LOAN DOCUMENT

PHOTOGRAPH THIS SHEET

DTIC ACCESSION NUMBER

LEVEL

INVENTORY

DOCUMENT IDENTIFICATION

DISTRIBUTION STATEMENT

HANDLE WITH CARE

ACCESSION FOR
NTIS GRA&I
DTIC TRAC
UNANNOUNCED
JUSTIFICATION

BY

DISTRIBUTION/

AVAILABILITY CODES

DISTRIBUTION	AVAILABILITY AND/OR SPECIAL	
A-1		

DISTRIBUTION STAMP

DATE ACCESSIONED

DATE RETURNED

19981223 071

DATE RECEIVED IN DTIC

REGISTERED OR CERTIFIED NUMBER

PHOTOGRAPH THIS SHEET AND RETURN TO DTIC-FDAC

DTIC FORM 70A JUN 90

DOCUMENT PROCESSING SHEET

PREVIOUS EDITIONS MAY BE USED UNTIL STOCK IS EXHAUSTED.

LOAN DOCUMENT

APPENDIX 13

EMITTER CLASSIFICATION PROCESSING

FINAL SOFTWARE REPORT

DATA ITEM NO. A005

INTEGRATED ELECTRONIC WARFARE SYSTEM
ADVANCED DEVELOPMENT MODEL (ADM)

PREPARED FOR:

NAVAL AIR DEVELOPMENT CENTER
WARMINSTER, PENNSYLVANIA

CONTRACT N62269-75-C-0070

RAYTHEON
ELECTROMAGNETIC
SYSTEMS DIVISION

1 OCTOBER 1977

APPENDIX 13

EMITTER CLASSIFICATION PROCESSING DESIGN SPECIFICATION

FINAL SOFTWARE REPORT

DATA ITEM A005

INTEGRATED ELECTRONIC WARFARE SYSTEM (IEWS)
ADVANCED DEVELOPMENT MODEL (ADM)

Contract No. N62269-75-C-0070

Prepared for:

Naval Air Development Center
Warminister, Pennsylvania

Prepared by:

RAYTHEON COMPANY
Electromagnetic Systems Division
6380 Hollister Avenue
Goleta, California 93017

1 OCTOBER 1977

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO.	SPEC NO.	
49956	53959-CT-0760	
	SHEET 1 OF 89	REV

TYPE OF SPEC

COMPUTER SUBPROGRAM DESIGN DOCUMENT

TITLE OF SPEC

EMITTER CLASSIFICATION PROCESSING - 1

FUNCTION	APPROVED	DATE	FUNCTION	APPROVED	DATE
WRITER	S. J. Stevens	12/7/76			
	George D. Thompson Jr.	12/7/76			

REVISIONS

CHK	DESCRIPTION	REV	CHK	DESCRIPTION	REV

REVISION																									
SHEET NO.																									
REV STATUS OF SHEETS	REVISION																								
	SHEET NO.																								

Table of Contents

1.0 SCOPE

1.1 IDENTIFICATION

This subprogram shall be known by the designator ECDR (Emitter Classification Driver).

Designators and brief functional descriptions of subroutines written specifically for use by ECDR are shown in Table 1.

Table 1

ECDR Subroutines - Designators & Function

Designator	Brief Functional Description
ECLV1	Emitter Classification Level 1 - Matches frequency, PRI and PW of input emitter track file to limits of emitter library 1 and returns with no match indication or else a candidate list. Called by ECDR.
BIRCH	Binary Search subroutine. Called by ECLV1.
TRNSL8	Translates candidate list in keyword - compressed format to standard candidate list format. Called by ECLV1.
INSECT	Generates the intersection of 2 or 3 candidate trunk lists in keyword - compressed format. Result in same format. Called by ECLV1 and TRNSL8.
PARITY	Generates the 1's bit count of A-Reg in A-Reg. Called by INSECT.
ECST1	Scan Test 1: Requests scan analysis if amplitude threshold exceeded. Else calls SCTCOM.

A seventh subroutine SCTCOM is shared by Scan Tests 1 and 2. It complements the ETF state indicator and if this is now = 1, sets scan type = circular and scan period = time out.

1.2 SUBPROGRAM TASKS

A brief description of each subprogram's tasks is provided in Table 1.

2.0 REFERENCE DOCUMENTS

2.1 PERFORMANCE SPECIFICATION

Computer Program Performance Specification (CPDS), IEWS, ADM (U), Raytheon Company, ESD, (No. 061290529), (1 June 1976), (U).

2.2 PROGRAM DESIGN SPECIFICATION

System Controller Program, IEWS, ADM - - CPDS Number 53959-GT-0750, 1976 September 2.

2.3 DATA BASE DESIGN DOCUMENT

Common Data Base Design Document (CDBDD), SC Unit, IEWS, ADM, Raytheon Company, ESD, (No. 53959-GT-0751), (13 October 1976), (U).

2.4 MISCELLANEOUS DOCUMENTS

Document No.	Document Title
53959-GT-0756	Computer Subprogram Design Document, Executive, IEWS, ADM
53959-GT-0759	Computer Subprogram Design Document, Data Extraction, IEWS, ADM

3.0 REQUIREMENTS

3.1 SUBPROGRAM DETAILED DESCRIPTION

3.1.1 ECDR - - Emitter Classification Driver

ECDR is the logical skeleton for emitter classification processing, first level. It accomplishes its work through two local subroutines:

ECLV1 - Emitter Classification Level 1, described hereafter (Section 3.1.2).

ECST1 - Emitter Classification Scan Test 1, described hereafter (Section 3.1.3).

both of which have dependent subroutines of their own; and one external, executive level subroutine EXMES. See flow diagram 3.2.0.

ECDR calls the two local subroutines cited, interprets their results, formats the scan-analysis request message, and sends this message to the Executive. A step-by-step account follows. The steps are either program labels or unlabeled sections headed by a semi-colon followed by a number.

ECDR - Save A-Reg. ECDR uses only A and X-Reg's; other registers are saved and restored by ECDR's dependent subroutines.

Upon entry, X—▷first word of a 3-word block. The third word contains the EFN (Emitter Track File #) as a whole-word item. This is fetched to the A-Reg, moved to the X-Reg, and saved on the stack.

;2 - Add the EC Return Module code (stored in the left byte as parameter RMCDLB) into the A-Reg (which still has EFN) to create a byte-split word:

Return Module Code	EFN

Store this in the Scan-Analysis request message block at label SRRMCD. See Section 3.3.1.4.

;3 - Call ECLV1. The input is X-Reg = EFN done above. A return to Call +1 implies that no Candidate List was created. The Call +1 location is a JMP to step ECEND below. Taking this path means returning from ECDR without sending any message to the Executive.

A return to Call +2 is to next step.

;4 - The X-Reg now points to the 1st word of the Candidate List. This is stored in the Scan-Analysis request message at label SRCLAD. See Section 3.3.1.4.

;5 - The X-Reg is reloaded with EFN (Non-destructively from top of stack). This is required as input for the forthcoming call on ECST1.

The A-Reg is loaded with the constant X'8ØØØ'. This will leave the AW bit of the Scan-Analysis request message set if ECST1 returns to Call +2.

Call ECST1. A return to Call +1 implies no scan analysis is required. The Call +1 location contains a clear instruction that resets the AW bit that was set above.

A return to Call +2 is to next step.

;6 - The A-Reg now contains either ØØØØ (No analysis) or X'8ØØØ' (Analysis wanted). This is now increased by 1 to set the type of analysis wanted or not wanted to "SCAN". The completed request word is stored in the Scan-Analysis request message at label SRREQW. See Section 3.3.1.4.

;7 - The X-Reg is loaded with the address of the 1st word of the now completed Scan-Analysis request message, and the message is sent by a call to EXMES.

ECEND - This step is reached from Step 7 or from Step 3 if ECLV1 returns to Call +1.

Load X-Reg from top of stack. This is not the input value of X-Reg, but = EFN.

Restore A-Reg from stack.

Return.

3.1.2 ECLV1 - Emitter Classification Level 1

This is the workhorse subroutine of ECDR. Its overall function is to create a list of Candidate Emitters each of which could match the input Emitter Track file in the three measurements: frequency, PRI and PW. The steps are as follows:

ECLV1 - Save A, B and E-Reg's on stack. Store the (local) address, CANDLE, of the Candidate List to be generated on the stack as the exit value of the X-Reg. Save the EFN, which is now in X-Reg in the first word of the Candidate List.

;1 - Make room on the stack for a block of 3 words which will later be used to hold the addresses of the Parameter-Matching Trunk lists as they are found. Push the address of the first word of this block to the stack. Symbolic name of this address is ala.(1)

;2 - Make room on the stack for nvps(1), the number of valid parameters and an un-named block of 17 words to be used to hold the intersection of 2 or 3 Parameter-Matching Trunk lists. Store the address of the first word of this block on the stack. Symbolic name of this address is adint.(1)

;3 - Create in X-Reg the address of ETF (EFN) = ETF + 16 * EFN. This stays in X-Reg until the end of the sub-routine, and is used to locate various items in the ETF.

;4 - Set nvps(1) = Ø.

LOOKFR - Fetch the word (EFVD rel to X) containing the frequency validity bit, mask it (Mask = EFVF), and test it for zero. If it = Ø, there is no valid frequency measurement: Go to step LOOKPI.

(1) We are using lower case letters to distinguish between the item (lower case) and the symbolic displacement (upper case) used with base S to access the item. See Section 3.3.2.1.

3.1.2 continued -

;5 - Fetch frequency (EFRQD rel to X) to A-Reg as Binary Search key, and call internal subroutine LOOK. 1 with these three arguments stored in order after the JSB: ODV. F, ODA. F and ZLS. F. See LOOK. 1 below.

If the serach is unsuccessful, control will go directly to Step FAIL. 1, else LOOK. 1 will return and proceed to next step.

LOOKPI - This step is reached from Steps LOOKFR and 5.

Fetch the word (EPIVD rel to X) containing the PRI validity bit, mask it (Mask = EPIVM), and test it for zero. If it = Ø, there is no valid PRI measurement: Go to Step LOOKPW.

;6 - Fetch average PRI (EAPID Rel to X) to A-Reg as Binary Search key (masking it with Mask = EAPIN) and call internal subroutine LOOK. 1 with these three arguments stored in order after the JSB: ODV. PI, ODA. PI and ZLS. PI. See LOOK. 1 below.

If the search is unsuccessful, control will go directly to Step FAIL. 1, else LOOK. 1 will return and proceed to next step.

LOOKPW - This step is reached from Steps LOOKPI and 6.

Fetch the word (EPWVD rel to X) containing the PW validity bit, mask it (Mask = EPWVM), and test it for zero. If it is = Ø, there is no valid PW measurement: Go to Step INSPCT.

;7 - Left concatenate long pulse flag (ELPD rel to X), using mask ELPM and shift count LP. RS to pulsewidth field (EPWD rel to X-Mask EPWM) to form a direct ODA-Look up index. Leave this in B-Reg.

;8 - Call internal subroutine LOOK. 2 with these two arguments stored in order after the JSB: ODA. PW, ZLS. PW. See LOOK. 2 below.

If the search is unsuccessful, control will go directly to Step FAIL. 1, else LOOK. 2 will return and proceed to next step.

3.1.2 continued -

INSPCT - This step is reached from Steps LOOKPW and 8.

Analyze nvps, the number of valid parameters. This was initialized to Ø (Step 4) and is augmented by 1 before each successful return from LOOK.1 or LOOK.2.

If nvps = Ø, all three validity bits were = Ø. Go to Step FAIL.2.

If nvps = 1, there was only one valid parameter. The address of the list of trunks which could display the same value of that parameter (a ZLS subsection) is stored in the location following ala. Go to Step TRL8.1.

If nvps = 2 or 3, there are nvps such list address stored after ala. These nvps lists must be intersected to identify the trunks which can match all nvps parameters. Load E-Reg with a NOP (JMP $ +1) if nvps = 2 or a JMP $ +2 if nvps = 3. This is an input to INSECT.

ISECT - Put B-Reg = adint to tell INSECT where to store intersection result.

Put A-Reg = ala to tell INSECT where to find the list of input list addresses.

Call INSECT. This will intersect the nvps input lists pointed to by input value of A-Reg and store the results in block pointed to by input value of B-Reg. The intersection will be 2-way if input E-Reg contained a NOP and 3-way if same contained a JMP $ +2. The result is also in KWC/BV format.

The value returned in A-Reg = keyword of the intersection. If this = Ø, intersection is null: Go to Step FAIL.1; else next step.

TRL8 - An intersection has been performed. It is stored at address adint. Put adint in A-Reg and go to Step 9.

TRL8.1 - This step is reached only from step INSPCT when nvps = 1. No intersection was needed. Put address of the singleton parameter-match list (in location following ala) in A-Reg. Go to Step 9.

3.1.2 continued -

;9 - Put address of result (CANDLE) in B-Reg, and call TRNSL8. This will translate the list whose address is in A-Reg from KWC/BV format to standard (CDBDD) Candidate List format, and store result in block pointed to by B-Reg on input. See Section 3.3.1.1.

NORMAL - A valid Candidtate List has been produced. ECLV1 communicates this to its caller by returning to Call +2. In this step, bump return address to Call +2 by rtad←rtad +1.

ABNORM - This step is reached from step NORMAL and from Step 11 which is the terminal step of the common sequel to the two failure steps FAIL. 1 and FAIL. 2.

Clean up stack, i.e., return to available status the 23 locations appropriated for local temporary storage in entry Steps 1 and 2. This is done by adding 23 to S-Reg. S-Reg then points to the exit value (CANDLE) of X-Reg.

Restore X, E, B and A-Reg's from stack and exit to Call +2 or Call +1 as Step NORMAL was or was not executed.

** - This ends the flow of ECLV1. We must, however, give account of the two failure steps FAIL. 1 and FAIL. 2 and the two internal subroutines LOOK. 1 and LOOK. 2.

FAIL. 1 - This step is reached from internal subroutines LOOK. 1/2 on no-match finding in any of the three calls on these subroutines (from Steps 5, 6, and 7), and from Step ISECT if the 2 or 3-way intersection proves to be null (A-Reg = Ø on return).

In this step put the failure code 'NOFA1' (None-of-the-above - - 1) in the B-Reg and go to Step 1Ø.

FAIL. 2 - This step is reached only from Step INSPCT when nvps = Ø.

Put the failure code 'UNCLS' (Unclassified) in the B-Reg and go to Step 1Ø.

3.1.2 continued -

;1Ø - Store failure code that is in B-Reg in the ETF Identity field (EIDD Rel to X), leaving other fields unchanged (Mask EIDM).

;11 - Store "Unknown" display code ('UNKNO') in ETF display code field (EDISD Rel to X), leaving other fields unchanged (Mask EDISM).

To to Step ABNORM.

LOOK.1 - Pop top of stack into E-Reg. This leaves E pointing at the word following the JSB, i.e., at the ODV argument.

Load ODV argument into B-Reg and set E←E + 1 (i.e., E→now to ODA argument).

Call BIRCH (Binary Search). The arguments are A-Reg = Search Key and B-Reg→Search List. A-Reg was set before the call LOOK.1 in Steps 5 and 6. Upon return B-Reg contains an index into ODA.

Go to Step LKCOM.

LOOK.2 - Pop top of stack into E-Reg. This leaves E pointing to the word following the JSB, i.e., at argument ODA.PW. The B-Reg was set to the index into ODA.PW prior to entry (Step 7).

Go to Step LKCOM.

LKCOM - This step is reached from Steps LOOK.1 and LOOK.2. In both cases, one reaches this step with B-Reg = Index into ODA List and E-Reg→address of ODA argument.

Add ODA argument to B-Reg (Index) and set E←E +1 (i.e., E→now to ZLS argument).

Pick up ODA at entry whose address was just been constructed in B-Reg. Call this D(L).

If D(L) < Ø, the parameter we are currently looking at matches none of the EL1 emitters: Go to Step FAIL.1, else next step.

3.1.2 continued -

;12 - Add ZLS argument to D(L) to form a ZLS-subsection address, and set E←—E +1 (i.e., E—→now to return point).

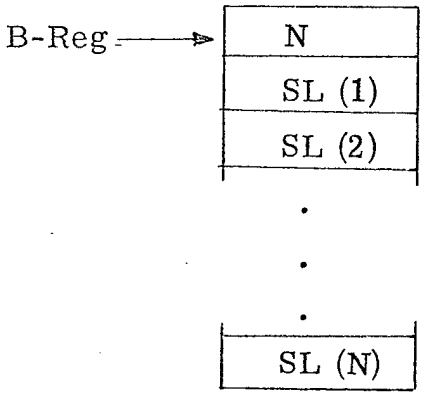
Store the address thus created at address ala + nvps.

Set nvps←—nvps +1.

Return to location pointed to by E-Reg.

3.1.2.1 BIRCH - Binary Search Subroutine

This is a subroutine of ECLV1. Its required inputs are A-Reg = Search Key (SK) and B-Reg = Address of Search List (SL):



where SL (1) < SL (2) <. . . < SL (N) and it is assumed that SL (1) is less than any value actually taken by SK.

The version of Binary Search used here is Algorithm U, Section 6.2.1, D. C. Knuth, The Art of Computer Programming, Vol. 3, P. 411. The steps are as follows:

BIRCH - Save A, X and E-Reg's on stack.

;2 - Set X-Reg = -1 for right-shift-1 count.

;3 - Set E-Reg = -N and B←—B +1 (i.e., B now —→SL (1)). Set A-Reg = 1's complement of N ≡ C1 (N). Save B-Reg = Addr [SL (1)] on stack.

3.1.2.1 continued -

LOOPG - This step is reached from Step 3 (First Loop Entry) and from Step GRTR, that is after a non-terminal comparison in which SK > current SL comparand.

At this point, E-Reg = -M where M is the termination criterion (Stop when M reaches Ø unless exact match found first).

Shift the register pair (X, E) right 1 bit. This keeps X = -1 and E ← - (M + 1)/2. (1)

Go to COMON.

LOOPL - This step is reached only from Step LESS, i.e., after a non-terminal comparison in which SK < current SL comparand.

The condition of E-Reg is as described in LOOPG.

Perform the same shift as in LOOPG, then change the sign of E-Reg so that E = (M + 1)/2. (1)

Go to Step COMON.

COMON - This step is reached from LOOPG with E = - (M + 1)/2 and LOOPL with E = + (M + 1)/2. In either case E contains - Delta where Delta is the change to the SL Lookup address (in B-Reg) required to find the next comparand.

Alter SL Lookup address in B-Reg by B ← B - E

B-Reg is increased if in last comparison SK was > last SL comparand and conversely.

Move C1 (M) from A-Reg to E-Reg and right shift register pair (X, E) one bit. E-Reg now contains C1(M/2) = C1 (New M). (1)

(1) Note: All divisions in this algorithm are truncated integer divisions as in FORTRAN.

3.1.2.1 continued -

;7 - Compare SK (at 3 rel to Top of Stack) to current SL item (at Ø, rel to B-Reg.)

The first four instructions of this step and step SIDIF are necessitated by assumption that SK and SL items are 16-bit unsigned values and by the fact that the RP-16 CSL instruction is an algebraic comparison (Signed, 15-bit #'s).

The first thing done is to exclusive-or together SK and SL item.

If result = Ø, comparands are equal: Go to Step DONE.

If result < Ø, comparands are of opposite signs: Go to Step SIDIF.

Else comparands are unequal and of like sign: Compare using CSL instruction.

If SK > SL item, go to Step GRTR. If SK < SL item, go to Step LESS.

SIDIF - This step reached from Step 7 when comparands are unequal and of opposite sign.

If SK ≥ Ø go to Step LESS. Else go to Step GRTR.

LESS - Save a copy of C1 (M) (Now in E-Reg) in A-Reg.

Do E←—E + 1. Test for zero. If E ≠ Ø, go to Step LOOPL:

Else C1 (M) = FFFF or M = Ø and we are done. However, there is a slight glitch: The index we are about to compute, B-Reg - Addr [SL (1)] will be of an element that is > SK whereas we went to end with index L such that

$$SL(L) \leq SK < SL(L+1)$$

Therefore do B←— B - 1 and then go to Step DONE.

3.1.2.1 continued -

GRTR - Everything is the same as in Step LESS except that after E←E + 1 and E ≠ Ø, go to Step LOOPG. If E = Ø, the B←B - 1 adjustment is not necessary (Just go to Step DONE).

DONE - Compute and leave in B-Reg an index L by subtracting from B-Reg the address of SL (1) which is on top of stack.

Restore E, X and A-Reg from stack.

Exit.

3.1.2.2 TRNSL8 - Subroutine of ECLV1

This subroutine accepts an input Candidate Trunk List in KWC/BV format (address in A-Reg) and translates it to standard (CDBDD) Candidate List format stored at the address supplied in B-Reg. The steps are as follows:

TRNSL8 - Save B-Reg on stack (resad). Make room for adr2 on stack. Store A-Reg on stack (adr1) and set ila (on stack above adr1) = Absolute address of adr1. Make room for 20 more locations on stack: grpno, limit, ncand, and un-named block of 17. Set ina (on stack above block of 17) = Absolute address of block of 17. See Section 3.3. for stack map and discussion of items thereon.

