD:=ACK_2;
    MESSAGE_STATUS:=EOMS_NOT_OUTSTANDING and NOT_REPLY_SEQ;
  when ACK_2_RCVD =>
    REPEAT_COUNTER:=0;
    RESET REPLY_TIMER;
    if ACK_SEQ_XPTD = ACK_2_RCVD then
      --> CONTINUE MESSAGE TRANSMISSION
    else
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- CANCEL MESSAGE
      --> PREPARE TO RETRANSMIT
    end if;
    ACK_SEQ_XPTD:=ACK_1;
    MESSAGE_STATUS:=EOMS_NOT_OUTSTANDING and NOT_CANCEL_SEQ
        and NOT_REPLY_SEQ;
  when CANCEL_SENT =>
    if MESSAGE_STATUS = EOMS_NOT_OUTSTANDING then
      MESSAGE_STATUS:=CANCEL_SEQ;
      ACK_SEQ_REPLY:=ACK_2;
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- TRANSMIT CANCEL_CODE
      if REPEAT_COUNTER = 3 then
        --> SET OPERATOR ALARM
      end if;
    else
      MESSAGE_STATUS:=REPLY_SEQ;
      SEND_ACKNOWLEDGEMENT_RESPONSE;
      -- TRANSMIT REPLY_CODE
```

```
            end if;
            START_REPLY_TIMER;
            REPEAT_COUNTER:=REPEAT_COUNTER + 1;
        end case;
    end GENERATE_ASYNCH_CTL_CHARS;

EOT..
```



A33 VALIDATE MESSAGE

```
        -- A331 ROUTE TO VALIDATION
        -- PARA 3.2.1.2.4.2.2.2
        -- REV BAAA
        -- 12/16/81 PD

        procedure ROUTE_TO_VALIDATION is
        begin
          if JANAP_128 then
            if PRECEDENCE = [W|Y|Z] then
              VALIDATE_HIPREC_JANAP_128;
            else
              VALIDATE_JANAP_128;
            end if;
          else  -- ACP-127
            if DOUBLE_PRECEDENCE_PROSIGN(1) = [Y|Z] or
                DOUBLE_PRECEDENCE_PROSIGN(2) = [Y|Z] or
                A (GARBLED) BELL SIGNAL IS PRESENT then
              VALIDATE_HIPREC_ACP_127;
            else
              VALIDATE_ACP_127;
            end if;
          end if;
        end ROUTE_TO_VALIDATION;

EOT..
```



A332 VALIDATE JANAP 128

```
--A3321 CHECK 128 HEADER
--PARA 3.2.1.2.10.1, 3.2.1.2.10.2, 3.2.1.2.8.4
--REV BAAAB
--2/3/82 PD

procedure CHECK_128_HEADER is
begin
  if PRECEDENCE /= [O|P|R] then
    raise INVALID_HEADER_REJ;
  end if;
  if LMF_PAIR not LEGAL then
    --LEGAL PAIRS ARE LISTED IN THE DATA DICTIONARY
    raise INVALID_HEADER_REJ;
  end if;
  if CLASS /= [M|A|T|S|C|R|E|U] then
    raise INVALID_SCTY_FIELD;
  end if;
  if CLASS >= LINE_DATA.SECURITY_PROSIGN then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (SCTY_MISMATCH,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
  end if;
  if not SINGLE_CARD then
    if REDUNDANT_CLASS(1..4) /= CLASS then
      raise INVALID_SCTY_FIELD;
    end if;
  end if;
  if CIC_CAI(1..3) not in LETTER or CIC_CAI(4) not in [LETTER
      |DIGIT] then
    raise INVALID_HEADER_REJ;
  end if;
  if OSRI(1..7) not in LETTER then
    raise INVALID_HEADER_REJ;
  end if;
  if OSSN(1..4) not in DIGIT then
    raise INVALID_HEADER_REJ;
  end if;
  if DATE_TIME (1..7) not in DIGIT then
    raise INVALID_HEADER_REJ;
  end if;
  --> CHECK SENTINELS AND SIGNALS
  --> BLANKS IN POS 9 & 21
  --> '-' IN POS 28 (33 IF RECORD_COUNT PRESENT)
  --> '--' IN POS 33 (38 IF RECORD_COUNT PRESENT)
  --> IF ANY ARE BAD RAISE INVALID_HEADER_REJ;
exception
  when INVALID_HEADER_REJ =>
    CALL GENERATE_SERVICE_MESSAGE( SERVICE_MESSAGE_INFO =>
        (INVALID_HEADER_REJ, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      CALL GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID
          );
    end if;
```

```
        when INVALID_SCTY_FIELD =>
          CALL GENERATE_SERVICE_MESSAGED(SERVICE_MESSAGE_INFO =>
              (INVALID_SCTY_FIELD, MESSAGE_ID));
          --> TERMINATE PROCESSING ON THIS MESSAGE
          if CHANNEL_MODE = [1|3] then
            CALL GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID
                );
          end if;
      end CHECK_128_HEADER;

EOT..
```

```
--A3322 A3343 CHECK_RECORD_COUNT
--PARA 3.2.1.2.10.1.7
--REV BAAAA
--1/13/82 PD

procedure CHECK_RECORD_COUNT is
begin
  if RECORD_COUNT /= ["MTMS"|"PLTS"] then
    if RECORD_COUNT(1) in LETTERS then
      if RECORD_COUNT /= VALID_RI then
        raise INVALID_HEADER_REJ;
      end if;
    elsif RECORD_COUNT(1) in digits then
      if RECORD_COUNT(2..4) not in digits then
        raise INVALID_HEADER_REJ;
      end if;
      if RECORD_COUNT <3 or RECORD_COUNT>500 then
        raise INVALID_HEADER_REJ;
      end if;
    else
      raise INVALID_HEADER_REJ;
    end if;
  end if;
exception
  when INVALID_HEADER_REJ =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_HEADER_REJ, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      CALL GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
end CHECK_RECORD_COUNT;

EOT..
```

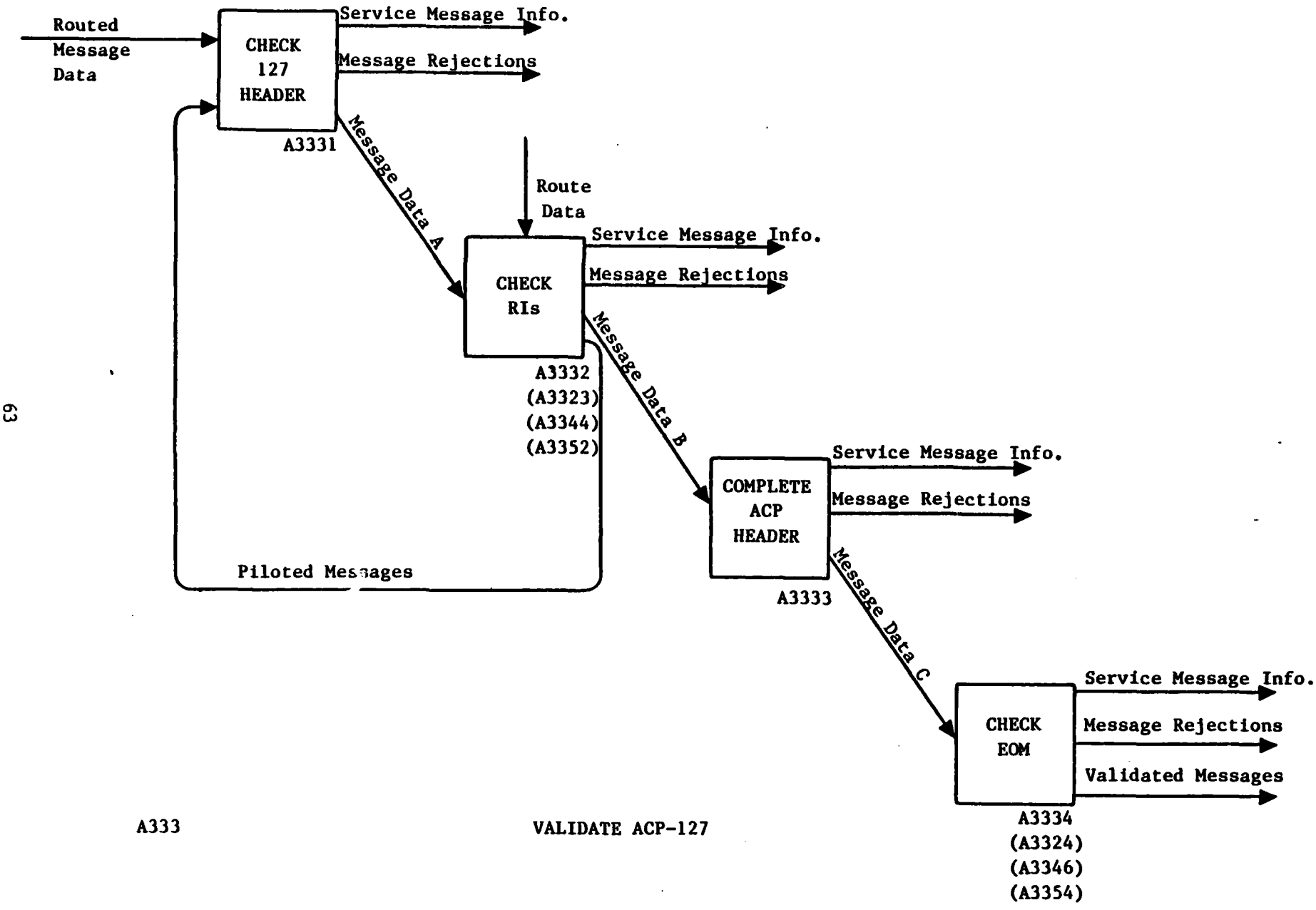
```
--A3323 A3332 A3344 A3352 CHECK_RIS
--PARA 3.2.1.2.10.8
--REV BAAAA
--1/6/82 PD

procedure CHECK_RIS is
begin
  loop
    --> FIND NEXT RI
    if A DELIMITER OTHER THAN [`B`|CR|LF] is FOUND then
      raise INVALID_RI_FIELD;
    end if;
    exit when END_OF_ROUTING is FOUND;
    -- END_OF_ROUTING IS '.' FOR JANAP-128
    -- AND CR CR LF Z OR CR CR LF D FOR ACP-127
    if RI(1..4) not in LETTER then
      raise INVALID_RI_FIELD;
    end if;
    if RI(1..4) = OUR_RI then
      --> CHECK ENTIRE RI FOR VALIDITY
    elsif RI(3..4) = "CR" then
      --> CHECK RI FOR VALID COLLECTIVE RI
    else
      --> CHECK RI(1..) FOR VALID RELAY RI
    end if;
    if RI GOOD then
      --> ADD RI TO VALID_RI_LIST
    else
      --> ADD RI TO INVALID_RI_LIST
    end if;
  end loop;
  if VALID_RI_LIST is EMPTY then
    raise ALL_RI_INVALID;
  elsif INVALID_RI_LIST not EMPTY then
    raise INVALID_RI;
  end if;
  if ACP-127 and END_OF_ROUTING = CR CR LF Z then
    --> RETURN TO CHECK_127_HEADER (THIS LINE WAS A PILOT)
  end if;
exception
  when INVALID_RI_FIELD =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_RI_FIELD,MESSAGE_ID));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when ALL_RI_INVALID =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (ALL_RI_INVALID,MESSAGE_ID,INVALID_RI_LIST));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when INVALID_RI =>
```

```
            CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                (INVALID_RI,MESSAGE_ID,INVALID_RI_LIST));
            if PRECEDENCE = ['W'|'Y'|'Z'] then
              CIC_CAI(4) := 'W';
            end if;
        end CHECK_RIS;

EOT..
```

```
--A3324 A3334 A3345 A3354 CHECK EOM
--PARA 3.2.1.2.4.2.1.16, 3.2.1.2.10.2.9, 3.2.1.2.10.2.10 B(2)
--REV BAAAA
--1/7/82 PD

procedure CHECK_EOM is
begin
  if OSSN /= EOM_VALIDATION(2..5) then
    -- USE ACP_SSN FOR ACP-127
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (SUSPECTED_STRAGGLER, MESSAGE_ID));
    if PRECEDENCE = [W|Y|Z] then
      CIC_CAI(4) := 'W';
    else
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  elsif EOM_SEQ DOES not CONTAIN LF & "NNNN" then
    CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
  if PRECEDENCE = ['W'|'Y'|'Z'] then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (HIGH_PRECEDENCE_ACCEPTANCE,MESSAGE_ID));
  end if;
  if CHANNEL_MODE = 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOM;

EOT..
```

```
--A3325 A3346 CHECK EOT
--PARA 3.2.1.2.4.2.1.16, 3.2.1.2.10.1.10, 3.2.1.2.10.2.10 B (2)
--REV BAAAA
--1/13/82 PD

procedure CHECK_EOT is
begin
  if not SINGLE_CARD then
    if EOT_CARD.OSSN /= ACP_SSN then
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
          (SUSPECTED_STRAGGLER, MESSAGE_ID));
      if PRECEDENCE = [X|Y|Z] then
        CIC_CAI(4) := 'W';
      else
        --> TERMINATE PROCESSING ON THIS MESSAGE
        if CHANNEL_MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID)
              ;
        end if;
      end if;
    end if;
    if JANAP_FMT_LN_2.RECORD_COUNT = "PLTS" then
      if EOT_CARD.RECORD_COUNT < 3 or EOT_CARD.RECORD_COUNT >
          500 then
        CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
            (INVALID_EOM_REJ, MESSAGE_ID));
        --> TERMINATE PROCESSING ON THIS MESSAGE;
        if CHANNEL_MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID)
              ;
        end if;
      end if;
    elsif JANAP_FMT_LN_2.RECORD_COUNT = "MTMS" then
      if EOT_CARD.RECORD_COUNT /= "MTMS" and
          EOT_CARD.RECORD_COUNT /= ACTUAL_RECORD_COUNT then
        -- THIS IMPLIES YOU MUST COUNT THE ACTUAL NUMBER OF
        -- RECORDS
        CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
            (INVALID_EOM_REJ, MESSAGE_ID));
        --> TERMINATE PROCESSING ON THIS MESSAGE
        if CHANNEL_MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID)
              ;
        end if;
      end if;
    else
      if EOT_CARD.RECORD_COUNT /= ACTUAL_RECORD_COUNT then
        CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
            (INVALID_EOM_REJ, MESSAGE_ID));
        --> TERMINATE PROCESSING ON THIS MESSAGE
        if CHANNEL_MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID)
              ;
        end if;
      end if;
```

```
        end if;
      else
        if SINGLE_CARD(80) /= 'N' then
          CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
              (INVALID_EOM_REJ, MESSAGE_ID));
          --> TERMINATE PROCESSING ON THIS MESSAGE
          if CHANNEL_MODE = [1|3] then
            GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
          end if;
        end if;
      end if;
      if PRECEDENCE = ['W'|'Y'|'Z'] then
        CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
            (HIGH_PRECEDENCE_ACCEPTANCE,MESSAGE_ID));
      end if;
      if CHANNEL_MODE := 5 then
        ECSN := ECSN + 1;
      end if;
    end CHECK_EOT;

EOT..
```



A333 VALIDATE ACP-127

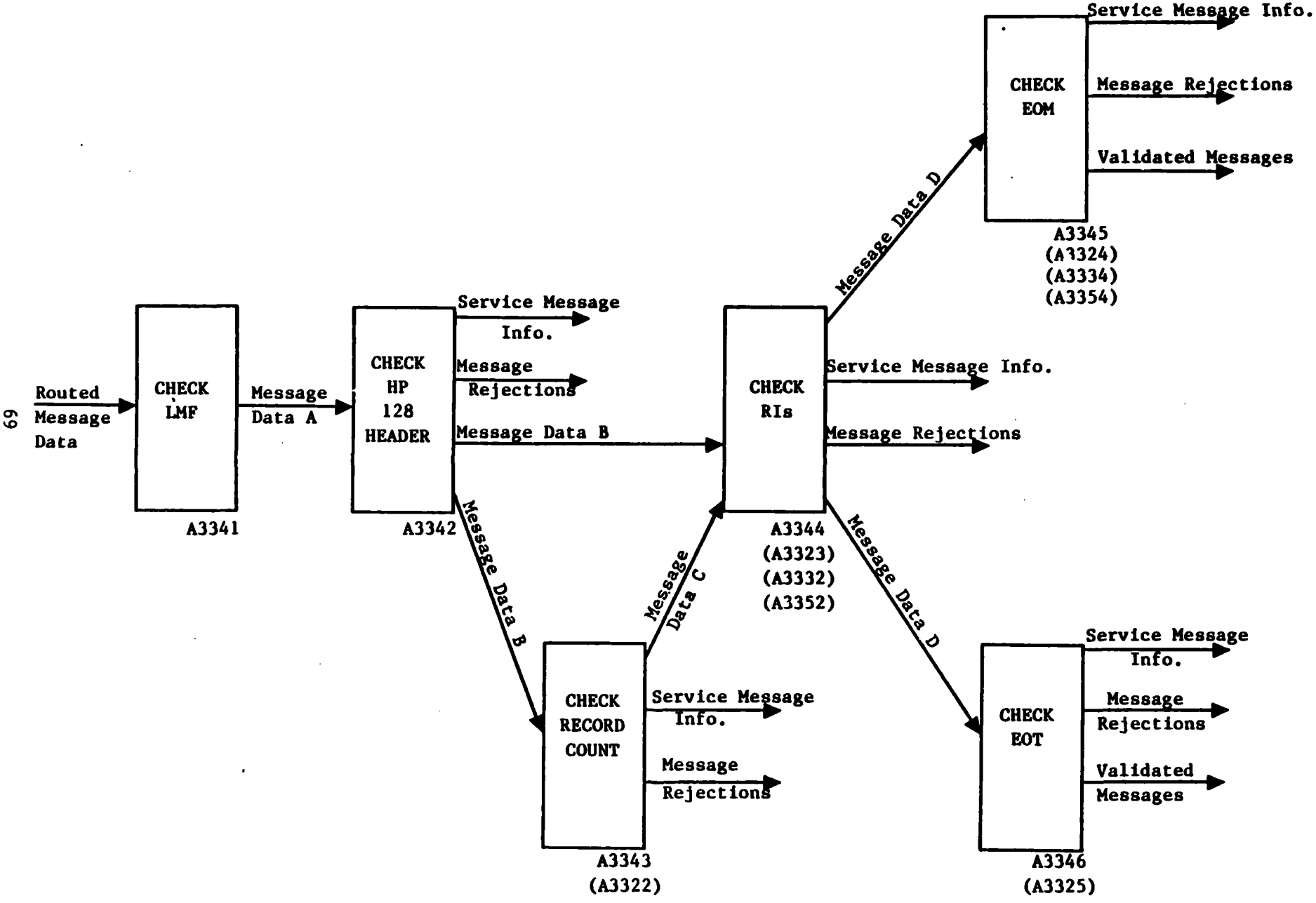
```
      -- A3331 CHECK 127 HEADER
      -- PARA 3.2.1.2.10.3
      -- REV BAAAA
      -- 1/7/82 PD

      procedure CHECK_127_HEADER is
      begin
        if DOUBLE_PRECEDENCE_PROSIGN /= ["OO"|"PP" |"RR"] then
          DOUBLE_PRECEDENCE_PROSIGN := "OO";
        end if;
        if ACP_FMT_LN_2(3) /= 'b' then
          CALL GENERATED_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
              (INVALID_HEADER_REJ,MESSAGE_ID));
          --> TERMINATE PROCESSING FOR THIS MESSAGE
          if CHANNEL_MODE = [1|3] then
            GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
          end if;
        end if;
      end CHECK_127_HEADER;

EOT..
```

```
--A3323 A3332 A3344 A3352 CHECK_RIS
--PARA 3.2.1.2.10.8
--REV BAAAA
--1/6/82 PD

procedure CHECK_RIS is
begin
  loop
    --> FIND NEXT RI
    if A DELIMITER OTHER THAN [`B`|CR|LF] is FOUND then
      raise INVALID_RI_FIELD;
    end if;
    exit when END_OF_ROUTING is FOUND;
    -- END_OF_ROUTING IS '.' FOR JANAP-128
    -- AND CR CR LF Z OR CR CR LF D FOR ACP-127
    if RI(1..4) not in LETTER then
      raise INVALID_RI_FIELD;
    end if;
    if RI(1..4) = OUR_RI then
      --> CHECK ENTIRE RI FOR VALIDITY
    elsif RI(3..4) = "CR" then
      --> CHECK RI FOR VALID COLLECTIVE RI
    else
      --> CHECK RI(1..) FOR VALID RELAY RI
    end if;
    if RI GOOD then
      --> ADD RI TO VALID_RI_LIST
    else
      --> ADD RI TO INVALID_RI_LIST
    end if;
  end loop;
  if VALID_RI_LIST is EMPTY then
    raise ALL_RI_INVALID;
  elsif INVALID_RI_LIST not EMPTY then
    raise INVALID_RI;
  end if;
  if ACP-127 and END_OF_ROUTING = CR CR LF Z then
    --> RETURN TO CHECK_127_HEADER (THIS LINE WAS A PILOT)
  end if;
exception
  when INVALID_RI_FIELD =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_RI_FIELD,MESSAGE_ID));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when ALL_RI_INVALID =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (ALL_RI_INVALID,MESSAGE_ID,INVALID_RI_LIST));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when INVALID_RI =>
```

```
            CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                (INVALID_RI,MESSAGE_ID,INVALID_RI_LIST));
            if PRECEDENCE = ['W'|'Y'|'Z'] then
              CIC_CAI(4) := 'W';
            end if;
        end CHECK_RIS;

EOT..
```

```
-- A3333 COMPLETE 127 HEADER
-- PARA 3.2.1.2.10.3, 3.2.1.2.8.4
-- REV BAAAA
-- 1/13/82 PD

procedure COMPLETE_127_HEADER is
begin
  if ACP_SSN PRECEDED BY '#' then
    --> RETAIN FOR EOM CHECK
  end if;
  --FMT_LN_4
  if OP_SIGNAL /= ["ZNY"|"ZNR"] or CLASS_X5 /= ["MMMMM"|"AAAAA"
      |"TTTTT"|"SSSSS"| "CCCCC"|"RRRRR"|"EEEEE"|"UUUUU"] then
    CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_SCTY_FIELD,MESSAGE_ID));
    --> TERMINATE PROCESSING FOR THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
  if CLASS >= LINE_DATA.SECURITY_PROSIGN then
    GENERATE_CONTROL_CHARACTER (REJECT_MESSAGE,MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (SCTY_MISMATCH,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
  end if;
end COMPLETE_127_HEADER;

EOT..
```

```
--A3324 A3334 A3345 A3354 CHECK EOM
--PARA 3.2.1.2.4.2.1.16, 3.2.1.2.10.2.9, 3.2.1.2.10.2.10 B(2)
--REV BAAAA
--1/7/82 PD

procedure CHECK_EOM is
begin
  if OSSN /= EOM_VALIDATION(2..5) then
    -- USE ACP_SSN FOR ACP-127
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (SUSPECTED_STRAGGLER, MESSAGE_ID));
    if PRECEDENCE = [W|Y|Z] then
      CIC_CAI(4) := 'W';
    else
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  elsif EOM_SEQ DOES not CONTAIN LF & "NNNN" then
    CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
  if PRECEDENCE = ['W'|'Y'|'Z'] then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (HIGH_PRECEDENCE_ACCEPTANCE,MESSAGE_ID));
  end if;
  if CHANNEL_MODE = 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOM;

EOT..
```



A334 VALIDATE HI PREC JANAP-128

```
--A3341 CHECK LMF
--PARA 3.2.1.2.4.2.1.3
--REV BAAAA
--1/13/82 PD

procedure CHECK_LMF is
begin
  case FIRST_LMF_CHARACTER is
    when 'A' =>
      if SELECT_CHARACTER = ['H'|'S'] then
        LMF_OK := TRUE;
      else
        LMF_OK := FALSE;
      end if;
    when 'B'|'D'|'I' =>
      if SELECT_CHARACTER = ['B'|'C'] then
        LMF_OK := TRUE;
      else
        LMF_OK := FALSE;
      end if;
    when 'C'|'S' =>
      if SELECT_CHARACTER = ['D'|'F'] then
        LMF_OK := TRUE;
      else
        LMF_OK := FALSE;
      end if;
    when 'F'|'Q'|'R'|'T' =>
      if SELECT_CHARACTER = 'A' then
        LMF_OK := TRUE;
      else
        LMF_OK := FALSE;
      end if;
    when others =>
      LMF_OK := FALSE;
  end case;
  if not LMF_OK then
    case SELECT_CHARACTER is
      when A =>
        FIRST_LMF_CHARACTER := 'T';
      when 'B'|'C' =>
        FIRST_LMF_CHARACTER := 'B';
      when 'D'|'F' =>
        FIRST_LMF_CHARACTER := 'C';
      when 'H'|'S' =>
        FIRST_LMF_CHARACTER := 'A';
    end case;
    CIC_CAI(4) := 'W';
  end if;
  --> CHECK FOR LEGAL LMF PAIR (LEGAL PAIRS ARE LISTED IN THE
      -- DATA DICTIONARY)
  if not LEGAL_PAIR then
    if FIRST_LMF_CHARACTER = ['R'|'Q'|'F'] then
      SECOND_LMF_CHARACTER := 'T';
    elsif FIRST_LMF_CHARACTER = 'S' then
      SECOND_LMF_CHARACTER := 'C';
```

```
            else
              SECOND_LMF_CHARACTER := FIRST_LMF_CHARACTER;
            end if;
            CIC_CAI(4) := 'W';
          end if;
        end CHECK_LMF;

EOT..
```

```
--A3342 CHECK HP 128 HEADER
--PARA 3.2.1.2.4.2.1, 3.2.1.2.8.4
--REV BAAAA
--1/13/82 PD

procedure CHECK_HP_128_HEADER is
begin
  --> SECURITY FIELD 3 OUT OF 5 CHECK
  --> 3 OF THE 5 SECURITY CHARACTERS MUST BE CORRECT AND MATCH
  --> ELSE GENERATE INVALID SCTY FIELD SERVICE MESSAGE AND
  --> TERMINATE AND REJECT MESSAGE
  if CLASS (MATCHING 3) >= LINE_DATA.SECURITY_PROSIGN then
    GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE( SERVICE_MESSAGE_INFO =>
        (SCTY_MISMATCH,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
  end if;
  if OSRI(1..4) = OUR_RI then
    --> CHECK ENTIRE RI FOR VALID TRIBUTARY
    if not VALID then
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
          (INVALID_HEADER_REJ,MESSAGE_ID));
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  else
    --> CHECK OSRI(1..4) FOR VALID RELAY
    if not VALID then
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
          (INVALID_HEADER_REJ, MESSAGE_ID));
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  end if;
  --> CHECK OSSN FOR EMBEDDED SPACES OR HYPHENS
  if FOUND then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_HEADER_REJ, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
  --> CHECK SENTENALS & SIGNALS
  --> BLANKS IN POS 9 & 21
  --> '-' IN POS 28 (33 IF RECORD_COUNT PRESENT)
  --> '--' IN POS 33 (38 IF RECORD_COUNT PRESENT)
  --> GENERATE INVALID HEADER REJ IF ANY ARE BAD
  --> TERMINATE AND REJECT MESSAGE
end CHECK_HP_128_HEADER;