;1 - ncand ← Ø
grpno ← Ø
limit ← 32

X-Reg ← Addr ODA. GT
B-Reg ← (ODA. GT) = # of groups to follow;
Simultaneously X ← X + 1
Save B-Reg on stack (ngrps)

3.1.2.2 continued -

CLHIB - This step begins a loop. It is entered first time from Step 1 and loop-entry-value-of-B-Reg-minus-one times from the end of CLHIB itself. On entry, X = ODA. GT +1, B = ngrps so that B + X - 1 is address of last ODA. GT entry. The loop is accomplished by diminishing B by 1 each iteration until it reaches Ø. Thus, X is unchanged by loop. The purpose of the loop is to clear the LSB of (ODA. GT +1), ... (ODA. GT + ngrps) -- the "Hit-Bit".

Fetch (X + B -1) to A-Reg AND with X'FFFE', store result back at (X + B -1), B ← B -1.

If B ≠ Ø go to Step CLHIB, else next.

GROOP - This step begins the major processing loop of the subroutine. It is entered first time from above and N -1 times from Step TALLY, where 32 ≤ N ≤ original value stored in ngrps. The lower limit, 32, is imposed by fact that there is room in the Candidate List output block for only 32 Candidates. On 1st entry to this step, X-Reg = ODA. GT +1.

grpno ← grpno +1

Pick up ((X)) ≡ (ODA. GT + I) ≡ D (I); I ≥ 1 and simultaneously, X ← X + 1.

Arithmetic right shift D (I) 1 bit to get rid of the Hit-Bit (LSB) in this working version of D (I). The bit remains in the stored version of D (I).

Test D (I): Of D (I) ≥ Ø go to Step ISECT else next.

;2 - D (I) < Ø: This means that D (I) = J - (I + 1) for some J such that 1 ≤ J ≤ I. At this point, (X-Reg) = ODA. GT + I + 1 (Because of X ← X + 1 in Step GROOP) so that the address

(X) + D (I) = ODA. GT + I + 1 + J - (I + 1) = ODA. GT + J

is the address of D (J) ≥ Ø which is a ZLS. GT entry displacement previously considered in Step ISECT. The results of that consideration are recorded (see Step 3) in the LSB of D (J) (Hit-Bit).

3.1.2.2

;2 - continued -

Fetch D (J) = ((X) + D (I))

Test LSB (D (J)): If = 1 go to step GOTHIT
If = Ø go to step TALLY

TALLY - This step is entered from Steps

2 : Some previous group was not a Hit.

ISECT: Current group is not a Hit.

6 : Current group is a Hit.

ngrps ← ngrps - 1

If ngrps ≠ Ø go to Step GROOP else to Step TRDUN.

ISECT - Add D (I) to address ZLS. GT and store in adr2. This is the address of a KWC/BV list in ZLS. GT whose elements indicate which trunks are associated with the Ith group.

Set up to call INSECT as follows:

A-Reg ← ila: Points to adr1, adr2

B-Reg ← ina: Points to result

E-Reg ← Ø = NOP: Indicates 2-way intersection is wanted.

Call INSECT (intersection of lists at adr1, adr2). If null result, A-Reg returns = Ø.

Test A-Reg: If = Ø go to Step TALLY

If ≠ Ø to next.

;3 - Pick up D (I) afresh from memory. D(I) is now at -1 relative to (X-Reg) because of the X ← X + 1 sub-step in Step GROOP.

Set LSB (D (I)) = 1 as flag that the call on INSECT scored a Hit. Store back in memory.

3.1.2.2 continued -

GOTHIT - This step is entered from Step 3 if the Trunk List of the PRESENT group had a non-null intersection with the input Candidate Trunk List- OR - from Step 2 if the present group's trunk list is identical to some previous (lower numbered) group's trunk list and the latter list had a non-null intersection, etc. In either case, the present group's number is grpno and the purpose of this step and Steps 4-6 is to record grpno in the output Candidate List.

Compute and leave in B-Reg the address of the EL2 file pertaining to grpno:

B←EL2 + 11 * (grpno -1) (call E2ADR)

;4 - Look up the Group identity field (right byte of MIDth word relative to B), and build in E-Reg byte-split word:

Group Ident	grpno

;5 - Augment Candidate count:

ncand←ncand + 1

and store byte-split result of Step 4 at address resad + ncand.

;6 - Test to see whether there is room in the output Candidate List to store any more Candidates: limit←limit -1

If limit $\neq \emptyset$ go to Step TALLY. Else go to Step TRDUN.

TRDUN - This step is entered from either step.

6 : No room left in Output Candidate List

-OR-

TALLY: No more Groups left to consider

Whichever occurs first.

Store ncand in left byte of Output Candidate List header (1st word), i.e., at address resad. Note that

$1 \leq \text{ncand} \leq 32$,

3.1.2.2 continued -

TRDUN - i.e., we do not enter with a null Candidate Trunk List and each Trunk therein is associated with at least one Group.

;7 - Clean up stack, i.e., return to available status the temporary storage locations on stack appropriated in Steps TRNSL8 and 1 (S-Reg←—S-Reg + 26).

Return.

3.1.2.3 INSECT - Intersection Subroutine

This subroutine is called both by ECLV1 and by TRNSL8. The required inputs are:

A-Reg = Address of a list of 2 or 3 Input List addresses.

B-Reg = Address where intersection result is to be stored.

E-Reg = Switch instruction

= NOP if 2-way intersection
= JMP $ +2 if 3-way intersection

Both input lists and output list are in KWC/BV format. (See Section 3.3.1.2.2). The steps of INSECT are as follows:

INSECT - Save X-Reg on stack. Save E and B Registers on stack in local temporary storage area with symbolic names instr and outad respectively.

;1 - Repeat the following (*) three (3) times:

* Fetch Input List address pointed to by A-Reg and A←—A + 1. Push Input List address to stack.

The symbolic names of these three addresses are adr1, adr2, and adr3.

Put the Absolute address of adr3 in X-Reg. Make room for eleven more temporary storage locations on stack (S-Reg←—S-Reg - 11). See map in 3.3.2.3.

3.1.2.3 continued -

;2 - Set A-Reg = Ø and B-Reg = X'FFFF'. A will be the head (most significant half) of the double words created in next 5 steps. B will be used to hold the keyword intersection KWI as it is developed.

;3 - Execute (E X Q) the Switch instruction instr. If it is a NOP control goes to the instruction following the EXQ and this is a jump to Step INSØ1. If instr is a JMP $ + 2, go to next step.

;4 - X-Reg was set to address of adr3 in Step 1. Load E-Reg with KW3 = (adr3) and X←—X + 1. Store double register (A, E) = (Ø, KW3) as dw3. B←—B∧KW3. Go to Step 5.

INSØ1 - This step is entered only from Step 3 and Step 4 has not been executed. Therefore advance X-Reg so that it points to adr2 (X←—X + 1).

;5 - Load E-Reg with KW2 = (adr2) and X←—X + 1. Store double register (A, E) = (Ø, KW2) as dw2. B←—B ∧ KW2.

Load E-Reg with KW1 = (adr1) and X←—X + 1. Store double register (A, E) = (Ø, KW1) as dw1. B←—B∧KW1.

;6 - Define B-Reg contents ≡ KWI (Keyword intersection). If KWI = Ø we have an early detection of null intersection: Go to Step INDUN1.

Else move KWI to E-Reg and store double register (A, E) = (Ø, KWI) as dwi.

;7 - rsi←—Ø (Result storage index)

cnt←—16 (Shift counter)

INLOOP - This major loop is entered as many times as there are 1's bits in KWI, initially from Step 7 and thereafter from Step INSØ5. All steps down to and including INSØ5 are inside this loop.

Set X-Reg = 1 and load double register (A, E)←—dwi.

3.1.2.3 continued -

INSHFT - Here begins a minor loop whose object is to shift (A, E) left one bit at a time until the LSB of A-Reg = 1 and to record the number of such shifts in X-Reg.

If E-Reg <Ø (MSB (E) = 1) go to Step INSØ2.

Else, shift (A, E) left 1 bit (Ø enters LSB (E))

X←—X + 1

Go to Step INSHFT

INSØ2 - Shift (A, E) left once more and store (A, E) as updated dwi. The shift count is in X-Reg and is maintained there through rest of loop. For ease of description we call it NSH from now on.

;11 - (A, E)←—dw1.

Shift (A, E) left NSH bits (NSH Ø's enter at right) dw1←—(A, E).

Call PARITY to do A-Reg←—1's bit Count (A-Reg), and add this to address adr1. We now have the List -1 binary vector word corresponding to the 1 in KWI which was just shifted into LSB [Head (dwi)]

Fetch this word and store as result word (rew).

;12 - Repeat Step 11 VIS-A-VIS dw2 except that word fetched from List -2 is ANDED with rew and held in A-Reg.

;13 - Execute (EXQ) switch instruction instr. As before, A NOP (2-way) takes us to a jump around next step (to Step INSØ3) while a JMP $ + 2 (3-way) takes us to next step.

;14 - Store partial result in A-Reg as rew. Repeat Step 12 VIS-A-VIS dw3.

INSØ3 - A-Reg now contains the intersection of 2 or 3 binary vector words.

If A-Reg ≠ Ø, A-Reg is a new, valid binary vector word of the result: Go to Step INSØ4. else next step.

3.1.2.3 continued -

;16 - The binary vector words corresponding to the 1 in KWI which is now in the LSB of the head of dwi have a null intersection which is not to be stored in the result. Therefore- -

Reset LSB [Head (dwi)] = Ø. Go to Step INSØ5.

INSØ4 - Do rsi←—rsi + 1

Create address outad + rsi and store result word threat.

INSØ5 - Tests for completion. This step is reached from Steps 16 and INSØ4.

Do cnt←—cnt - NSH

If cnt now = Ø go to Step INDUN2

Else, do (A, E)←—dwi.

If TAIL (dwi), i.e., E-Reg ≠ Ø, there are still 1's in KWI that have to be examined: Go to Step INLOOP else out of loop to next step.

;19 - Out of major loop. cnt was not = Ø but there were no more 1's left in KWI to be examined. dwi is in (A, E).

Shift (A, E) left cnt bits.

INDUN1 - This step is reached from Steps 6 (KWI = Ø to start, A-Reg = Ø), 19 (dwi is in (A, E) and cnt has been forced to Ø) and from step INDUN2 (cnt was = Ø, INDUN2 loaded A with head (dwi)).

In any case, A-Reg = true keyword of the result. Store this at address = (outad).

Clean up stack, i.e., return to available status the 16 words of temporary stack storage appropriated in entry steps INSECT and 1. This is done by S-Reg←—S-Reg + 16.

Restore X-Reg from stack.

Return to caller at Call + 1.

3.1.2.3 Continued -

INDUN2 - This step is entered only from Step INSØ5 when cnt = Ø. This can only come about if LSB of original KWI (Step 6) was = 1.

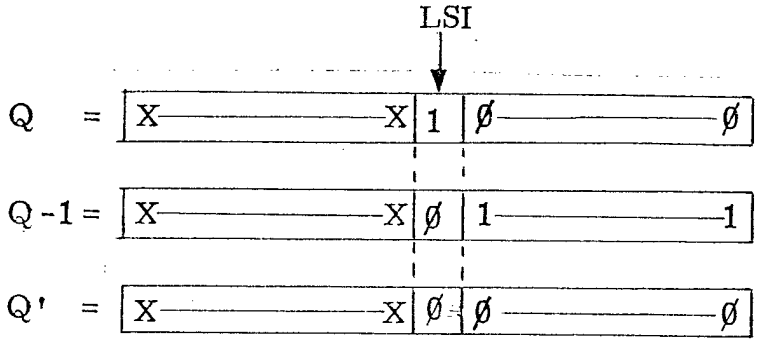
Set up proper exit conditions: Load A-Reg with head (dwi).

3.1.2.3.1 PARITY - Count 1's in A-Reg

This is a subroutine of INSECT. It is based on the following theorem:

Theorem: Let Q be an unsigned, non-zero binary number represented in a register of finite length. Then Q' = Q ∧ (Q - 1) has one fewer 1 bit than Q.

Proof: Theorem is obviously true for Q = 1. Therefore, consider some Q > 1 and define LSI as the position of Q's least significant 1 bit.



Input: Argument and result are both in A-Reg.

Steps:

PARITY - If A-Reg = Ø go to Step DONE.

;1 - Save B & E-Reg's on stack

Set B←Ø. B-Reg will hold count until end.

PLOOP - E←A
E←E - 1
A←A ∧ E
B←B + 1

3.1.2.3.1 continued -

;2 - If A-Reg ≠ Ø go to Step PLOOP, else next step.

;3 - Move count from B-Reg to A-Reg. Restore E and B-Reg's from stack.

DONE - This step reached from Step PARITY (input argument was = Ø) and Step 3 (algorithm complete).

In either case, A-Reg = result.

Return to caller at Call +1.

3.1.3 ECST1 - Scan Test 1

This is a subroutine of ECDR. The required input is X-Reg = Emitter Track File Number (EFN).

ECST1 determines whether or not ECDR shall make a true (AW Bit = 1) Scan-Analysis request or a false one (AW Bit = Ø). In the former case, it returns to Call +2 and in the latter to Call +1.

ECST1 - Save A & B-Reg's on stack.

;2 - Build address of ETF (EFN) = ETF + 16 * EFN in X-Reg and leave it there as X-Reg exit value.

;3 - Fetch ETF Peak Amplitude field (EPMPD rel to X) masking out rest of word (Mask EPMPM) and compare to Amplitude Threshold (AMTHR) aligned with Peak Amplitude (avoids dynamic shift).

If Peak Amp ≤ Amp Thres go to Step 4.

If Peak Amp > Amp Thres go to Step 5.

;4 - Call SCTCOM. This is logic common to Scan Tests 1 and 2. This returns to Step 6.

;5 - Bump return address to Call +2.

;6 - Reach here from Steps 4 or 5.

Restore B and A-Regs

Return to Call +1 or Call +2 as Step 5 wasn't/was executed.

3.1.3.1 SCTCOM - Common Logic of Scan Tests 1 & 2

Called by Scan Tests 1 (ECST1) and 2 (ANST2 - in Document #53959-GT-0761) X-Reg = Address ETF (EFN) on entry and exit.

Complements ETF State Indicator and if now on sets ETF scan type to circular and scan-period to time-out. Always exits to Call +2.

3.1.3.1 continued -

SCTCOM - Bump return address to Call +2.

;1 - B←State Indicator Mask (ESINM)

A←State Indicator (ESIND rel to X)

A←X OR (A, B) - This complements State Indicator. Store result (ESIND rel to X).

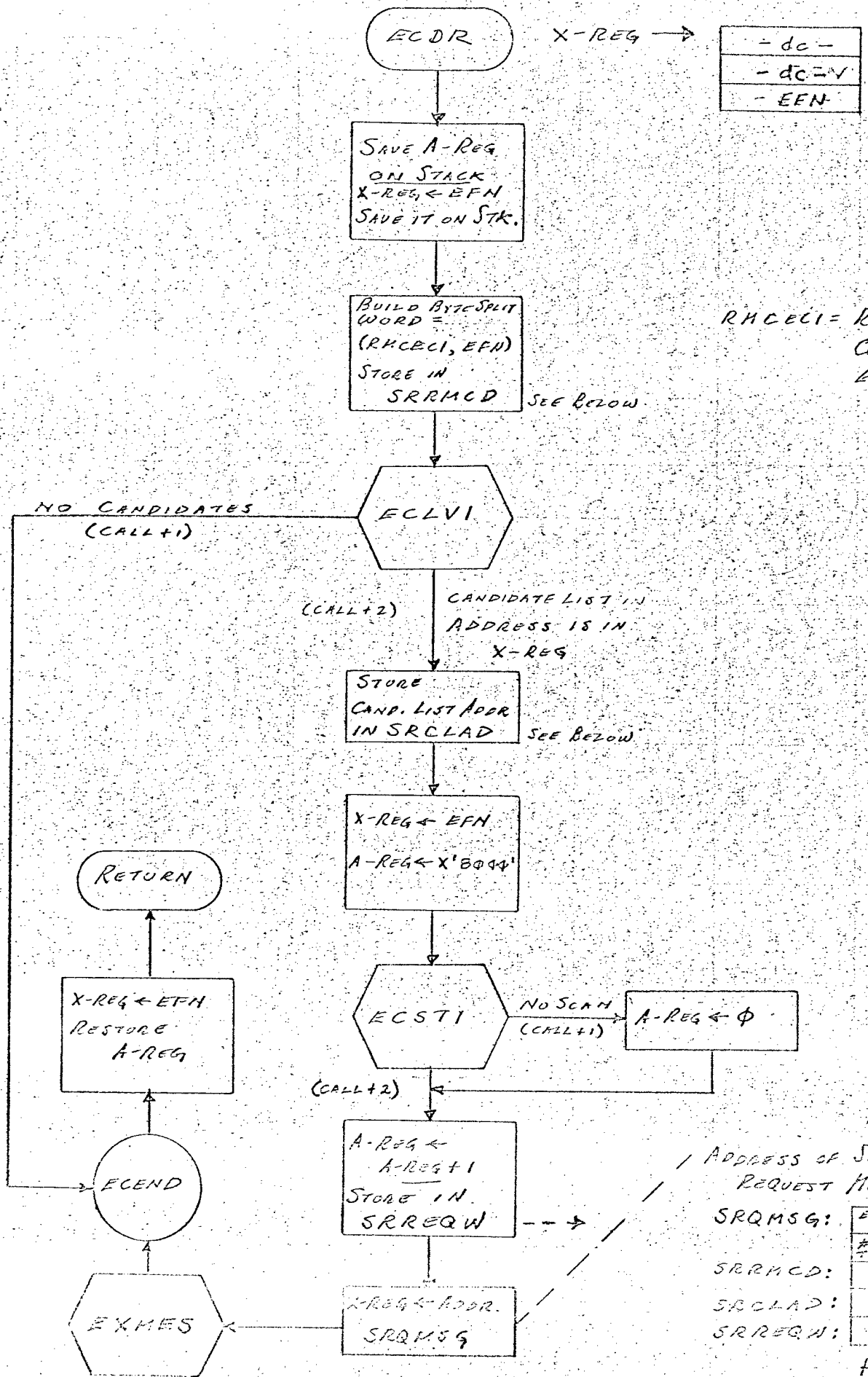
;2 - Isolate present value of State Indicator in A-Reg by A←A ∧ B.

If State Indicator (A-Reg) = Ø go to Step OUT, else next step.

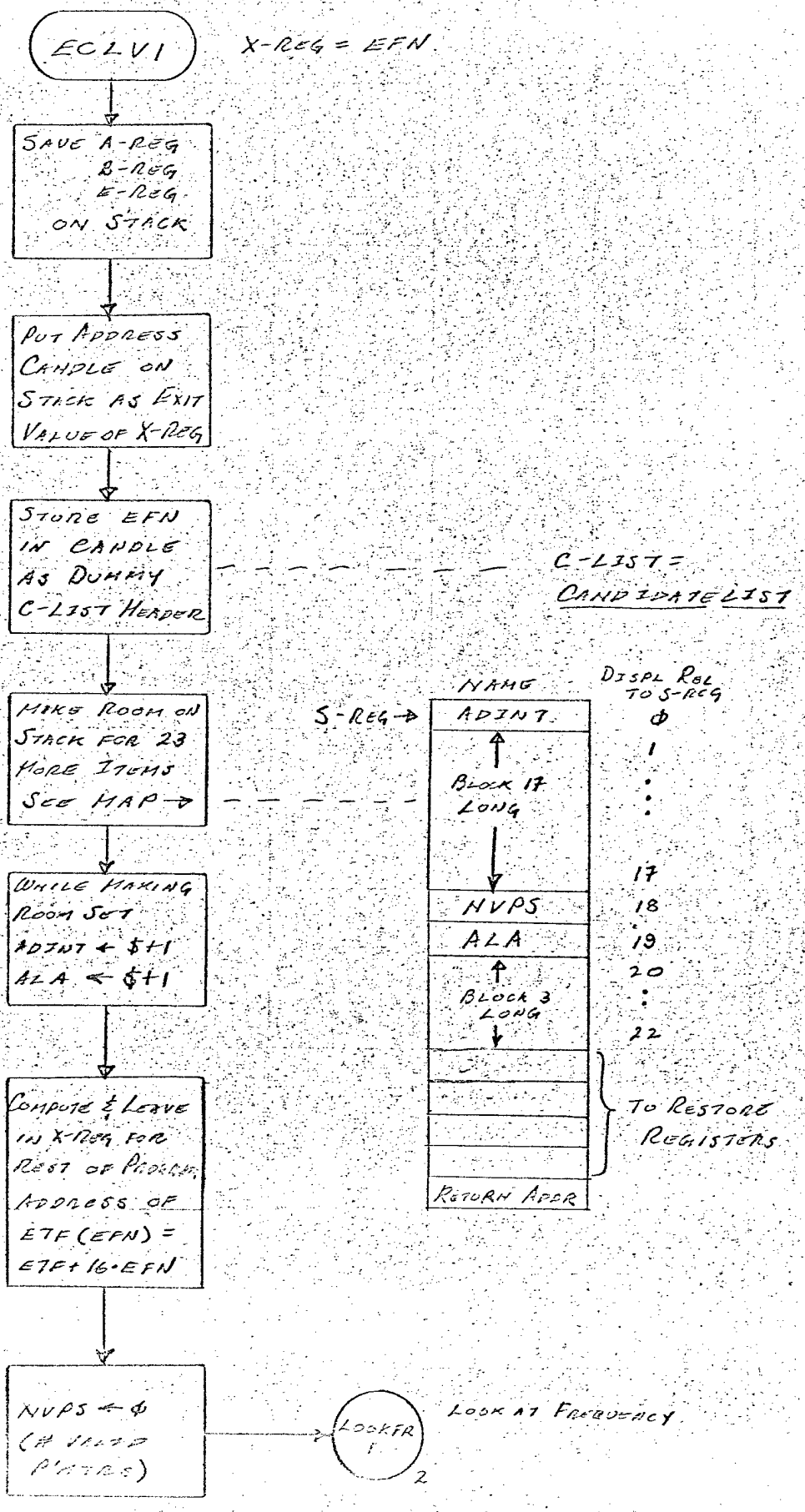
;3 - Fetch statically encoded word containing both the circular scan type (CIRC) and the time-out scan period (TOUT) aligned to their respective fields. Store this in the common location of the two fields (ESTYD rel to X).

OUT - Reached from Steps 2 and 3.

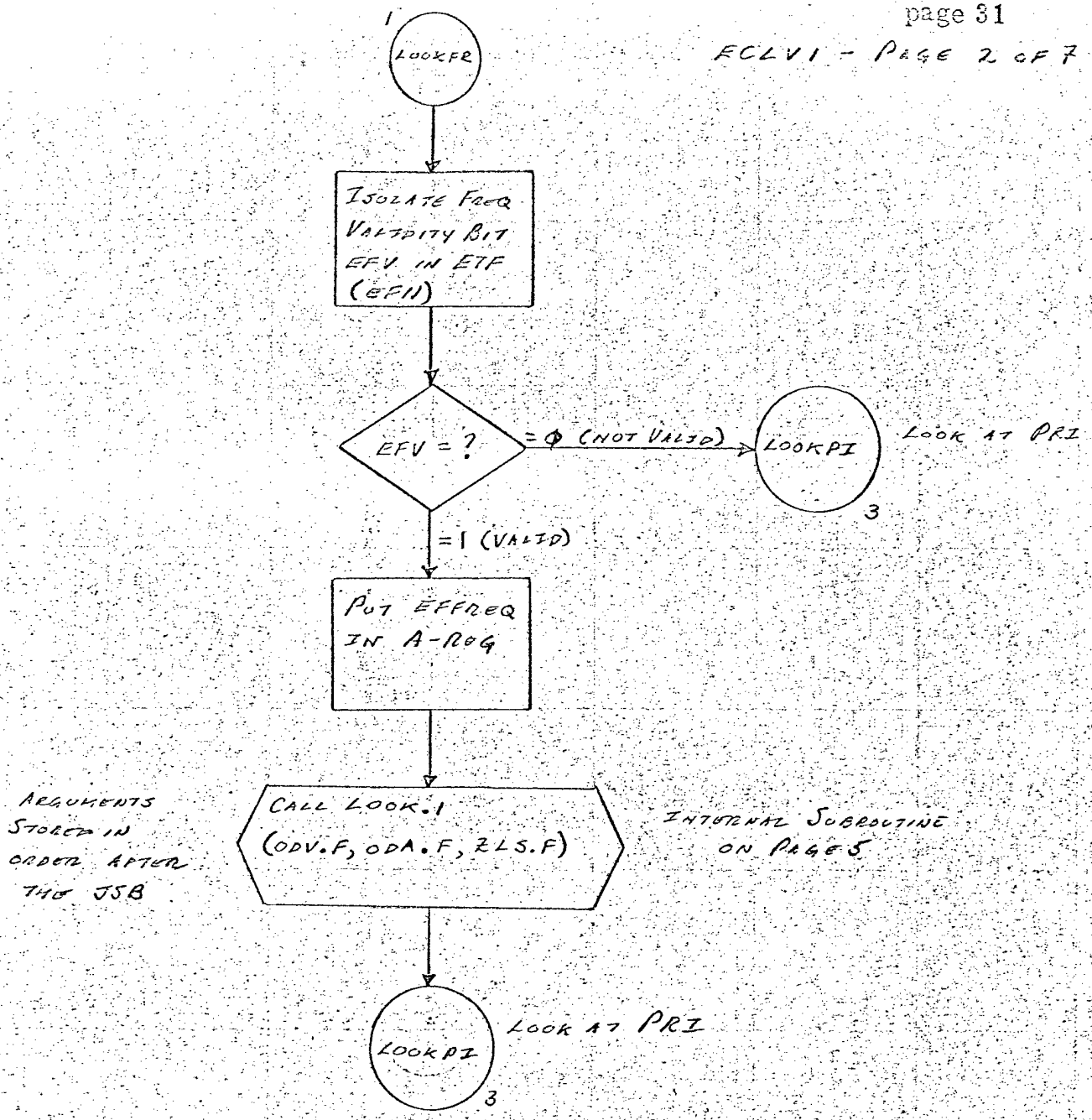
Return to Call +2.

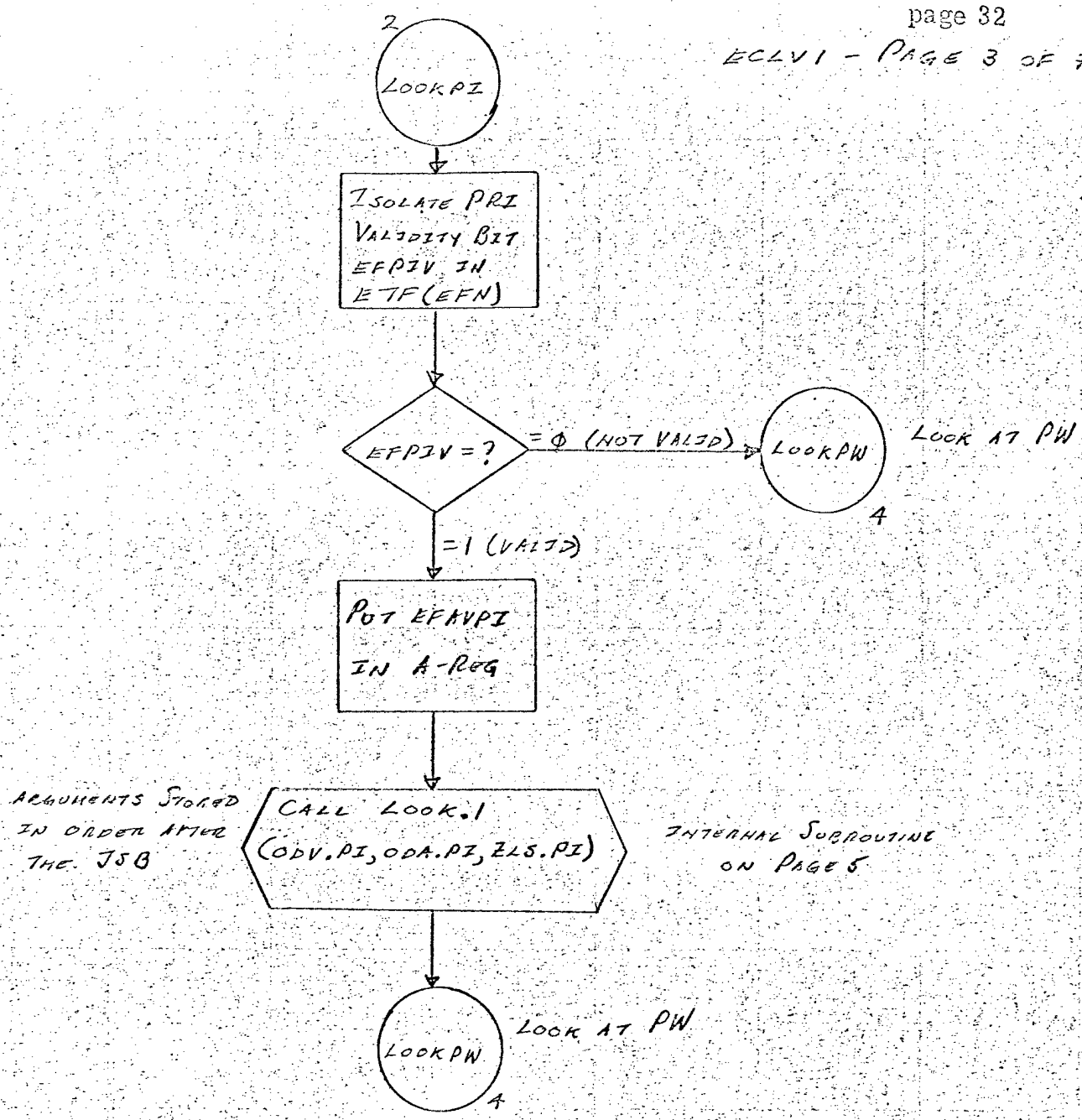

ECDR
X-REG →
- dc -
- dc=V
- EFN
SAVE A-REG ON STACK
X-REG ← EFN
SAVE IT ON STK.
BUILD BYTE SPLIT WORD = (RMCECI, EFN)
STORE IN SRRMCD
SEE BELOW
RMCECI = RETURN MODULE CODE FOR EMITTED CLASS. 1
ECLVI
NO CANDIDATES (CALL+1)
(CALL+2)
CANDIDATE LIST IN ADDRESS IS IN X-REG
STORE CAND. LIST ADDR IN SRCLAD
SEE BELOW
X-REG ← EFN
A-REG ← X'8004'
RETURN
X-REG ← EFN
RESTORE A-REG
ECSTI
NO SCAN (CALL+1)
A-REG ← 0
(CALL+2)
A-REG ← A-REG+1
STORE IN SRREQW
ECEND
EXMES
X-REG ← ADDR. SRQMSG
ADDRESS OF SCAN-ANALYSIS REQUEST MESSAGE --
SRQMSG:
EXEC. MSG. #
WDS TO FOLLOW = 3
SRRMCD:
f
f
SRCLAD:
f
SRREQW:
f
f: filled by this routine
SEND MESSAGE TO EXECUTIVE

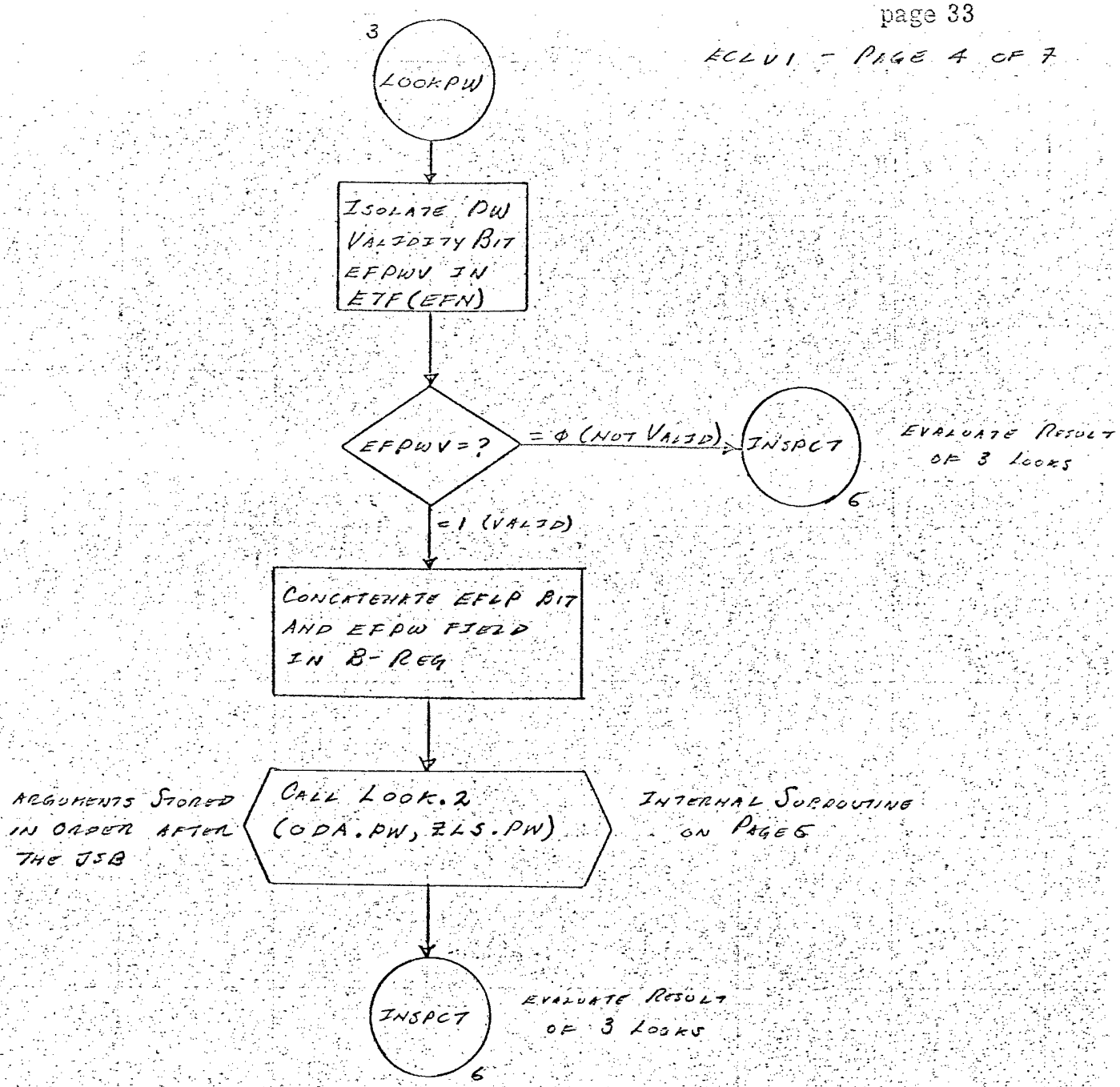
3.2.2 ECLVI

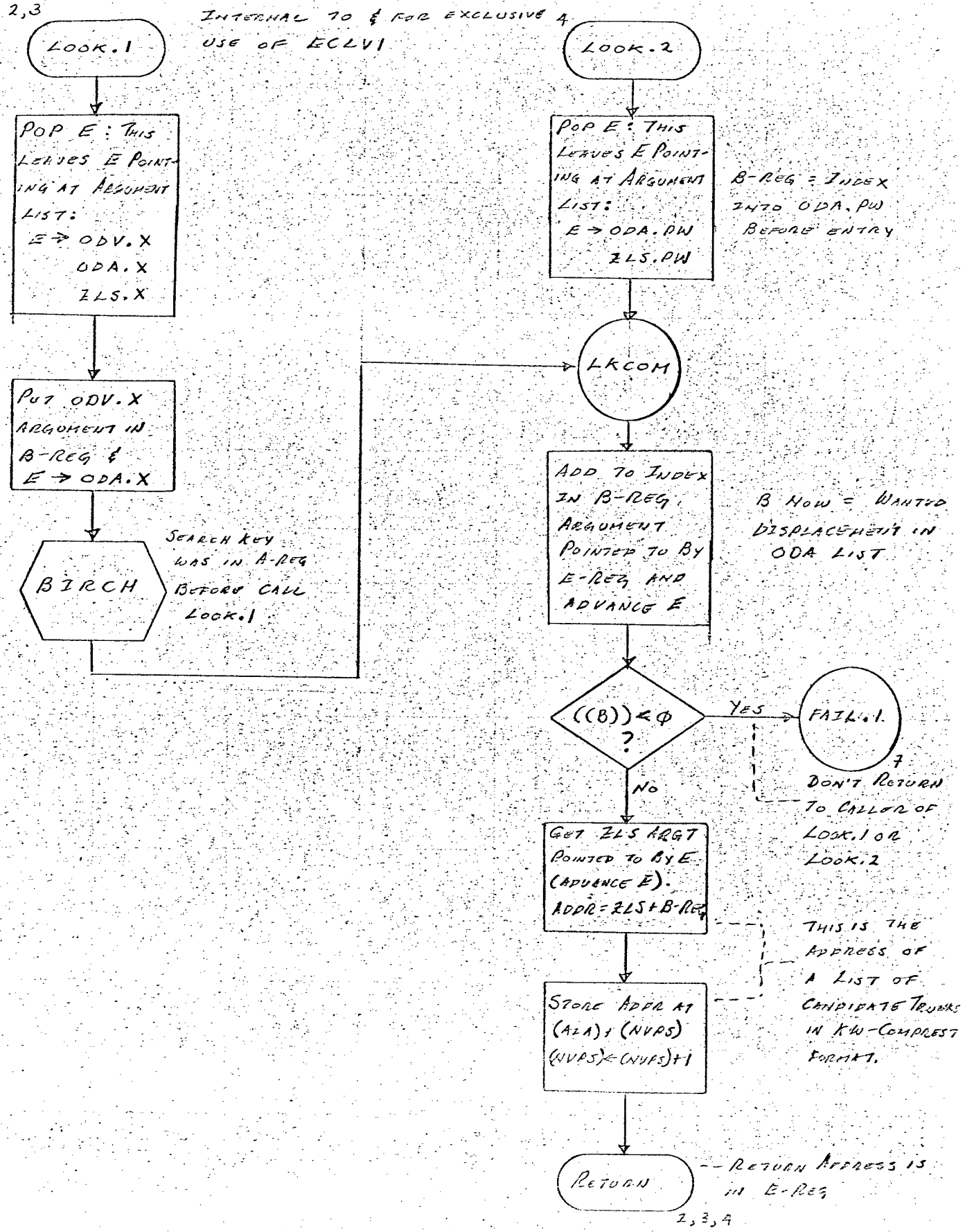

ECLVI
X-REG = EFN
SAVE A-REG
B-REG
E-REG
ON STACK
PUT ADDRESS
CANDLE ON
STACK AS EXIT
VALUE OF X-REG
STORE EFN
IN CANDLE
AS DUMMY
C-LIST HEADER
C-LIST =
CANDIDATE LIST
MAKE ROOM ON
STACK FOR 23
MORE ITEMS
SEE MAP →
S-REG →
NAME
DISPL REL TO S-REG
ADINT
0
1
BLOCK 17 LONG
17
NVPS
18
ALA
19
20
BLOCK 3 LONG
22
TO RESTORE REGISTERS
RETURN ADDR
WHILE MAKING
ROOM SET
ADINT ← $+1
ALA ← $+1
COMPUTE & LEAVE
IN X-REG FOR
REST OF PROGRAM
ADDRESS OF
ETF(EFN) =
ETF+16·EFN
NVPS ← 0
(# SAVED
PTRS)
LOOKFR
1
2
LOOK AT FREQUENCY