EOT..
```

```
        --A3322 A3343 CHECK_RECORD_COUNT
        --PARA 3.2.1.2.10.1.7
        --REV BAAAA
        --1/13/82 PD

        procedure CHECK_RECORD_COUNT is
        begin
          if RECORD_COUNT /= ["MTMS"|"PLTS"] then
            if RECORD_COUNT(1) in LETTERS then
              if RECORD_COUNT /= VALID_RI then
                raise INVALID_HEADER_REJ;
              end if;
            elsif RECORD_COUNT(1) in digits then
              if RECORD_COUNT(2..4) not in digits then
                raise INVALID_HEADER_REJ;
              end if;
              if RECORD_COUNT <3 or RECORD_COUNT>500 then
                raise INVALID_HEADER_REJ;
              end if;
            else
              raise INVALID_HEADER_REJ;
            end if;
          end if;
        exception
          when INVALID_HEADER_REJ =>
            CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                (INVALID_HEADER_REJ, MESSAGE_ID));
            --> TERMINATE PROCESSING ON THIS MESSAGE
            if CHANNEL_MODE = [1|3] then
              CALL GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
            end if;
        end CHECK_RECORD_COUNT;

EOT..
```

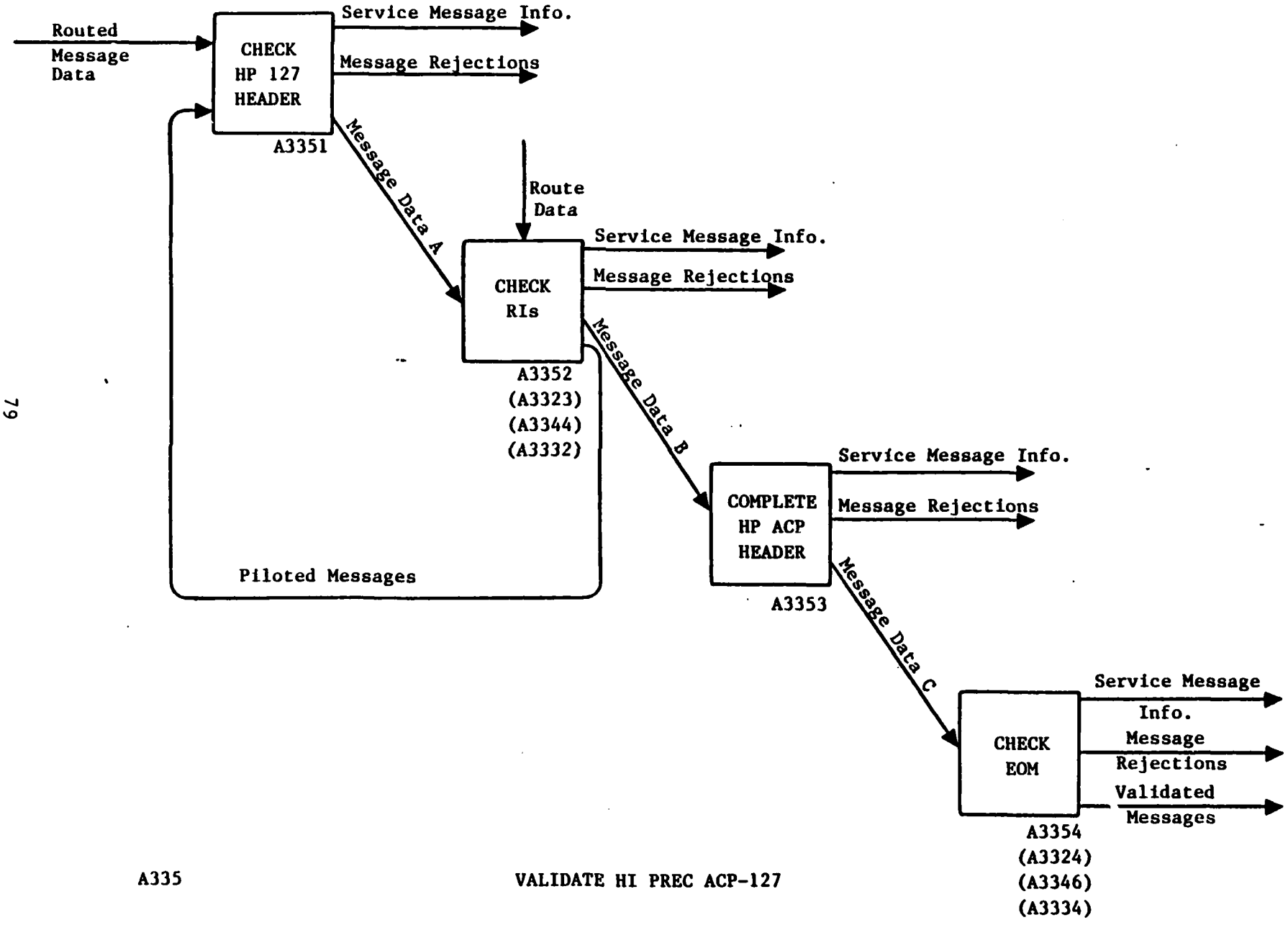
```
--A3323 A3332 A3344 A3352 CHECK_RIS
--PARA 3.2.1.2.10.8
--REV BAAAA
--1/6/82 PD

procedure CHECK_RIS is
begin
  loop
    --> FIND NEXT RI
    if A DELIMITER OTHER THAN [`B`|CR|LF] is FOUND then
      raise INVALID_RI_FIELD;
    end if;
    exit when END_OF_ROUTING is FOUND;
    -- END_OF_ROUTING IS '.' FOR JANAP-128
    -- AND CR CR LF Z OR CR CR LF D FOR ACP-127
    if RI(1..4) not in LETTER then
      raise INVALID_RI_FIELD;
    end if;
    if RI(1..4) = OUR_RI then
      --> CHECK ENTIRE RI FOR VALIDITY
    elsif RI(3..4) = "CR" then
      --> CHECK RI FOR VALID COLLECTIVE RI
    else
      --> CHECK RI(1..) FOR VALID RELAY RI
    end if;
    if RI GOOD then
      --> ADD RI TO VALID_RI_LIST
    else
      --> ADD RI TO INVALID_RI_LIST
    end if;
  end loop;
  if VALID_RI_LIST is EMPTY then
    raise ALL_RI_INVALID;
  elsif INVALID_RI_LIST not EMPTY then
    raise INVALID_RI;
  end if;
  if ACP-127 and END_OF_ROUTING = CR CR LF Z then
    --> RETURN TO CHECK_127_HEADER (THIS LINE WAS A PILOT)
  end if;
exception
  when INVALID_RI_FIELD =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_RI_FIELD,MESSAGE_ID));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when ALL_RI_INVALID =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (ALL_RI_INVA ID,MESSAGE_ID,INVALID_RI_LIST));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when INVALID_RI =>
```

```
            CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                (INVALID_RI,MESSAGE_ID,INVALID_RI_LIST));
            if PRECEDENCE = ['W'|'Y'|'Z'] then
              CIC_CAI(4) := 'W';
            end if;
        end CHECK_RIS;

EOT..
```

```
--A3324 A3334 A3345 A3354 CHECK EOM
--PARA 3.2.1.2.4.2.1.16, 3.2.1.2.10.2.9, 3.2.1.2.10.2.10 B(2)
--REV BAAAA
--1/7/82 PD

procedure CHECK_EOM is
begin
  if OSSN /= EOM_VALIDATION(2..5) then
    -- USE ACP_SSN FOR ACP-127
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (SUSPECTED_STRAGGLER, MESSAGE_ID));
    if PRECEDENCE = [W|Y|Z] then
      CIC_CAI(4) := 'W';
    else
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  elsif EOM_SEQ DOES not CONTAIN LF & "NNNN" then
    CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
  if PRECEDENCE = ['W'|'Y'|'Z'] then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (HIGH_PRECEDENCE_ACCEPTANCE,MESSAGE_ID));
  end if;
  if CHANNEL_MODE = 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOM;

EOT..
```

```
--A3325 A3346 CHECK EOT
--PARA 3.2.1.2.4.2.1.16, 3.2.1.2.10.1.10, 3.2.1.2.10.2.10 B (2)
--REV BAAAA
--1/13/82 PD

procedure CHECK_EOT is
begin
  if not SINGLE_CARD then
    if EOT_CARD.OSSN /= ACP_SSN then
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
          (SUSPECTED_STRAGGLER, MESSAGE_ID));
      if PRECEDENCE = [X|Y|Z] then
        CIC_CAI(4) := 'W';
      else
        --> TERMINATE PROCESSING ON THIS MESSAGE
        if CHANNEL_MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID)
              ;
        end if;
      end if;
    end if;
    if JANAP_FMT_LN_2.RECORD_COUNT = "PLTS" then
      if EOT_CARD.RECORD_COUNT < 3 or EOT_CARD.RECORD_COUNT >
          500 then
        CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
            (INVALID_EOM_REJ, MESSAGE_ID));
        --> TERMINATE PROCESSING ON THIS MESSAGE;
        if CHANNEL_MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID)
              ;
        end if;
      end if;
    elsif JANAP_FMT_LN_2.RECORD_COUNT = "MTMS" then
      if EOT_CARD.RECORD_COUNT /= "MTMS" and
          EOT_CARD.RECORD_COUNT /= ACTUAL_RECORD_COUNT then
        -- THIS IMPLIES YOU MUST COUNT THE ACTUAL NUMBER OF
        -- RECORDS
        CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
            (INVALID_EOM_REJ, MESSAGE_ID));
        --> TERMINATE PROCESSING ON THIS MESSAGE
        if CHANNEL_MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID)
              ;
        end if;
      end if;
    else
      if EOT_CARD.RECORD_COUNT /= ACTUAL_RECORD_COUNT then
        CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
            (INVALID_EOM_REJ, MESSAGE_ID));
        --> TERMINATE PROCESSING ON THIS MESSAGE
        if CHANNEL_MODE = [1|3] then
          GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID)
              ;
        end if;
      end if;
```

```
          end if;
        else
          if SINGLE_CARD(80) /= 'N' then
            CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
                (INVALID_EOM_REJ, MESSAGE_ID));
            --> TERMINATE PROCESSING ON THIS MESSAGE
            if CHANNEL_MODE = [1|3] then
              GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
            end if;
          end if;
        end if;
        if PRECEDENCE = ['W'|'Y'|'Z'] then
          CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
              (HIGH_PRECEDENCE_ACCEPTANCE,MESSAGE_ID));
        end if;
        if CHANNEL_MODE := 5 then
          ECSN := ECSN + 1;
        end if;
      end CHECK_EOT;

EOT..
```



A335 VALIDATE HI PREC ACP-127

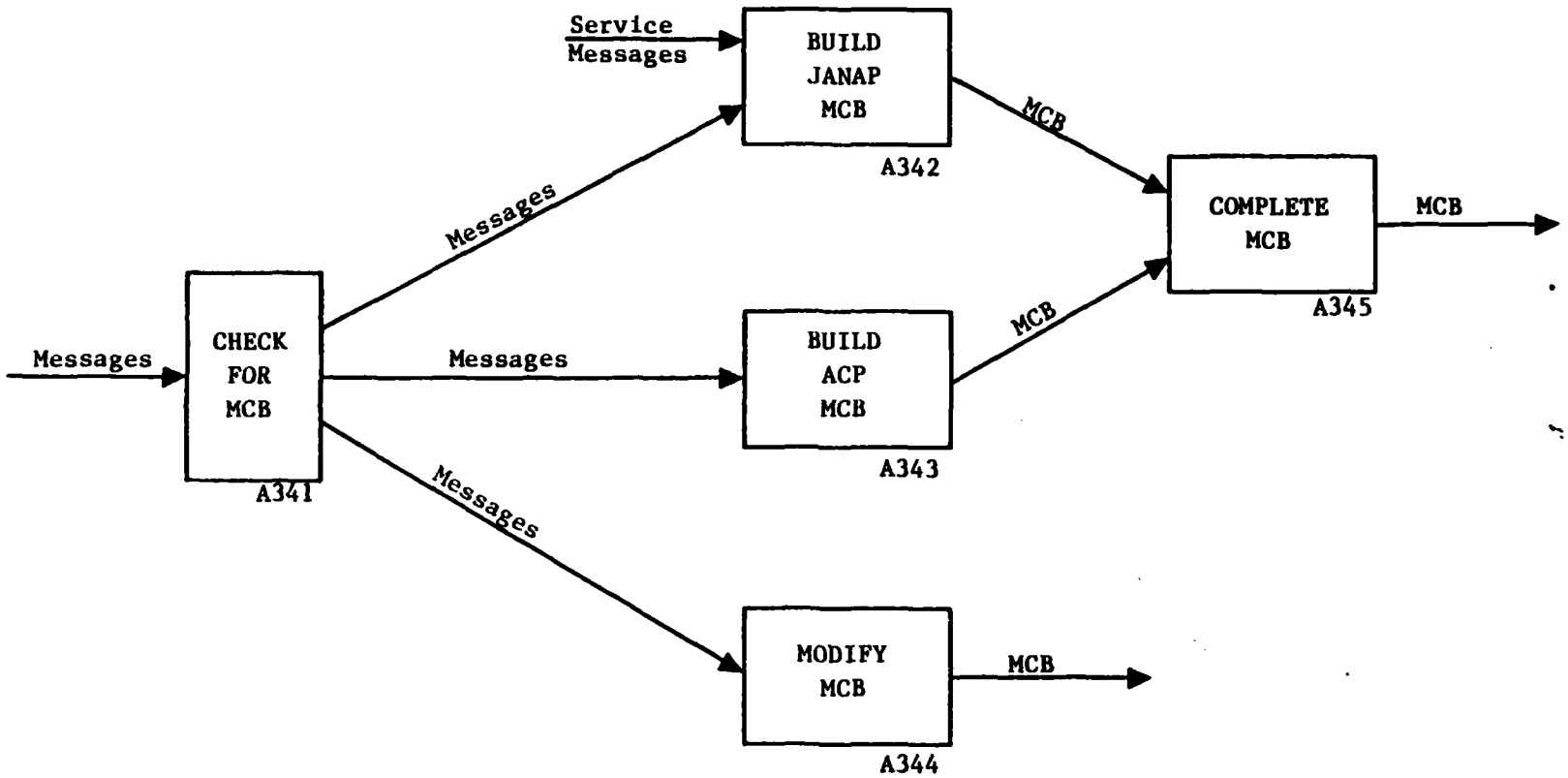
```
--A3351 CHECK HP 127 HEADER
--PARA 3.2.1.2.4.2.2
--REV BAAAA
--1/14/82 PD

procedure CHECK_HP_127_HEADER is
begin
  if BELL SIGNAL (or GARBLED BELL SIGNAL) is PRESENT then
    --> SEARCH FIRST 18 CHARACTERS FOR 'b'
    if not FOUND then
      CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
          (INVALID_HEADER_REJ, MESSAGE_ID));
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  elsif ACP_FMT_LN_2(3) /= 'b' then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_HEADER_REJ, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
end CHECK_HP_127_HEADER;

EOT..
```

```
--A3323 A3332 A3344 A3352 CHECK_RIS
--PARA 3.2.1.2.10.8
--REV BAAAA
--1/6/82 PD

procedure CHECK_RIS is
begin
  loop
    --> FIND NEXT RI
    if A DELIMITER OTHER THAN [`B`|CR|LF] is FOUND then
      raise INVALID_RI_FIELD;
    end if;
    exit when END_OF_ROUTING is FOUND;
    -- END_OF_ROUTING IS '.' FOR JANAP-128
    -- AND CR CR LF Z OR CR CR LF D FOR ACP-127
    if RI(1..4) not in LETTER then
      raise INVALID_RI_FIELD;
    end if;
    if RI(1..4) = OUR_RI then
      --> CHECK ENTIRE RI FOR VALIDITY
    elsif RI(3..4) = "CR" then
      --> CHECK RI FOR VALID COLLECTIVE RI
    else
      --> CHECK RI(1..) FOR VALID RELAY RI
    end if;
    if RI GOOD then
      --> ADD RI TO VALID_RI_LIST
    else
      --> ADD RI TO INVALID_RI_LIST
    end if;
  end loop;
  if VALID_RI_LIST is EMPTY then
    raise ALL_RI_INVALID;
  elsif INVALID_RI_LIST not EMPTY then
    raise INVALID_RI;
  end if;
  if ACP-127 and END_OF_ROUTING = CR CR LF Z then
    --> RETURN TO CHECK_127_HEADER (THIS LINE WAS A PILOT)
  end if;
exception
  when INVALID_RI_FIELD =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_RI_FIELD,MESSAGE_ID));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when ALL_RI_INVALID =>
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (ALL_RI_INVALID,MESSAGE_ID,INVALID_RI_LIST));
    --> TERMINATE MESSAGE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  when INVALID_RI =>
```

```
            CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
                (INVALID_RI,MESSAGE_ID,INVALID_RI_LIST));
            if PRECEDENCE = ['W'|'Y'|'Z'] then
              CIC_CAI(4) := 'W';
            end if;
        end CHECK_RIS;

EOT..
```

```
--A3353 COMPLETE HP 127 HEADER
--PARA 3.2.1.2.4.2.2, 3.2.1.2.8.4
--REV BAAAA
--1/14/82 PD

procedure COMPLETE_HP_127_HEADER is
begin
  if ACP_SSN PRECEDED BY '#' then
    --> RETAIN FOR STRAGGLER CHECK
  end if;
  --> MAKE 3 OF 5 CHECK ON CLASS_X5
  --> 3 OUT OF THE 5 CLASS CHARACTERS MUST BE CORRECT AND AGREE
  --> TO ESTABLISH THE CLASSIFICATION OF THE MESSAGE
  --> IF THIS CHECK FAILS, GENERATE AN INVALID_SCTY_FIELD SVC
  --> MESSAGE, TERMINATE AND REJECT THE MESSAGE
  if (CLASS = 'U' and OP_SIGNAL /= "ZNR") or (CLASS /= 'U' and
      OP_SIGNAL /= "ZNY") then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (INVALID_SCTY_FIELD, MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
  if CLASS >= LINE_DATA.SECURITY_PROSIGN then
    GENERATE_CONTROL_CHARACTER (REJECT_MESSAGE,MESSAGE_ID);
    GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (SCTY_MISMATCH,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
  end if;
end COMPLETE_HP_127_HEADER;

EOT..
```

```
--A3324 A3334 A3345 A3354 CHECK EOM
--PARA 3.2.1.2.4.2.1.16, 3.2.1.2.10.2.9, 3.2.1.2.10.2.10 B(2)
--REV BAAAA
--1/7/82 PD

procedure CHECK_EOM is
begin
  if OSSN /= EOM_VALIDATION(2..5) then
    -- USE ACP_SSN FOR ACP-127
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (SUSPECTED_STRAGGLER, MESSAGE_ID));
    if PRECEDENCE = [W|Y|Z] then
      CIC_CAI(4) := 'W';
    else
      --> TERMINATE PROCESSING ON THIS MESSAGE
      if CHANNEL_MODE = [1|3] then
        GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
      end if;
    end if;
  elsif EOM_SEQ DOES not CONTAIN LF & "NNNN" then
    CALL GENERATE_SERVICE_MESSAGE (SERVICE_MESSAGE_INFO =>
        (INVALID_EOM_REJ,MESSAGE_ID));
    --> TERMINATE PROCESSING ON THIS MESSAGE
    if CHANNEL_MODE = [1|3] then
      GENERATE_CONTROL_CHARACTER(REJECT_MESSAGE,MESSAGE_ID);
    end if;
  end if;
  if PRECEDENCE = ['W'|'Y'|'Z'] then
    CALL GENERATE_SERVICE_MESSAGE(SERVICE_MESSAGE_INFO =>
        (HIGH_PRECEDENCE_ACCEPTANCE,MESSAGE_ID));
  end if;
  if CHANNEL_MODE = 5 then
    ECSN := ECSN + 1;
  end if;
end CHECK_EOM;

EOT..
```



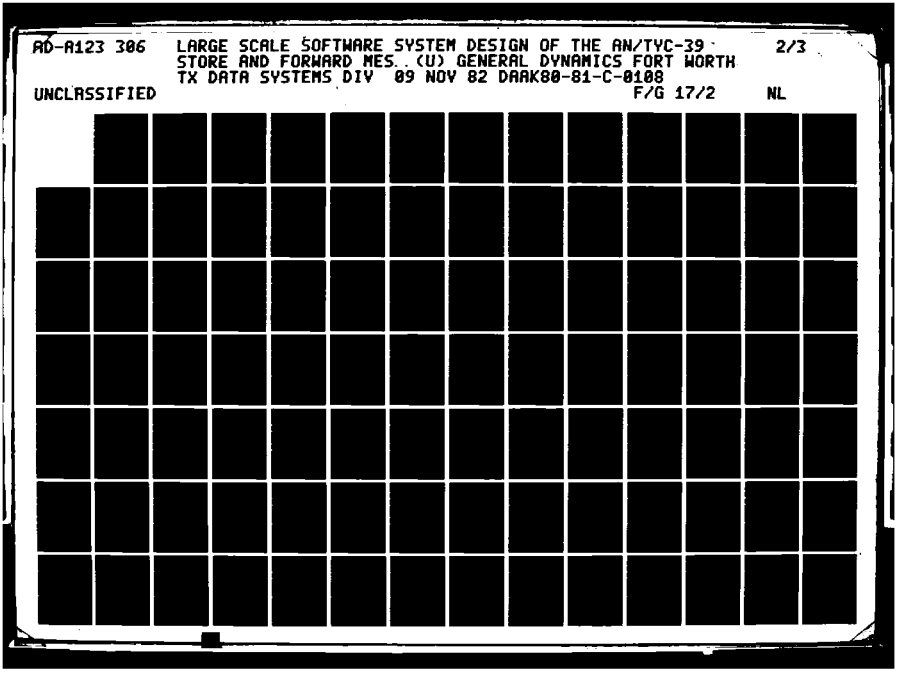
A34 BUILD/MODIFY MCB

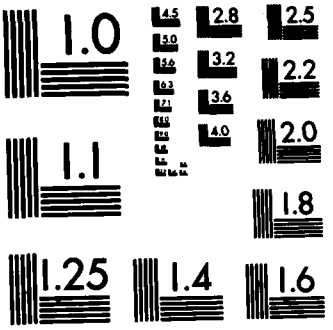
```
      --A341 CHECK FOR MCB
      --REV BAAA
      --12/14/81 PD

      procedure CHECK_FOR_MCB is
      begin
        if MCB_RECEIVED then
          MODIFY_MCB;
        elsif JANAP_128 then
          BUILD_JANAP_MCB;
        else   -- ACP_127
          BUILD_ACP_MCB;
        end if;
      end CHECK_FOR_MCB;

EOT..
```

```
        --A342 BUILD JANAP MCB
        --DCAC-370-D175-1 TABLE 8-1
        --REV BAAA
        --12/14/81 PD

        procedure BUILD_JANAP_MCB is
        begin
          LMF_PAIR := LMF_PAIR (FROM HEADER);
          CIC_CAI := CIC_CAI (FROM HEADER);
          OSRI := OSRI (FROM HEADER);
          OSSN := OSSN (FROM HEADER);
          DATE_TIME := DATE_TIME (FROM HEADER);
          if MODE = 2 or MODE = 4 or MODE = 5 then
            --> ADD ICD
            --> ADD ICSN
          else
            ICD := 'bbb';
            ICSN := 'bbbb';
          end if;
        end BUILD_JANAP_MCB;

EOT..
```

```
      --A343 BUILD ACP MCB
      --DCAC-370-D175-1 TABLE 8-1
      --REV BAAA
      --12/14/81 PD

      procedure BUILD_ACP_MCB is
      begin
        LMF_PAIR := FT;
        CIC_CAI := ZYUW;
        OSRI := HOME_RELAY (?) & ICD;
        OSSN := '0' & ICSN;
        DATE_TIME := SOM_IN_DATE_TIME;
        ICD := 'bbb';
        ICSN := 'bbbb';
      end BUILD_ACP_MCB;

EOT..
```

```
--A344 MODIFY MCB
--PARA 3.2.1.2.13.1,DCAC-370-D175-1 TABLE 8-1
--REV BAAA
--12/14/81 PD

procedure MODIFY_MCB is
  type CT is 0..2;
  COUNT : CT;
begin
  COUNT := 0;
  for all CHARACTERS in TDW loop
    if CHARACTER = SWITCH_ID then
      COUNT := COUNT + 1;
    end if;
  end loop;
  case COUNT is
    when 0 =>
      --> ADD SWITCH ID TO FIRST ZERO FIELD
    when 1 =>
      -- ZERO ALL FIELDS AFTER POSITION OF SWITCH_ID
      --> ADD SWITCH_ID SECOND TIME
    when 2 =>
      --> HOLD MESSAGE
      --> NOTIFY OPERATOR (PROVIDE ORBIT INFORMATION)
  end case;
  BP := MAC(1);
  for I in 2..83 loop
    BP := BP xor MCB(I);
  end loop;
  --> ADD SOH_TIME
end MODIFY_MCB;

EOT..
```


AD-A123 306
LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE AN/TYC-39
STORE AND FORWARD MES. (U) GENERAL DYNAMICS FORT WORTH
TX DATA SYSTEMS DIV 09 NOV 82 DAAK80-81-C-0108
2/3
UNCLASSIFIED
F/G 17/2
NL

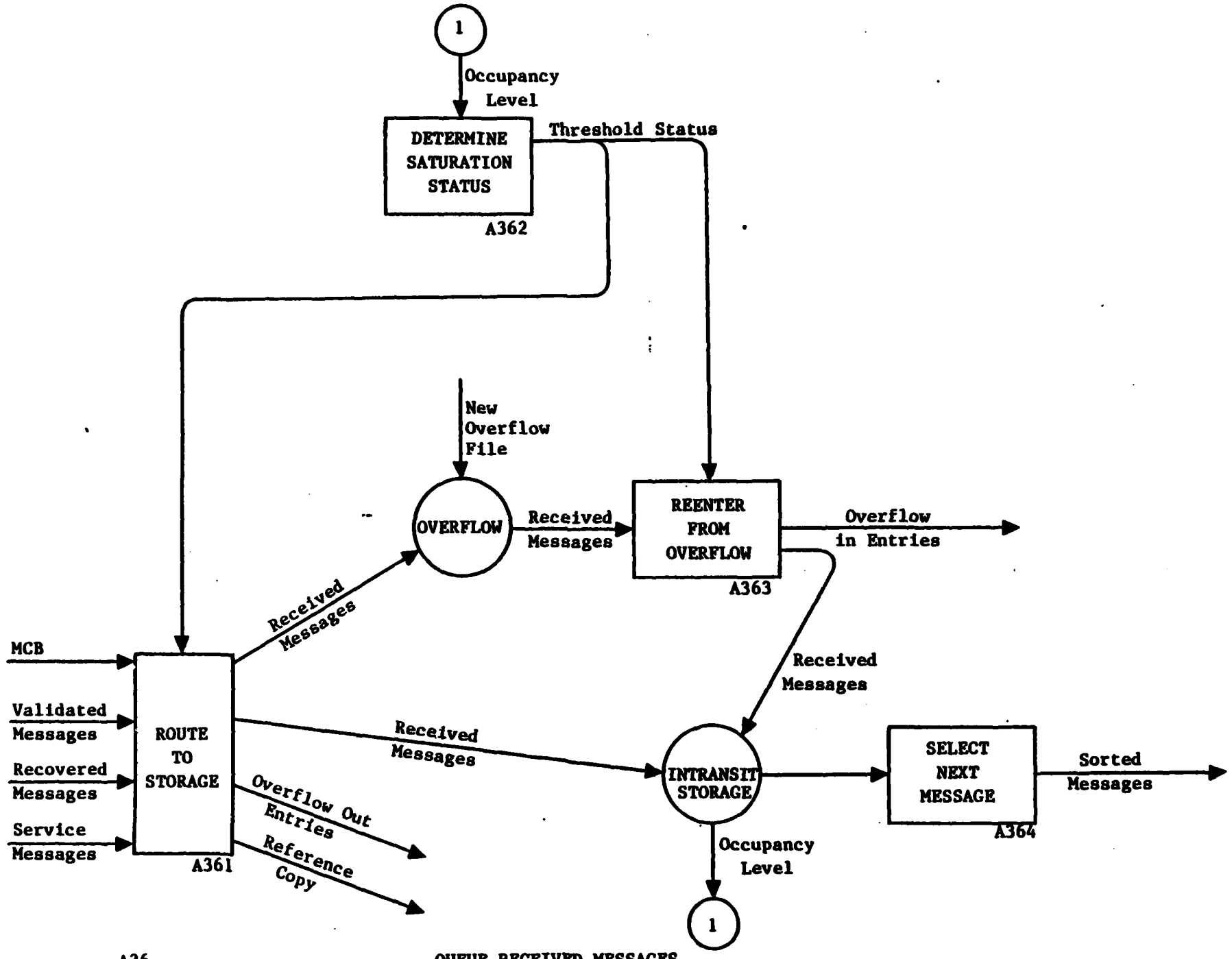


MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```
--A345 COMPLETE MCB
--DCAC-370-D175-1 TABLE 8-1
--REV BAAA
--12/14/81 PD

procedure COMPLETE_MCB is
begin
  --> PICK SELECT_CHARACTER BASED ON LMF
  PRECEDENCE := PRECEDENCE (FROM HEADER);
  CLASS := CLASS (FROM HEADER);
  NUMBER_BLOCKS := NUMBER of BLOCKS RECEIVED;
  --> ADD SOH_TIME
  if COLLECTIVE RIS in HEADER then
    TRIP = 'C';
  else
    TRIP = 'N';
  end if;
  if SERVICE MESSAGE then
    SDC := 'S';
  elsif PRECEDENCE = W then
    -- CRITIC
    if COLLECTIVE RIS in HEADER then
      if MESSAGE COMPLETE then
        SDC := 'M';
      else
        SDC := 'I';
      end if;
    else
      -- NO COLLECTIVE RIS
      if MESSAGE COMPLETE then
        SDC := 'C';
      else
        SDC := 'P';
      end if;
    end if;
  else
    SDC := 'b';
  end if;
  TDW := (1..12 => 0);
  OASC := SWITCH_ID;
  if SERVICE MESSAGE then
    SMRI := RI(2..6) (FROM HEADER);
  else
    SMRI := 'bbbbb';
  end if;
  SDO := 'b';
  TASC := 'b';
  SASC := 'b';
  BP := MCB(1);
  for I in 2..83 loop
    BP := BP xor MCB(I);
  end loop;
end COMPLETE_MCB;
```


:OT..

```
      -- A35 GENERATE SERVICE MESSAGE
      -- PARA 3.2.1.2.3, JANAP-128 PARA 319
      -- REV BAA
      -- 12/15/81 PD

      procedure GENERATE_SERVICE_MESSAGE is
      begin
        --> OBTAIN SVC MESSAGE FORMAT BASED ON SERVICE_MESSAGE_TYPE
            -- AND MESSAGE FORMAT
        --> FILL IN OSRI, OSSN, DATE_TIME
        --> ADD RI FROM ORIG MESSAGE OSRI
        --> ADD ORIG MESSAGE OSRI, OSSN, DATE_TIME AS REFERENCE
        if SERVICE_MESSAGE_TYPE = [INVALID_RI|OUTPUT_SCTY_MISMATCH
            |ILLEGAL_EXCHANGE| ALL_RI_INVALID] then
          --> ADD RIS
        elsif SERVICE_MESSAGE_TYPE = OPEN_CSN or INCORRECT_CSN then
          --> ADD CSNS
        end if;
      end GENERATE_SERVICE_MESSAGE;

EOT..
```



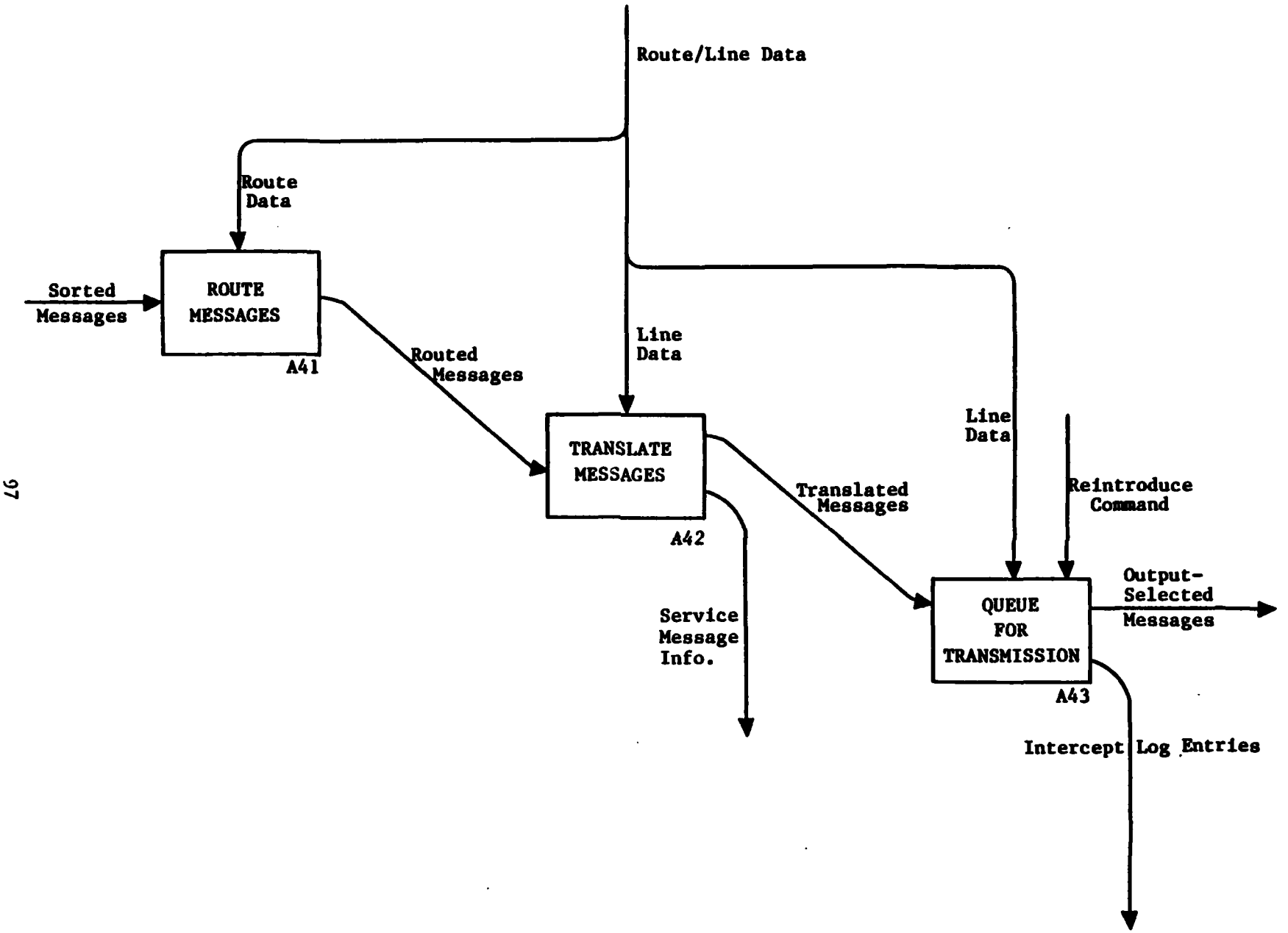
A36 QUEUE RECEIVED MESSAGES

```
      --A361 ROUTE TO STORAGE
      --PARA 3.2.1.2.14.4
      --REV BAAA
      --12/11/81 PD

      procedure ROUTE_TO_STORAGE is
      begin
        -->SEND A COPY OF THE MESSAGE & MCB TO REFERENCE STORAGE
        if PRECEDENCE /= W and PRECEDENCE /= Y and PRECEDENCE /= Z
            and THRESHOLD_STATUS = UPPER then
          --> ROUTE MESSAGE AND MCB TO OVERFLOW
          -->MAKE AN OVERFLOW_OUT_ENTRY
        else
          --> ROUTE MESSAGE TO INTRANSIT STORAGE
        end if;
      end ROUTE_TO_STORAGE;

EOT..
```

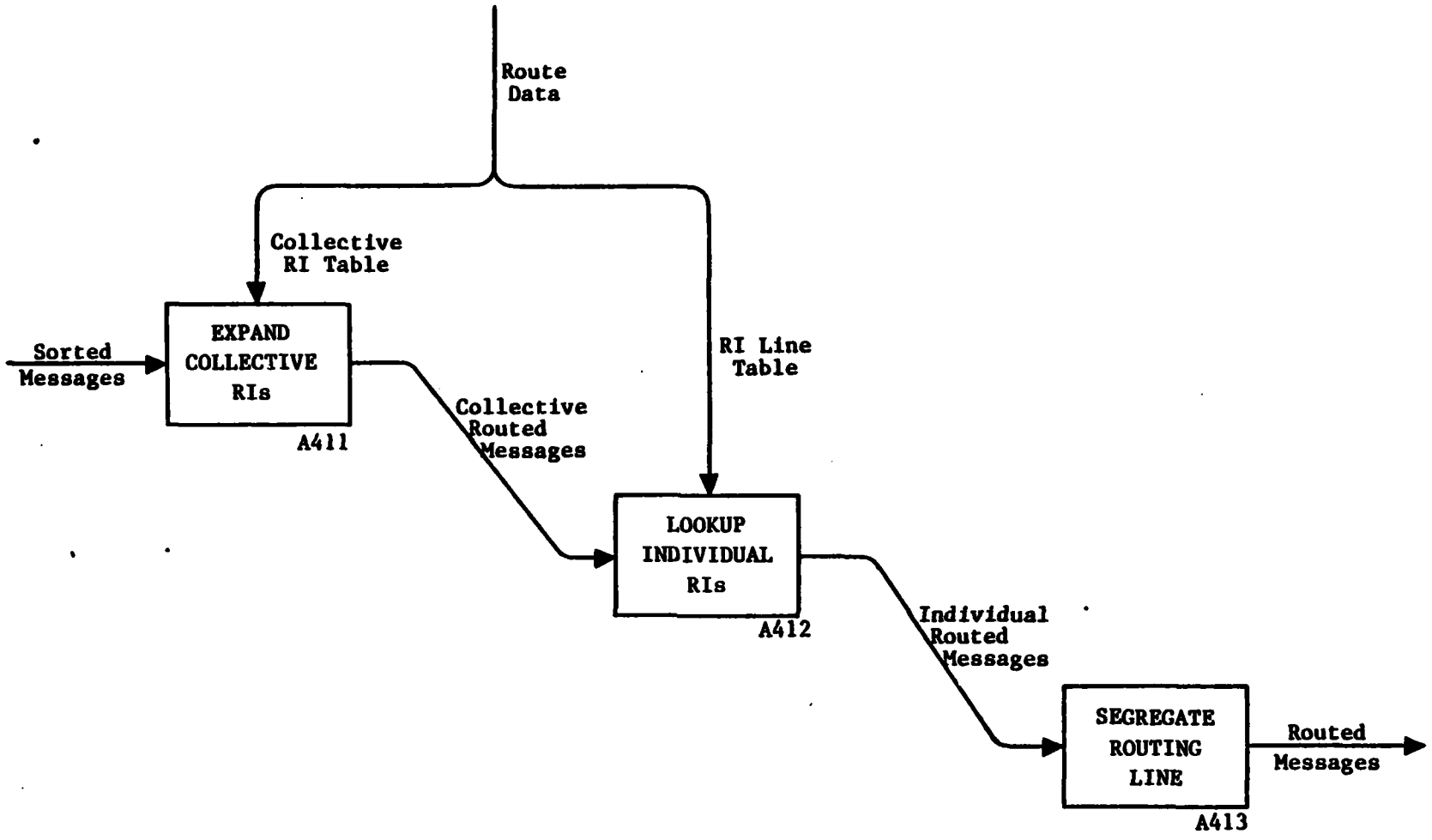
```
      --A362 DETERMINE SATURATION LEVEL
      --PARA 3.2.1.2.14.4
      --REV BAAA
      --12/11/81 PD

      procedure DETERMINE_SATURATION_LEVEL is
      begin
        if OCCUPANCY_LEVEL > UPPER_THRESHOLD then
          THRESHOLD_STATUS = UPPER;
        elsif OCCUPANCY_LEVEL > MIDDLE_THRESHOLD then
          THRESHOLD_STATUS = UPPER_MIDDLE;
        elsif OCCUPANCY_LEVEL > LOWER_THRESHOLD then
          THRESHOLD_STATUS = LOWER_MIDDLE;
        else
          THRESHOLD_STATUS = LOW;
        end if;
      end DETERMINE_SATURATION_LEVEL;

EOT..
```

```
      --A363 REENTER FORM OVERFLOW
      --PARA 3.2.1.2.14.4
      --REV BAAA
      --12/11/81 PD

     procedure REENTER_FROM_OVERFLOW is
     begin
       loop
         if THRESHOLD_STATUS = LOW then
           for all MESSAGES in OVERFLOW_STORAGE loop
             --> ROUTE MESSAGE TO INTRANSIT_STORAGE
             --> MAKE OVERFLOW_IN_ENTRY
           end loop;
         end if;
       end loop;
     end REENTER_FROM_OVERFLOW;

EOT..
```

```
      --A364 SELECT NEXT MESSAGE
      --PARA 3.2.1.4.1, 3.2.1.2.3
      --REV BAAA
      --12/11/81 PD

      procedure SELECT_NEXT_MESSAGE is
      begin
        --> WITHIN THE HIGHEST PRIORITY MESSAGES AVAILABLE
        --> SELECT THE EARLIEST SERVICE MESSAGE
        if NO SERVICE MESSAGES then
          --> SELECT THE EARLIEST MESSAGE
        end if;
      end SELECT_NEXT_MESSAGE;

EOT..
```



A4 PROCESS MESSAGES

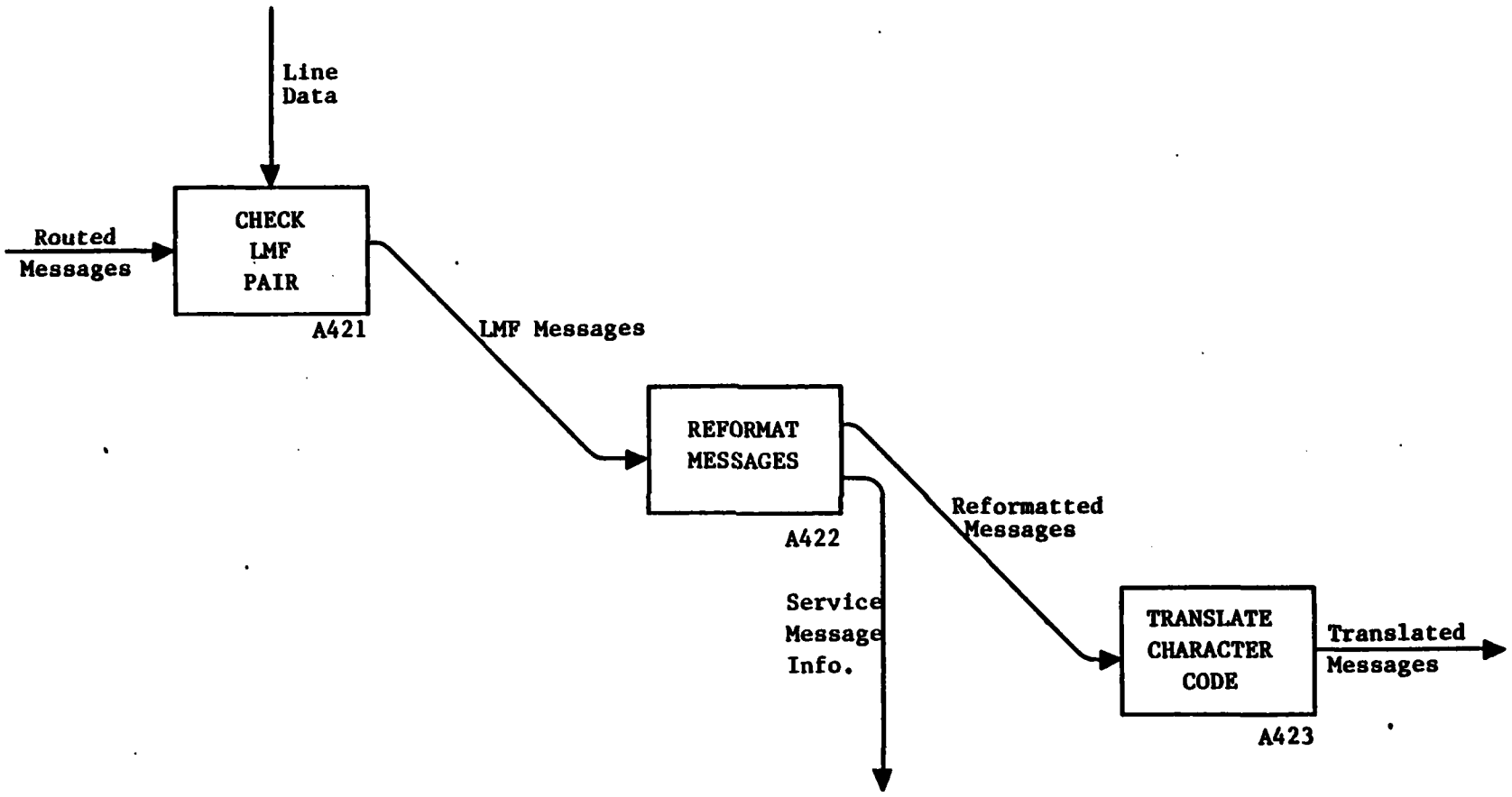


A41 ROUTE MESSAGES

```
      --A411 EXPAND COLLECTIVE RIS
      --REV BAAA
      --11/5/81 PD

      procedure EXPAND_COLLECTIVE_RIS is
      begin
        for all RIS in HEADER loop
          if RI(3..4) = "CR"then
            --> LOOK UP in COLLECTIVE_RI_TABLE
            --> ASSOCIATE RI WITH LINE OR LINES FOR OUTPUT
          end if;
        end loop;
      end EXPAND_COLLECTIVE_RIS;

EOT..
```

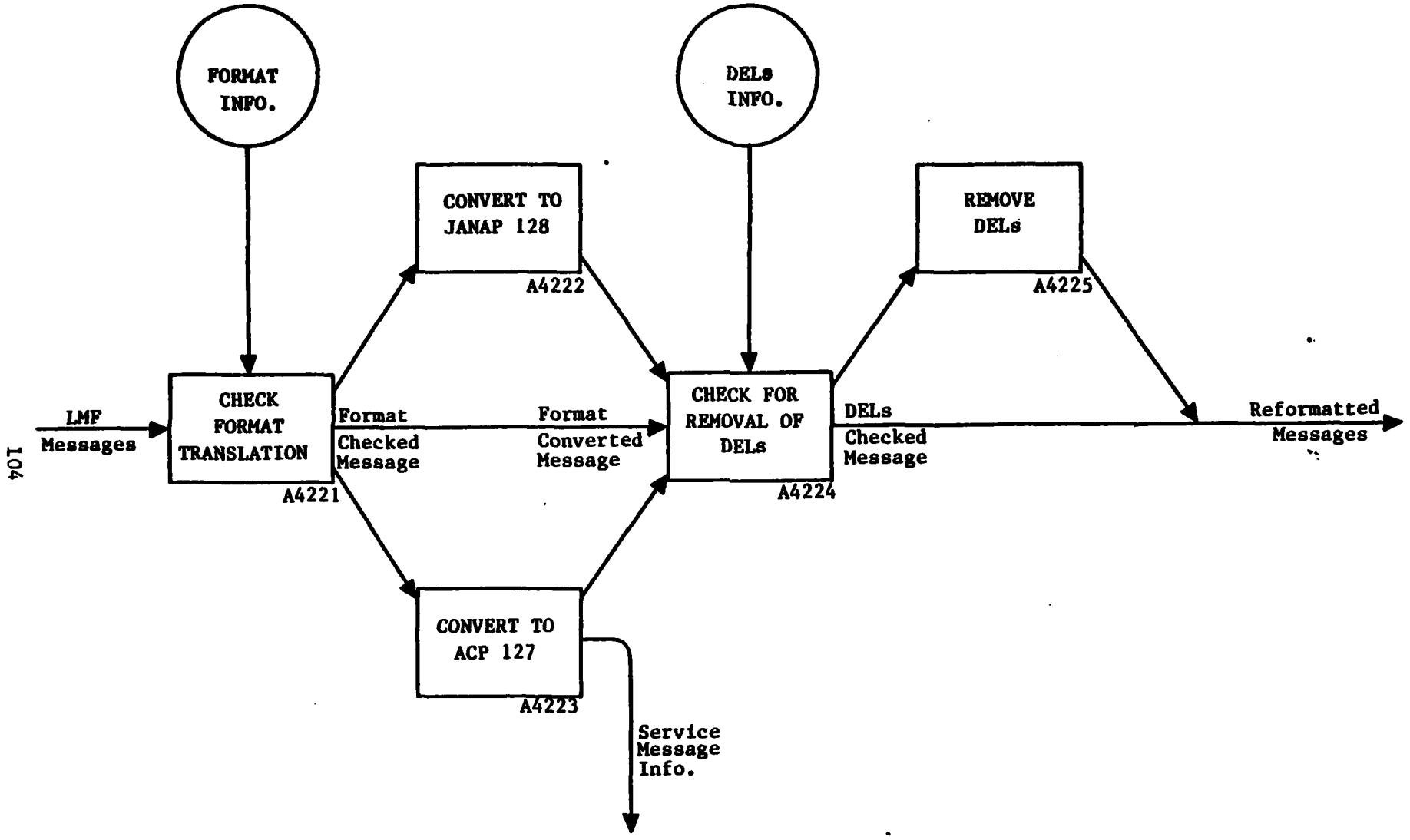
```
      -- A412 LOOK UP INDIVIDUAL RIS
      -- REV BAAA
      -- 11/5/81 PD

      procedure LOOK_UP_INDIVIDUAL_RIS is
      begin
        for all RIS ON ROUTING_LINE loop
          if RI(3..4) /= "CR" then
            --> LOOK UP RI IN ROUTING_TABLE
            --> ASSOCIATE LINE FOR OUTPUT WITH RI
          end if;
        end loop;
      end LOOK_UP_INDIVIDUAL_RIS;

EOT..
```

```
--A413 PERFORM ROUTING LINE SEGREGATION
--PURPOSE : GENERATES MULTIPLE ROUTES, SEGREGATED BY OUTPUT
    -- TRUNK TYPE
--REQUIRED BY PARAGRAPH: 3.2.1.2.7.6, FURTHER REFERENCES IN
-- DCAC-370-D175-1, SECTION 9-4.
-- REV BAAA
-- 11/5/81 HF/PD

procedure SEGREGATE is
begin
  for NEXT_RI in ROUTE_LINE loop
    --OBTAIN NEXT_RI IN HEADER
    if NEXT_RI = RI_TERMINATOR (2EH) then
      --> COPY RI_TERMINATOR TO NEW_ROUTE_LINE;
      exit loop;
    end if;
    if NEXT_RI = RI in GROUP_RI then
      --> COPY NEXT_RI TO NEW_ROUTE_LINE;
    else
      --> REPEAT UNTIL ALL RI IN GROUP_RI HAVE BEEN COMPARED TO
          --> NEXT_RI;
    end if;
    if NO RI=NEXT_RI then
      if OUTPUT DEVICE is DTE then
        if LMF_PAIR = "SC","CC","BB","DD" or "II"then
          --> COPY AN EQUIVALENT NUMBER OF SPACES TO
              --> NEW_ROUTE_LINE;
        end if;
      elsif LMF_FIRST = "S","C","B","D" or "I"then
        --> COPY A SINGLE SPACE TO NEW_ROUTE_LINE;
      elsif LMF_FIRST = "T","R","F","Q" or "A"then
        --> COPY "SI"(0FH) TO NEW_ROUTE_LINE;
      else
        raise LMF_ERROR;
      end if;
    end if;
    --> COPY ONE SPACE CHARACTER TO NEW_ROUTE_LINE;
  end loop;
end SEGREGATE;

EOT..
```