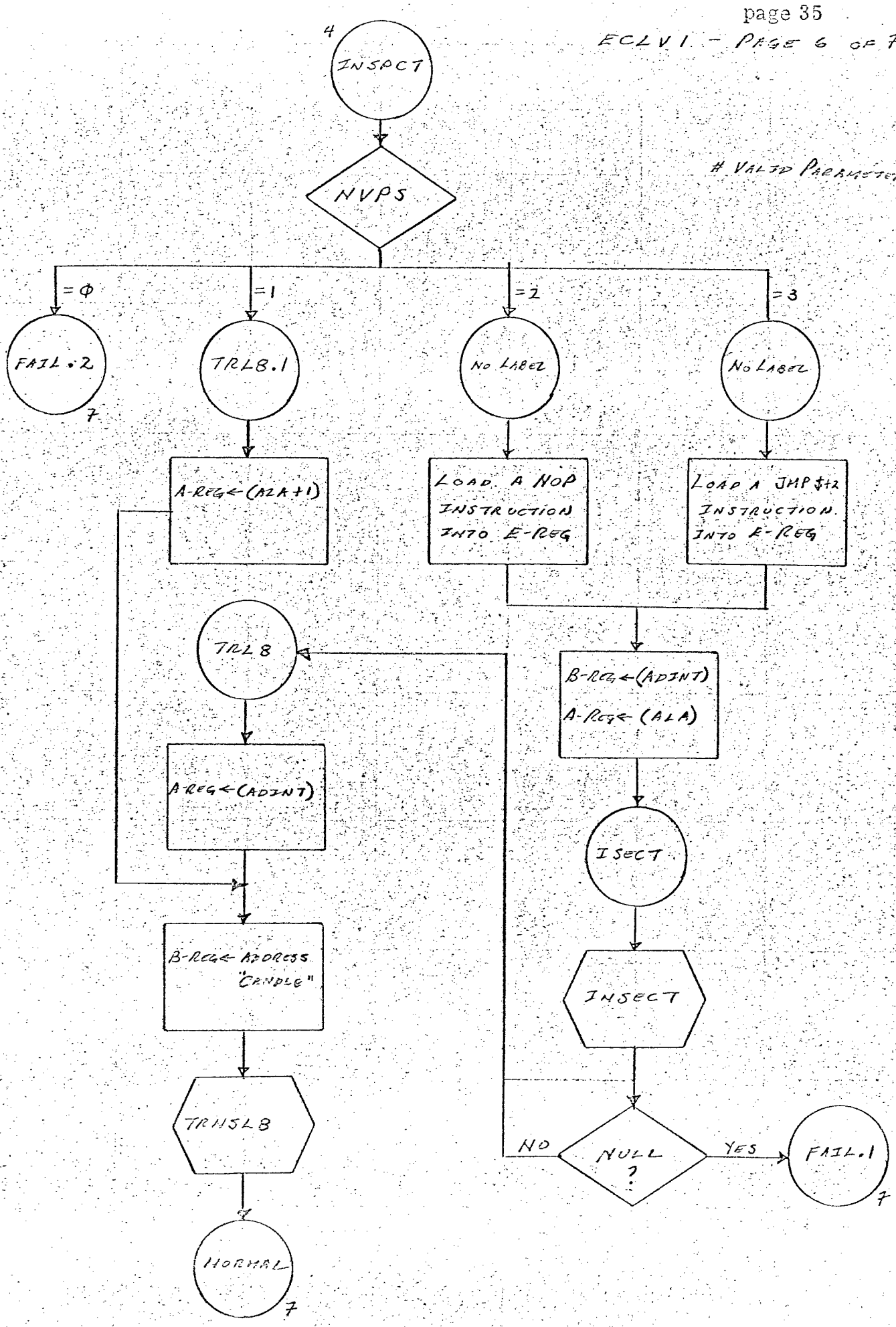
ECLV1 - PAGE 2 OF 7

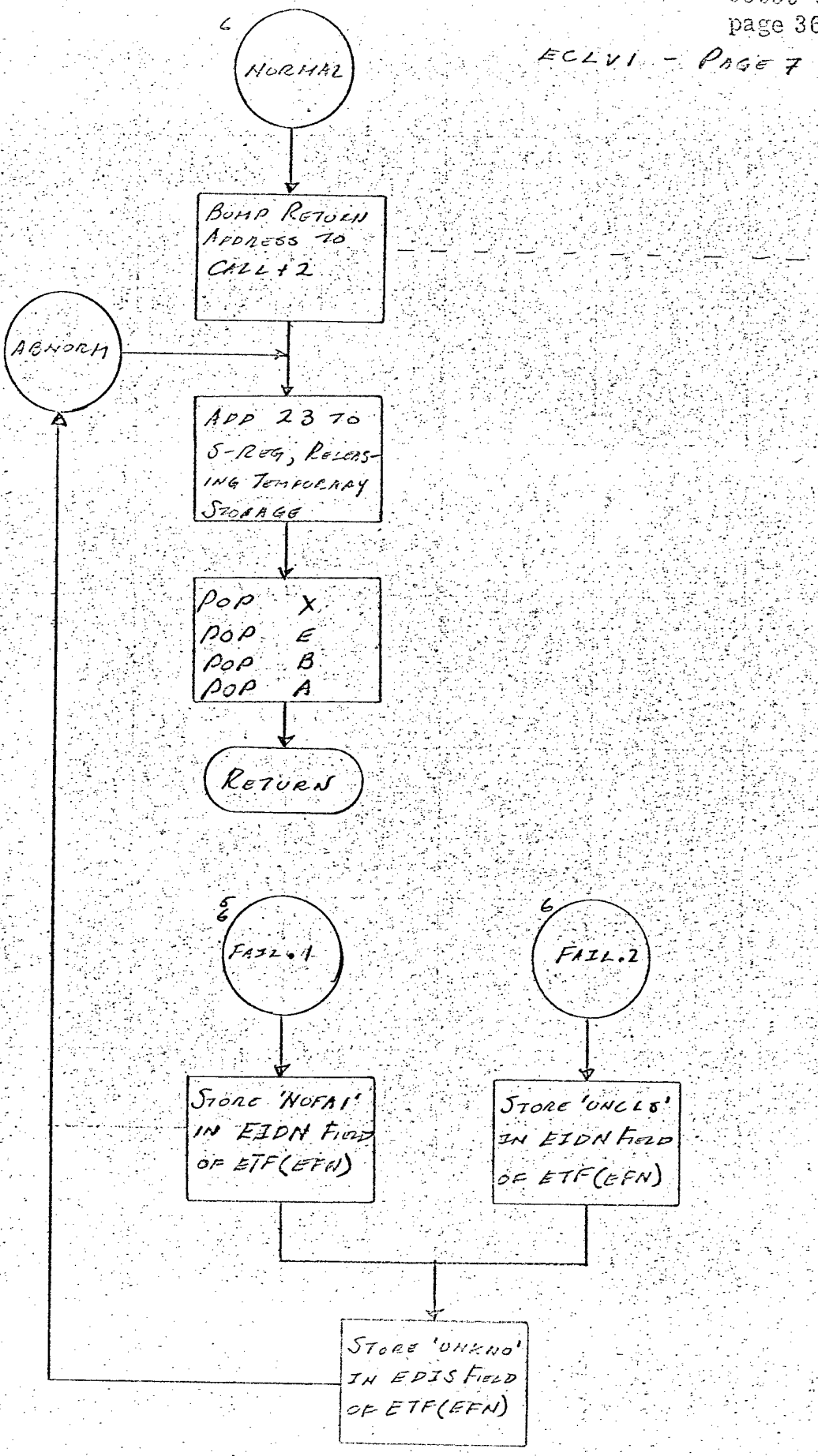



2
LOOKPI
ISOLATE PRI VALIDITY BIT EFPIV IN ETF(EFN)
EFPIV = ?
=0 (NOT VALID)
LOOKPW
4
LOOK AT PW
=1 (VALID)
PUT EFAVPI IN A-REG
ARGUMENTS STORED IN ORDER AFTER THE JSB
CALL LOOK.1 (ODV.PI, ODA.PI, ZLS.PI)
INTERNAL SUBROUTINE ON PAGE 5
LOOKPW
4
LOOK AT PW


3
LOOKPW
ISOLATE PW VALIDITY BIT EFPWV IN E7F(EFN)
EFPWV = ?
= 0 (NOT VALID)
INSPCT
6
EVALUATE RESULT OF 3 LOOKS
= 1 (VALID)
CONCATENATE EFLP BIT AND EFPW FIELD IN B-REG
ARGUMENTS STORED IN ORDER AFTER THE JSB
CALL LOOK.2 (ODA.PW, ZLS.PW)
INTERNAL SUBROUTINE ON PAGE 6
INSPCT
6
EVALUATE RESULT OF 3 LOOKS

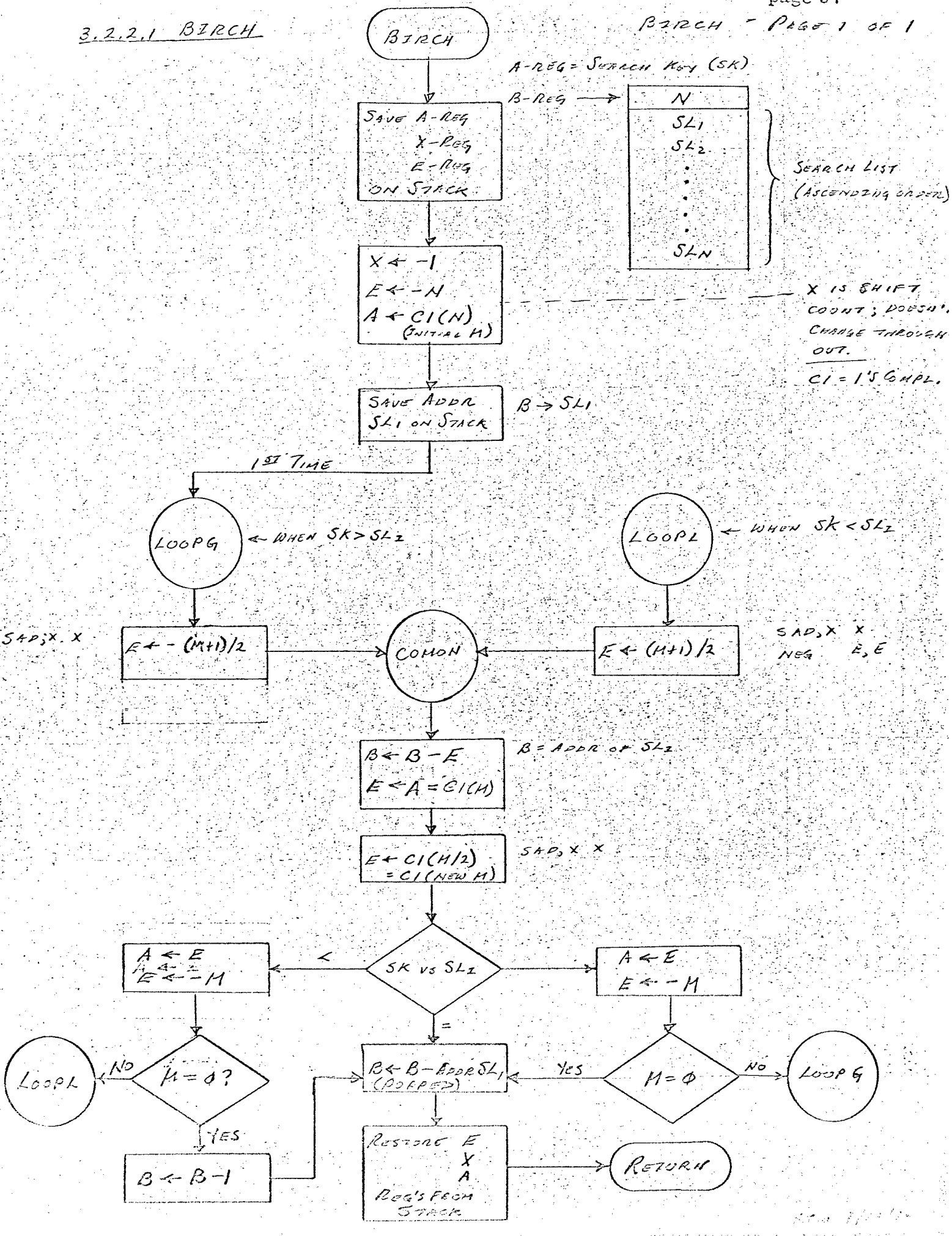

2,3
INTERNAL TO ϕ FOR EXCLUSIVE USE OF ECLV1
LOOK.1
POP E: THIS LEAVES E POINTING AT ARGUMENT LIST:
E → ODV.X
ODA.X
ZLS.X
PUT ODV.X ARGUMENT IN B-REG & E → ODA.X
BIRCH
SEARCH KEY WAS IN A-REG BEFORE CALL LOOK.1
LOOK.2
POP E: THIS LEAVES E POINTING AT ARGUMENT LIST:
E → ODA.PW
ZLS.PW
B-REG = INDEX INTO ODA.PW BEFORE ENTRY
LKCOM
ADD TO INDEX IN B-REG, ARGUMENT POINTED TO BY E-REG AND ADVANCE E
B NOW = WANTED DISPLACEMENT IN ODA LIST
((B)) < ϕ ?
YES
FAIL.1.
7
DON'T RETURN TO CALLER OF LOOK.1 OR LOOK.2
NO
GET ZLS ARGT POINTED TO BY E (ADVANCE E). ADDR = ZLS + B-REG
THIS IS THE ADDRESS OF A LIST OF CANDIDATE TRUNKS IN KW-COMPREST FORMAT.
STORE ADDR AT (ALA) + (NVPS)
(NVPS) ← (NVPS) + 1
RETURN
-- RETURN ADDRESS IS IN E-REG
2,3,4


4
INSPCT
NVPS
VALID PARAMETERS
=0
=1
=2
=3
FAIL.2
7
TRLB.1
NO LABEL
NO LABEL
A-REG←(ALA+1)
LOAD A NOP INSTRUCTION INTO E-REG
LOAD A JMP $+2 INSTRUCTION INTO E-REG
TRLB
B-REG←(ADINT)
A-REG←(ALA)
A-REG←(ADINT)
ISECT
B-REG← ADDRESS "CANDLE"
INSECT
TRNSLB
NO
NULL ?
YES
FAIL.1
7
NORMAL
7

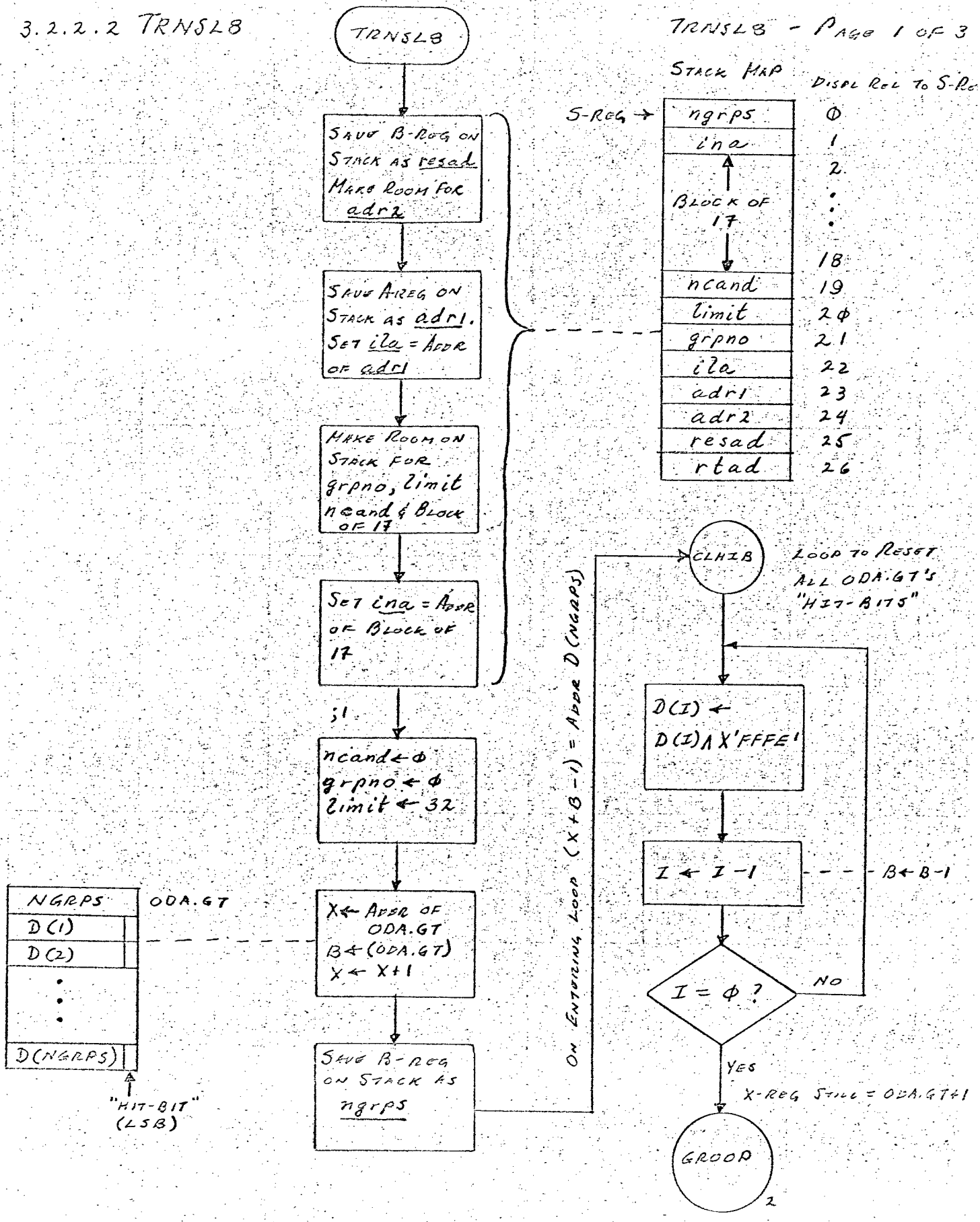

6
NORMAL
BUMP RETURN ADDRESS TO CALL+2
28 LOCN'S FROM TOP OF STACK
ABNORM
ADD 23 TO S-REG, RELEASING TEMPORARY STORAGE
POP X
POP E
POP B
POP A
RETURN
6
6
FAIL.1
FAIL.2
STORE 'NOFAI' IN EIDN FIELD OF ETF(EFN)
STORE 'UNCLS' IN EIDN FIELD OF ETF(EFN)
STORE 'UNKNO' IN EDIS FIELD OF ETF(EFN)