A42 TRANSLATE MESSAGES

```
      -- A421 CHECK LMF PAIR
      -- DCAC 370-D175-1 PARA 9-1A TABLE 9-1
      -- REV BAAA
      -- 11/20/81 PD

      procedure CHECK_LMF_PAIR is
      begin
        if SECOND_LMF_CHARACTER not ACCEPTABLE TO LINE then
          TRANSLATION_PAIR = FIRST_LMF_CHARACTER &
              LINE_LMF_CHARACTER;
        else
          TRANSLATION_PAIR = LMF_PAIR;
        end if;
        if TRANSLATION_PAIR /= LEGAL_EXCHANGE then
          --> REMOVE MESSAGE FROM SYSTEM
          --> TREAT RI AS BAD RI
          --> GENERATE SERVICE MESSAGE
          --> INFORM OPERATOR
        end if;
      end CHECK_LMF_PAIR;

EOT..
```



A422 REFORMAT MESSAGE

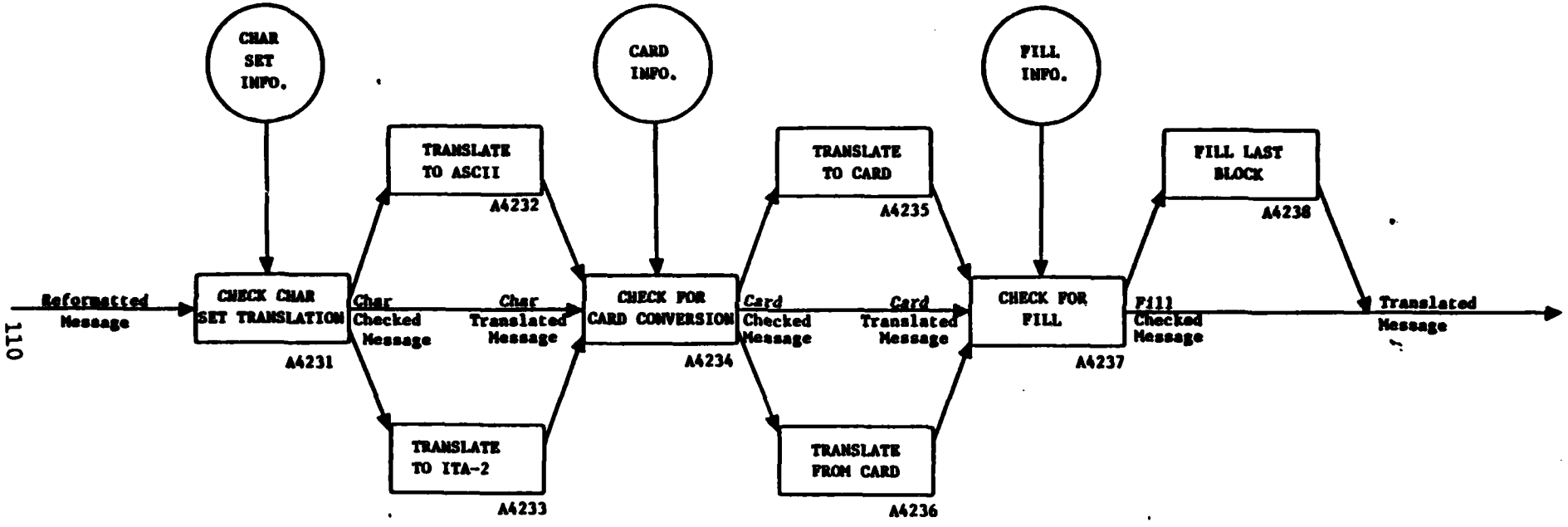
```
      -- A4221 CHECK FORMAT TRANSLATION
      -- DCDC-370-D175-1 TABLE 9-2
      -- REV BAAAA
      -- 11/6/81 PD

      procedure CHECK_FORMAT_TRANSLATION is
      begin
        --> READ FORMAT_INFO_FILE (KEY => TRANSLATION_PAIR)
        if TO_127 then
          CONVERT_TO_ACP_127;
        elsif TO_128 then
          CONVERT_TO_JANAP_128;
        else
          null;
        end if;
      end CHECK_FORMAT_TRANSLATION;

EOT..
```

```
      -- A4222 CONVERT TO JANAP 128
      -- DCAC-370-D175-1 PARA 9-2 TABLE 9-2
      -- ALSO SPEC PARA 3.2.1.2.10.4
      -- REV BAAAA
      -- 11/20/81 PD

      procedure CONVERT_TO_JANAP_128 is
      begin
        --> DELETE ACP_FMT_LN_2, SAVING INFO
        --> GENERATE JANAP_FMT_LN_2 AS FOLLOWS
        PRECEDENCE := PRECEDENCE FROM FMT_LN_2;
        LMF_PAIR := "FT";
        CLASS := CLASS FROM FMT_LN_4;
        CIC_CAI = ????????;
        if ORIGINATING_STATION is RELAY or TRIBUTARY OFF S&F then
          OSRI(1..4) := RELAY_RI(1..4);
        else
          OSRI(1..4) := S&F_RI(1..4);
        end if;
        OSRI(5..7) := CHANNEL_ID;
        OSSN(5..7) := '0' & INCOMING_CSN;
        DATE_TIME := DATE_TIME_RECEIVED;
        RIS := ACP_RIS;
        --> ADD SPACES, DASHES AND END OF ROUTING INDICATOR AS
            -- REQUIRED
      end CONVERT_TO_JANAP_128;

EOT..
```

```
      -- A4223 CONVERT_TO_ACP_127
      -- DCAC-370-D175-1 TABLE 9-2
      -- ALSO SPEC PARA 3.2.1.2.10.3.10, 3.2.1.2.10.3.11,
          -- 3.2.1.2.10.3.12
      -- REC BAAAA
      -- 11/20/81 PD

      procedure CONVERT_TO_ACP_127 is
      begin
        if FMT_LINE_4 is CORRECT or MESSAGE is SUSP_DUP then
          --> DELETE JANAP_FMT_LN_2, SAVING INFO
          -- GENERATE ACP_FMT_LN_2 AS FOLLOWS
          DOUBLE_PRECEDENCE_PROSIGN := (1,2 => PRECEDENCE);
          --> COPY RIS FROM JANAP_FMT_LN_2
          -- GENERATE ACP_FMT_LN_3 AS FOLLOWS
          PROSIGN := "DE";
          RI := OSRI;
          ACP_SSN := OSSN;
          DATE_TIME := DATE_TIME;
          if PRECEDENCE = 'Z' or PRECEDENCE = 'Y' then
            --> INSERT <FIGS JJJJJSSSSS LTRS> BEFORE FMT LN 2
          end if;
          if FMT_LN_4 not CORRECT then
            -- GENERATE FMT LN 4 (ONLY ON SUSP DUP)
            if CLASS = 'U' then
              OP_SIGNAL := "ZNR";
            else
              OP_SIGNAL := "ZNY";
            end if;
            CLASS_X5 := (1,2,3,4,5 => CLASS);
          end if;
          --> ADD 12 LTRS AFTER EOM SEQUENCE
        else
          --> REMOVE MESSAGE FROM SYSTEM
          --> INFORM SUPERVISOR
          --> GENERATE INVALID_RI SERVICE MESSAGE
        end if;
      end CONVERT_TO_ACP_127;

EOT..
```

```
      -- A4224 CHECK FOR REMOVAL OF DELETES
      -- DCAC-370-D175-1 TABLE 9-2
      -- REV BAAAA
      -- 11/6/81 PD

      procedure CHECK_FOR_DELETE_REMOVAL is
      begin
        --> READ DELETE_INFO_FILE (KEY => TRANSLATION_PAIR)
        if YES then
          REMOVE_DELETES;
        end if;
      end CHECK_FOR_DELETE_REMOVAL;

EOT..
```

```
      --A4225 REMOVE DELETES
      --DCAC-370-D175-1 TABLE 9-2
      --REV BAAAA
      --11/6/81 PD

      procedure REMOVE_DELETES is
      begin
        for all CHARACTERS in BODY loop
          if CHARACTER = SI or CHARACTER = SO then
            --> PLACE ' ' IN NEW_BODY
          elsif CHARACTER /= DEL then
            --> PLACE CHARACTER IN NEW_BODY
          end if;
        end loop;
        BODY := NEW_BODY;
      end REMOVE_DELETES;

EOT..
```



A423 TRANSLATE CHARACTER CODE

```
--A4231 CHECK FOR CHARACTER SET TRANSLATION
--DCAC-370-D175-1 TABLE 9-2
--REV BAAAA
--11/6/81 PD

procedure CHECK_FOR_CHARACTER_SET_TRANSLATION is
begin
  --> READ CHAR_SET_INFO_FILE (KEY => TRANSLATION_PAIR)
  if ASCII then
    TRANSLATE_TO_ASCII;
  elsif ITA then
    TRANSLATE_TO_ITA_2;
  else
    null;
  end if;
end CHECK_FOR_CHARACTER_SET_TRANSLATION;

EOT..
```

```
      --A4232 TRANSLATE TO ASCII
      --DCAC-370-D175-1 9-6C TABLE 9-2
      --REV BAAAA
      --11/6/81 PD

      procedure TRANSLATE_TO_ASCII is
      begin
        CURRENT_CASE := LTRS;
        for all CHARACTERS in MESSAGE loop
          if CHARACTER = LTRS or CHARACTER = FIGS then
            CURRENT_CASE := CHARACTER;
            -->DELETE CHARACTER;
          else
            --> CONVERT CHARACTER TO ASCII BASED ON CURRENT_CASE;
          end if;
        end loop;
      end TRANSLATE_TO_ASCII;

EOT..
```

```
--A4233 TRANSLATE TO ITA-2
--DCAC-370-D175-1 TABLE 9-2
--REV BAAAA
--11/6/81 PD

procedure TRANSLATE_TO_ITA_2 is
begin
  --> INSERT LTRS IN NEW_MESSAGE
  CURRENT_SHIFT := LTRS;
  for all CHARACTERS in MESSAGE loop
    --> LOOK UP ITA_EQUIV AND REQUIRED_SHIFT FOR
        -- CHARACTER
    if REQUIRED_SHIFT /= CURRENT_SHIFT then
      --> INSERT REQUIRED_SHIFT
      CURRENT_SHIFT := REQUIRED_SHIFT;
    end if;
    --> INSERT ITA_EQUIV IN NEW_MESSAGE;
  end loop;
  MESSAGE := NEW_MESSAGE;
end TRANSLATE_TO_ITA_2;

EOT..
```

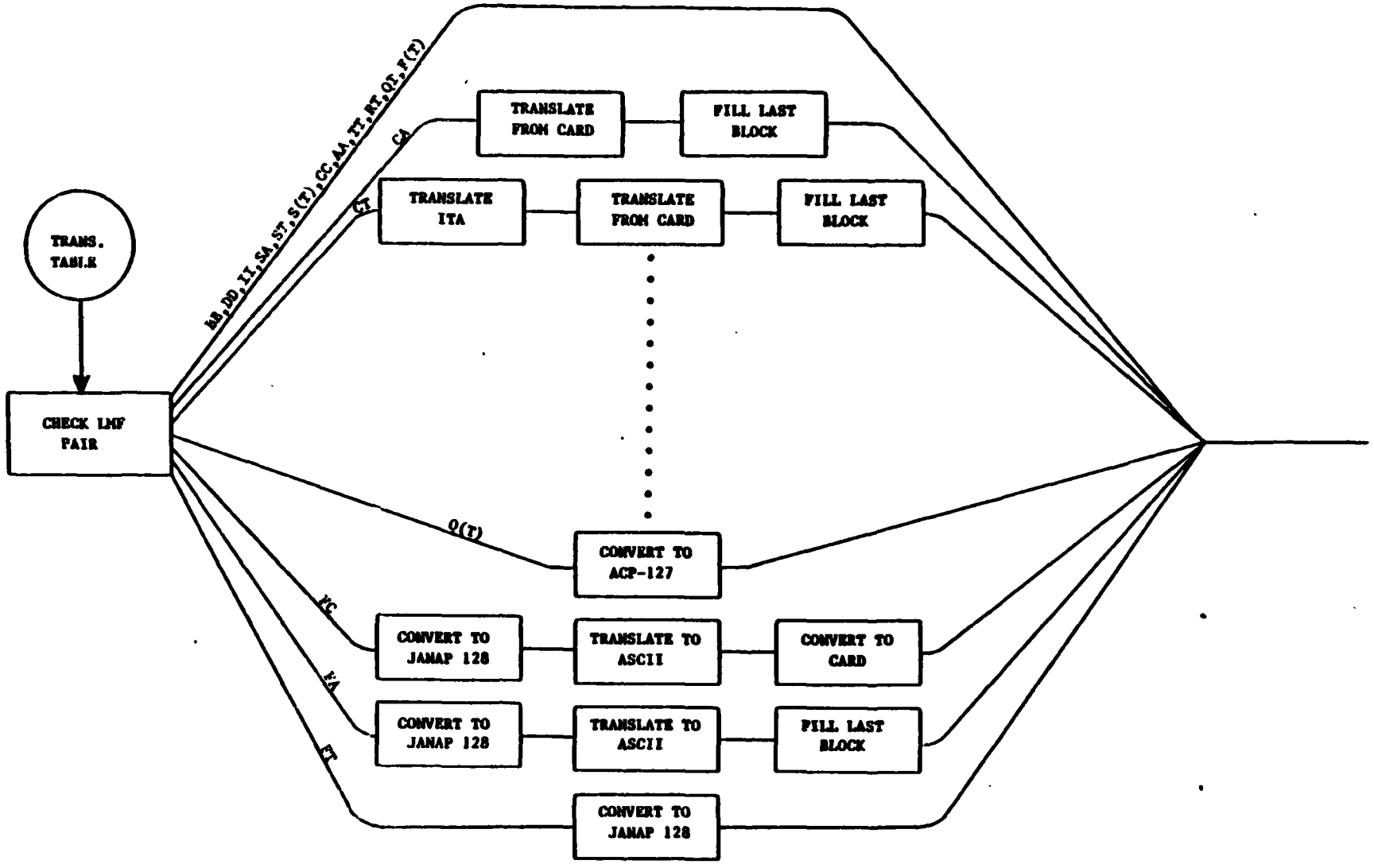
```
      --A4234 CHECK FOR CARD CONVERSION
      --DCAC-370-D175-1 TABLE 9-2
      --REV BAAAA
      --11/9/81 PD

      procedure CHECK_FOR_CARD_CONVERSION is
      begin
        --> READ CARD_INFO_FILE ( KEY => TRANSLATION_PAIR)
        if TO_CARD then
          TRANSLATE_TO_CARD;
        elsif FROM_CARD then
          TRANSLATE_FROM_CARD;
        else
          null;
        end if;
      end CHECK_FOR_CARD_CONVERSION;

EOT..
```

```
      --A4235 TRANSLATE TO CARD
      --DCAC-370-D175-1 CHAPTER 9 TABLE 9-2
      --REV BAAAA
      --11/20/81

      procedure TRANSLATE_TO_CARD is
      begin
        --> PLACE "MTMS" IN RECORD_COUNT OF HEADER
        for all LINES in BODY loop
          -- A LINE IS THE SEQUENCE OF CHARACTERS BETWEEN CR-CR-LF
              -- SEQUENCES
          --> DETERMINE LINE_LENGTH
          if LINE_LENGTH = 80 then
            OUT_LINE := LINE;
          elsif LINE_LENGTH < 80 then
            OUTLINE := (1..LINE_LENGTH => LINE, others => ' ');
          else
            -- LINE_LENGTH > 80
            OUT_LINE := LINE(1..80);
            SECOND_OUT_LINE := (1..LINE_LENGTH-80 => LINE (81..
                LINE_LENGTH),others => ' ');
          end if;
        end loop;
        --> DELETE EOM_SEQUENCE
        --> GENERATE EOT_CARD WITH "MTMS" IN RECORD_COUNT
      end TRANSLATE_TO_CARD;

EOT..
```

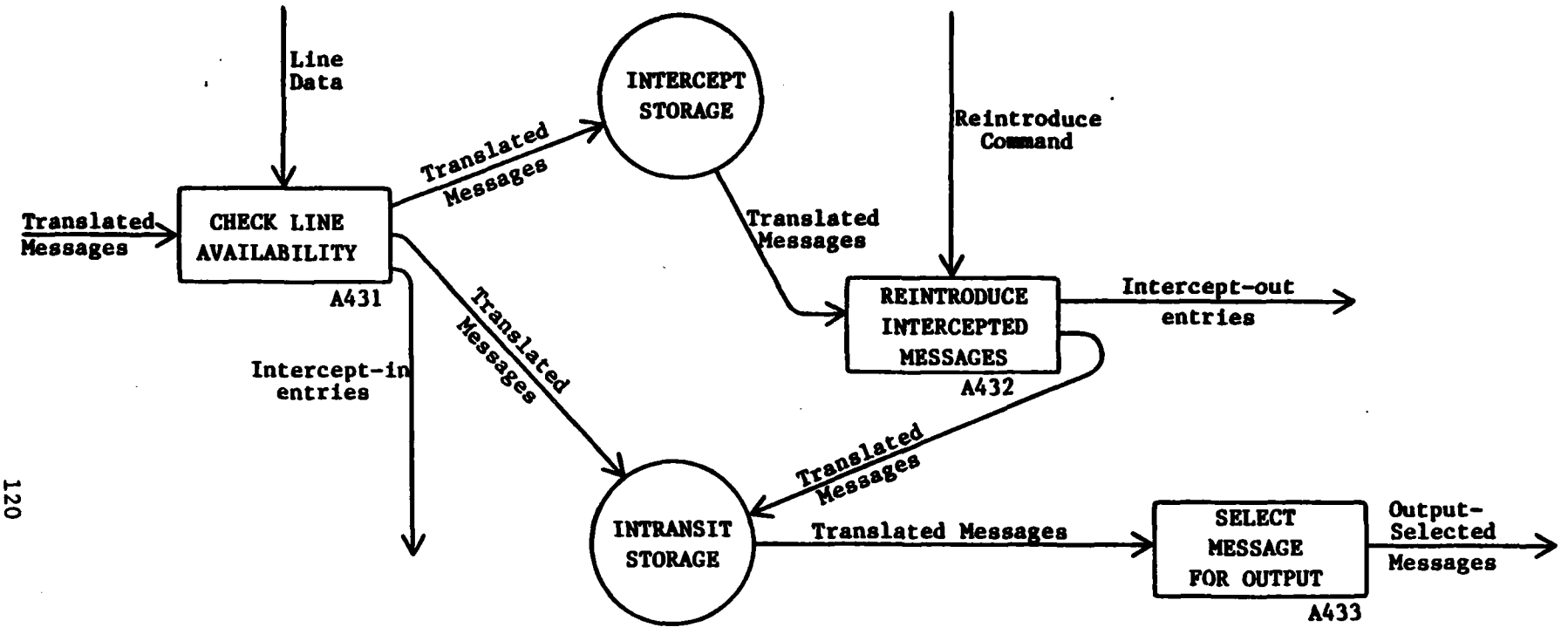
```
--A4236 TRANSLATE FROM CARD
--DCAC-370-D175-1 CH. 9 PARAGRAPH 5,8
--REV BAAAA
--11/20/81 PD

procedure TRANSLATE_FROM_CARD is
begin
  if JANAP_128 then
    for EACH LINE (CARD) in body loop
      --> DELETE TRAILING SPACES
      --> ADD CR CR LF
    end loop;
  else
    for EACH LINE in body loop
      --> COUNT PRINTABLE CHARACTERS IN LINE
      if > 69 then
        --> BREAK INTO TWO LINES OF MAX 69 CHARACTERS
        --> BREAK AT WORD AS CLOSE TO 69 CHARACTERS AS POSSIBLE
      end if;
      --> ADD CR CR LF TO EACH LINE PRODUCED
    end loop;
  end if;
  --> DELETE EOT_CARD
  --> ADD EOM_SEQUENCE
end TRANSLATE_FROM_CARD;

EOT..
```

```
      --A4237 CHECK FOR FILL
      --DCAC-370-D175-1 TABLE 9-2
      --REV BAAAA
      --11/20/81 PD

      procedure CHECK_FOR_FILL is
      begin
        --> READ FILL_INFO_FILE (KEY => TRANSLATION_PAIR)
        if YES then
          FILL_LAST_BLOCK;
        end if;
      end CHECK_FOR_FILL;

EOT..
```

```
      --A4238 FILL LAST BLOCK
      --DCAC-370-D175-1 CH. 9 PARAGRAPH 10
      --REV BAAAA
      --11/20/81 PD

      procedure FILL_LAST_BLOCK is
      begin
        --> FILL LAST 80 CHARACTER BLOCK OF MESSAGE (AFTER LAST 'N')
        if ASCII then
          --> FILL WITH SI
        else
          --> FILL WITH LTRS
        end if;
      end FILL_LAST_BLOCK;

EOT..
```



A421-3 CHECK IMF PAIR/REFORMAT MESSAGE/TRANSLATE CHARACTER CODE OVERVIEW

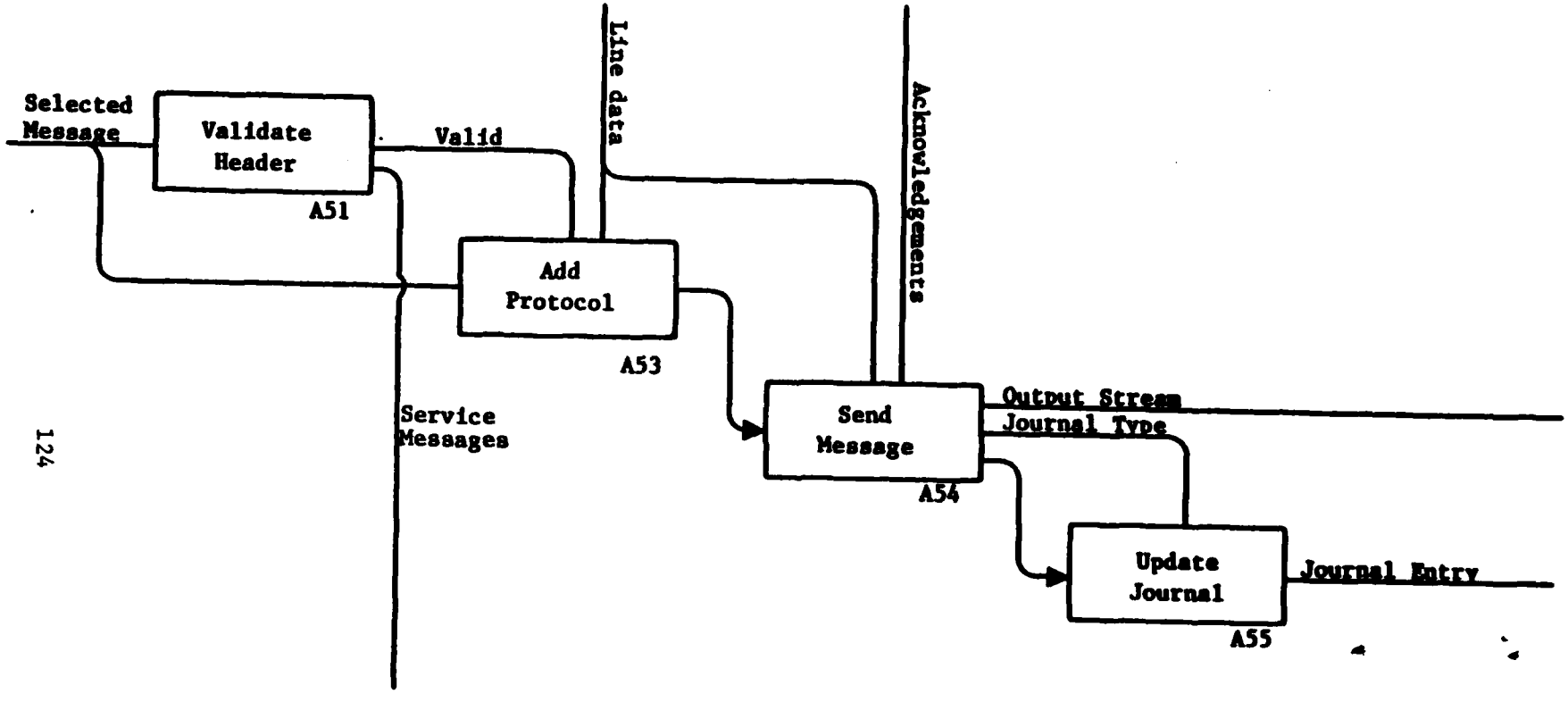


A43 QUEUE FOR TRANSMISSION

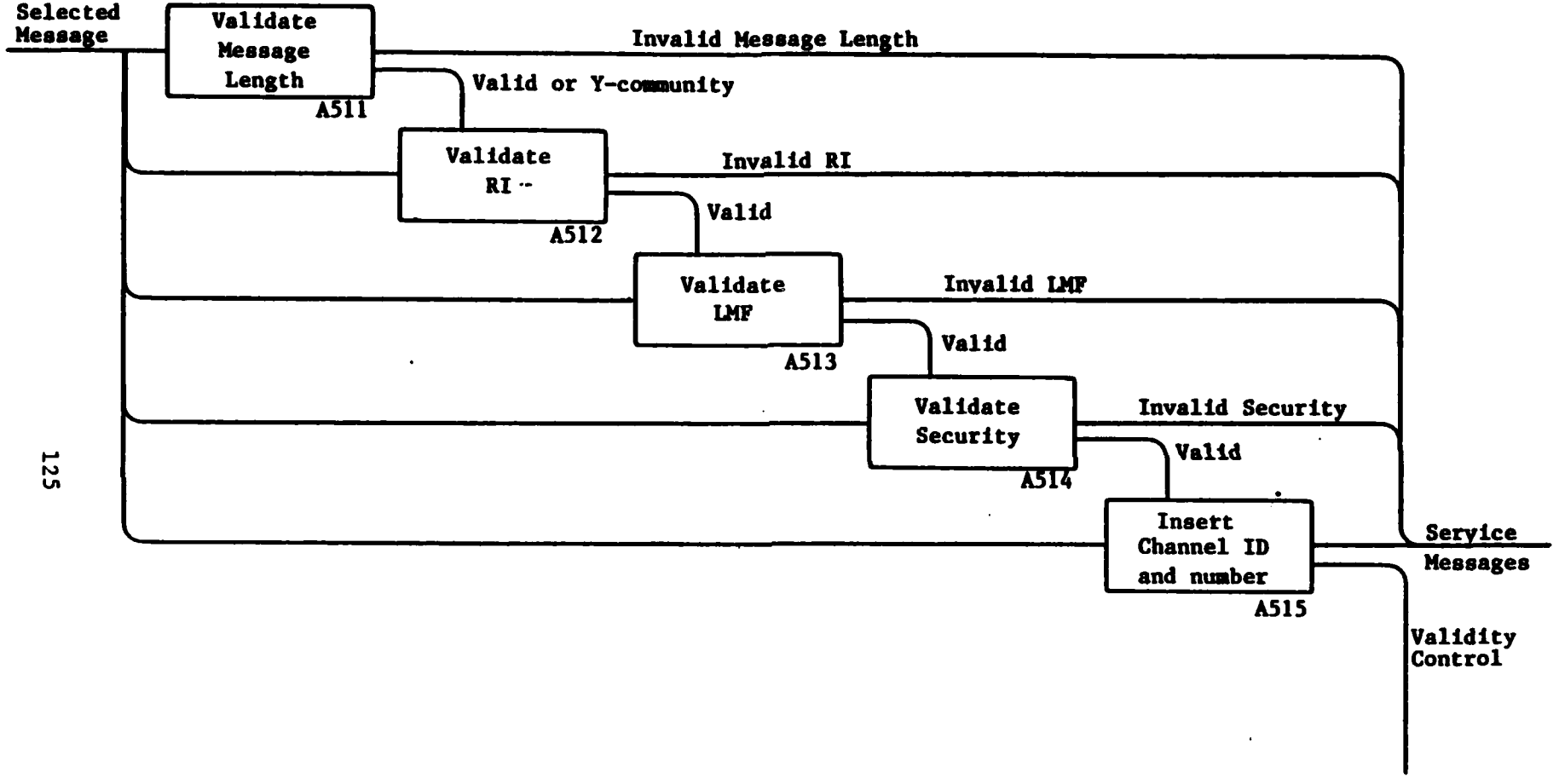
```
      --A431 CHECK LINE AVAILABILITY
      --PARA 3.2.1.2.14.3
      --REV BAAA
      --11/5/81 PD

      procedure CHECK_LINE_AVAILABILITY is
      begin
        --> LOOK UP LINE AVAILABILITY IN LINE TABLE
        if LINE_AVAILABLE then
          --> PLACE MESSAGE IS INTRANSIT_STORAGE
        else
          --> MAKE SOM_ENTRY IN JOURNAL
          --> PLACE MESSAGE ON INTERCEPT_STORAGE
          --> MAKE EOM_ENTRY IN JOURNAL
          if VOLUME_FULL then
            -->REQUEST NEW VOLUME
            -->ENTER NEW VOLUME IN INTERCEPT_VOLUMES LIST
          end if;
        end if;
      end CHECK_LINE_AVAILABILITY;

EOT..
```

```
--A432 REINTRODUCE INTERCEPTED MESSAGES
--PARA 3.2.1.2.14.3
--REV BAAA
--11/5/81 PD

procedure REINTRODUCE_INTERCEPTED_MESSAGES is
begin
  -- DONE ON SSF COMMAND
  for all VOLUMES ON INTERCEPT_VOLUMES_LIST loop
    if VOLUME NOT_MOUNTED then
      --> REQUEST MOUNT BY OPERATOR
    end if;
    for all MESSAGES ON VOLUME loop
      if MESSAGE_OUTPUT_LINE = LINE(S) TO BE REINTRODUCED then
        --> MAKE SOM ENTRY ON JOURNAL
        --> PLACE MESSAGE IN INTRANSIT_STORAGE
        --> MAKE EOM_ENTRY ON JOURNAL
      end if;
    end loop;
  end loop;
end REINTRODUCE_INTERCEPTED_MESSAGES;

EOT..
```

```
      -- A433 SELECT MESSAGE FOR OUTPUT
      -- REV BAAA
      -- 11/5/81 PD

      procedure SELECT_MESSAGE_FOR_OUTPUT is
      begin
        --> SELECT NEXT MESSAGE FOR OUTPUT BASED ON HIGHEST PRIORITY
        --> WITHIN PRIORITY, SELECT ON EARLIEST TIME RECEIVED
      end SELECT_MESSAGE_FOR_OUTPUT;

EOT..
```



A5 TRANSMIT MESSAGE



A51 VALIDATE HEADER

```
      --
      --
      -- A511
      -- VALIDATE MESSAGE LENGTH
      -- REQUIRED BY PARAGRAPH 3.2.1.2.15.1 (C)
      -- THIS PROCEDURE VALIDATES MESSAGE BLOCK COUNT BY
      -- RE-COUNTING BLOCKS UNTIL AN END-OF-MESSAGE IS
      -- ENCOUNTERED.
      -- 12/11/81 RCR

      procedure VALIDATE_MESSAGE_LENGTH is
      begin
        COMPUTED_NUMBER_OF_BLOCKS := 0;
        while FILE_STATUS/=END_OF_MESSAGE loop
          --> BUMP_TO_NEXT_BLOCK(FILE_STATUS);
          COMPUTED_NUMBER_OF_BLOCKS := COMPUTED_NUMBER_OF_BLOCKS +1;
        end loop;
        if COMPUTED_NUMBER_OF_BLOCKS /= NUMBER_BLOCKS
            then
          ERROR_CODE := INVALID_LENGTH_CODE;
        end if;
      end;

EOT..
```

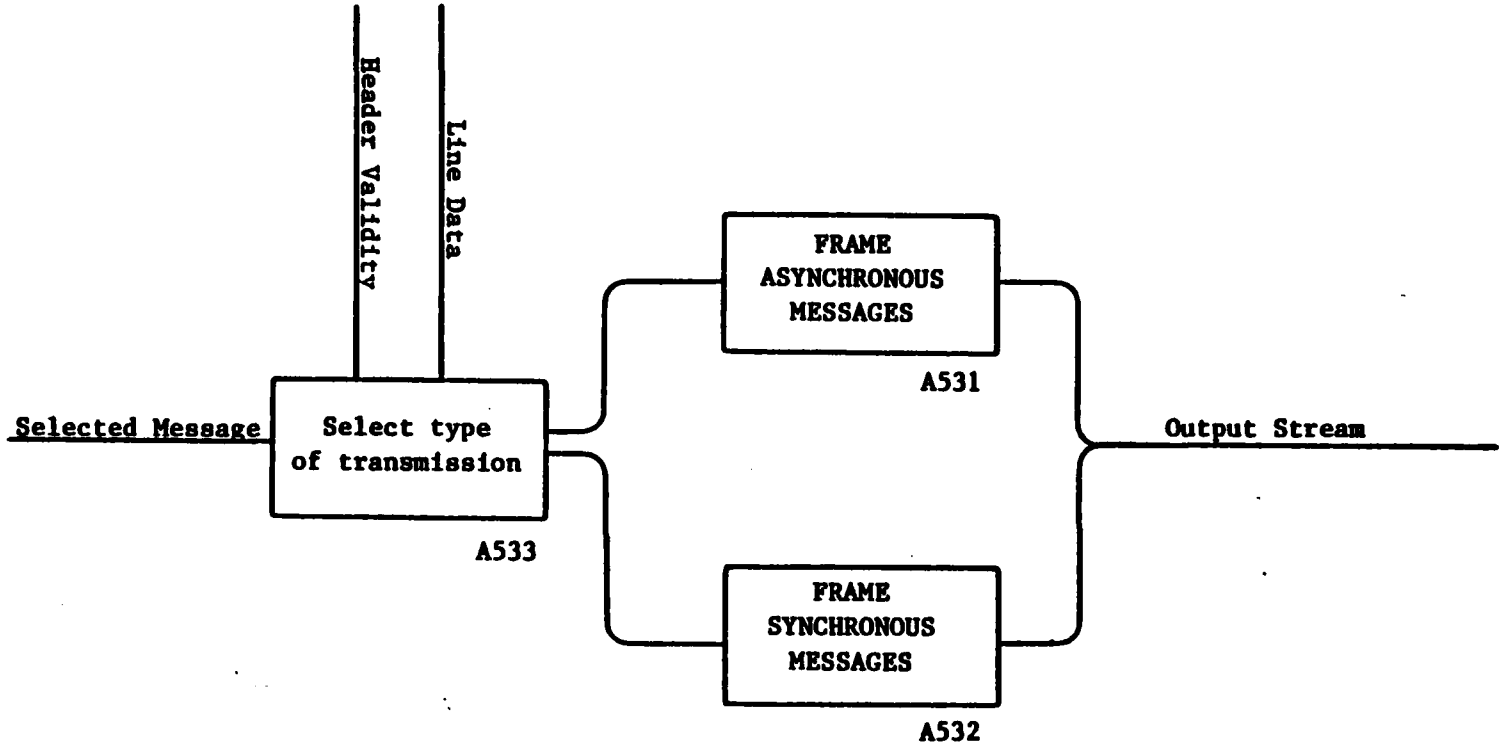
```
      --
      --
      -- A512
      -- THIS PROCEDURE VALIDATES THE ROUTING INDICATOR OF
      -- THE MESSAGE AS BEING THE ROUTING INDICATOR OF THE
      -- OUTPUT LINE IN USE. THIS VALIDATION IS REQUIRED
      -- BY PARAGRAPH 3.2.1.2.1.15.1 B.
      -- 12/11/81 RCR

      procedure VALIDATE_OUTPUT_RI is
      begin
        --> DETERMINE RI THIS LINE
        --> DETERMINE RI OF CURRENT TRANSMISSION
        if RI_THIS_OUTPUT_LINE /= RI_THIS_MESSAGE then
          ERROR_CODE := INVALID_LINE_RI;
        end if;
      end;

EOT..
```

```
--
--
-- A513
-- THIS PROCEDURE VALIDATES THE LANGUAGE MEDIA FORMAT
-- AS DESCRIBED BY PARAGRAPH 3.2.1.2.15.1 (C).
-- THIS IS ACCOMPLISHED BY ISOLATING THE FIRST CHARACTER
-- AND INSURING THAT THE COMBINATION OF THE TWO CHARACTERS
-- IS PROPER. SHOULD THE COMBINATION BE INVALID, AN ERROR
-- CODE IS PRODUCED FOR USE IN THE SERVICE MESSAGE ROUTINE.
-- VALID LMF COMBINATIONS ARE SHOWN IN DCAC 370-D175-1,
-- PAGE 9-2. LMF CODES ARE DEFINED IN DCAC 370-D175-1,
-- PAGE 3-4.
-- 12/11/81 RCR

procedure VALIDATE_LMF is
begin
  case FIRST_LMF_CHARACTER is
    when B =>
      if SECOND_LMF_CHARACTER /= 'B' then
        ERROR_CODE := INVALID_LMF;
      end if;
    when D =>
      if SECOND_LMF_CHARACTER \= 'D' then
        ERROR_CODE := INVALID_LMF;
      end if;
    when I =>
      if SECOND_LMF_CHARACTER /= 'I' then
        ERROR_CODE := INVALID_LMF;
      end if;
    when S =>
      case SECOND_LMF_CHARACTER is
        when C|A|T =>
          null;
        when others =>
          ERROR_CODE := INVALID_LMF;
      end case;
    when C|A|T|Q|F =>
      case SECOND_LMF_CHARACTER is
        when C|A|T|T_PAREN =>
          null;
        when others =>
          ERROR_CODE := INVALID_LMF;
      end case;
    when R =>
      case SECOND_LMF_CHARACTER is
        when T|T_PAREN =>
          null;
        when others =>
          ERROR_CODE := INVALID_LMF;
      end case;
    when others =>
      ERROR_CODE := INVALID_LMF;
  end case;
end;
```


EOT..

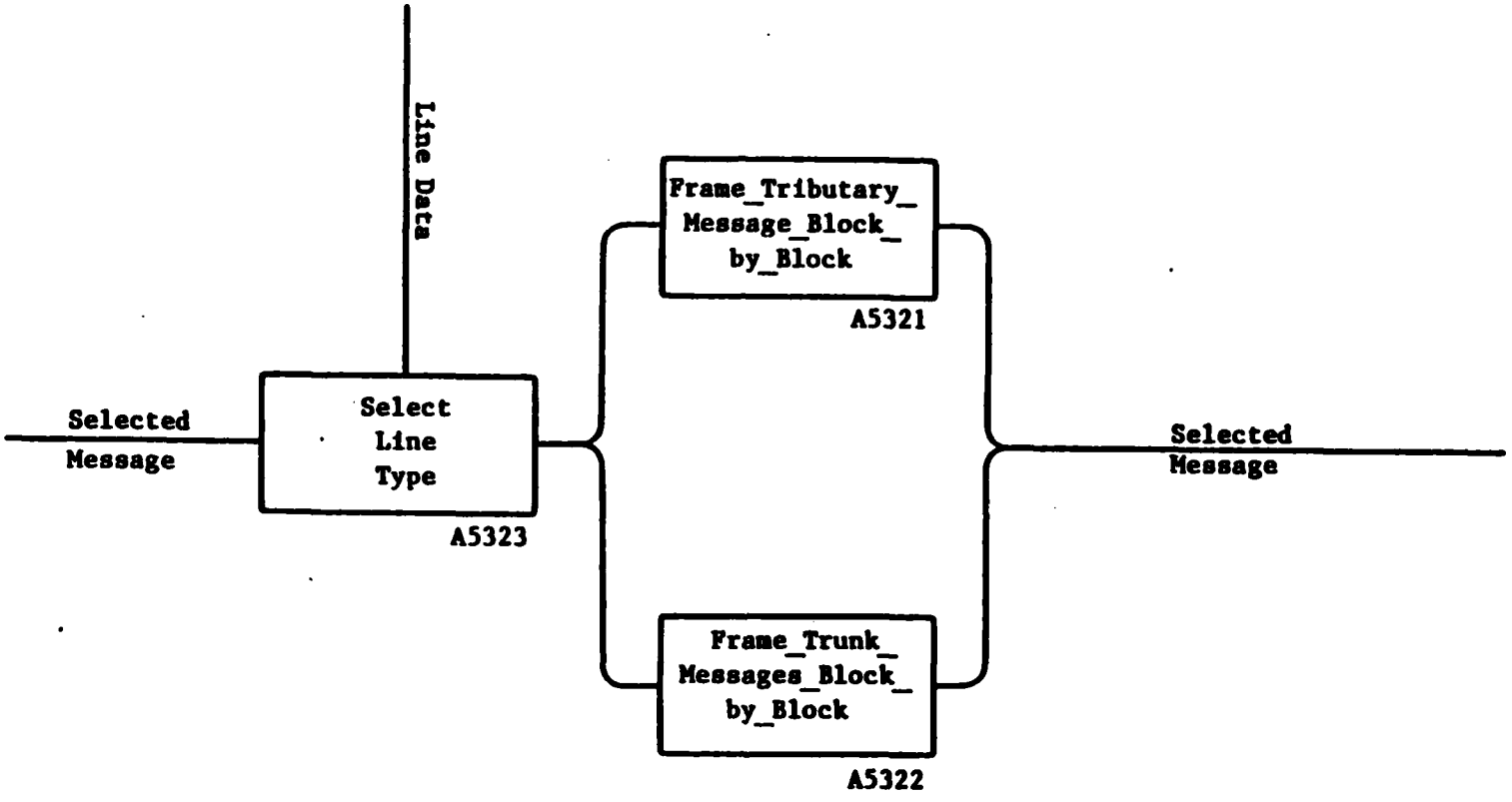
```
      --
      --
      -- A514
      -- THIS PROCEDURE MAKES SECURITY CHECKS AS REQUIRED
      -- BY PARAGRAPHS 3.2.1.2.15.1 (D) AND 3.2.1.2.8.6 .
      -- 12/11/81 RCR

      procedure VALIDATE_HEADER_SECURITY is
      begin
        NUMBER_OF_TRANSMISSIONS  := 0;
        NUMBER_OF_DESTINATIONS := NUMBER_RIS_THIS_MESSAGE;
        while NUMBER_OF_DESTINATIONS >0 loop
          -->DETERMINE SECURITY OF CURRENT RI
          if CLASS > SECURITY_OF_CURRENT_RI then
            -->SAVE BAD RI
            -->DO NOT SEND TO THIS RI
            ERROR_CODE := INVALID_RI_SECURITY;
          else
            NUMBER_OF_TRANSMISSIONS := NUMBER_OF_TRANSMISSIONS +1;
          end if;
        end loop;
        -- DETERMINE IF ANY RI'S WERE MISSED
        if NUMBER_OF_TRANSMISSIONS /= NO_OF_DESTINATIONS then
          -- PRODUCE NON-TRANSMITTED RI'S ON SERVICE MESSAGE
          GENERATE_SERVICE_MESSAGE;
        end if;
      end;

EOT..
```

```
-- A515
-- THIS PROCEDURE INSERTS  OUTPUT CHANNEL IDENTIFICATION
-- AND NUMBERING AS REQUIRED IN PARAGRAPH 3.2.1.2.15.1 (E)
-- AND JANAP-128.

procedure INSERTS_OUTPUT_CHANNEL_ID_AND_NUMBER is
begin
  case CHANNEL_MODE is
    when 2|4|5 =>
      -->APPEND SOH ON HEADER
      -->INSERT CHANNEL ID IN TRANSMISSION IDENTIFIER
      -->INSERT CHANNEL SEQUENCE IN TRANSMISSION IDENTIFIER
      if PRECEDENCE = 'Y' or 'Z' and
         MESSAGE_MODE = JANAP_128 then
        -->INSERT BELL CODE AFTER TRANSMISSION IDENTIFIER
      end if;
    when others =>
      null;
  end case;
  if MESSAGE_CANCELLED then
    if MESSAGE_IN_TRANSMISSION then
      --> TRANSMIT_CANTRAN_SEQUENCE
      --> INCREMENT_CSN
    end if;
  end if;
end;

EOT..
```



A53 ADD PROTOCOL

```
 -- A531
 -- THE PURPOSE OF THIS ROTINE IS TO DESCRIBE THE REQUIREMENTS FOR THE
-- FRAMING OF ASYNCHRONOUS MESSAGES AS DESCRIBED IN THE FOLLOWING:
--                MANUAL                      PARAGRAPH(S)
--             JANAP 128                        401 (g).
-- 01/28/82
   procedure FRAME_ASYNCHRONOUS_MESSAGE is
     begin
       if MESSAGE_MODE = ACP_127 | ACP_127_MOD then
         --> PRECEDE TEXT WITH SIX "BLANKS" AND SIX "LTRS"
       elsif MESSAGE_MODE = JANAP_128 then
         --> PRECEDE TEXT WITH SIX "NULLS" AND SIX "DELETES"
       end if;
       --> FOLLOW TEXT WITH ?????????
     end FRAME_ASYNCHRONOUS_MESSAGE;
EOT..
```



A532

FRAME SYNCHRONOUS MESSAGES

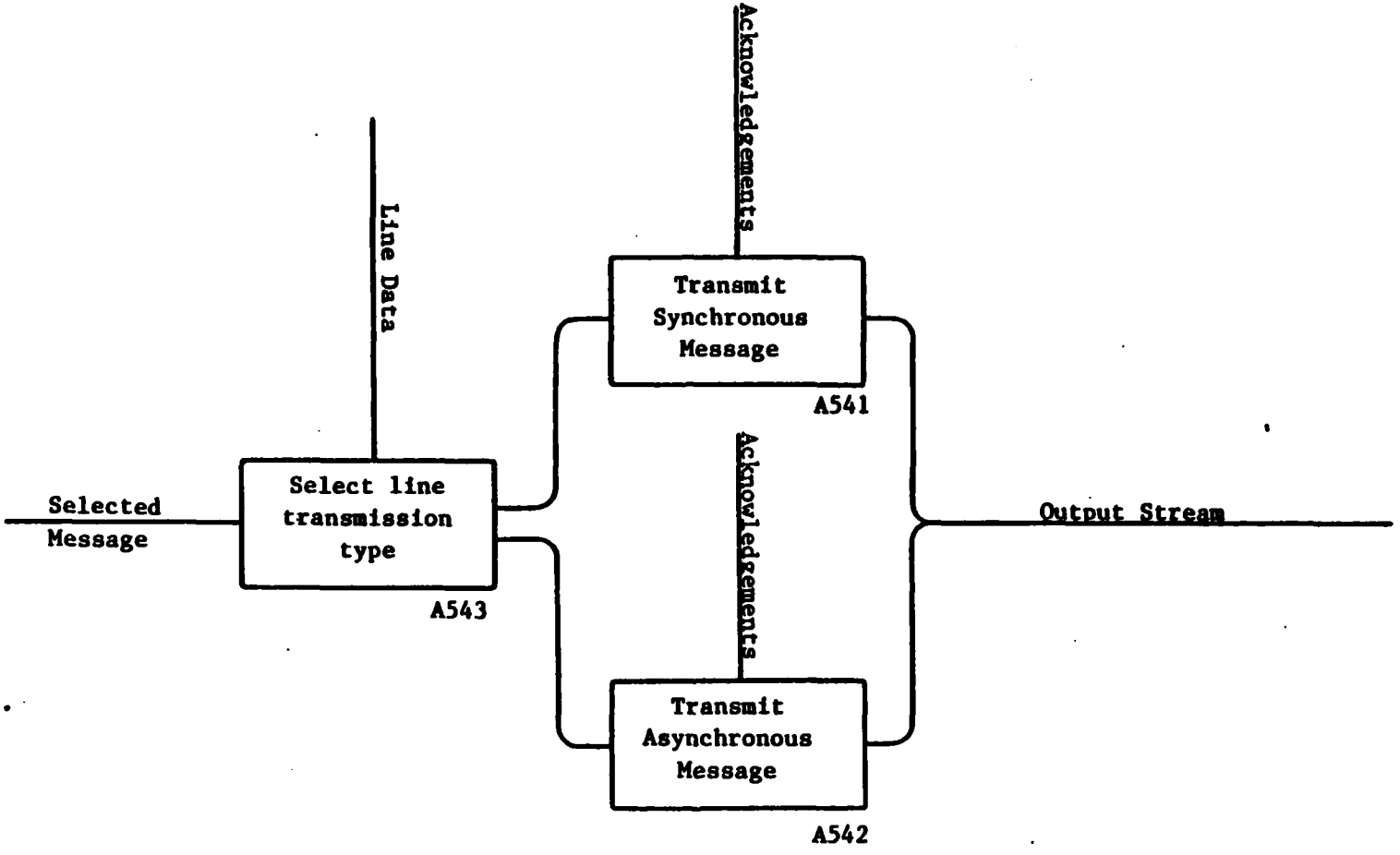
```
--
--
-- A5321
-- THE PURPOSE OF THIS PROCEDURE IS TO FRAME SYNCHRONOUS
-- MESSAGES TO TRIBUTARIES
-- WITH THE PROPER PROTOCOL AS DICTATED BY THE
-- LMF OF THE RECEIVING STATION. THIS REQUIREMENT IS
-- CONTAINED IN DCAC 370-D175-1, CHAPTER 8.
-- 12/16/81 RCR

procedure FRAME_TRIBUTARY_MESSAGE_BLOCK_BY_BLOCK is
begin
  case SECOND_LMF_CHARACTER is
    when C|R|T|A|Q =>
      if BLOCK_FRAMED = 1 then
        --> COPY MCB TO OUTPUT BUFFER
      elsif BLOCK_FRAMED /= NUMBER_BLOCKS then
        --> PRECEDE CURRENT_BLOCK WITH "STX" AND "DEL"
        --> AND APPEND IT WITH "ETB" AND "BP"
      end if;
      if BLOCK_FRAMED = NUMBER_BLOCKS then
       --> PRECEDE BLOCK WITH "STX" AND "DEL"
       --> APPEND "ETX" AND "BP" TO BLOCK
      end if;
    when S =>
      --> PRECEDE BLOCK WITH "SOH" AND "D"
      --> APPEND "ETX" AND "BP" TO BLOCK
    when D =>
      --> THE DCAC 370-D175-1 DOCUMENT REFERS
      --> TO NON-EXISTENT
      --> FIGURE 8-1 IN DEFINING THE FORMAT FOR LMF "D"
    when B|I =>
      if BLOCK_FRAMED = 1 then
       --> PRECEDE THE MCB WITH "SOH" AND "D" CHARACTERS
       --> FOLLOW IT WITH "ETX" AND "BP" CHARACTERS
      elsif BLOCK_FRAMED = 2|NUMBER_BLOCKS then
       --> BUILD MODE CHANGE BLOCK
       --> TRANSMIT MODE CHANGE BLOCK
      else
       --> PRECEDE MESSAGE BLOCK WITH "SOH" AND "D"
       --> CHARACTERS
       --> AND FOLLOW IT WITH ??????????????
      end if;
    when G =>
      if BLOCK_FRAMED=1 then
       --> PRECEDE BLOCK OF MESSAGE WITH "SOH" AND SEL
       --> CHARACTERS
       --> AND FOLLOW IT WITH "EM", "ETB", AND "BP" CHARACTERS
      elsif BLOCK_FRAMED /= NUMBER_BLOCKS then
       --> PRECEDE BLOCK WITH "STX" AND "DEL" CHARACTERS
       --> AND FOLLOW IT WITH "ETB" AND "BP" CHARACTERS
      elsif BLOCK_FRAMED = NUMBER_BLOCKS then
       --> PRECEDE BLOCK WITH "STX" AND "DEL" CHARACTERS
       --> AND FOLLOW IT WITH "ETB" AND "BP" CHARACTERS
      end if;
```

```
          when others =>
            ERROR_CODE := INVALID_LMF;
        end case;
      end FRAME_TRIBUTARY_MESSAGE_BLOCK_BY_BLOCK;
EOT..
```