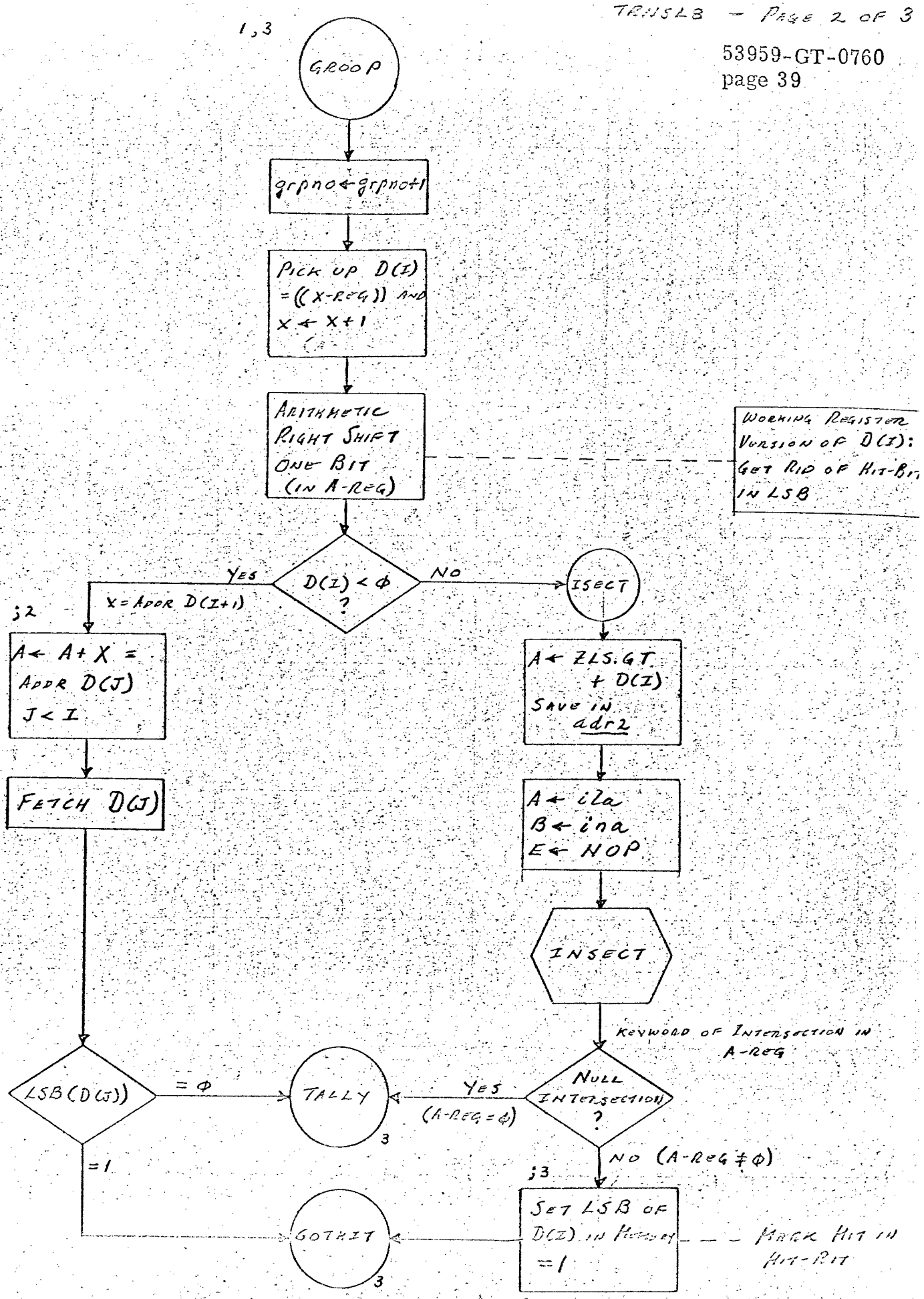
3.2.2.1 BIRCH

BIRCH - PAGE 1 OF 1

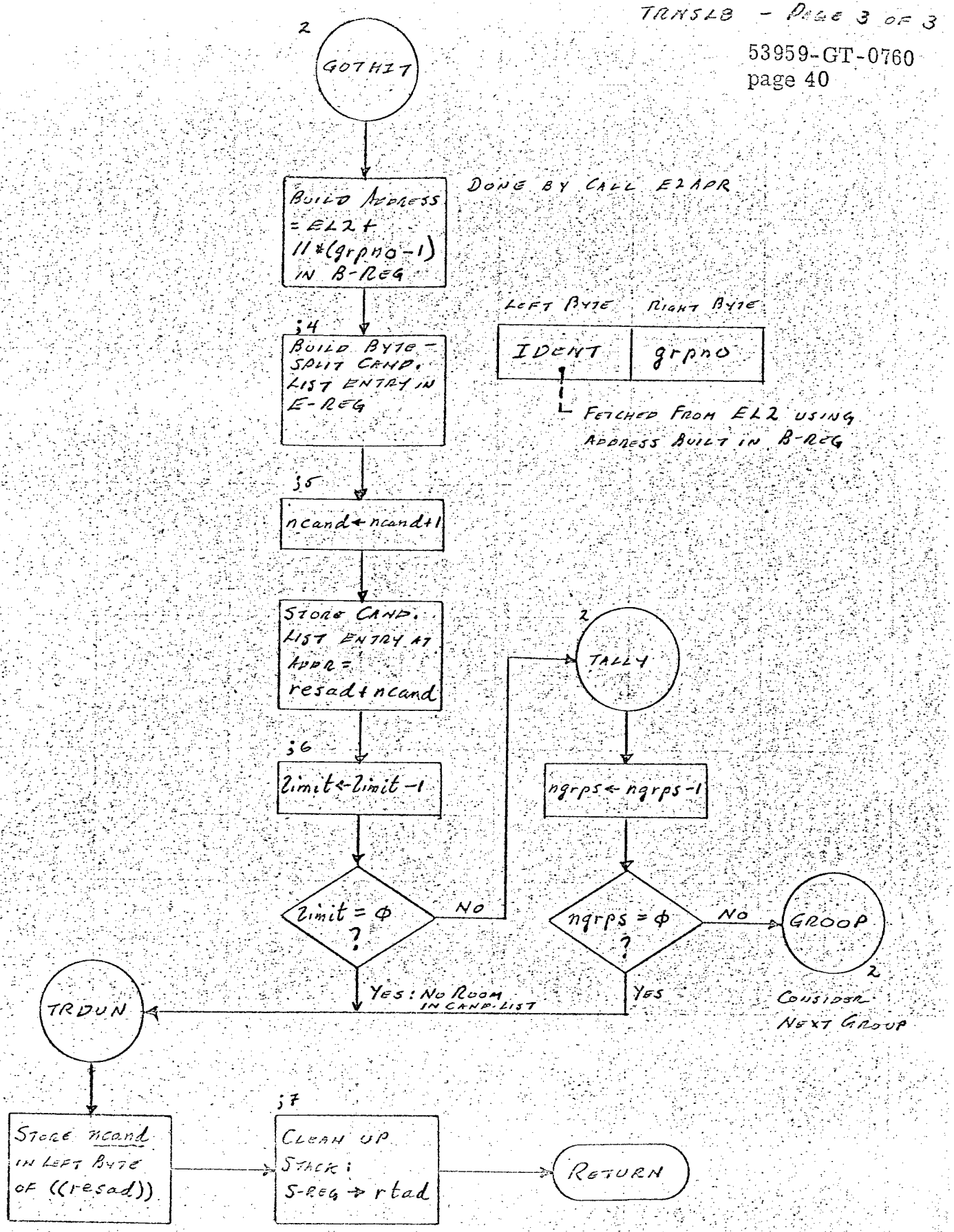


3.2.2.2 TRNSLB

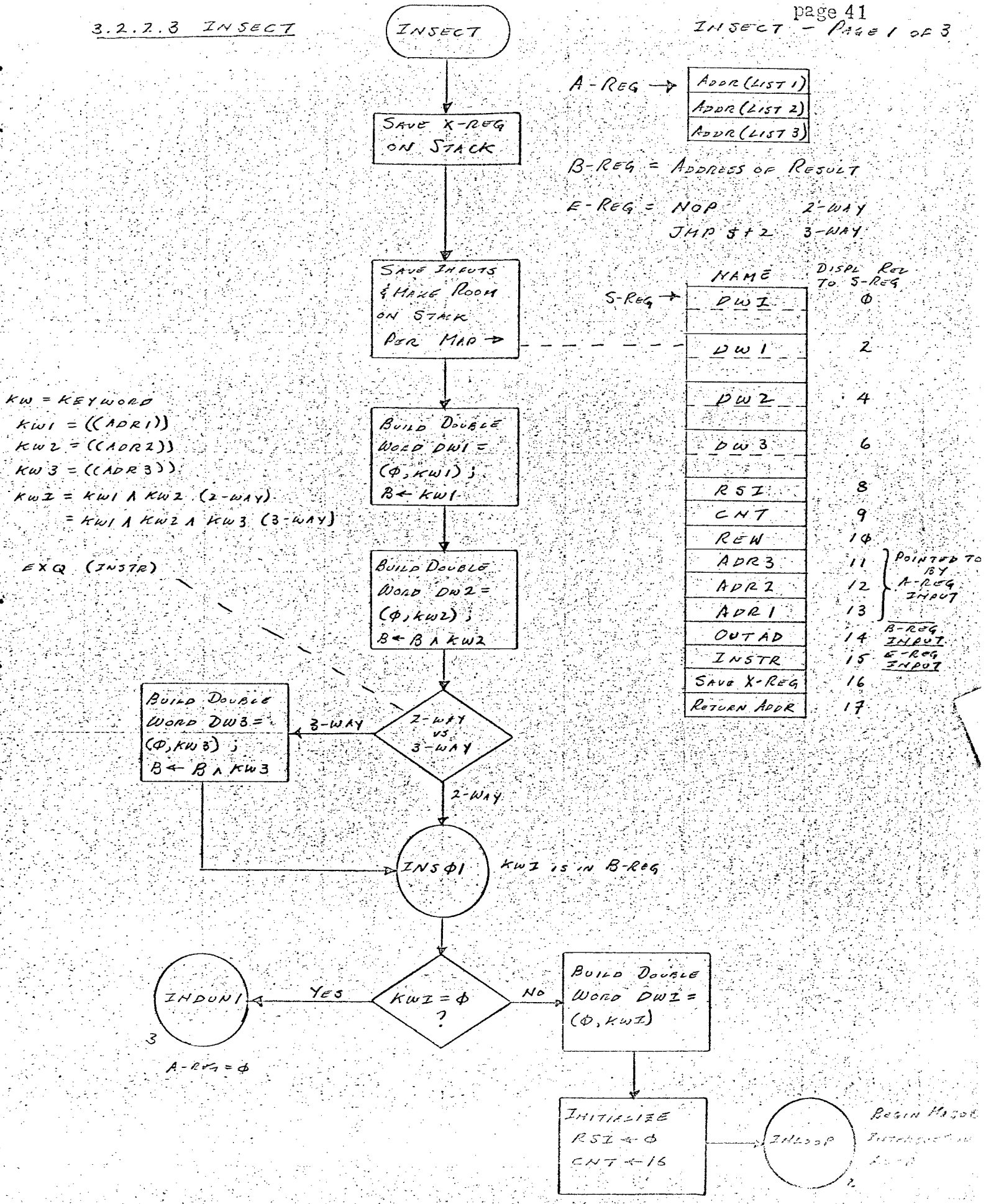

TRNSLB
TRNSLB - PAGE 1 OF 3
STACK MAP
DISPL REL TO S-REG
S-REG →
ngrps 0
ina 1
BLOCK OF 17 2 ... 18
ncand 19
limit 20
grpno 21
ila 22
adr1 23
adr2 24
resad 25
rtad 26
SAVE B-REG ON STACK AS resad
MAKE ROOM FOR adr2
SAVE A-REG ON STACK AS adr1.
SET ila = ADDR OF adr1
MAKE ROOM ON STACK FOR grpno, limit ncand & BLOCK OF 17
SET ina = ADDR OF BLOCK OF 17
;1
ncand ← 0
grpno ← 0
limit ← 32
X ← ADDR OF ODA.GT
B ← (ODA.GT)
X ← X+1
SAVE B-REG ON STACK AS ngrps
NGRPS
D(1)
D(2)
D(NGRPS)
ODA.GT
"HIT-BIT" (LSB)
ON ENTERING LOOP (X+B-1) = ADDR D(NGRPS)
CLHIB
LOOP TO RESET ALL ODA.GT'S "HIT-BITS"
D(I) ← D(I) ∧ X'FFFE'
I ← I-1
B ← B-1
I = 0 ?
NO
YES
X-REG STILL = ODA.GT+1
GROOP
2

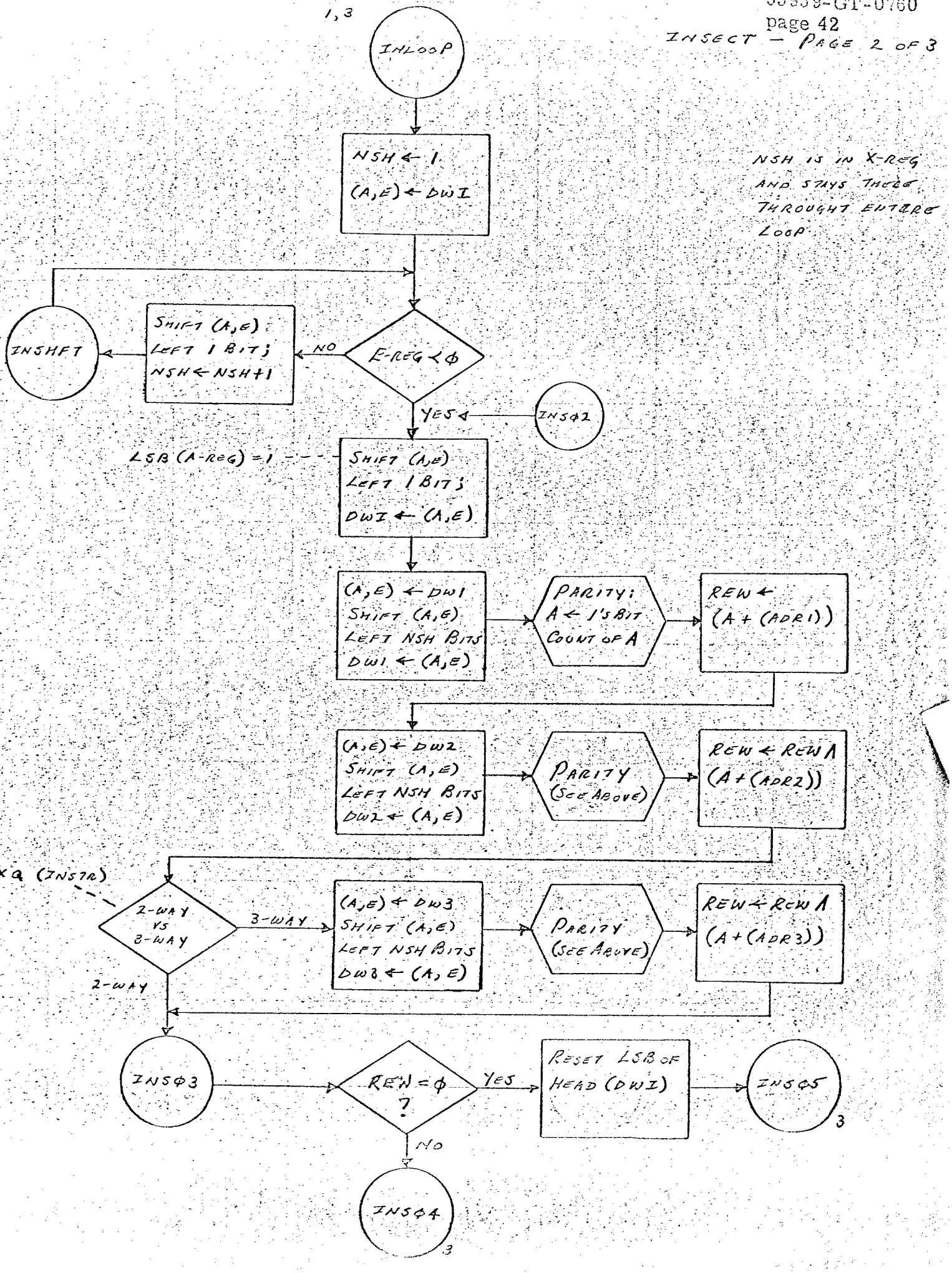

1,3
GROOP
grpno ← grpno+1
PICK UP D(I) = ((X-REG)) AND X ← X+1
ARITHMETIC RIGHT SHIFT ONE BIT (IN A-REG)
WORKING REGISTER VERSION OF D(I): GET RID OF HIT-BIT IN LSB
D(I) < Φ ?
YES
NO
X = ADDR D(I+1)
;2
A ← A+X = ADDR D(J) J < I
FETCH D(J)
ISECT
A ← ZLS.GT. + D(I) SAVE IN adr2
A ← ila B ← ina E ← NOP
INSECT
KEYWORD OF INTERSECTION IN A-REG
NULL INTERSECTION ?
YES (A-REG = Φ)
NO (A-REG ≠ Φ)
LSB(D(J))
= Φ
= 1
TALLY
3
;3
SET LSB OF D(I) IN MEMORY = 1
MARK HIT IN HIT-BIT
GOTHIT
3

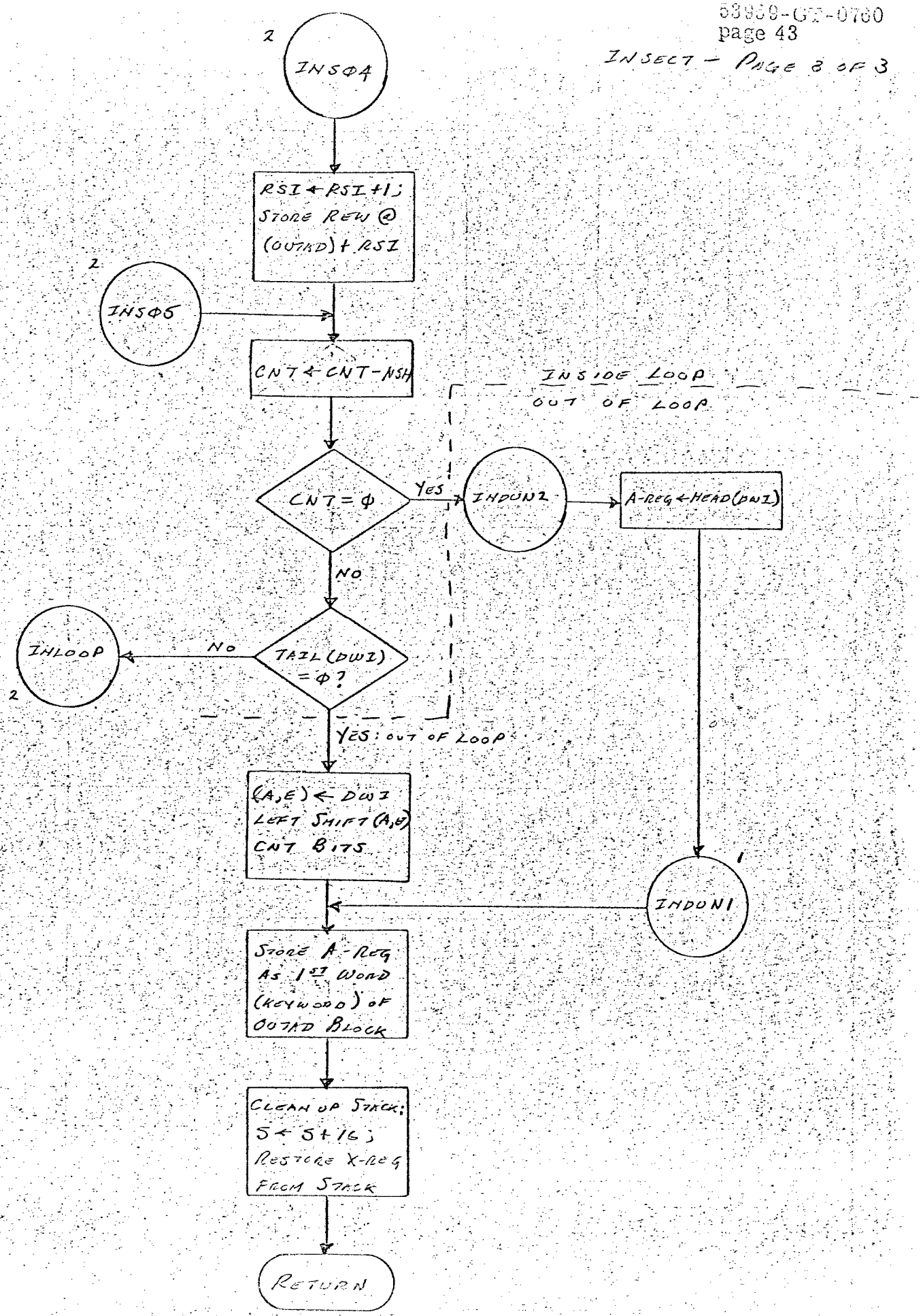
SJW 10/6/76

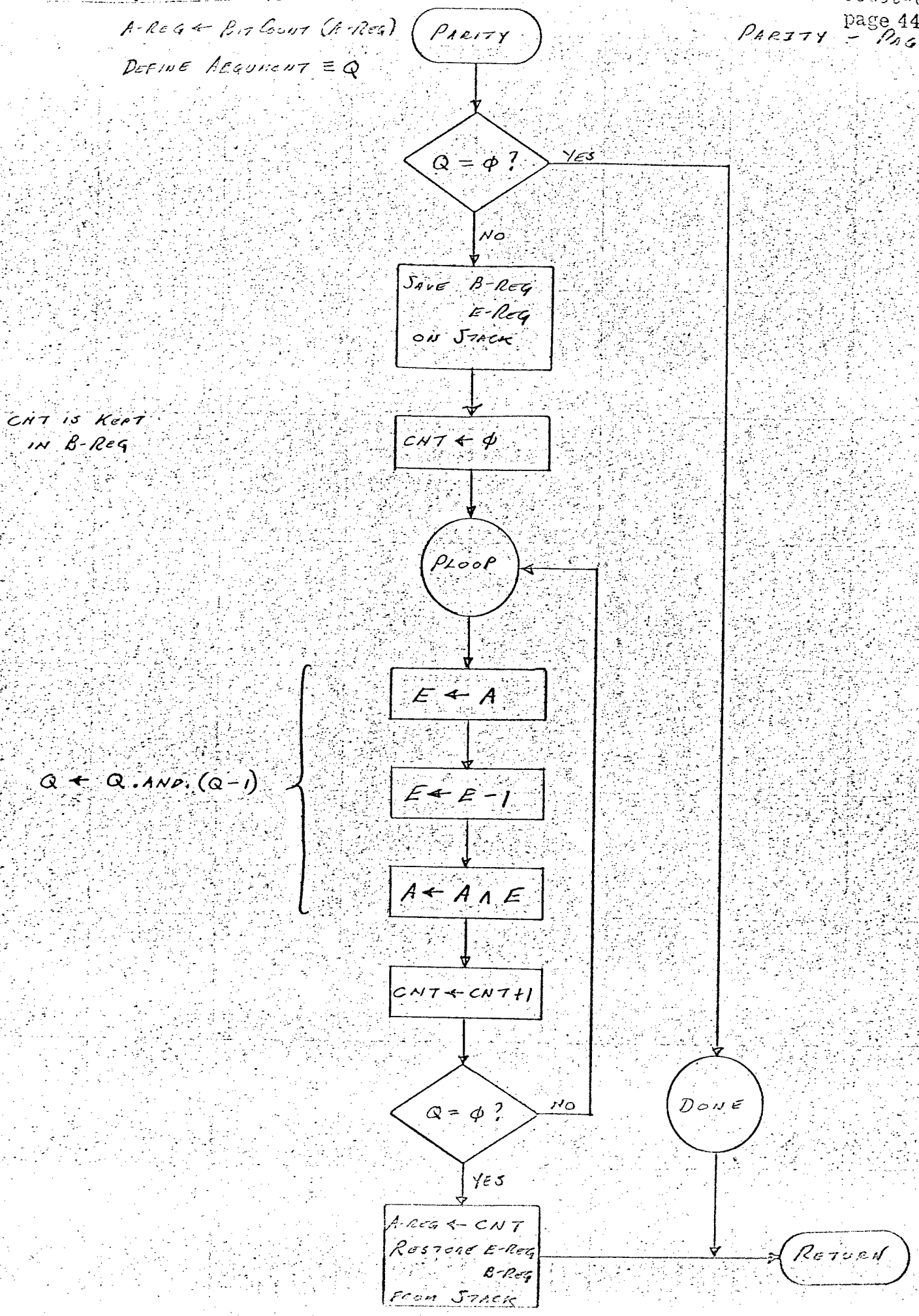

2
GOTHIT
BUILD ADDRESS = EL2+ 11*(grpno-1) IN B-REG
DONE BY CALL E2APR
LEFT BYTE
RIGHT BYTE
IDENT
grpno
FETCHED FROM EL2 USING ADDRESS BUILT IN B-REG
;4
BUILD BYTE - SPLIT CAND. LIST ENTRY IN E-REG
;5
ncand←ncand+1
STORE CAND. LIST ENTRY AT ADDR = resad+ncand
;6
limit←limit-1
limit = ϕ ?
NO
YES: NO ROOM IN CAND. LIST
2
TALLY
ngrps←ngrps-1
ngrps = ϕ ?
NO
GROOP
2
CONSIDER NEXT GROUP
YES
TRDUN
STORE ncand IN LEFT BYTE OF ((resad))
;7
CLEAN UP STACK: S-REG → rtad
RETURN

3.2.2.3 INSECT

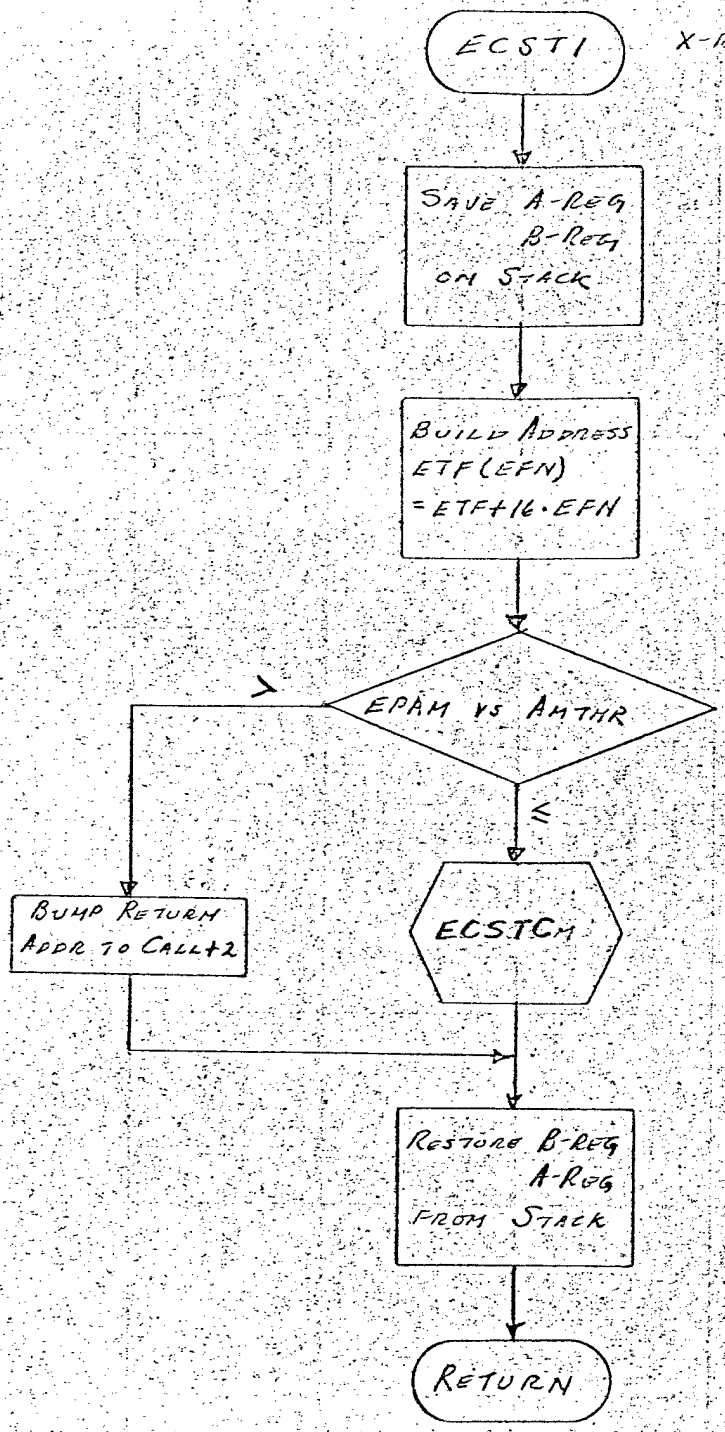

INSECT
INSECT – PAGE 1 OF 3
SAVE X-REG ON STACK
A-REG → ADDR(LIST 1)
ADDR(LIST 2)
ADDR(LIST 3)
B-REG = ADDRESS OF RESULT
E-REG = NOP 2-WAY
JMP $+2 3-WAY
SAVE INPUTS & MAKE ROOM ON STACK PER MAP →
S-REG →
NAME
DISPL REL TO S-REG
DW1 0
DW1 2
DW2 4
DW3 6
RSI 8
CNT 9
REW 10
ADR3 11
ADR2 12
ADR1 13
OUTAD 14
INSTR 15
SAVE X-REG 16
RETURN ADDR 17
POINTED TO BY A-REG INPUT
B-REG INPUT
E-REG INPUT
KW = KEYWORD
KW1 = ((ADR1))
KW2 = ((ADR2))
KW3 = ((ADR3))
KW1 = KW1 ∧ KW2 (2-WAY)
= KW1 ∧ KW2 ∧ KW3 (3-WAY)
BUILD DOUBLE WORD DW1 = (Φ, KW1); B ← KW1
BUILD DOUBLE WORD DW2 = (Φ, KW2); B ← B ∧ KW2
EXQ (INSTR)
2-WAY vs. 3-WAY
3-WAY
BUILD DOUBLE WORD DW3 = (Φ, KW3); B ← B ∧ KW3
2-WAY
INSΦ1
KW1 IS IN B-REG
KW1 = Φ ?
YES
INDUN1
3
A-REG = Φ
NO
BUILD DOUBLE WORD DW1 = (Φ, KW1)
INITIALIZE RSI ← Φ CNT ← 16
INLOOP
2
BEGIN MAJOR


1, 3
INLOOP
NSH ← 1
(A,E) ← DWI
NSH IS IN X-REG AND STAYS THERE THROUGHT ENTIRE LOOP
INSHFT
SHIFT (A,E) LEFT 1 BIT; NSH ← NSH+1
NO
E-REG < Φ
YES
INSΦ2
LSB (A-REG) = 1
SHIFT (A,E) LEFT 1 BIT; DWI ← (A,E)
(A,E) ← DWI SHIFT (A,E) LEFT NSH BITS DWI ← (A,E)
PARITY: A ← 1'S BIT COUNT OF A
REW ← (A + (ADR1))
(A,E) ← DW2 SHIFT (A,E) LEFT NSH BITS DW2 ← (A,E)
PARITY (SEE ABOVE)
REW ← REW A (A + (ADR2))
EXQ (INSTR)
2-WAY VS 3-WAY
3-WAY
(A,E) ← DW3 SHIFT (A,E) LEFT NSH BITS DW3 ← (A,E)
PARITY (SEE ABOVE)
REW ← REW A (A + (ADR3))
2-WAY
INSΦ3
REW = Φ ?
YES
RESET LSB OF HEAD (DWI)
INSΦ5
3
NO
INSΦ4
3

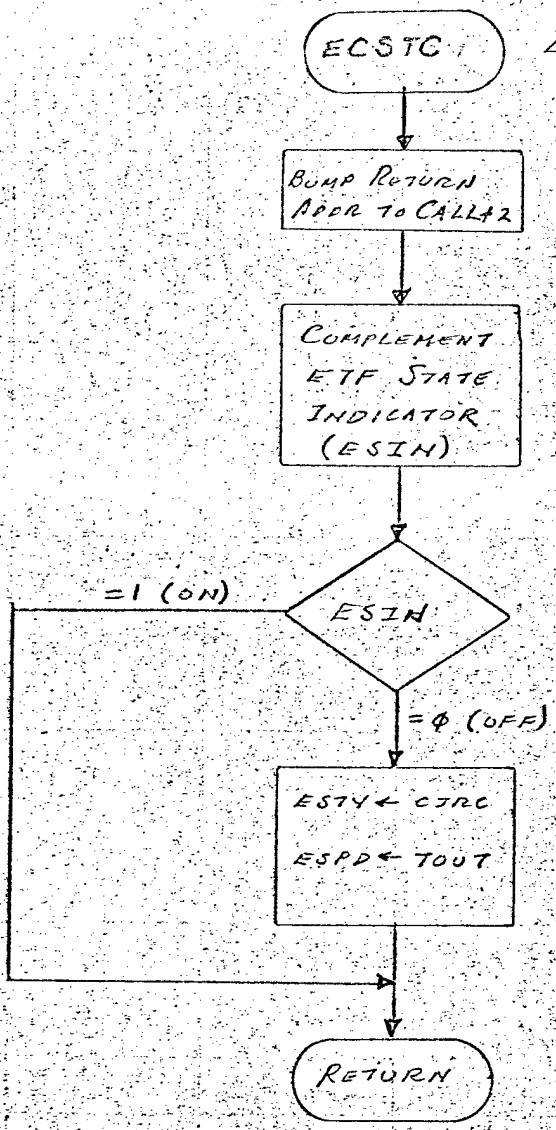

2
INSØ4
RSI ← RSI + 1;
STORE REW @
(OUTAD) + RSI
2
INSØ5
CNT ← CNT - NSH
INSIDE LOOP
OUT OF LOOP
CNT = Ø
Yes
INDON2
A-REG ← HEAD(DWI)
NO
INLOOP
2
NO
TAIL(DWI)
= Ø ?
YES: OUT OF LOOP
(A,E) ← DWI
LEFT SHIFT (A,E)
CNT BITS
1
INDON1
STORE A-REG
AS 1ST WORD
(KEYWORD) OF
OUTAD BLOCK
CLEAN UP STACK:
S ← S + 16;
RESTORE X-REG
FROM STACK
RETURN


A-REG ← BIT COUNT (A-REG)
DEFINE ARGUMENT = Q
PARITY
Q = Φ ?
YES
NO
SAVE B-REG
E-REG
ON STACK
CNT IS KEPT IN B-REG
CNT ← Φ
PLOOP
E ← A
Q ← Q .AND. (Q-1)
E ← E-1
A ← A ∧ E
CNT ← CNT+1
Q = Φ ?
NO
YES
DONE
A-REG ← CNT
RESTORE E-REG
B-REG
FROM STACK
RETURN

3.2.3 ECSTI


ECSTI
X-REG = EFN
SAVE A-REG
B-REG
ON STACK
BUILD ADDRESS
ETF(EFN)
= ETF+16·EFN
STAYS IN X-REG
THRU WHOLE SUBROUTINE
EPAM VS AMTHR
ETF PEAK AMPLITUDE
VS
AMPLITUDE THRESHHOLD
>
≤
BUMP RETURN
ADDR TO CALL+2
ECSTCM
LOGIC COMMON TO
SCAN TESTS 1 & 2
– ALWAYS RETURNS TO
CALL+2
RESTORE B-REG
A-REG
FROM STACK
RETURN

3.2.3.1 ECSTC


ECSTC
Logic Common to Scan Tests 1 & 2
Bump Return Addr to Call+2
Complement ETF State Indicator (ESIN)
ESIN
=1 (ON)
=0 (OFF)
ESTY ← CIRC
ESPD ← TOUT
Set ETF
Scan Type ← Circular Code
Scan Period ← Time-out Code
Return

3.3 COMPUTER SUBPROGRAM ENVIRONMENT

The Data Environment of this subprogram consists of:

(a) Global Tables - These are defined in the CDBDD. The specific tables used herein are -

ETF - - Emitter Track File

EL2 - - Emitter Library 2

Local parameters are defined for the displacement and Bit layout templates of these tables. These names are prescribed by the CDBDD. See Section 3.3.3.

(b) Global Parameters - Values of certain codes. These include:

Name	Subroutine	Function
EMNSRQ	ECDR	Executive Message Number of Scan-Analysis-Request
RMCEC1	ECDR	Return Module Code of Emitter Classification Processing - 1
AMTHR	ECST1	Amplitude Threshold
CIRC	SCTCOM	Circular Scan-Type Code
TOUT	SCTCOM	Time-Out Scan Period Code
NOFA1	ECLV1	"None-of-Above" Identity Code
UNCLS	ECLV1	"Unclassified" Identity Code
UNKNO	ECLV1	"Unknown" Display Code

(c) Local Permanent Storage - - These are tables with fixed addresses. They are described in Section 3.3.1.

(d) Local Ephemeral Storage - - These are local tables, variables, indices, etc., which are assigned space on the stack upon entry to the subroutines wherein they are needed and disappear upon return therefrom. In our estimation, this factor outweighs their mixed nature. Therefore, we discuss the stack storage for each subroutine as a unit in Section 3.3.2.

3.3.1 Local Permanent Tables

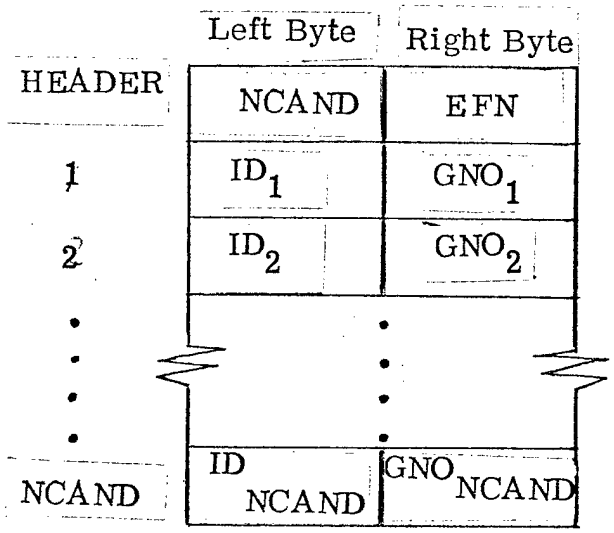
3.3.1.1 CANDLE

(a) This table is named "CANDLE". It is defined in and known only to subroutine ECLV1.

(b) CANDLE is used to store the final, Level 1 Candidate List. As such, its address is sent to subroutine TRNSL8 which loads CANDLE with this List. Its address is also returned by ECLV1 to the driver ECDR which stores this address in the scan-analysis request message sent to the Executive.

(c) CANDLE has 33 locations allotted to it. The first word is a header and the table proper consists of the following 32 words which are indexed by addition of an index NCAND = 1, 2, . . ., 32 to the header's address.

(d) The structure and bit layout are shown in the accompanying diagram.



NCAND ≤ 32: Number of Candidates
EFN: Emitter File Number (Input)
ID_i: Identity Code of i^{th} Candidate
GNO_i: Group Number of i^{th} Candidate

ID_i is taken from the $GNO_i{}^{th}$ entry of EL2.

If no Candidate List is generated, the header = EFN, i.e., NCAND = Ø

3.3.1.2 Emitter Library 1 (EL1)

EL1 consists of data structures whose purpose is to enable the generation of a Candidate List each of whose members (trunks) could match the input Emitter Track File (ETF) on the three parameters frequency (f), pulse repetition interval (π) and pulse width (p).

Accordingly, EL1 consists of three major divisions, one for each physical parameter. Each of these divisions consists of:

- an Outer Directory. Therefore, we have OD. F, OD. PI and OD. PW
- a Zone-List-Section. Therefore, we have ZLS. F, ZLS. PI and ZLS. PW.

We economize on the requisite description by noting that OD. F and OD. PI are, except for length and physical significance, of identical structure. Therefore, we describe them as OD. X where X stands for either F or PI. Moreover, all three ZLSs are of identical structure, so we describe them as ZLS. Y where Z stands for either F, PI or PW.

3.3.1.2.1 Outer Directories

3.3.1.2.1.1 OD. X: Outer Directories OD. F and OD. PI

(a) The name of this locally defined and used table is "OD. F" ("OD. PI"). It consists of two sections: a value section "ODV. F" ("ODV. PI"), and an address section "ODA. F" ("ODA. PI").

(b) ODV. F (ODV. PI) contains N. F (N. PI) values of frequency (PRI) in ascending order. The first value is much less than any expected value of frequency (PRI) and the last much greater. The table exists from the beginning of the program (not created dynamically), and is used as input to the binary search subroutine (BIRCH). The result of this search is an index L such that the input value of frequency, f (PRI, π) is

$$\text{ODV. F (L)} \leq f < \text{ODV. F (L + 1)}$$
$$[\text{ODV. PI (L)} \leq \pi < \text{ODV. PI (L + 1)}]$$

3.3.1.2.1.1

(b) -continued -

ODA. F (ODA. PI) contains N. F (N. PI) pointers. If L is as just described and ODA. F (L) < Ø (ODA. PI (L) <Ø) then there are no known emitters with the frequency f (PRI π); whereas, if ODA. F (L) ≥ Ø (ODA. PI (L) ≥Ø) then this pointer is in displacement D (L) such that the address

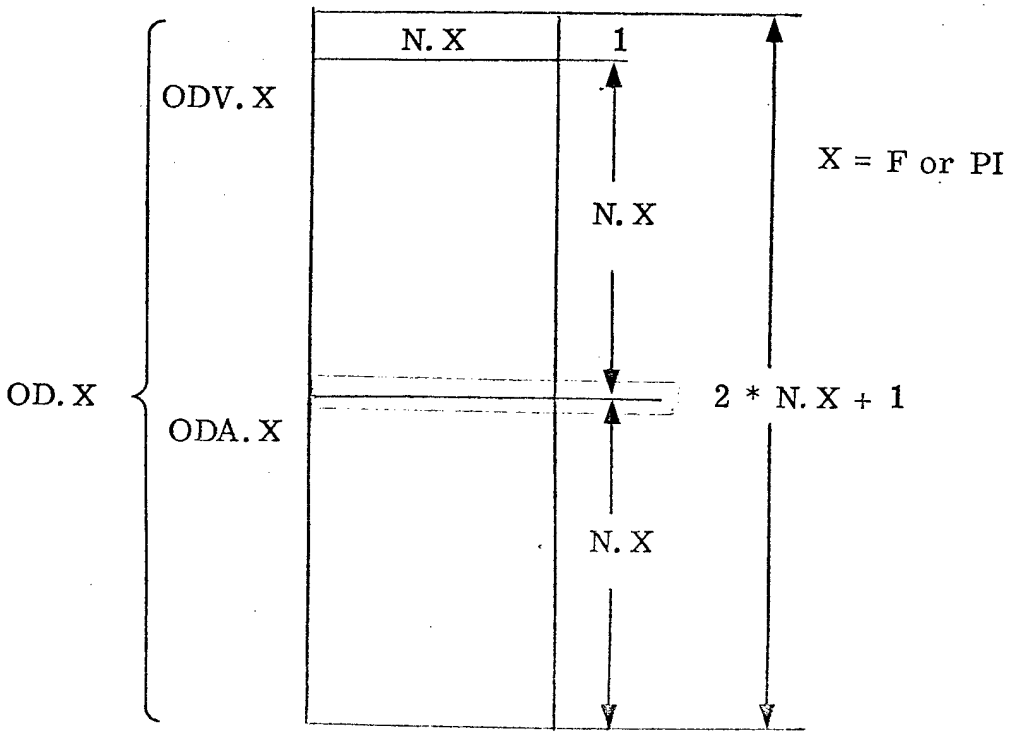
$$A\ (L) = ZLS.\ F + D\ (L)$$
$$[A\ (L) + ZLS.\ PI + D\ (L)]$$

points to a list of emitters in ZLS. F (ZLS. PI) which are exhibit the given value of f (π).

(c) The length of OD. F is 2 * N. F + 1.
(OD. PI) (2 * N. PI + 1)

Indexing is applied to the two subsections ODV. F and ODA. F (ODV. PI and ODA. PI) by adding the index of the desired entry to the addresses ODV. F and ODA. F (ODV. PI and ODA. PI) where the index runs from Ø to N. F - 1 (N. PI - 1).

(d) OD. F (OD. PI) is structured as depicted below. Bit layout is not applicable; each item is a whole word datum.



3.3.1.2.1.2 OD. PW

(a) The name of this locally defined and used table is "OD. PW". It consists of, and is co-extensive with, an address table "ODA. PW".

(b) ODA. PW is indexed directly by p = one of the 32 pulse width codes. Each such code already stands for an interval of physical pulse width. Therefore, selection of a ZLS. PW pointer can be accomplished directly, and does not require the mediation of an interval limits list (the non-existent ODV. PW) and a binary search.

As above, ODA. PW (p) < Ø (Ø ≤ 1 ≤ 31) implies no emitters with given code p while ODA. PW (p) ≥ Ø is = a displacement D (p) such that address

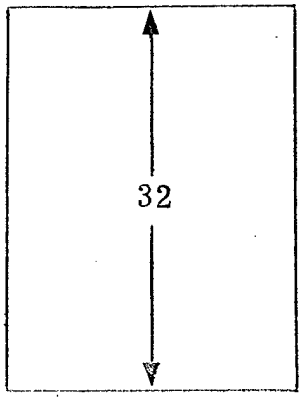
$$A(p) = ZLS.PW + D(p)$$

points to a list of emitters in ZLS. PW which all could exhibit the given value of p.

(c) The length of OD. PW is 32 words. It is indexed by addition of the code p to address OD. PW.

(d) OD. PW is structured as depicted below. Bit layout is not applicable; each item is a whole word datum.

OD. PW =
ODA. PW:



Index = p
Ø
1
.
.
.
3Ø
31

3.3.1.2.2 Zone List Sections

(a) The name of these tables are "ZLS.Y" where Y = F, PI or PW.

(b) Each such table contains more than 1 but no more than N.Y subsections (N.PW ≡ 32). Each subsection is an encloded list of the emitter trunks which can exhibit the value of the corresponding parameter as described in Section 3.3.1.2.1.1(b) and 3.3.1.2.1.2(b). Each such table exists from the beginning of the program (not created dynamically).

(c) The size of ZLS.Y depends on the number of non-negative entries in ODA.Y. This number is of no importance to the execution of the ECLV1 programs and so is not assigned a symbolic name. However, for the sake of this sizing discussion alone we define

$$\S.Y(I) = \begin{cases} \emptyset & \text{if ODA.Y(I)} < \emptyset \\ 1 & \text{if ODA.Y(I)} \geq \emptyset \end{cases}$$

$$L.Y(I) = \begin{cases} \text{if ODA.Y(I)} < \emptyset \\ \in [2, 17] & \text{if ODA.Y(I)} \geq \emptyset \end{cases}$$

Then the size of ZLS.Y $\leq \sum_{c=1}^{N.Y} \S.Y(I) * L.Y(I)$ (1)

ZLS.Y consists of $\leq \sum_{c=1}^{N.Y} \S.Y(I)$ (1) subsections

each of variable length (see d below) ranging from 2 to 17 words. Indexing is accomplished in two stages. First, locate the subsection initial address A(Y) as described in the above cited sections. Then use the bit-encoding as described below.