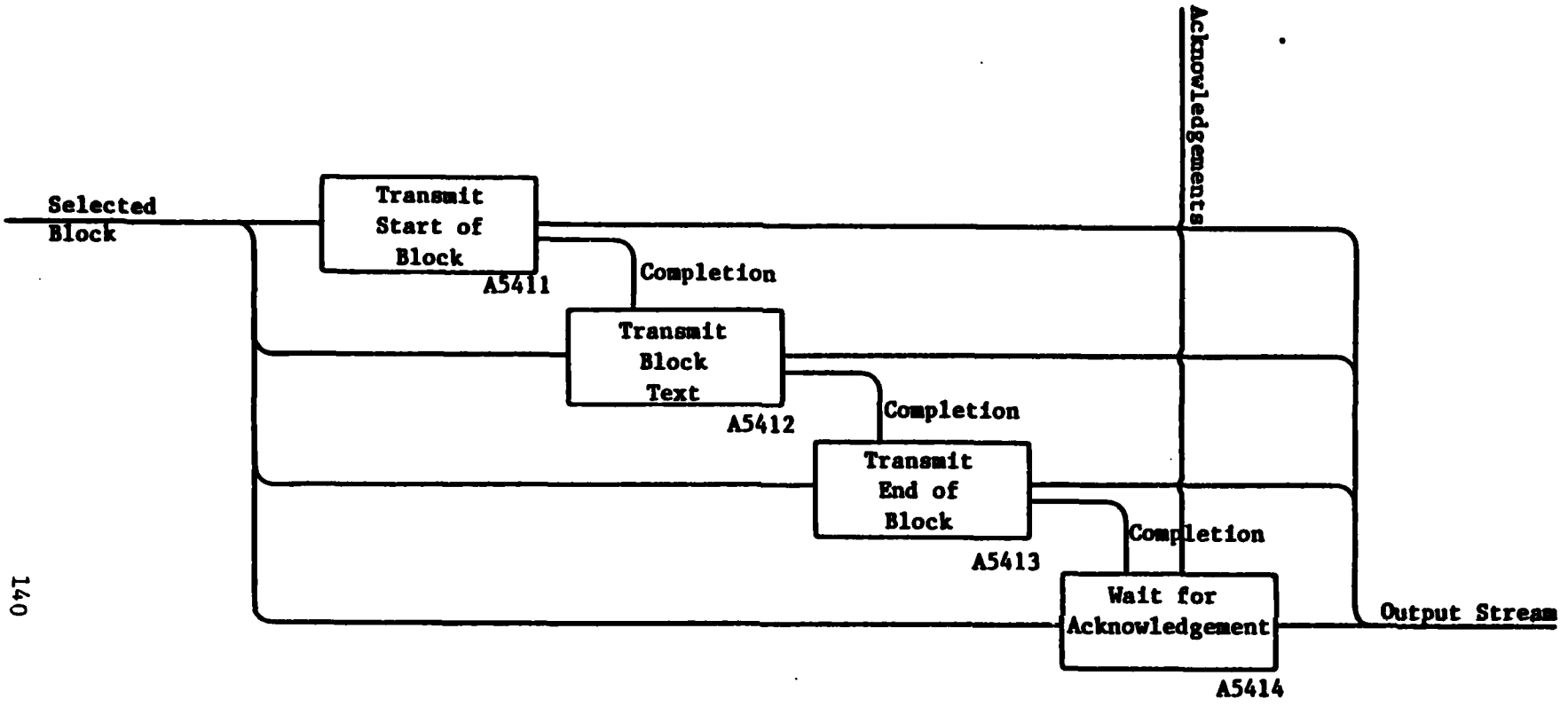
```
--
--
-- A5322
-- THE PURPOSE OF THIS PROCEDURE IS TO FRAME SYNCHRONOUS
-- MESSAGES OOUTPUT ON TRUNKS WITH THE PROPER PROTOCOL AS
-- DICTATED BY THE
-- LMF OF THE RECEIVING STATION. THIS REQUIREMENT IS
-- CONTAINED IN DCAC 370-D175-1, CHAPTER 8.
-- 12/16/81 RCR

procedure FRAME_TRUNK_MESSAGE_BLOCK_BY_BLOCK is
begin
  case SECOND_LMF_CHARACTER is
    when C|R|T|A|Q =>
      if BLOCK_COUNT = 1 then
       --> PRECEDE HEADER WITH "SOH" AND SEL CHARACTERTS
       --> FOLLOW HEADER WITH "ETB" AND "BP"
      elsif BLOCK_COUNT /= NUMBER_BLOCKS then
       --> PRECEDE BLOCK WITH "STX" AND "SEC" CHARACTERS
       --> FOLLOW IT WITH "ETB" AND "BP" CHARACTERS
      elsif BLOCK_COUNT = NUMBER_BLOCKS then
       --> PRECEDE BLOCK WITH "STX"
       --> FOLLOW BLOCK WITH "ETX" AND "BP"
      end if;
    when S =>
      if BLOCK_COUNT = 1 then
       --> PRECEDE HEADER WITH "SOH" AND "D"
       --> FOLLOW HEADER WITH "ETB" AND "EB".
      else
       --> PRECEDE BLOCK WITH "STX" AND "U"
       --> FOLLOW BLOCK WITH "ETX" AND "BP"
      end if;
  end case;
end FRAME_TRUNK_MESSAGE_BLOCK_BY_BLOCK;
EOT..
```

```
      --
      --
      -- A5323
      -- THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE FRAMING OF
      -- SYNCHRONOUS MESSAGES WITH THE PROPER PROTOCOL AS DICTATED BY
      -- THE LMF OF THE RECEIVING STATION. THIS REQUIREMENT IS
      -- CONTAINED IN DCAC 370-D175-1, CHAPTER 8.
      -- 12/11/81 RCR

      procedure FRAME_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK is
      begin
        case OUTPUT_LINE_TYPE is
          when TRIBUARY =>
            FRAME_TRIBUTARY_MESSAGE_BLOCK_BY_BLOCK;
          when TRUNK =>
            FRAME_TRUNK_MESSAGE_BLOCK_BY_BLOCK;
        end case;
        -- A BLOCK OF A SYNCHRONOUS BLOCK-BY-BLOCK MESSAGE
        -- IS READY TO TRANSMIT.
        -- SYNCHRONOUS CONTINUOUS MESSAGES CANNOT BE
        -- TRANSMITTED UNTIL ALL BLOCKS ARE FRAMED.
      end FRAME_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK;
EOT..
```

```
      --
      --
      -- A533
      -- THE PURPOSE OF THIS ROUTINE IS TO ADD PROTOCOL
      -- TO A GIVEN MESSAGE AS A FUNCTION OF:
      --       SYNCHRONOUS/ASYNCHROPNOUS
      --       LMF
      --       TRUNK/TRIBUTARY
      --       MODE OF TRANSMISSION
      -- THIS PROTOCOL IS REQUIRED BY THE FOLLOWING BY THE
      -- FOLLOWING MANUAL REFERENCES.
      --       MANUAL                    PARAGRAPH
      --  TT-B1-1101-0001A               3.2.1.2.15
      --                                 3.2.1.2.8.6
      --  DCAC 370-D175-1                CHAPTER 5,8,11
      --  12/15/81 RCR

      procedure ADD_PROTOCOL is
      begin
        case TYPE_OF_TRANSMISSION is
          when ASYNCHRONOUS =>
            -- THE BASIC UNIT OF TRANSMISSION IS THE MESSAGE.
            -- IT IS FRAMED BY START OF MESSAGES AND END OF MESSAGE
            -- SEQUENCES AS DEFINED IN JANAP-128 AND ACP-127.
            -- THE ASYNCHRONOUS COORDINATION IS A DERIVATIVE
            -- OF MODE I SYNCHRONOUS CONTROL.
            FRAME_ASYNCHRONOUS_MESSAGE;
          when SYNCHROUNOUS =>
            -- THE BASIC UNIT OF TRANSMISSION FOR SYNCHRONOUS
            -- MESSAGES IS AN 80 CHARACTER(OR LESS) BLOCK
            -- FRAMED BY SPECIAL CHARACTERS. THE PROTOCOL
            -- DEFINES UNIQUE FRAMING FOR "FIRST", "INTERMEDIATE"
            -- AND "LAST" BLOCKS.
            BLOCK_FRAMED := 1;
            while BLOCKED_FRAMED <= NUMBER_BLOCKS loop
              FRAME_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK;
              BLOCK_FRAMED := BLOCKED_FRAMED + 1;
            end loop;
            -- SINCE ALL BLOCKS ARE FRAMED , NOW
            -- READY TO TRANSMIT SYNCHRONOUS CONTINUOUS MESSAGES
        end case;
      end;
EOT..
```



A54 SEND MESSAGE



A541 TRANSMIT A BLOCK

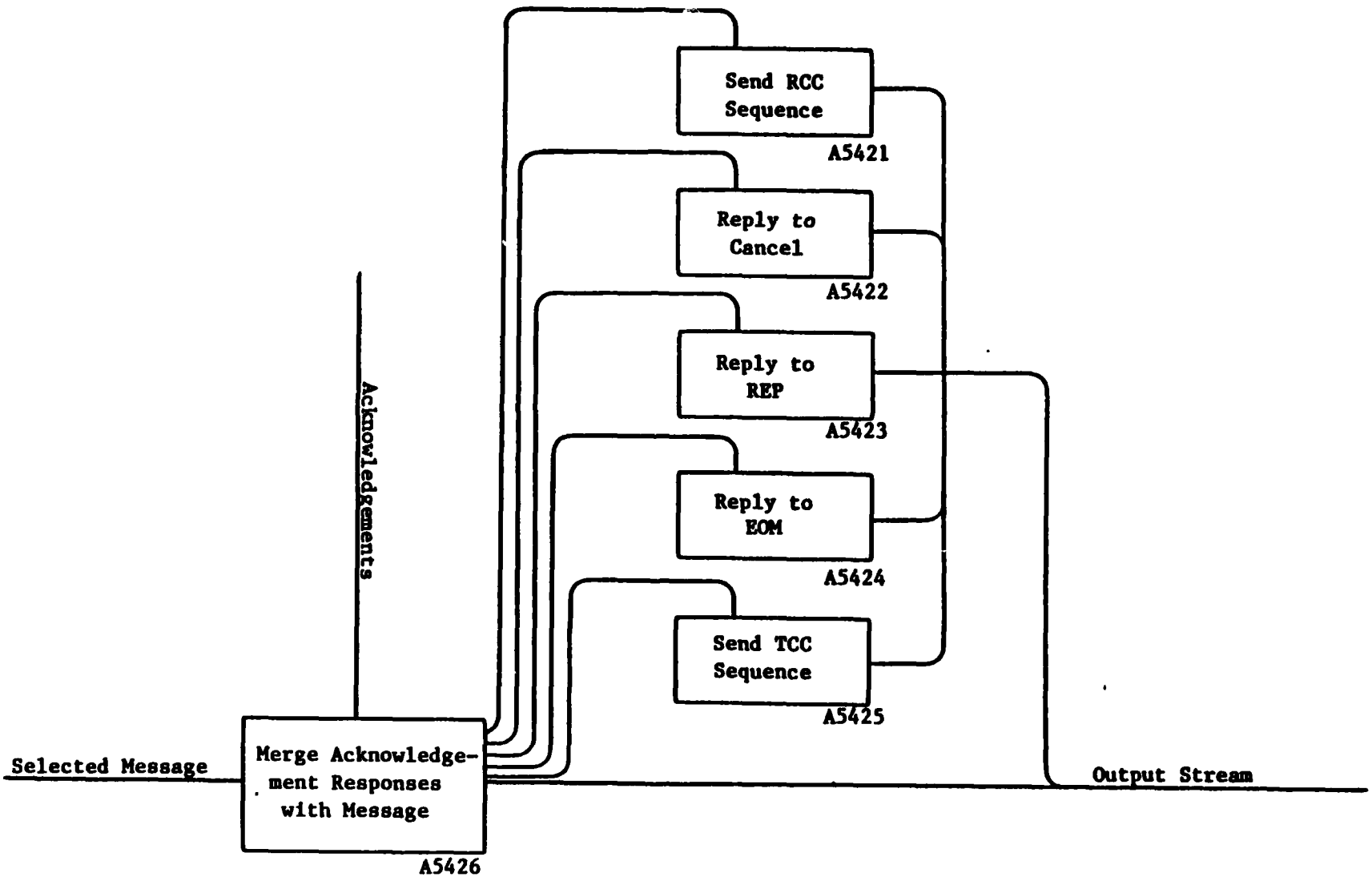
```
      --
      --
      -- A541
      -- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
      -- FOR THE TRANSMISSION OF A BLOCK OF A BLOCK-BY-BLOCK SYNCHRO-
      -- NOUS MESSAGE AS DESCRIBED BY THE FOLLOWING:
      --          MANUAL                PARAGRAPH
      --   TT-B1-1101-0001A             3.2.1.2.15
      --   DCAC 370-D175-1              CHAPTER 5
      -- 02/03/81 RCR
        procedure TRANSMIT_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK is
        begin
         loop
          loop
           GENERATE_CONTROL_CHARACTERS;
           if BLOCK_READY_TO_SEND then
            exit;
           else
            --> SEND SYNC CHARACTERS
           end if;
          end loop;
          TRANSMIT_START_OF_BLOCK;
          TRANSMIT_BLOCK_TEXT;
          if SYNC_DATA_MODE /= BLOCK_BY_BLOCK then
           if ACK_RECEIVED_FOR_LAST_BLOCK /= TRUE then
            TRANSMIT_CONTROL;
           end if;
          end if;
          GENERATE_CONTROL_CHARACTERS;
          TRANSMIT_END_OF_BLOCK;
          if SYNC_DATA_MODE = BLOCK_BY_BLOCK then
           TRANSMIT_CONTROL;
          end if;
          WAIT_FOR_BLOCK_ACKOWLEDGEMENT;
          -- START AND END OF MESSAGE JOURNAL ENTRIES MAY
          -- BE MADE AT THIS POINT
          if BLOCK_COUNT = 1 then
           OUTPUT_JOURNAL_ENTRY_TYPE := START_OF_MESSAGE_OUT_CODE;
          elsif BLOCK_COUNT = NUMBER_BLOCKS then
           OUTPUT_JOURNAL_ENTRY_TYPE := END_OF_MESSAGE_OUT_CODE;
          end if;
          --> GENERATE JOURNAL ENTRY
         end loop;
        end;
EOT..
```

```
    -- A5411
    -- THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE
    -- REQUIREMENTS FOR THE TRANSMISSION OF THE START OF
    -- A SYNCHRONOUS BLOCK AS DESCRIBED BY THE FOLLOWING:
    --              MANUAL              PARAGRAPHS
    --         TT-B1-1101-0001A         3.2.1.2.15
    --         DCAC 370-D175-1          CHAPTER 5
    -- 02/03/81 RCR

       procedure TRANSMIT_START_OF_MESSAGE is
       begin
        -- THE FIRST TWO FRAMING CHARACTERS WERE INSERTED
        -- IN THE TEXT ACCORDING TO MODULE A532
         --> TRANSMIT FRAMING CHARACTERS
       end;
EOT..
```

```
      -- A5412
      -- THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE
      -- REQUIREMENTS FOR THE TRANSMISSION OF THE TEXT PORTION
      -- OF A MESSAGE BLOCK AS CONTAINED BY THE FOLLOWING:
      --            MANUAL                  PARAGRAPH
      --       TT-B1-1101-0001A             3.2.1.2.15
      --       DCAC 370-D175-1              CHAPTER 5-4(d)
      -- 02/03/81 RCR
         procedure TRANSMIT_BLOCK_TEXT is
         begin
          CHARACTERS_THIS_BLOCK := 2;
          -- COUNTER INCLUDES TWO FRAMING CHARACTERS
          loop
           GENERATE_CONTROL_CHARACTERS;
           if CURRENT_CHARACTER = TEXT_CHARACTER|MODE_CONTROL|
                                  END_OF_MESSAGE then
            CHARACTERS_THIS_BLOCK := CHARACTERS_THIS_BLOCK +1;
            --> UPDATE PARITY OF BLOCK
            --> SEND CURRENT CHARACTER
            if CHARACTERS_THIS_BLOCK = 82 or
                                       CURRENT_CHARACTER = EOM then
             exit;
            end if;
           end if;
          end loop;
         end;
EOT..
```

```
     -- A5413
     -- THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE
     -- REQUIREMENTS FOR THE TRANSMISSION OF THE END-OF-BLOCK
     -- SEQUENCE AS CONTAINED IN THE FOLLOWING:
     --           MANUAL              PARAGRAPH
     --       TT-B1-1101-0001A        3.2.1.2.15
     --       DCAC 370-D175-1         CHAPTER 5-4(e)
     -- 02/03/81 RCR
        procedure TRANSMIT_END_OF_BLOCK is
        begin
         --> SEND END-OF-BLOCK SEQUENCE
         --> SEND BLOCK PARITY
         --> SET EOMS_SENT_CODE
         GENERATE_CONTROL_CHARACTERS;
        end;
EOT..
```

```
-- A5414
-- THE PURPOSE OF THIS PROCEDURE IS TO DESCRIBE THE
-- REQUIREMENTS FOR THE RECEIPT OF ACKNOWLEDGEMENTS
-- TECHNIQUE AS CONTAINED IN THE FOLLOWING:
--          MANUAL              PARAGRAPH
--      TT-B1-1101-0001A        3.2.1.2.15
--      DCAC 370-D175-1         CHAPTER 5-4(e)
--                              CHAPTER 3-4(a)
-- 02/03/81 RCR
   procedure WAIT_FOR_BLOCK_ACKNOWLEDGEMENT is
   begin
    GENERATE_CONTROL_CHARACTERS;
    -- SATISFIES THE FOLLOWING REQUIREMENTS:
    -- ACK_RECEIVED := FALSE;
    -- TIMER_TRYS := 0;
    -- while TIMER_TRYS <= 3 loop
    --  RESET ACK_TIMER TO MAX VALUE
    --   (A DECREASING COUNTER ASSUMED)
    --  while ACK_TIMER > 0 loop
    --   if CONTROL_CHARACTERS_TO_BE_SENT then
    --    SEND INDICATED CONTROLS
    --   else
    --    SEND SYNC CHARACTERS
    --   end if;
    --   if ACK_RECEIVED then
    --    return;
    --   end if;
    --  end loop;
    --  TIMER_TRYS := TIMER_TRYS +1
    --  SEND REPLY CHARACTER
    -- end loop;
    -- TIMER HAS EXPIRED THREE TIMES
    -- ACTIVATE ALARM
   end WAIT_FOR_BLOCK_ACKNOWLEDGEMENT;
EOT..
```



A542 TRANSMIT ASYNCHRONOUS MESSAGE

```
-- A5421
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR SENDING A RCC , RECEIVER CONTROL CHARACTER , SEQUENCE
-- AS DESCRIBED IN THE FOLLOWING:
--                MANUAL                    PARAGRAPH(S)
--          DCAC 370-D175-1                    11-28
-- 01/19/82 RCR
   procedure SEND_RCC_SEQUENCE is
    begin
     GENERATE_CONTROL_CHARACTERS;
--    THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:
--      if SEND_GEN_CODE = FALSE then
--       return;
--      elsif STOP_CODE = TRUE then
--       SEND_STOP_SEQUENCE;
--      elsif RT_CODE = TRUE then
--       SEND_RT_SEQUENCE;
--       else if ACK_ALTERNATE = ACK_1 then
--        SEND_ACK1_SEQUENCE;
--       else
--        SEND_ACK2_SEQUENCE;
--       end if;
--      end if;
--      RCC_TO_SEND := FALSE;
--      SEND_GEN_CODE := FALSE;
   end SEND_RCC_SEQUENCE;
EOT..
```

```
-- A5422
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR REPLYING TO CANCEL COMMAND DESCRIBED IN THE FOLLOWING:
--                MANUAL                    PARAGRAPH(S)
--             DCAC 370-D175-1                 11-29
-- 01/14/82  RCR
   procedure REPLY_TO_CAN is
    begin
     GENERATE_CONTROL_CHARACTERS;
--    THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:
--      if STOP_RECD_CODE = TRUE then
--       STOP_RECD_CODE := FALSE;
--       THREE_RPT_COUNTER := 0;
--       elsif ACK1_RECD_CODE = TRUE then
--        CAN_CODE := FALSE;
--        REPLY_TIMER := 0;
--        THREE_RPT_COUNTER := 0;
--       if CAN_SENT_WITHIN_MESSAGE then
--        --> RETURN MESSAGE TO QUEUE
--        --> PREPARE TO SEND CANTRAN
--       end if;
--      end if;
--      if REPLY_TIMER_EXPIRED then
--       --> RESET REPLY TIMER
--       if THIRD_CAN_SENT_W/O RESPONSE then
--        THREE_RPT_ALARM := TRUE;
--        TCC_TO_SEND := TRUE;
--       else
--        TCC_TO_SEND := TRUE;
--       end if;
--      else
--       STOP_RECD_CODE := FALSE;
--       THREE_RPT_COUNTER   := 0;
--      end if;
   end REPLY_TO_CAN;
EOT..
```

```
      -- A5423
      -- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
      -- FOR REPLYING TO A "REP" OR "STOP" COMMAND AS DESCRIBED IN
      -- THE FOLLOWING:
      --       MANUAL                    PARAGRAPH(S)
      -- DCAC 370-D175-1                    11-30
      -- 02/04/82
      procedure REPLY_TO_REP_OR_STOP is
      begin
        GENERATE_CONTROL_CHARACTERS;
--        THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:
--        if STOP_RECD_CODE = TRUE then
--           STOP_RECD_CODE := FALSE;
--           THREE_RPT_COUNTER := 0;
--        elsif ACK_RCD_CODE = FALSE then
--           REP_CODE := FALSE;
--           REPLY_TIMER := 0;
--           THREE_RPT_COUNTER := 0;
--           if RCVD_ACK = TRANS_ACK_ALT then
--             if ACK_TO_COMPLETE_MSG = TRUE then
--               PREPARE_TO_COMPLETE_MSG := TRUE;
--             else
--               CAN_CODE := TRUE;
--           end if;
--           exit;
--         elsif RT_RECD_CODE = TRUE then
--           CAN_CODE := TRUE;
--           REPLY_TIMER := 0;
--           THREE_RPT_COUNTER := 0;
--           exit;
--         end if;
--         if REPLY_TIMER \= 0 then
--           exit;
--         else
--           REPLY_TIMER := 0;
--           TCC_TO_SEND := TRUE;
--           if THIRD_REP_SENT_W/O_RESPONSE then
--             THREE_RPT_ALARM := TRUE;
--           end if;
--         end if;
      end REPLY_TO_REP_OR_STOP;

EOT..
```

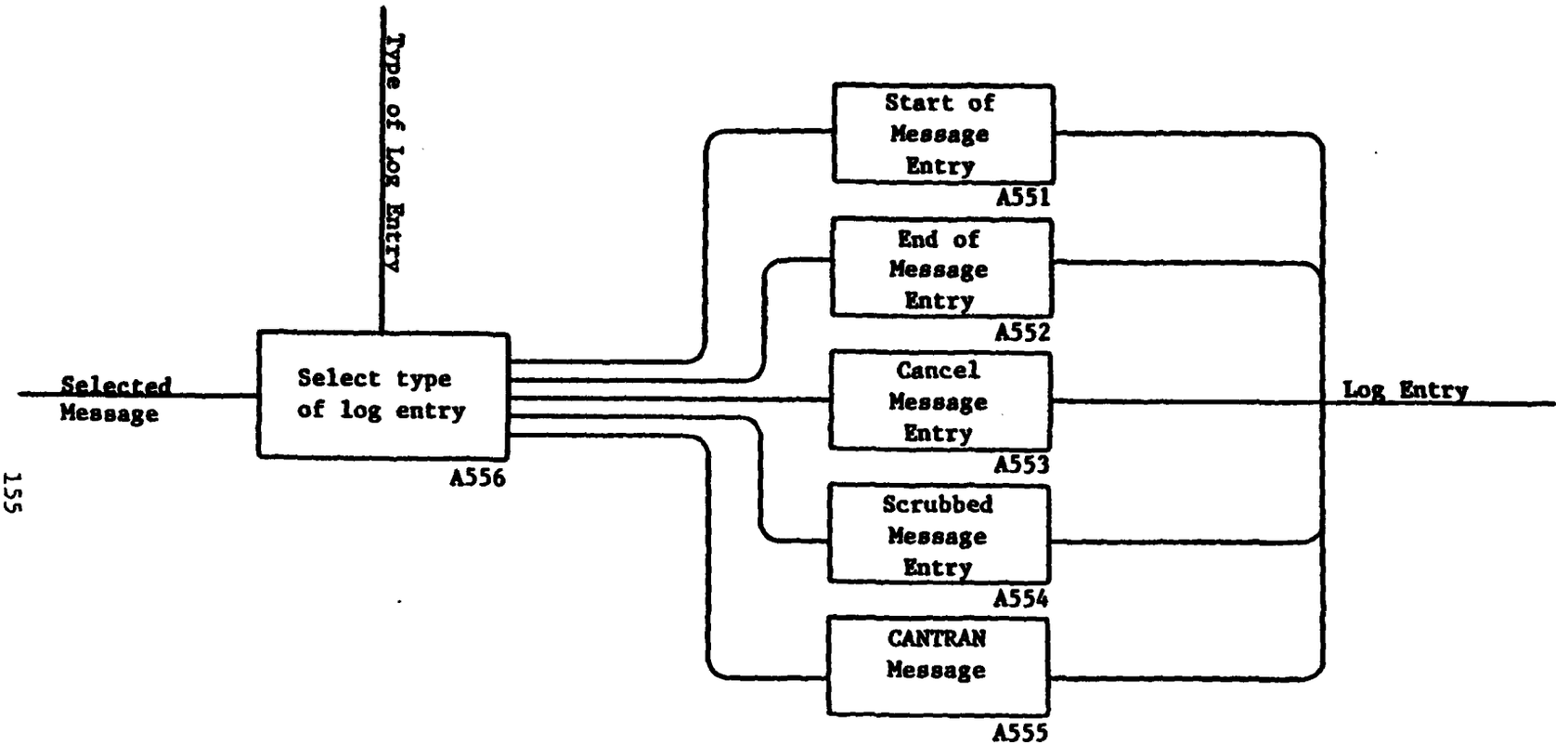
```
-- A5424
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR REPLYING TO EOM COMMAND AS DESCRIBED IN THE FOLLOWING:
--              MANUAL                    PARAGRAPH(S)
--          DCAC 370-D175-1                   11-31
-- 01/19/82  RCR
   procedure REPLY_TO_EOM is
    begin
     GENERATE_CONTROL_CHARACTERS;
--     THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:
--      if STOP_RECD_CODE = TRUE then
--       STOP_RECD_CODE := FALSE;
--       THREE_RPT_COUNTER := 0;
--      elsif ACK_RECD_CODE = TRUE then
--       REPLY_TIMER := 0;
--       if RECD_ACK = TRANS_ACK_ALT then
--        PREPARE_TO_SEND_NEXT_MESSAGE := TRUE;
--       else
--        CAN_CODE := TRUE;
--       end if;
--       exit;
--      elsif RT_RECD_CODE = TRUE then
--       REPLY_TIMER := 0;
--       CAN_CODE := TRUE;
--       if REPLY_TIMER_EXPIRED then
--        REPLY_TIMER := 0;
--        REP_CODE := TRUE;
--        TCC_TO_SEND := TRUE;
--       end if;
--      end if;
   end REPLY_TO_EOM;
EOT..
```

```
-- A5425
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR SENDING A TCC SEQUENCE AS DESCRIBED IN THE FOLLOWING:
--                MANUAL                    PARAGRAPH(S)
--            DCAC 370-D175-1                  11-32
-- 01/15/82   RCR
   procedure SEND_TCC_SEQUENCE is
    begin
     GENERATE_CONTROL_CHARACTERS;
--    THIS PROCEDURE SATISFIES THE FOLLOWING REQUIREMENTS:
--      if SEND_GEN_CODE = TRUE then
--       SEND_REP_SEQUENCE;
--       elsif CAN_CODE = TRUE then
--        SEND_CAN_SEQUENCE;
--       end if;
--      end if;
--      if REP_CODE = TRUE and CAN_CODE = TRUE then
--       THREE_RPT_COUNTER := THREE_RPT_COUNTER + 1;
--       if ACK1 = TRUE then
--        ACK1 := FALSE;
--       end if;
--       if ACK2 = TRUE then
--        ACK2 := FALSE;
--       end if;
--       if RT_RECD_CODE = TRUE then
--        RT_RECD_CODE := FALSE;
--       end if;
--       if STOP_RECD_CODE = TRUE then
--        STOP_RECD_CODE := FALSE;
--       end if;
--       START_REPLY_TIMER;
--      else
--       SEND_START_SEQUENCE;
--      end if;
--      TCC_TO_SEND := FALSE;
--      SEND_GEN_CODE := FALSE;
--     end if;
  end SEND_TCC_SEQUENCE;
EOT..
```

```
-- A5426
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR THE TRANSMISSION OF THE TEXT PORTION OF A MESSAGE
-- AS DESCRIBED IN THE FOLLOWING:
--                MANUAL                    PARAGRAPH(S)
--          DCAC 370-D175-1                   11-24,25
-- 01/19/82
   procedure SELECT_ACKNOWLEDGEMENT_RESPONSE is
    begin
     CAN_CODE := TRUE;
     loop
      loop
       if PAUSE_GENERATED = TRUE then
        SEND_GEN_CODE := TRUE;
        SEND_DET_CODE := FALSE;
       elsif SEND_DETECT_GENERATED = TRUE then
        SEND_DET_CODE := TRUE;
       end if;
       if RCC_TO_SEND = TRUE then
        SEND_RCC_SEQUENCE;
        exit;
       end if;
       if REPLY_TIMER_RUNNING = TRUE then
        if TIMING_REPLY_TO_CAN then
         REPLY_TO_CAN;
         exit;
         else if TIMING_REP_OR_STOP then
          REPLY_TO_REP_OR_STOP;
          exit;
         else
          REPLY_TO_EOM;
          exit;
         end if;
        end if;
       end  if;
        if TCC_TO_SEND = TRUE then
         SEND_TCC_SEQUENCE;
         exit;
        end if;
        if STOP_RECD = TRUE then
         --> START REPLY TIMER
         exit;
         elsif CAN_CODE = TRUE then
          exit;
          elsif MESSAGE_CHAR_TO_SEND \= TRUE then
           if CANTRAN_CHAR_TO_SEND \= TRUE then
            exit;
           end if;
        end if;
   --   ********************************************
   --   connector 2 of requirements starts here
   --   ********************************************
        if SEND_GEN_CODE = TRUE then
         --> SEND ST ART SEQUENCE
         SEND_GEN_CODE := FALSE;
```

```
            elsif SEND_DET_CODE = TRUE then
             exit;
           end if;
           --> SEND A CHARACTER
           if LAST_CHAR_OF_EOMS then
            --> COMPLIMENT TRANSMIT ACK ALTERNATOR
            --> SET ALL RECEIVE CODES
            --> START REPLY TIMER
            REPLY_TO_EOM;
            exit;
           end if;
          end loop;
          if PREPARE_TO_SEND_NEXT_MESSAGE = TRUE then
           exit;
          end if;
         end loop;
        end SELECT_ACKNOWLEDGEMENT_RESPONSE;
EOT..
```

```
-- A543
-- THE PURPOSE OF THIS ROUTINE IS TO TRANSMIT MESSAGES AS
-- A FUNCTION OF THE MODE OF TRANSMISSION OF THE LINE IN
-- USE. THIS REQUIREMENT IS CONTAINED IN THE FOLLOWING:
--          MANUAL                  PARAGRAPH(S)
--    TT-B1-1101-0001A              3.2.1.2.15
--    DCAC 370 D175-1               CHAPTER 5
--    12/16/81 RCR
      procedure TRANSMIT_MESSAGE is
      begin
        case LINE_TRANSMISSION_TYPE is
          when SYNCHRONOUS =>
           if SYNCHRONOUS_MODE = SYNCHRONOUS_BLOCK_BY_BLOCK then
             -- TRANSMIT BLOCKED WHICH WAS JUS FRAMED
             BLOCK_COUNT := BLOCK_FRAMED;
             TRANSMIT_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK;
           elsif SYNCHRONOUS_MODE = SYNCHRONOUS_CONTINUOUS then
             BLOCK_COUNT := 1;
             while BLOCK_COUNT <= NUMBER_BLOCKS loop
               -- TRANSMIT BLOCK INDICATED BY BLOCK_COUNT
               TRANSMIT_SYNCHRONOUS_MESSAGE_BLOCK_BY_BLOCK;
               BLOCK_COUNT := BLOCK_COUNT + 1;
             end loop;
           end if;
          when ASYNCHRONOUS =>
            TRANSMIT_ASYNCHRONOUS_MESSAGE;
          when OTHERS => ERROR_CODE := INVALID_TRANSMISSION_CODE;
        end case;
      end TRANSMIT_MESSAGE;
EOT..
```