(d) Each subsection of ZLS.Y consists of a leading (key) word and one or more non-zero binary vector words. (2)

Each subsection contains an emitter trunk presence (1)/absence (∅) bit-vector where the number of trunks

(1) Inequality applies because identical subsections need be stored only once.

(2) This format is referred to keyword-compressed/binary-vector format or KWC/BV format elsewhere in this document and in the commentary of the program listings.

3.3.1.2.2

(d) -continued-

NTRUNX ≤ 256[(3)]. Accordingly, a subsection covering all 256 trunks would be 16 words long with MSB (word 1) corresponding to trunk #1 and LSB (word 16) corresponding to trunk #256. Since, in actuality, most bits and most words will be = Ø, space is saved by using the keyword as a map of the bit-vector words that are non-zero (Have ≥ 1 bits = 1) and omitting all pure zero words.

Thus, by way of example, consider a subsection which is to include trunks 1, 3, 18, 8Ø and 113. The bit vector before the keyword compression would be as shown in Figure 1a. The subsection after the keyword compression as actually stored is shown in Figure 1b.

(3) NTRUNX = N. F

Word #																Trunk #'s
1	1	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1, 3
2	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	18
3	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	
4	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	
5	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	8Ø
6	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	
7	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	
8	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	113
9	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	
10								.								
11								.								
12								.								
13								.								
14								.								
15								.								
16	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

Figure 1a. Bit Map of Trunks 1, 3, 18, 8Ø, 113 Before Keyword Compression

Word #																Keyword: Words
1	1	1	Ø	Ø	1	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1, 2, 5, 8 ≠ Ø
2 (1)	1	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	
3 (2)	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	
4 (5)	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1	
5 (8)	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	

Figure 1b. Bit Map of Trunks 1, 3, 18, 8Ø, 113 After Keyword Compression

3.3.1.3 Library-to-Library Linkage Tables

Library EL1 (Local to the subroutines of this document) is based on emitter trunks which are characterized by unique sextuplets of the physical measurements

Frequency - minimum and maximum
PRI - minimum and maximum
PW - minimum and maximum

arranged and numbered on ascending minimum frequency.

Library EL2 is defined in the CDBDD. It is based on emitter groups which are derived from classification according to scan to type, minimum and maximum scan period, EW technique and related factors.

The two tables of this section are used to associated groups with trunks. The first such table is an outer directory ODA. GT (GT = Group/Trunk), and the second, ZLS. GT, is entirely analogous to the ZLS tables described in Section 3.3.1.2.

3.3.1.3.1 ODA. GT

(a) The name of this table is ODA. GT.

(b) An element in ODA. GT at address ODA. GT. + I, $I \geq 1$ will be referred to as D (I). Each D (I) $\geq \emptyset$ is a displacement which when added to the address ZLS. GT yields the address of a KWC/BV trunk list in ZLS. GT whose members are associated with the I^{th} group.

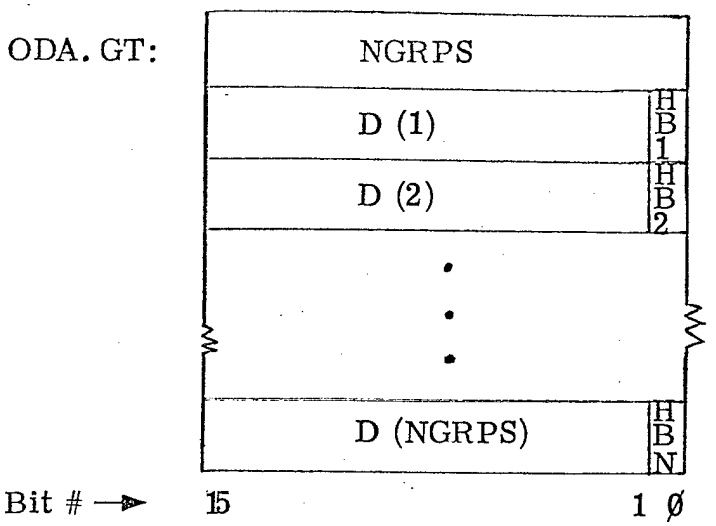
A value of D (I) $< \emptyset$ is taken to be = J - (I + 1) for some $J < I$. This indicates that the I^{th} group has the same KWC/BV trunk list in ZLS. GT as the J^{th} group.

Each element D(I) whether $\geq$ or $< \emptyset$ is stored as twice the nominal value discussed above, so that upon being fetched it must be arithmetically shifted right one bit position. This allows the LSB of each (D (J) to be used to hold a Hit-Bit (flag). See Section 3.1.2.2.

3.3.1.3.1 -continued-

(c) ODA. GT is NGRPS in length. NGRPS is stored at location ODA. GT itself (see below). Indexing is accomplished by adding the desired index I, $1 \leq I \leq$ NGRPS to address ODA. GT.

(d) ODA. GT structure and bit layout are indicated on the accompanying diagram.



ODA. GT: Structure & Bit Layout

3.3.1.3.2 ZLS. GT: - Group Lists of Trunks

(a) The name of this table is ZLS. GT. The name has been chosen because of its exact identity in use and structure to the ZLS tables of EL1 (Section 3.3.1.2). It has nothing to do with measurement zones of physical parameters.

The other revelant descriptions of ZLS. GT can be gotten from Section 3.3.1.2.

3.3.1.4 Scan Analysis Request Message

(a) The name of this table is SRQMSG

(b) This table contains the codes and parameters required to cause the Executive to schedule a scan analysis. It is defined in and known only to the driver ECDR which passes its address to the Executive when it calls EXMES.

(c) This table is five long; only three of its elements are filled by ECDR. These are accessed by individual labels assigned to the three target elements.

(d) The structure, bit layout and labels of SRQMSG are shown below.

Label		Contents
SRQMSG:	EMNSRQ	Executive Message # of Scan Analysis Request
	3	Number of Words to Follow
SRRMCD:	RMCEC2 | EFN	Left Byte: Return module code to EC2 — Right Byte: Emitter file # - Input to ECDR
SRCLAD:	CANDLE	Address of Candidate List
SRREQW:	AW | - dc - | 1	AW = {0 / 1} as analysis {is / is not} wanted; Bit Ø = 1 Scan analysis

15 Ø

Items filled in by ECDR are underlined.

3.3.2 Local Ephemeral Storage

Three of the subroutines of this subprogram use the stack for their local tables and variables during execution, releasing the storage upon exit. The other four subroutines and the driver (ECDR) use only small amounts of stack storage to save and restore registers. This latter use of the stack will not be described.

An oversimplified example of stack usage for ephemeral storage can be gained from the following example.

A subroutine preserves all registers and needs room for 3 variables named "AL", "EE", and "OOP" and a table 5 long named "THING".

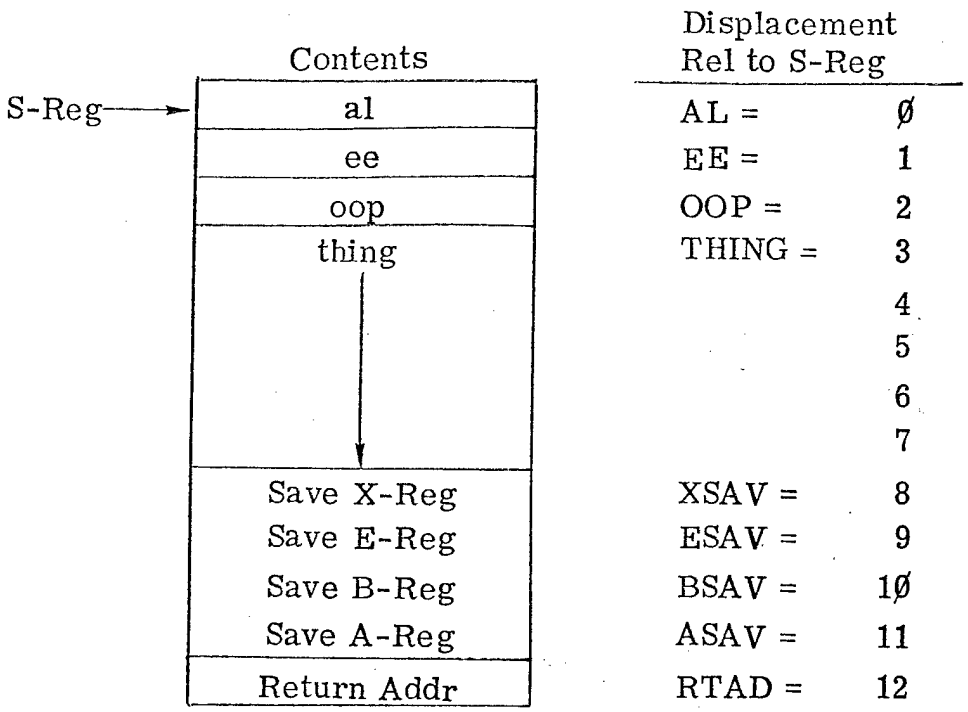
1) Define the stack map with parameters

AL =	Ø	Local Ephemeral Storage
EE =	1	
OOP =	2	
THING =	3	
XSAV =	8	Register Save Area
ESAV =	9	
BSAV =	1Ø	
ASAV =	11	
RTAD =	12	Return Address

2) On Entry do:

```
PSH     A
PSH     B
PSH     E
PSH     X
SUB, S  P, INC
        XSAV
```

3) The stack memory map is



4) In the body of the program, stack items, including the saved registers and the return address can be accessed symbolically and mnemonically. For example

- oop ← opp +1

Is coded ISZ OOP, S

- al ← al + ee

Is coded LDS AL, S
ADD EE, S
STS AL, S etc.

5) Upon leaving do:

```
ADD, S   P, INC
         X SAV
POP      X
POP      E
POP      B
POP      A
EXIT
```

3.3.2 -continued-

In the following paragraphs, we present the stack map for each of the three subroutines and describe the function of each entity in each map in top-of-map to bottom order. We shall use the same convention that was used in the above example, namely -

Upper case symbol (e.g., AL) for the displacement-relative to S-Reg of an item.

Lower case symbol (e.g., al) for the item itself (i.e., contents of S-Reg + U.C. Symbol)

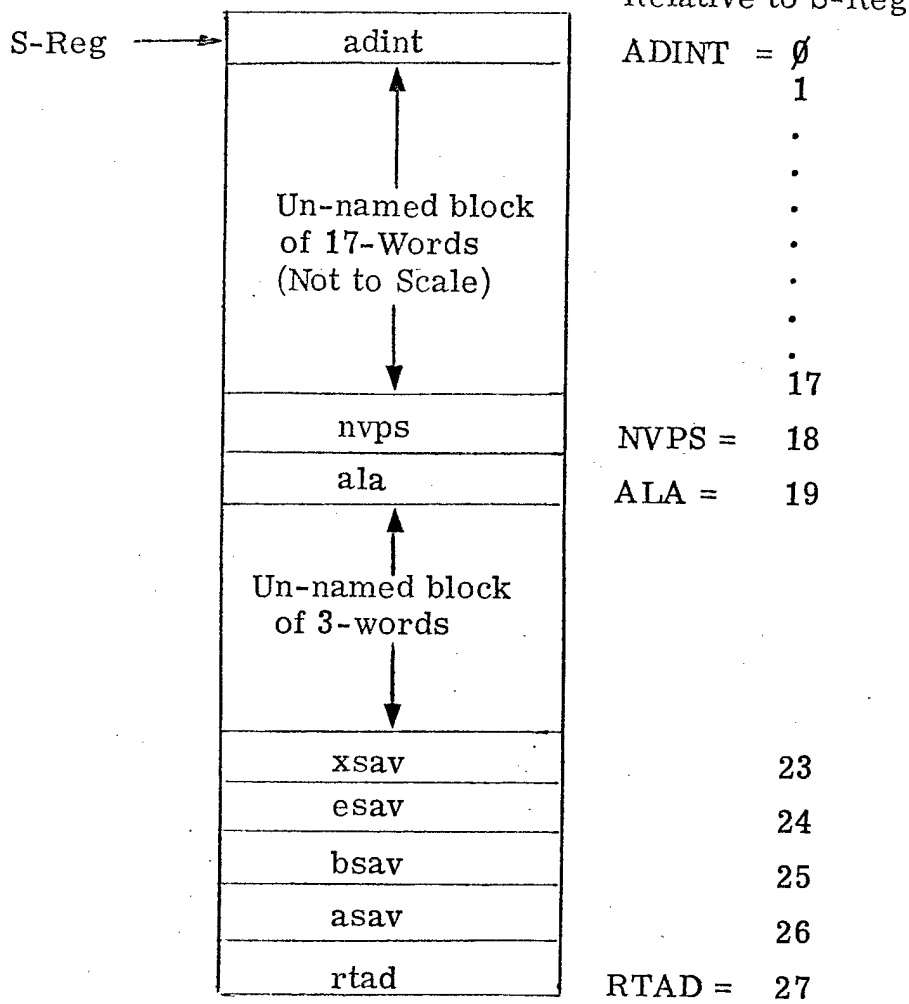
We shall use the special lower case symbols "asav", "bsav", etc to mark stack space for storing the corresponding registers and "rtad" for return address although U.C. symbols for these items are generally not defined.

3.3.2.1 Stack Storage For Subroutine ECLV1

The stack map is shown in an accompanying diagram. The table and item descriptions follow. (Note, there is a disparity between the map presented here with depth 28 and the maximum depth of 29 shown in flowchart 3.2.Ø. This is because ECLV1 uses an internal subroutine - - LOOK.1 or LOOK.2 - - which temporarily adds a 29th location to the stack).

adint = Address of the intersection. This is initialized at the start of ECLV1 to = the absolute address of the following location (1st word of block of 17 words), and does not change throughout the execution of ECLV1.

adint is loaded into the B-Reg before calling INSECT. This tells INSECT where to store the intersection of its input lists.



3.3.2.1 Stack Map for ECLV1

adint is also loaded into the A-Reg before calling TRNSL8. This tells TRNSL8 where to find the list it is to translate.

un-named block of 17-words — As implied by the description of adint, this holds the KWC/BV format intersection output of INSECT which is also input to TRNSL8.

nvps = Number of valid parameters = 1's bit count of the three validity bits

EFV
EPIV
EPWV

It is set = Ø at start of ECLV1 and augmented each time one of the validity bits is inspected and found non-zero.

ala = Address of list of addresses. This is initialized at start of ECLV1 to = the absolute address of the following location (1st word of block of 3 words), and does not change throughout the execution of ECLV1.

After a validity bit has been found ≠Ø and after the index into the ODA list has been found by binary search or direct lookup, but before nvps has been augmented, ala + nvps is used as the store address for the address.

ODA element + ZLS base address

ala is loaded into the A-Reg before calling INSECT It points to the 2 or 3 input list addresses used by INSECT.

un-named block of 3-words — Use is as described under ala.

xsav - Requires a special comment. This is not the input value (EFN) of X-Reg, but = address of table CANDLE. Restoring X from stack on leaving sets X as required.

3.3.2.2 Stack Storage for Subroutine TRNSL8

The stack map is shown in an accompanying diagram. The table and item descriptions follow.

ina = Intersection address. Exactly analogous to adint described above as regards contents and use in calling INSECT. It is not, of course, used in calling TRNSL8.

un-named block of 17-words } INSECT must have a place to store its result. This is it when INSECT is called from TRNSL8. However, the result itself is not wanted or used, only the indication of nullity/non-nullity of the intersection. (Candidate for future modification - - this is silly)

ncand - Number of candidates. Initialized to Ø. Each time a candidate group is recognized, it is incremented and then used as an index to create address at which to store candiate entry (resad + ncand).

When all candidates have been found (hit bottom of ODA. GT or run out of room), ncand is stored in the left byte of the result address (resad).

grpno - A running group number index initialized at Ø and augmented each time a new entry of ODA. GT is picked up. It is used as an entry index into EL2 by means of the address computation

EL2 (grpno) = EL2 + 11 * grpno

grpno is also stored in the right byte of the entries of CANDLEas discussed in Section 3.3.1.1.

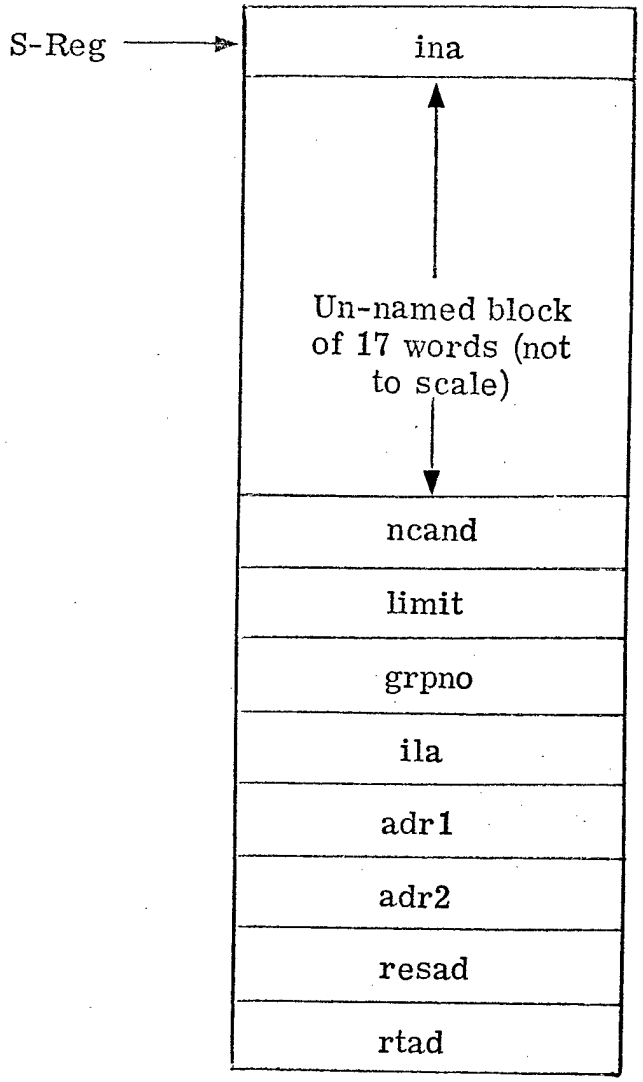
ila - Absolute address of following item (adr1). Exactly analogous to ala above.

adr1 - Place where the address of the KWC/BV format emitter trunk list (input in A-Reg) is stored.

adr2 - Place where address

ODA. GT(I) + ZLS. GT is stored.

resad - Result address. This was input in B-Reg.



Displacement Rel to S-Reg

INA =	Ø
	1
	.
	.
	.
	.
	.
	.
	17
NCAND =	18
LIMIT =	19
GRPNO =	2Ø
ILA =	21
ADR1 =	22
ADR2 =	23
RESAD =	24
	25

3.3.2.2 Stack Map for TRNSL8

3.3.2.3 Stack Storage for Subroutine INSECT

The stack map is shown in an accompanying diagram. The item descriptions follow.

dwi - Double Word Intersection = Initialized to (Ø, KWI) where

$$KWI = KW1 \wedge KW2 \quad \text{or}$$

$$KWI = KW2 \wedge KW2 \quad KW3$$

For 2 and 3-way intersections, respectively and kwi is the keyword of the i^{th} input list.

At the end of the routine, DWI = (KWI', Ø) where KWI' is the true keyword of the result.

dwi1 = Double Word 1 = (Ø, KW1) initially.

dw2 = Double Word 2 = (Ø KW2) initially.