Type of Log Entry
Selected Message
Select type of log entry
A556
Start of Message Entry
A551
End of Message Entry
A552
Cancel Message Entry
A553
Scrubbed Message Entry
A554
CANTRAN Message
A555
Log Entry

```
-- A551
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR START OF MESSAGE JOURNAL ENTRIES AS DESCRIBED IN THE FOLOWING:
--                MANUAL                      PARAGRAPH(S)
--          TT-B1-1101-0001A              3.2.1.2.14.1.2(a)
-- 12/16/81 RCR
   procedure START_OF_MESSAGE_OUT_JOURNAL_ENTRY is
     begin
      OUTGOING_LOG_ENTRY := SOM_OUT_ENTRY;
      -- CREATE A JOURNAL ENTRY WHICH CONTAINS:
      --    OUTPUT CHANNEL ID
      --    CHANNEL SEQUENCE NUMBER, IF APPLICABLE
      --    CHANNEL MODE OF OPERATION
      --    ROUTING INDICATOR
      --    HEADER INFORMATION
      --    TIME SOM TRANSMITTED
      --    S&F ASSIGNED SERIAL NUMBER
     end START_OF_MESSAGE_OUT_JOURNAL_ENTRY;
EOT..
```

```
-- A552
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR END OF MESSAGE JOURNAL ENTRIES AS DESCRIBED IN THE FOLOWING:
--                MANUAL                    PARAGRAPH(S)
--          TT-B1-1101-0001A              3.2.1.2.14.1.2(b)
-- 01/19/82 RCR
   procedure END_OF_MESSAGE_OUT_JOURNAL_ENTRY is
     begin
      OUTGOING_LOG_ENTRY := EOM_OUT_ENTRY;
      -- CREATE A JOURNAL ENTRY WHICH CONTAINS:
      --    ENTRY IDENTIFICATION
      --    OUTPUT CHANNEL ID
      --    TIME OF TRANSMISSION
      --    CHANNEL SEQUENCE NUMBER
      --    DESTINATIONS FOR THIS MESSAGE
      --    S&F ASSIGNED SERIAL NUMBER
      --    BLOCK COUNT
     end END_OF_MESSAGE_OUT_JOURNAL_ENTRY;
EOT..
```

```
-- A553
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR CANCEL MESSAGE JOURNAL ENTRIES AS DESCRIBED IN THE FOLOWING:
--                MANUAL                    PARAGRAPH(S)
--          TT-B1-1101-0001A              3.2.1.2.14.1.2(c)
-- 01/19/82 RCR
   procedure CANCEL_MESSAGE_OUT_JOURNAL_ENTRY is
     begin
      OUTGOING_LOG_ENTRY := CANCEL_OUT_ENTRY;
      -- CREATE A JOURNAL ENTRY WHICH CONTAINS:
      --     CHANNEL ID AND CSN
      --     CHANNEL MODE OF OPERATION
      --     TIME OF CANCELLATION
      --     REASON FOR CANCELLING
      --     ROUTING INDICATORS OF MESSAGE
      --     S&F MODULE ASSIGNED SERIAL NUMBER
      --     BLOCK COUNT
     end CANCEL_MESSAGE_OUT_JOURNAL_ENTRY;
EOT..
```

```
-- A554
-- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE REQUIREMENTS
-- FOR SCRUBBED MESSAGE JOURNAL ENTRIES AS DESCRIBED IN THE FOLOWING:
--                MANUAL                    PARAGRAPH(S)
--          TT-B1-1101-0001A              3.2.1.2.14.1.2(d)
-- 01/19/82 RCR
   procedure SCRUBBED_MESSAGE_OUT_JOURNAL_ENTRY is
     begin
      OUTGOING_LOG_ENTRY := SCRUB_ENTRY;
      -- CREATE A JOURNAL ENTRY WHICH CONTAINS:
      --    CHANNEL ID AND CSN
      --    CHANNEL MODE OF OPERATION
      --    TIME SCRUBBED
      --    S&F MODULE ASSIGNED SERIAL NUMBER
      --    ROUTING INDICATORS OF MESSAGE
      --    REASON FOR SCRUB
      --    BLOCK COUNT
      if MESSAGE_OUT_SENT_TO_OVERFLOW then
       -- INCLUDE OVERFLOW MESSAGE SERIAL NUMBER
      end if;
      if MESSAGE_SENT_TO_INTERCEPT then
       -- INCLUDE INTERCEPT MESSAGE SERIAL NUMBER
      end if;
     end SCRUBBED_MESSAGE_OUT_JOURNAL_ENTRY;
EOT..
```

```
        -- A556
        -- THE PURPOSE OF THIS ROUTINE IS TO DESCRIBE THE
        -- REQUIREMENTS FOR JOURNAL ENTRY CREATION AS A RESULT
        -- OF MESSAGE OUTPUT PROCCESSING AS DESCRIBED IN THE
        -- FOLLOWING:
        -- MANUAL PARAGRAPH(S)
        -- TT-B1-1101-0001A 3.2.1.2.14.1.2
        -- 12/15/81

        procedure UPDATE_JOURNAL is
        begin
          case JOURNAL_ENTRY_TYPE is
            when START_OF_MESSAGE_CODE =>
              START_OF_MESSAGE_OUT_JOURNAL_ENTRY;
            when END_OF_MESSAGE_CODE =>
              END_OF_MESSAGE_OUT_JOURNAL_ENTRY;
            when CANCEL_MESSAGE_CODE =>
              CANCEL_MESSAGE_OUT_JOURNAL_ENTRY;
            when SCRUBBED_MESSAGE_CODE =>
              SCRUBBED_MESSAGE_OUT_JOURNAL_ENTRY;
            when CANTRAN_MESSAGE_CODE =>
              CANTRAN_MESSAGE_OUT_JOURNAL_ENTRY;
          end case;
        end UPDATE_JOURNAL;

    EOT..
```

O1A3 C1A5	ACKNOWLEDGMENT	= [ACK_REQUEST¦ACK_RECEIVED] * USED TO PASS REQUEST TO SEND ACK OR TO PASS ACKS WHICH HAVE BEEN RECEIVED TO THE TRANSMIT FUNCTION *
	ACP_FMT_LN_1	= 'VZCZC' + ICD + FIGS + ICSN + LTRS + <5>('b') + ([UU¦HH]) + LN_END * TRANSMISSION IDENT (TI) *
	ACP_FMT_LN_2	= (BELL_SIGNAL) + DOUBLE_PRECEDENCE_PROSIGN + <..>('b' + RI) + LN_END
	ACP_FMT_LN_3	= 'DEb' + RI + 'b' + ('#') + ACP_SSN + [DATE_TIME¦DATE + '/' + TIME + 'Z'] + LN_END
	ACP_FMT_LN_15	= (CORRECTION) + (EOM_SEQ)
	ACP_HEADER	= ACP_FMT_LN_1 + ACP_FMT_LN_2 + ACP_FMT_LN_3 + (FMT_LN_4) + (FMT_LN_5) + (FMT_LN_6) + (FMT_LN_7) + (FMT_LN_8) + (FMT_LN_9) + (FMT_LN_10)
	ACP_MESSAGE	= ACP_HEADER + SEPARATOR + BODY + SEPARATOR + ACP_TRAILER
	ACP_PILOT	= ACP_FMT_LN_2 + FMT_LN_4 + ICD + ICSN + <5>('b') + ([UU¦HH]) + LN_END
	ACP_SSN	= <3>(DIGIT) + (LETTER) * ORIGINATING STATION SERIAL NUMBER *
	ARRI	= <7>(['b'¦LETTER])
	ACP_TRAILER	= (FMT_LN_14) + ACP_FMT_LN_15

```
ASN                              = TBD
                                   * ASSIGNED SER NO
                                   USED TO DIFFERENTIATE A
                                   PARTICULAR MESSAGE FROM ALL
                                   OTHER SYSTEM MESSAGES *

ASYNCHRONOUS_MODE                = [ NORMAL ¦ STEPPED ]
                                   *TRANSMISSION METHOD*

ASYNCH_RECEIVE_CONTROL_CHAR      = [ACK_1¦ACK_2¦STOP¦RT]
                                   * CHARACTER *

BELL_SIGNAL                      = FIGS + <5>('J') +
                                   <5>('S') + LTRS

BODY                             = <0..>(CHARACTER)
                                   * TEXT OF MESSAGE *

BP                               = CHARACTER
                                   * BLOCK PARITY OF MESSAGE
                                     FORMED BY THE BINARY
                                     ADDITION WITHOUT CARRY
                                     (XOR) OF THE BITS IN EACH
                                     ROW OF A BLOCK AND PLACED
                                     IN THE LAST FRAMING
                                     CHARACTER OF A
                                     SYNCHRONOUS MESSAGE. *

BP_CALC                          = * CUMULATIVE BLOCK PARITY
                                     CALCULATION DURING THE
                                     SYNCHRONOUS RECEIVE
                                     PROCESS STARTING WITH THE
                                     2ND FRAMING CHAR AND
                                     ENCING WITH "ETX" OR
                                     "ETB" *

CAN_CODE                         = [TRUE ¦ FALSE ]

CANCEL_OUT_ENTRY                 = OCD + OCSN + DATE_TIME +
                                   + ASN
                                   + CHANNEL_MODE
                                   + CANCEL_REASON
                                   + <1..50>(RI)
                                   + NUMBER_BLOCKS
                                   * MADE WHEN A CANCELLATION
                                   IS TRANSMITTED *

CANCEL_REASON                    = TBD
```

```
          CANCEL_REC_ENTRY              = ICD + ICSN + DATE_TIME +
                                          CHANNEL_MODE + NUMBER_BLOCKS
                                          + ASN
                                          * MADE WHEN A CANCEL IS
                                            RECEIVED *

          CANTRAN_CHAR_TO_SEND          = [ TRUE ¦ FALSE ]

          CANTRAN_REC_ENTRY             = ICD + ICSN + DATE_TIME +
                                          CHANNEL_MODE + NUMBER_BLOCKS
                                          + ASN
                                          * MADE WHEN A CANTRAN IS
                                           RECEIVED *

          CANTRAN_OUT_ENTRY             = OCD + OCSN + DATE_TIME +
                                          CHANNEL_MODE + NUMBER_BLOCKS
                                          + ASN
                                          * MADE WHEN A CANTRAN IS
                                          TRANSMITTED *

OxA4234   CARD_CHECKED_MESSAGES         = MESSAGE + MCB +
I1A4235                                   TRANSLATION_PAIR
I1A4236

          CARD_FMT_LN_2                 = PRECEDENCE + LMF_PAIR +
                                          CLASS + CIC_CAI + 'b' +
                                          OSRI + OSSN + 'b' +
                                          DATE_TIME + 'b' +
                                          RECORD_COUNT + '-' +
                                          '--' + RI + <0..49>(RI) +
                                          '.'

C1A4234   CARD_INFO                     = <28>([FROM_CARD¦TO_CARD¦NO])
                                          * TABLE KEYED BY
                                          TRANSLATION_PAIR.
                                          (CA¦CT¦CT' => FROM_CARD,
                                          TC¦QC¦AC¦FC => TO_CARD,
                                          OTHERS => NO) *

          CARD_MESSAGE                  = (JANAP_FMT_LN_1) +
                                          (JANAP_PILOT) +
                                          [CARD_FMT_LN_2 +
                                          <1..500>(<80>(CHARACTER)) +
                                          EOT_CARD¦SINGLE_CARD]

O1A4235   CARD_TRANSLATED_MESSAGES      = MESSAGE + MCB +
O1A4236                                   TRANSLATION_PAIR
IxA4237

          CHANNEL_MODE                  = [1¦2¦3¦4¦5]
```

	CHAR	= SEE CHARACTER
OxA4231 I1A4232 I1A4233	CHAR_CHECKED_MESSAGE	= MESSAGE + MCB + TRANSLATION_PAIR
O1A4232 O1A4233 IxA4234	CHAR_TRANSLATED_MESSAGE	= MESSAGE + MCB + TRANSLATION_PAIR
C1A4231	CHAR_SET_INFO	= <28>([ITA¦ASCII¦NO]) * TABLE KEYED BY TRANSLATION_PAIR. (CT¦CT'¦AT¦$\overline{AT}$' => ITA, TC¦TA¦QC¦QA¦FC¦FA => ASCII, OTHERS => NO) *
	CHARACTER	= LEVEL + [TEXT_CHARACTER ¦ CONTROL_CHARACTER]
	CHARACTER_PARITY	= ASCII + [EVEN¦ODD]
	CHNL_DES	= <3>(LETTER) * CHANNEL DESIGNATOR *
	CIC_CAI	= <3>(LETTER) + [LETTER¦DIGIT] * CONTENT INDICATOR CODE -- CONTENT ACTION INDICATOR *
	CLASS	= [M¦A¦T¦S¦C¦R¦E¦U] * CLASSES ARE: M -- DSSCS A -- SPECAT T -- TOP SECRET S -- SECRET C -- CONFIDENTIAL R -- RESTRICTED E -- EFTO (ENCRYPT FOR TRANSMISSION ONLY) U -- UNCLASSIFIED *
	CLASS_X5	= <5>(CLASS)
C1A411	COLLECTIVE_RI_TABLE	= <0..200>(RI + <1..50>(LINE_NO))
O1A411 I1A412	COLLECTIVE_ROUTED_MESSAGES	= MESSAGE + MCB + <1..50>(OUTPUT_LINE NUMBER)

```
         COMMUNITIES_SERVED          = (R) + (U) + (Y)

         CONTROL_CHARACTER           = [ASCII.NUL..ASCII.US|
                                        ASCII.DEL|ITA.LTRS|
                                        ITA.FIGS|ITA.CR|ITA.CR|
                                        ITA.LF|ITA.BLANK]

         CONTROL_CODE                = [SEND_STOP|RT_RCVD|SEND_RT|
03A31                                   EOMS_RCVD|STOP_RCVD|
01A314                                  ACK_1_RCVD|ACK_2_RCVD|
01A315                                  CANCEL_RCVD|REPLY_RCVD|
01A3141                                 SEND_RM|SEND_ACK|SEND_NAK|
01A3143                                 WBT_RCVD|RM_RCVD|NAK_RCVD|
01A3144                                 INVALID_RCVD|
01A3153                                 INVALID_CHAR |
01A3156                                 ENQUIRY_RCVD]
01A3157                                 * USED TO SIGNAL CONDITIONS
                                          TO GENERATE_CONTROL_
                                           CHARACTERS *

         CORRECTION                  = 'C' + <..>(CHARACTER) +
                                       LN_END

01A121   CURRENT_LIST                = TBD
I1A122

         DATA_MODE                   = FORMAT + LEVEL

         DATA_STATE                  = [TRUE|FALSE]
                                     * TRUE, SYNCHRONOUS RECEIVER
                                       WILL ACCEPT TEXT CHARACTERS
                                       FALSE, WILL ACCEPT CONTROL
                                       AND SYNC CHARACTERS *

         DATE                        = <2>(DIGIT)
                                       * DAY OF MONTH *

         DATE_TIME                   = <7>DIGIT
                                       * JULIAN DATE PLUS
                                         ZULU TIME *

0xA4224  DELS_CHECKED_MESSAGE        = MESSAGE + MCB +
I1A4225                              TRANSLATION_PAIR

C1A4224  DELS_INFO                   = <28>([YES|NO])
                                       * TABLE KEYED BY
                                       TRANSLATION_PAIR
                                       (AC|AT|AT' => YES,
                                       OTHERS => NO ) *
```

```
                DIRECTION                       = (INPUT) + (OUTPUT)
                                                  * BOTH SET IMPLIES 2-WAY *

                DOUBLE_PRECEDENCE_PROSIGN       = <2>(PRECEDENCE)
                                                  * USED IN
                                                  ACP_FMT_LN_2 *

                ECSN                            = <3>(DIGIT)
                                                  * EXPECTED CHANNEL
                                                  SEQUENCE NUMBER *

                EOM_IN_ENTRY                    = ICD + ICSN + DATE_TIME +
                                                  ASN + NUMBER_BLOCKS +
                                                  CHANNEL_MODE
                                                  * MADE WHEN EOM IS
                                                  RECEIVED *

                EOM_OUT_ENTRY                   = OCD + OCSN + DATE_TIME +
                                                  ASN + <1..50>(RI) +
                                                  NUMBER_BLOCKS
                                                  * MADE WHEN EOM IS
                                                  TRANSMITTED *

                EOM_SEQ                         = <2>(CR) + <8>(LF) +
                                                  <4>('N') + <8>(LTRS)

                EOM_VALIDATION                  = '#' + <4>(DIGIT)

                EOT_CARD                        = PRECEDENCE + LMF_PAIR +
                                                  CLASS + CIC_CAI + 'b' + OSRI
                                                  + OSSN + 'b' + DATE_TIME +
                                                  'b' + RECORD_COUNT + '-' +
                                                  REDUNDANT_CLASS +
                                                  <44>('b') + <4>('N')
                                                  * N'S START IN COLUMN 72 *

                ERROR_CODE                      = [ NO_ERROR |
                                                    INVALID_LENGTH |
                                                    INVALID_LINE_RI |
                                                    INVALID_LMF |
                                                    INVALID_RI_SECURITY |
                                                    BAD_BLOCK_COUNT |
                                                    INVALID_LINE_TRANSMISSION
                                                        TYPE ]
                                                    * CODES TO INDICATE ERROR
                                                      CONDITION  *

OxA4237
I1A4238         FILL_CHECKED_MESSAGE            = MESSAGE + MCB +
                                                  TRANSLATION_PAIR
```

```
C1A4237  FILL_INFO                    = <28>([YES|NO])
                                        * TABLE KEYED BY
                                          TRANSLATION PAIR
                                          (CA|CT|CT'|AT|AT'|QA|FA =>
                                        YES, OTHERS => NO) *

         FIRST_LMF_CHARACTER          = [B|D|I|S|C|A|T|R|Q|F]

         FMT_LN_4                     = OP_SIGNAL + CLASS_X5 +
                                        <0..>(CHARACTER) + LN_END

         FMT_LN_5                     = PRECEDENCE +
                                        ('b' + PRECEDENCE)
                                        + 'b' + DATE + TIME + 'b' +
                                        MONTH + 'b' + YEAR
                                        + <0..>(CHARACTER) + LN_END

         FMT_LN_6                     = 'FMb' + <..>(CHARACTER) +
                                        LN_END

         FMT_LN_7                     = 'TOb' + <..>(CHARACTER) +
                                        LN_END

         FMT_LN_8                     = 'INFOb' + <..>(CHARACTER) +
                                        LN_END

         FMT_LN_9                     = 'XMTb' + <..>(CHARACTER) +
                                        LN_END

         FMT_LN_10                    = <..>(CHARACTER) + LN_END

         FMT_LN_14                    = <..>(CHARACTER) + LN_END

         FORMAT                       = [JANAP_128 | ACP_127]
                                        * MESSAGE FORMAT *

OxA4221  FORMAT_CHECKED_MESSAGE       = MESSAGE + MCB +
IA14222                                 TRANSLATION_PAIR
I1A4223

O1A4222  FORMAT_CONVERTED_MESSAGE     = MESSAGE + MCB +
O1A4223                                 TRANSLATION_PAIR
IxA4224

C1A4221  FORMAT_INFO                  = <28>([TO_127|TO_128|NO])
                                        * TABLE KEYED BY
                                        TRANSLATION_PAIR.
                                        (CT'|AT'|TT'|RT'|QT' =>
                                        TO_127, FC|FA|FT =>
                                        TO_128, OTHERS => NO) *
```

```
            HEADER                          = [JANAP_HEADER|
                                               CARD_FMT_LN_2|
                                               SINGLE_CARD|
                                               ACP_HEADER]

            ICD                             = <3>(CHARACTER)
                                               * INCOMING CHANNEL ID *

            ICSN                            = <3>(DIGIT)
                                               * INCOMING CHANNEL SER NO *

            INCOMING_LOG_ENTRY              = [SOM_IN_ENTRY|
                                               EOM_IN_ENTRY|
                                               REJECT_ENTRY|
                                               CANCEL_REC_ENTRY|
                                               CANTRAN_REC_ENTRY]

O1A412      INDIVIDUAL_ROUTED_MESSAGES      = MESSAGE + MCB +
I1A413                                         <1..50>(OUTPUT_LINE NUMBER)

I3A3        INPUT_STREAM                    = <..>(BIT)

O3A431      IN..RCEPT_IN_ENTRY              = HEADER + ASN +DATE_TIME +
                                               MCB + NUMBER_BLOCKS
                                               * MADE WHEN MESSAGE
                                               RECEIVED FORM INTERCEPT *

O1A1331     INTERCEPT_LIST                  = TBD
I1A1332

O2A43       INTERCEPT_LOG_ENTRY             = [INTERCEPT_IN_ENTRY |
                                               INTERCEPT_OUT_ENTRY]

O1A432      INTERCEPT_OUT_ENTRY             = ASN + HEADER + DATE_TIME +
                                               <1..50>(RI) +NUMBER_BLOCKS
                                               * MADE WHEN MESSAGE SENT
                                               TO INTERCEPT *

            INVALID_CHAR                    = ASCII.BEL(16#07#)

            JANAP_FMT_LN_1                  = 'VZCZC' + ICD + FIGS
                                               + ICSN + LTRS + LN_END
                                               * TRANSMISSION IDENT (TI) *

            JANAP_FMT_LN_2                  = PRECEDENCE + LMF_PAIR +
                                               CLASS + CIC_CAI + 'b' + OSRI
                                               + OSSN + 'b' + DATE_TIME +
                                               '-' + REDUNDANT_CLASS +
                                               '--' + RI + <0..49>(RI) +
                                               '.' + LN_END
```

```
JANAP_FMT_LN_3          = 'DEb' + RI + OSSN +
                          DATE_TIME + LN_END

JANAP_FMT_LN_15         = (CORRECTION) +
                          (EOM_VALIDATION)

JANAP_FMT_LN_16         = EOM_SEQ

JANAP_HEADER            = (JANAP_FMT_LN_1) +
                          (JANAP_PILOT) +
                          (JANAP_FMT_LN_3) +
                          (JANAP_FMT_LN_3) +
                          (FMT_LN_4) + (FMT_LN_5) +
                          (FMT_LN_6) + (FMT_LN_7) +
                          (FMT_LN_8) + (FMT_LN_9) +
                          (FMT_LN_10)

JANAP_MESSAGE           = [CARD_MESSAGE|TTY_MESSAGE]

JANAP_PILOT             = PRECEDENCE + LMF_PAIR +
                          CLASS + CIC_CAI + 'b' + OSRI
                          + OSSN + 'b' + DATE_TIME +
                          '-' + REDUNDANT_CLASS
                          + RECORD_COUNT +
                          '--' + RI + <0..49>(RI) +
                          '.' + LN_END
                          * RECORD_COUNT MUST BE RI *

JANAP_TRAILER           = (FMT_LN_14) +
                          (JANAP_FMT_LN_15) +
                          JANAP_FMT_LN_16

LEVEL                   = [5|8]
                          * 5 LEVEL = ITA_#2
                            8 LEVEL = ASCII *

LINE_AVAILABLE          = TBD
```

```
C1A42     LINE_DATA                    = CHANNEL_MODE +
C1A43                                    [SYNCHRONOUS_MODE |
C1A421                                    ASYNCHRONOUS_MODE ] +
C1A431                                   DATA_MODE+LOGICAL_LINE_NO +
                                         PHYSICAL_LINE_NO
                                         LOOP_SPEED +
                                         COMMUNITIES_SERVED +
                                         (FIRST_LINK) +
                                         MAX_NO_RIS_PER_DELIV +
                                         (SOM_SEQ) + DIRECTION +
                                         ECSN + OCSN + CHNL_DES +
                                         (RCSN) +
                                         (NO_STOP_BITS) +
                                         (SECURITY_PROSIGN) +
                                         (SPEC_TERM)+
                                         LINE_AVAILABLE

O1A421    LMF_MESSAGES                 = MESSAGE + MCB +
I1A422                                   TRANSLATION_PAIR
I1A4221

          LMF_PAIR                     = FIRST_LMF_CHARACTER +
                                         SECOND_LMF_CHARACTER
                                         * LEGAL PAIRS ARE:
                                         TT TA TC AT AA AC
                                         CC CA CT RT SC BB
                                         CC II QT FT *

          LN_END                       = <2>(CR) + LF

I2A1      LOG_DATA                     = <0..>(LOG_ENTRY)
I2A2
I2A12
I2A13
I2A121
I2A132
I2A133
I1A1321
I1A1331

O3A3      LOG_ENTRY                    = [INCOMING_LOG_ENTRY |
O3A4                                     OUTGOING_LOG_ENTRY |
O3A5                                     OVERFLOW_LOG_ENTRY |
                                         INTERCEPT_LOG_ENTRY |
                                         SVC_GEN_ENTRY]

          LOG_ENTRY_INFO               = INCOMING_LOG_ENTRY
                                         * EXCEPT CANTRAN *

          MAX_NO_RIS_PER_DELIV         = [1|6|14|50|500]
```

```
                MCB                             = 'SOH' + SELECT_CHARACTER +
                                                  PRECEDENCE + LMF_PAIR +
                                                  CLASS + CIC_CAI + 'b' +
                                                  OSRI + OSSN + 'b' +
                                                  DATE_TIME + NUMBER_BLOCKS +
                                                  SOH_TIME + TRIP + SDC +
                                                  TDW + SDO + TASC +
                                                  SASC + ARRI + SMRI + ICD +
                                                  TMID + ICSN + 'ETB' + BP

                MESSAGE                         = [JANAP_MESSAGE |
                                                   ACP_MESSAGE]

IxA5426         MESSAGE_CHAR_TO_SEND            = [ TRUE | FALSE ]
                                                  * CHARACTER AVAILABILITY

I1A33           MESSAGE_DATA                    = MESSAGE
I1A331

02A3321         MESSAGE_DATA_A                  = MESSAGE
04A3321                                           * NOTE: ON
I1A3322                                           MESSAGE_DATA_A THROUGH
I1A3323                                           MESSAGE_DATA_D,
03A3331                                           LEXICALLY GREATER
I1A3332                                           DISTINGUISHING LETTERS
01A3341                                           INDICATE PROGRESSIVELY
I1A3342                                           MORE COMPLETE
03A3351                                           VALIDATION *
I1A3352

01A3322         MESSAGE_DATA_B                  = MESSAGE
I2A3323
03A3332
I1A3333
03A3342
04A3342
I1A3343
I1A3344
03A3352
I1A3353
```

```
01A3323  MESSAGE_DATA_C                  = MESSAGE
04A3323
I1A3324
I1A3325
03A3333
I1A3334
01A3343
I2A3344
03A3353
I1A3354

01A3344  MESSAGE_DATA_D                  = MESSAGE
04A3344
I1A3345
I1A3346

         MESSAGE_CANCELLED               = [ TRUE ¦ FALSE ]

         MESSAGE_IN_TRANSMISSION         = [ TRUE ¦ FALSE ]

         MESSAGE_MODE                    = [ JANAP_128 ¦ ACP_127 ¦
                                             ACP_127_MOD ]
                                           * FORMATTING CODE *

02A332   MESSAGE_REJECTION              = SEND_RM
02A333                                     * USED TO SIGNAL
02A334                                     MESSAGE REJECTION
02A335                                     TO GENERATE CONTROL
03A3321                                    CHARACTER FUNCTION *
03A3322
03A3323
02A3324
02A3325
02A3331
02A3332
02A3333
02A3334
02A3342
03A3343
03A3344
02A3345
02A3345
02A3351
02A3352
02A3353
02A3354
04A33

         MONTH                           = [JAN¦FEB . . . DEC]
                                           * 3 LETTER MONTH NAMES *
```

```
03A1      NEW_INTERCEPT_FILE            = <0..>(MESSAGE + MCB)
02A13
01A133
01A1332
I1A4

04A1      NEW_OVERFLOW_FILE             = <0..>(MESSAGE + MCB)
02A13
01A132
01A1322
I2A3

          NEW_ROUTE_LINE                = ROUTE_LINE

          NO_ERROR
                                          * NO ERROR DETECTED *

          NUMBER_BLOCKS                 = <3>(DIGIT)

          OASC                          = ['b'|LETTER]

          OCCUPANCY_LEVEL               = [0..100]
                                          * % INTRANSIT STORAGE *

          OCD                           = <3>(CHARACTER)
                                          * OUTGOING CHANNEL ID *

          OCSN                          = <3>(DIGIT)
                                          * OUTGOING CHANNEL SER NO *

          OP_SIGNAL                     = ['ZNY'|'ZNR']

01A2      OPERATOR_COMMAND              = [REINTRODUCE_COMMAND|
C1A3                                      TRACE_COMMAND|
C1A4                                      RETRIEVE_COMMAND|
C3A5                                      RECOVERY_COMMAND|
                                          THROTTLE_COMMAND|
                                          DRY_UP_COMMAND|
                                          CHANNEL_COMMAND|
                                          TAPE_COMMAND|
                                          STATISTICS_COMMAND|
                                          MESSAGE_CONTROL_COMMAND|
                                          DESTRUCT_COMMAND|
                                          ROUTING_COMMAND|
                                          EQUIPMENT_COMMAND|
                                          TABLE_COMMAND]

01A2      OPERATOR_PRINTOUTS            = TBD
```

	OSRI	= RI * ORIGINATING STATION ROUTING INDICATOR *
	OSSN	= <4>(DIGIT) * ORIGINATING STATION SERIAL NUMBER *
	OUTGOING_LOG_ENTRY	= [SOM_OUT_ENTRY¦ EOM_OUT_ENTRY¦ CANCEL_OUT_ENTRY¦ SCRUB_ENTRY¦ CANTRAN_OUT_ENTRY]
	OUTPUT_LINE_TYPE	= [TRIBUTARY ¦ TRUNK]
	OUTPUT_LINE_NUMBER	= [1..50]
O2A4 O1A43 O1A433 I1A5	OUTPUT_SELECTED_MESSAGE	= MESSAGE + MCB
O2A5	OUTPUT_STREAM	= <..>(BIT)
	OVERFLOW_IN_ENTRY	= HEADER + ASN + DATE_TIME + MCB + NUMBER_BLOCKS * MADE WHEN MESSAGE RECEIVED FORM OVERFLOW *
O1A1321 I1A1322	OVERFLOW_LIST	= TBD
	OVERFLOW_LOG_ENTRY	= [OVERFLOW_IN_ENTRY¦ OVERFLOW_OUT_ENTRY]
	OVERFLOW_OUT_ENTRY	= ASN + HEADER + DATE_TIME + NUMBER_BLOCKS * MADE WHEN MESSAGE SENT TO OVERFLOW *
IxA5426	PAUSE_GENERATED	= [TRUE ¦ FALSE]

	PRECEDENCE	= [W|Y|Z|O|P|R] * PRECEDENCES ARE: W -- CRITIC Y -- ECP Z -- FLASH O -- IMMEDIATE P -- PRIORITY R -- ROUTINE *
	PREPARE_TO_SEND_MESSAGE	= [TRUE | FALSE] * INDICATES COMPLETION OF MESSAGE *
I3A1	PROGRAM_LOAD_FILE	= TBD
O4A2	PROGRAM_LOAD_INSTRUCTIONS	= TBD
IxA5426	RCC_TO_SEND	= [TRUE | FALSE]
	RCSN	= <3>DIGIT * REJECTED CHANNEL SEQUENCE NUMBER *
	RECORD_COUNT	= [<4>(DIGIT)|'MTMS'|'PLTS'|RI] * RI--MS RI (4 LETTERS) *
O3A1 O1A12 O1A121 I1A3	RECOVERED_MESSAGES	= <0..>(MESSAGE + MCB)
O2A11 C1A12 C1A121	RECOVERY_COMMAND	= TBD
	REDUNDANT_CLASS	= <4>(CLASS)
O4A3	REFERENCE_COPY	= MESSAGE + MCB
I1A1 I1A2 I1A12 I1A13 I2A122 I1A132 I1A133 I1A1322 I1A1332	REFERENCE_DATA	= <0..>(MESSAGE + MCB)

O1A422
O1A4225
I1A423
I1A4231
REFORMATTED_MESSAGE = MESSAGE + MCB + TRANSLATION_PAIR

C2A4
C2A43
C2A432
REINTRODUCE_COMMAND = TBD

REJECT_ENTRY = DATE_TIME + REJECT_REASON + ASN + CHANNEL_MODE + ICD + ICSN
* MADE WHEN A MESSAGE IS REJECTED BY THE SWITCH *

REJECT_REASON = TBD

REPLY = PROTOCOL COMMAND
* SENT BY THE TRANSMITTING STATION TO DIRECT THE RECEIVER TO SEND ITS CURRENT STATUS *

REPLY_CHARACTER = ASCII.DC1(16#11#)

REQUEST_FOR_ANSWER_CHAR = [TRANSMIT_CONTROL_CHAR | BP_CHAR]

RI = [R|U|Y] + <3..6>(LETTER)

C1A412
RI_LINE_TABLE = <0..1500>(RI+LINE_NO)

C1A41
C1A332
C1A333
C1A334
C1A335
C1A3323
C1A3332
C1A3344
C1A3352
ROUTE_DATA = TBD

ROUTE_LINE = [JANAP_FMT_LN_2 | ACP_FMT_LN_2 | CARD_FMT_LN_2]

C2A4
ROUTE_LINE_DATA = TBD

O1A41
O1A413
I1A42
I1A421

ROUTED_MESSAGE = MESSAGE + MCB

O1A331
O2A331
O3A331
O4A331
I1A332
I1A333
I1A334
I1A335
I1A3321
I1A3331
I1A3341
I1A3351

ROUTED_MESSAGE_DATA = MESSAGE

SASC = ['b'|LETTER]

IxA5426 SEND_DET_CODE = [TRUE | FALSE]

IxA5426 SEND_DET_GENERATED = [TRUE | FALSE]

SCRUB_ENTRY = OCD + OCSN + CHANNEL_MODE +
DATE_TIME + ASN +
SCRUB_REASON + NUMBER_BLOCKS
<1..50>(RI)
* MADE WHEN A MESSAGE
IS SCRUBBED *

SEND_GEN_CODE = [TRUE | FALSE]

SCRUB_REASON = TBD

SDC = ['b'|S|C|P|M|T|A]

SDO = ['b'|LETTER]

SECOND_LMF_CHARACTER = [B|D|I|C|A|T]

SECURITY_PROSIGN = [A|T|S|C|R|E|U|M]

SELECT_CHARACTER = [A|B|C|D|E|F|G|H|J|K|M|P|S]

SEPARATOR = 'BT' + LN_END

01A4
02A42
02A422
02A4223
01A5
01A332
01A333
01A334
01A335
01A3321
02A3322
02A3323
01A3324
01A3325
01A3331
01A3332
01A3333
01A3334
01A3342
02A3343
02A3344
01A3345
01A3346
01A3351
01A3352
01A3353
01A3354
C2A3

SERVICE_MESSAGE_INFO = SVC_MESSAGE_TYPE + MESSAGE + <0..50>(RI> + (<2>)(ICSN)

SINGLE_CARD = PRECEDENCE + LMF_PAIR + CLASS + CIC_CAI + 'b' + OSRI + OSSN + 'b' + DATE_TIME + '--' + RI + '.' + <41>(CHARACTER) + 'N'
* FINAL N MUST BE IN COLUMN 80 *

SMRI = <5>(LETTER)

SOH_TIME = DATE_TIME
* TIME OF RECEPTION OF START OF HEADER *

SOM_IN_ENTRY = CHANNEL_MODE + HEADER + DATE_TIME + ASN + ICD + ICSN + MCB
* MADE WHEN START OF MESSAGE RECEIVED *

	SOM_OUT_ENTRY	= OCD + OCSN + CHANNEL_MODE + <1..50>(RI) + HEADER + DATE_TIME + ASN * MADE WHEN START OF MESSAGE TRANSMITTED *
	SOM_SEQ	= [FULL | ABBREVIATED]
O2A3 I2A4 I1A41 I1A411	SORTED_MESSAGE	= MESSAGE + MCB
	SPEC_TERM	= [ACCESS | INTERSWITCH | TECH_CONTROL]
C1A1 C1A11	START_COMMAND	= TBD
	STOP_RECD_CODE	= [TRUE | FALSE]
	SVC_GEN_ENTRY	= HEADER + ASN + DATE_TIME + MCB + NUMBER_BLOCKS * MADE WHEN SERVICE MESSAGE IS GENERATED *
	SVC_MESSAGE_TYPE	= [INVALID_RI| OUTPUT_SCTY_MISMATCH| EXCESSIVE_ROUTING_REJ| ILLEGAL_EXCHANGE| SUSPENDED_TRANSMISSION| SUSPECTED_STRAGGLER| OPEN_CSN| INPUT_SCTY_MISMATCH| INVALID_SCTY_FIELD| INVALID_HEADER_REJ| INVALID_HEADER_ACC| HI_PREC_ACC| INVALID_RI_FIELD| ALL_RI_INVALID| INVALID_TI_REJ| INVALID_TI_ACC| TWO_CONSEC_SOM| INVALID_EOM_REJ| INVALID_EOM_ACC| INCORRECT_CSN| INVALID_BLOCK_COUNT| TRAFFIC_CHECK| NO_EOM]

	SYNCH_RECEIVE_CONTROL_CHAR	= [ACK_1 ¦ ACK_2 ¦ NAK ¦ RM ¦ WBT] * CHARACTER *
	SYNCHRONOUS_MODE	= [BLOCK_BY_BLOCK ¦ CONTINUOUS]
O3A2	TABLE_CHANGES	= TBD
O1A1 O1A11	TABLE_INITIALIZATION	= TBD
	TASC	= ['b'¦LETTER]
IxA5426	TCC_TO_SEND	= [TRUE ¦ FALSE]
	TDW	= <12>(CHARACTER)
	TEXT_CHARACTER	= [ASCII.BLANK..ASCII.TILDE ¦ * PRINTABLE ITA CHARACTERS]
	THRESHOLD_STATUS	= [UPPER¦UPPER_MIDDLE¦ LOWER_MIDDLE¦LOW]
	TIME	= <4>(DIGIT) + 'Z'
	TMID	= <8>(CHARACTER)
IxA5426	TIMING_REPLY_TO_CAN	= [TRUE ¦ FALSE]
IxA5426	TIMING_REP_OR_STOP	= [TRUE ¦ FALSE]
O1A42 O1A423 O1A4238 O1A431 O2A431 O2A432 I1A43 I1A431 I1A432 I1A433	TRANSLATED_MESSAGE	= MESSAGE + MCB
	TRANSLATION_PAIR	= LMF_PAIR
	TRANSMIT_CONTROL_CHAR	= [REPLY_CHAR¦CANCEL_CHAR¦ ENQUIRY_CHAR]
	TRIP	= [N¦C¦O¦P¦R¦D]

TTY_MESSAGE = JANAP_HEADER + SEPARATOR + BODY + SEPARATOR + JANAP_TRAILER
* NOTE: THE MAXIMUM LENGTH FOR A PHYSICAL LINE IN A TTY_MESSAGE IS 69 PRINTABLE CHARACTERS. SINCE A FORMAT LINE MAY EXCEED THIS NUMBER, LN_END'S MUST BE ADDED AS REQUIRED TO KEEP EACH PHYSICAL LINE AT OR BELOW THIS LENGTH *

03A332 VALIDATED_MESSAGE = MESSAGE
03A333
03A334
03A335
03A3324
03A3325
03A3334
03A3345
03A3346
03A3354

YEAR = <2>(DIGIT)

Message Switch Non-Functional Requirements

I. Messages shall be accepted without regard to the immediate availability of an outgoing line or trunk. The receive module will have the ability of throttling incoming traffic during overload periods or during periods of high priority messages. Two types of message codes will be handled.

A. ASCII Code

(1) Odd parity
(2) Seven (7) bits plus parity
(3) One (1) start unit interval
(4) One (1) or two (2) stop unit interval(s)

B. ITA #2 Code

(1) 5 bits data
(2) One (1) start unit interval
(3) One (1) or two (2) stop unit interval(s)

II. Synchronous Operation - Messages Received From an Autodin Switch

A. Block-by-Block

(1) 75-16000 b/s
(2) ASCII character set
(3) Parity Criteria - (7 bits for information and the 8th bit for parity)
(4) Bits received serially - (low order bit first and the parity bit last)
(5) Message characters will have odd parity
(6) Control characters will have even parity
(7) Modes I or III
(8) Acquire frame sync. before receive
(9) A message control block (MCB) will be employed if the message is via an interswitch trunk
(10) Bit stream - bit synchronous where each bit is accompanied with a clock pulse
(11) Recognize accepted idle pattern for channel sync.
(12) SOH/STX characters of a block stream will be separated by a syn. period from the previous block.

D. Continuous (block-by-block)

(1) SOH/STX characters of a block stream will be contiguous to the block parity character of the previous block

(2) Reception of continuous (block-by-block) is the same as block-by-block above except for item 12

III. Asynchronous Operations Modes II, IV (without Automatic Channel and Error Control)

A. ASCII and ITA#2 Codes (Mode II)

(1) Free running characters
(2) Character by character release
(3) Independent of simultaneous two-way operation

B. ASCII and ITA#2 Code (Mode IV)

(1) Unidirectional operation (send only or receive only)
(2) Equivalent to half duplex operation of Mode II
(3) Free running characters

IV. Asynchronous Operation Mode V (with automatic channel and error control)

A. ASCII Code

(1) 75-300 baud
(2) Continuous two-way transmission

B. ITA #2 Code

(1) 45.45 to 75 baud
(2) Continuous two-way transmission

V. Message statistics. The maximum length message is 44,000 characters (or 550 blocks). The maximum length teletype (TTY) message is 6900 characters. The average message length is 2400 characters. There will be an average of 1.75 addresses per message. The average collective RI received will have 3.9 destinations. The messages will be distributed among the various precedences as follows:

ECP(Y)/CRITIC(W)	0.1%
FLASH(Z)	2.9%
IMMEDIATE(O)	30.0%
PRIORITY(P)	33.0%
ROUTINE(R)	34.0%

VI. Maximum character handling - characters per time period

25 line switch	1 second	1 hour	1 day
Input	3600	6,000,000	18,000,000
Output	2400	11,000,000	36,000,000

50 line switch	1 second	1 hour	1 day
Input	5400	9,000,000	27,000,000
Output	3600	16,500,000	54,000,000

VII. Errors. Bit errors within the switch or delivered to output terminals shall not exceed 1 bit in 10 billion consecutive bits. No more than 1 message in 10 million messages shall be misrouted. No more than 1 in 10 million messages shall be lost due to nondetection or errors in the following: SOM, EOM, cancel transmission (CANTRAN), or header. Loss of receipted messages shall not exceed 1 in 1 billion.

VIII. Processing time. Processing time will be the sum of the time for reception of the EOM until the message is placed on the output queue and the time from the output line becoming available until the first bit of the SOM is transmitted. Mean processing time will be 2 seconds per message. No more than 1 in 1,000 CRITIC, ECP, or FLASH messages will have in processing time greater than 6 seconds. No more than 1 in 1,000 messages of lower precedence will have a processing time in excess of 8 seconds.

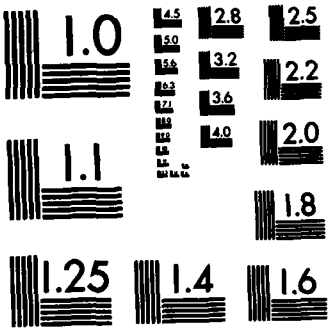
IX. Routing table size. The routing table shall contain room for:

1500 single address RIs
200 collective RIs with an average 20 RIs each
600 RIs for other centers

X. Intransit storage size. For a 25 line switch, intransit storage shall be capable of holding a minimum of 2000 average length messages. For 50 lines this number shall be increased to 2500.

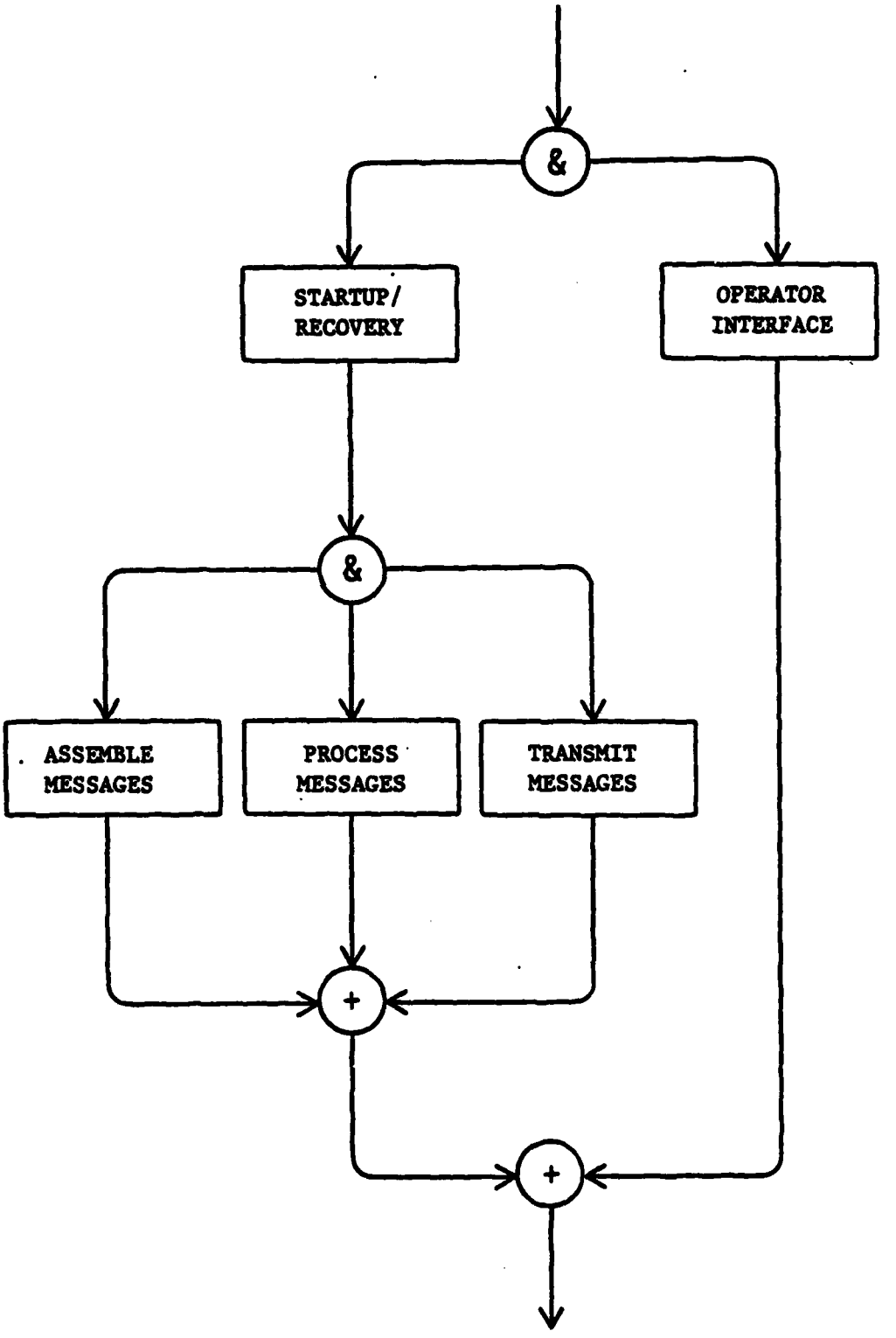
XI. Recovery and retrieval. Reentry of messages from intercept storage shall be such that the first message is ready for reentry with 10 minutes of the reentry directive. Retrieval of CRITIC, ECP, and FLASH messages less than 24 hours old shall be accomplished in 7 minutes or less. All other messages less than 24 hours old shall be retrieved within 15 minutes. Messages older than 24 hours must be capable of being retrieved within 30 minutes.

AD-A123 306 LARGE SCALE SOFTWARE SYSTEM DESIGN OF THE AN/TYC-39 · 3/3
STORE AND FORWARD MES.. (U) GENERAL DYNAMICS FORT WORTH
TX DATA SYSTEMS DIV 09 NOV 82 DAAK80-81-C-0108
UNCLASSIFIED F/G 17/2 NL



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

Recovery of messages after switch failure must take no more than 30 minutes.



TOP LEVEL CONCURRENCY DIAGRAM