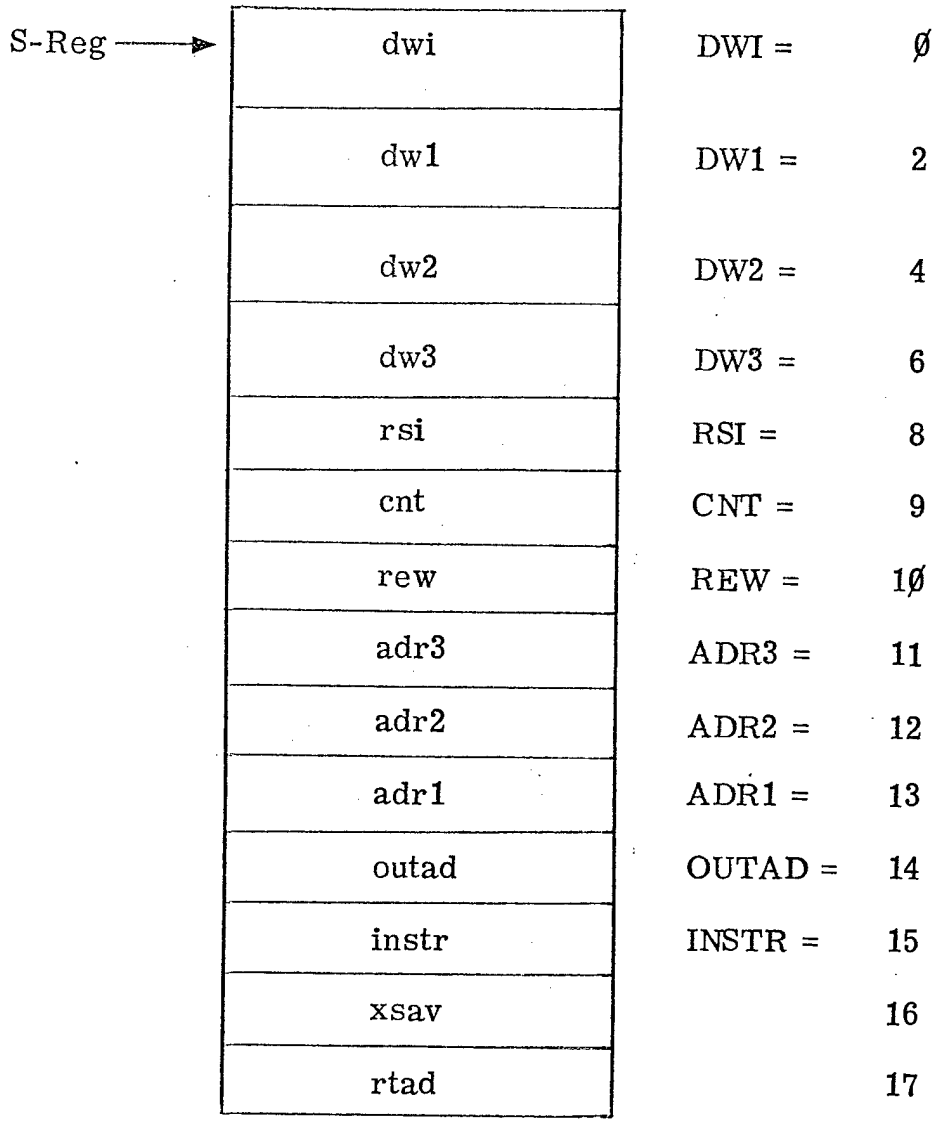
dw3 = Double Word 3 = (Ø, KW3) initially. If the call on Insect requires a 2-way intersection, the space for dw3 is still allotted but it is not used.

rsi = Result Storage Index. Initially = Ø, rsi is incremented each time a non-zero result word (see rew) is found. This result word is stored at address

$$outad + rsi, \quad rsi = 1, 2, \ldots, 16.$$

cnt = Count of bits not yet shifted left out of DWI - tail into DWI-head. Initially = 16, cnt is diminished at the end of each iteration by nsh (maintained in X-Reg) the number of bits DWI had to be shfted to get LSB [Head (DWI)] = 1. The condition cnt = Ø is a sufficient, but not necessary, condition for the end of the algorithm.

rew = Result Word. This location is used as a temporary repository of the partial 2 or 3-way intersection of words from the input lists.



3.3.2.3 Stack Map for INSECT

adr3
adr2
adr1

At entry, A-Reg → ADDR (List 1)
ADDR (List 2)
ADDR (List 3)

These 3 items are transferred to adr1, adr2 and adr3 (even if ADDR (List 3) is fictitious). The inverted order is due to the fact that stack building proceeds in bottom-to-top order.

outad = Output Address, i.e., where to store the intersection result. This was input at entry time in B-Reg. It is used as indicated under rsi for storing binary vector result words. At end of algorithm, KWI is stored at (outrad).

instr = Instruction to switch INSECT's logic to 2 or 3-way intersection as required. This is input at entry time in E-Reg.

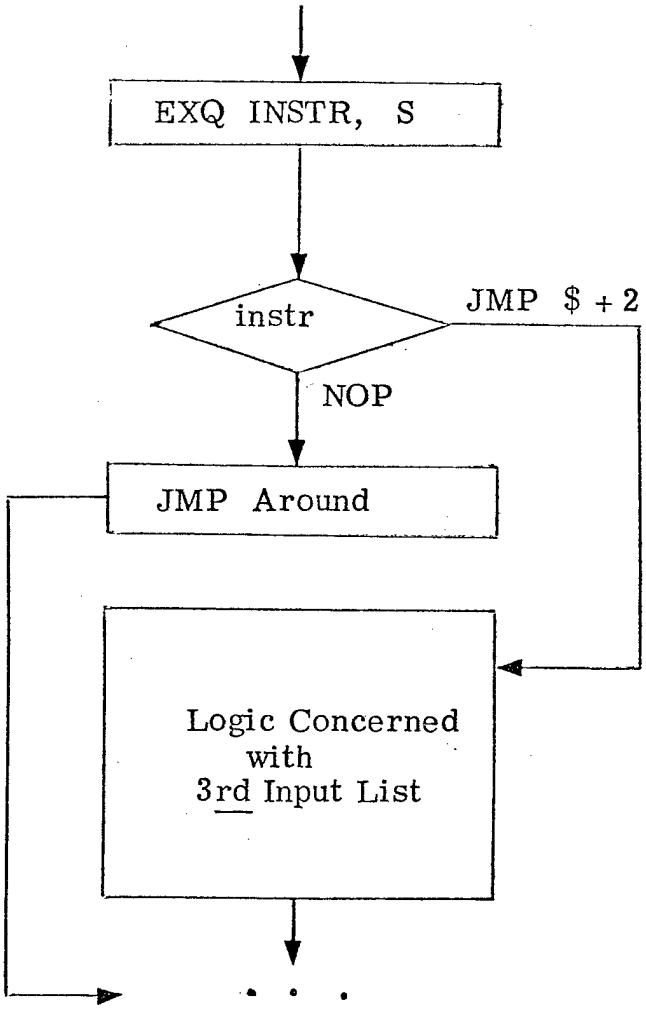
instr = NOP for a 2-way inserction
= JMP $ + 2 for a 3-way intersection

The mechanics are shown in the accompanying diagram. instr is made active by use of the RP-16 EXQ (Execute) instruction and is used at two points in the logic:

a) At point of fetching KW3, storing DW3 and ANDing KW3 into KWI.

b) At point of fetching binary vector word from third input list and ANDing it into the partial result word (rew).

Mechanics of instr Switch



3.3.3 Common Data Base References

A number of items in the common data base are referenced by the use of CDBDD - Prescribed but locally-defined (and therefore purgeable) symbols for word displacements and bit-layout parameters. We shall give here only the root symbols and prefix of items used, omitting the functional attribute suffixes. A table of Global Parameters used was given at the start of Section 3.3.

3.3.3.1 Reference to Emitter Library 2 (EL2)

The identify field (MID) is used in subroutine TRNSL8.

3.3.3.2 References to Emitter Track File (ETF)

References to the Emitter Track File are shown in the following table by subroutine.

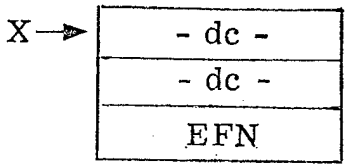
EMITTER CLASSIFICATION -1: ETF REFERENCES

Subroutine	Item Mnemonic	Item Description	Type
ECST1	EPMP	Peak Amplitude	Field
SCTCOM	ESIN	State Indicator	Flag
	ESTY	Scan Type	Field
	ESPD	Scan Period	Field
ECLV1	ELP	Long Pulse	Flag
	EFV	Frequency Validity	"
	EPIV	PRI Validity	"
	EPWV	PW Validity	"
	EFRQ	Frequency	Field
	EAPI	Average PRI	"
	EPW	Pulse Width	"
	EID	Identification	"
	EDIS	Display Code	"

3.4 INPUT/OUTPUT FORMATS

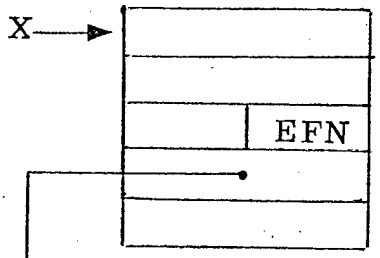
3.4.1 ECDR - - Driver

Inputs: On input X-Reg points to (contains the address of, abbreviated: X→) a three word block of which the = the number of the emitter track file to be classified. Symbolically:

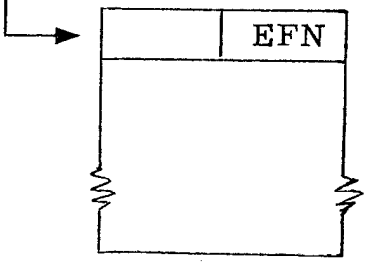


(dc = don't care)

Outputs: The output of ECDR is accomplished by a call on EXMES before ECDR returns to its caller. At the time of this call



Five-word block containing message See Section 3.3.1.4 for internal format.



The 4th word of the message points to a 1 to 17-word Candidate List. See 3.3.1.1 for internal format.

Other outputs include permanent changes made to ETF (EFN). These will be shown in the subroutines where they are made.

3.4.2 ECLV1 (Subroutine of ECDR)

Inputs: X = [EFN] $0 \leq EFN \leq 127$

AND the emitter track file ETF (EFN) at address = ETF + 16 * EFN to which this refers.

Outputs: Directly in ECLV1 Proper

EFN planted in 1st word (header) of Candidate List.

If no Candidate List (Failure):

- Failure codes in ETF ident field (EID)
- Display code in ETF display code field (EDIS)

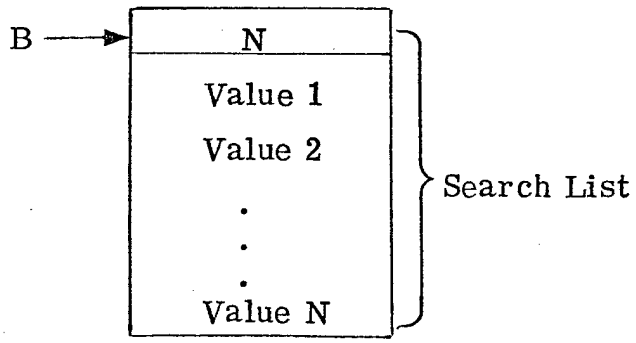
Fail/no fail indication by return to Call +1/Call +2 respectively. In either case, X → Candidate List.

Indirectly in Subroutines

Create balance of Candidate List.

3.4.2.1 BIRCH (Subroutine of ECLV1)

Inputs: A = Search Key

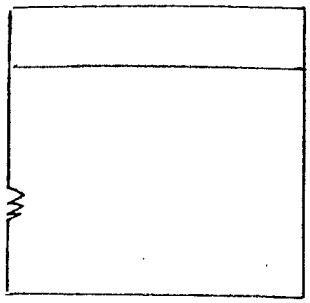


Outputs: B = Index L such that

$$Value_L \leq \text{Search Key} < Value_{L+1}$$

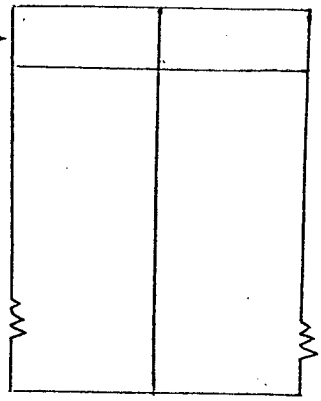
3.4.2.2 TRNSL8 (Subroutine of ECLV1)

Inputs: A →



Candidate List in KWC/BV format 2-17 words long. See Section 3.3.1.2.2 for format description.

B →



33-Words area for storing translated Candidate List. See Section 3.3.1.1 for internal format.

Outputs: Translated list stored as directed.

3.4.2.3 Insect (Subroutine of ECLV1 and TRNSL8)

Inputs: A →

ADDR (List 1)
ADDR (List 2)
ADDR (List 3)

List i is a 2 to 17-word list in KWC/BV format. See Section 3.3.1.2.2.

B →

17-word block for storing intersection result

E = | Switch Instruction |

= NOP — For 2-way intersection (ADDR (List 3) not used)

= JMP $ + 2 — For 3-way intersection

Outputs: Intersection Null

A-Reg = Ø and return to Call +1

Intersection Not Null

A-Reg ≠ Ø (= keyword of intersection)

Intersection stored as directed in KWC/BV format (Section 3.3.1.2.2).

3.4.2.3.1 Parity (Subroutine of INSECT)

Inputs: A-Reg = Argument

Outputs: A-Reg = # of 1's bits in input argument.

3.4.3 ECST1 (Subroutine of ECDR)

Inputs: X = | EFN |

Outputs: If peak amplitude > amplitude threshold
(E PMP) (AMTHR)

Return to Call +2, no changes to ETF.

Else call SCTCOM and return to Call +1.

3.4.3.1 SCTCOM (Subroutine of ECST1 and ECST2)

Inputs: X = Addr ETF (EFN) = ETF + 16 * EFN

Output: (X-unchanged)

Complement ETF State Indicator (ESIN)

If ESIN now = Ø return, else

Set ETF Scan Type (ESTY) = Circular (CIRC)
Set ETF Scan Period (ESPD) = Time-out (TOUT)
Return

3.5 REQUIRED EXTERNAL SUBROUTINES

Only one subroutine external to this nest is called. This is

EXMES

To return a message to the Executive, see Section 3.4.1.

3.6 CONDITIONS FOR INITIATION

This subprogram shall have unconditional entry and shall require no special initialization procedure.

3.7 SUBPROGRAM LIMITATIONS

3.7.1 Timing

No timing sensitivities or limitations.

3.7.2 Algorithm Limits

The present design is based on the KWC/BV format which is limited to the representation of trunk lists 256 in length. To go beyond 256 would require a second keyword (taking the limit up to 512) and this is not a trivial modification.

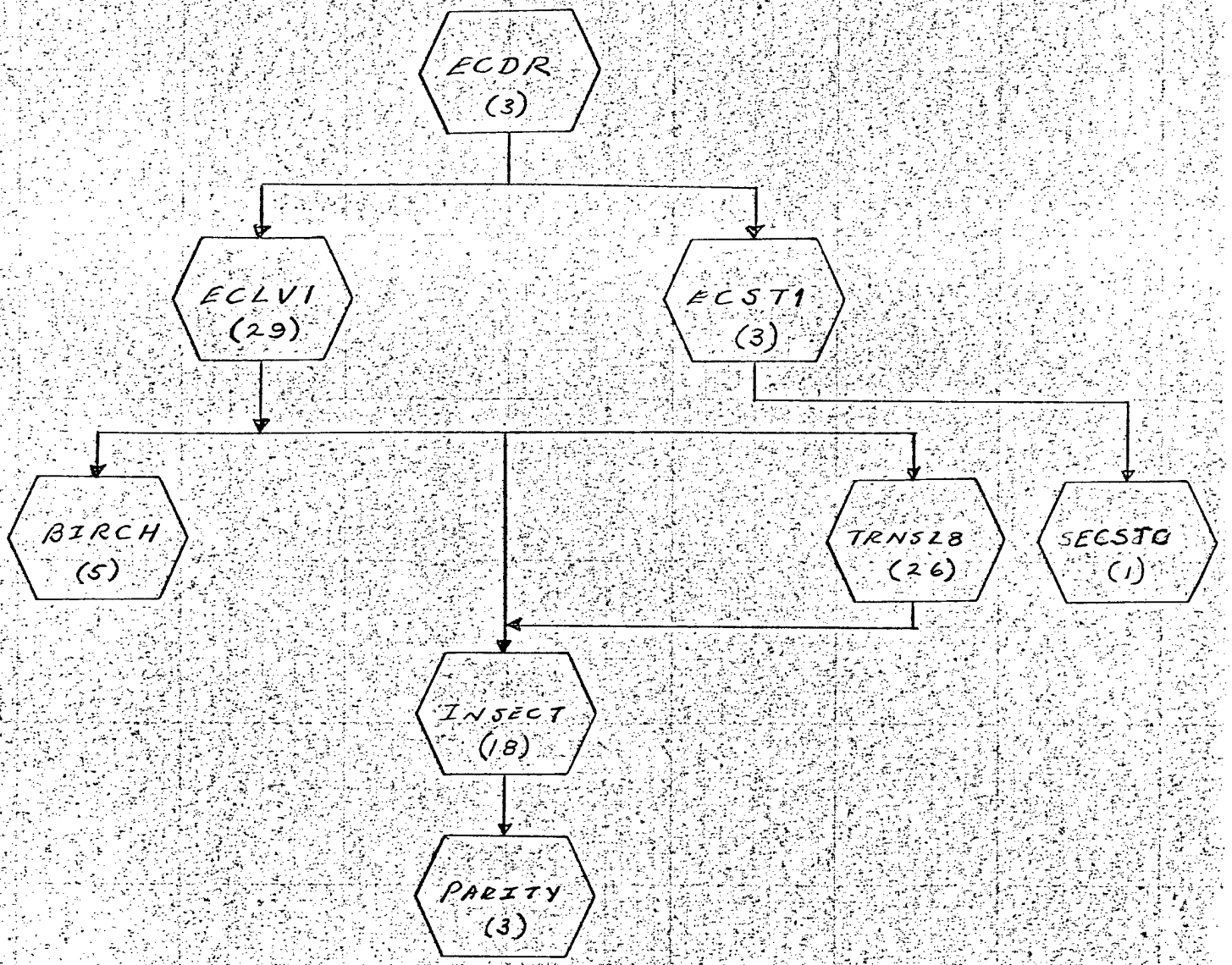
3.7.3 Error Testing/Reasonableness Checks

None.

3.8 INTERFACE DESCRIPTION

3.8.0 Emitter Classification Processing -1

Who calls whom



Numbers in parentheses = Maximum Stack Depth Increase caused by Module. This includes the entry return address, but not additions to stack by dependent subroutines.

Total Maximum = 79 (In Parity when Insect called by TRNSLB)

SJO 9/12/76

APPENDIX

Detailed Examples of Algorithms

A. Binary Search

B. Intersection

C. Parity (1's Bit Counting)

APPENDIX

Detailed Examples of Algorithms

This appendix presents instruction-by-instruction trace of three of the algorithms. The lines are keyed to subroutine internal lables or internal, unlabeled sections (; N). ALL numbers are hexidecimal (Base 16).

A. Binary Search (BIRCH)

Entry Conditions:

A-Reg = Ø Ø 2 7

B-Reg = 3 Ø Ø Ø : Ø Ø Ø 8

3	Ø	Ø	1	Ø	Ø	Ø	Ø
3	Ø	Ø	2	Ø	Ø	Ø	5
3	Ø	Ø	3	Ø	Ø	1	Ø
3	Ø	Ø	4	Ø	Ø	1	5
3	Ø	Ø	5	Ø	Ø	2	Ø
3	Ø	Ø	6	Ø	Ø	2	5
3	Ø	Ø	7	Ø	Ø	3	Ø
3	Ø	Ø	8	Ø	1	Ø	Ø

List to Follow: 8 long

S-Reg = 1 Ø 3 Ø before Call

Appendix A - Binary Search (1 of 2)

Instruction	A	B	E	X	S	Remarks
BIRCH: PSH A	Ø Ø 2 7	3 Ø Ø Ø	e e e e	x x x x	1 Ø 2 E	(1 Ø 2 E) = Ø Ø 2 7
PSH X	.	.	.	.	1 Ø 2 D	(1 Ø 2 D) = x x x x
PSH E	.	.	.	.	1 Ø 2 C	(1 Ø 2 C) = e e e e
(; 2) CLR X	.	.	.	Ø Ø Ø Ø	.	
COM X, X	.	.	.	F F F F	.	F F F F = -1
(; 3) LDN, E						
B, INC	.	3 Ø Ø 1	F F F 8	.	.	
LDC -1, B	F F F 7	.	.	.	.	
PSH B	.	.	.	.	1 Ø 2 B	(1 Ø 2 B) = 3 Ø Ø 1
LOOPG: SAD, X X	.	.	F F F C	.	.	JMP Comon
COMON: SUB, B E	.	3 Ø Ø 5	.	.	.	
MOV A, E	.	.	F F F 7	.	.	
SAD, X X	.	.	F F F B	.	.	
(; 7) LDS 3, S	Ø Ø 2 7	.	.	.	.	Ø Ø 2 7 vs. Ø Ø 2 Ø
XOR Ø, B	Ø Ø Ø 7	.	.	.	.	JEZ, JNG Not Taken
LDS 3, S	Ø Ø 2 7	.	.	.	.	
CSL Ø, B	.	.	.	.	.	No Skip
GRTR: MOV E, A	F F F B	.	.	.	.	
ISZ, E E	.	.	F F F C	.	.	No Skip
LOOPG: SAD, X X	.	.	F F F E	.	.	JMP Comon
COMON: SUB, B E	.	3 Ø Ø 7	.	.	.	
MOV A, E	.	.	F F F B	.	.	
SAD, X X	.	.	F F F D	.	.	
(; 7) LDS 3, S	Ø Ø 2 7	.	.	.	.	Ø Ø 2 7 vs. Ø Ø 3 Ø
XOR Ø, B	Ø Ø 1 7	.	.	.	.	JEZ JNG Not taken
LDS 3, S	Ø Ø 2 7	.	.	.	.	
CSL Ø, B	.	.	.	.	.	Skip taken
LESS: MOV E, A	F F F D	.	.	.	.	
IS2, E E	.	.	F F F F	.	.	No Skip

RAYTHEON COMPANY
LEXINGTON, MASS. 02173
CODE IDENT NO. 49956
SPEC NO. 53959-GT-0760
SHEET 82 OF
REV

10-1349 CONT (11-68) VELLUM PRINTED IN U.S.A.
10-1349 CONT (11-68) FILM

Appendix A - Binary Search (continued) (2 of 2)

Instruction	A	B	E	X	S	Remarks
LOOPL: SAD, X X	.	.	F F F F	.	.	
NEG E, E	.	.	Ø Ø Ø	.	.	
COMON: SUB, B E	.	3 Ø Ø Ø	.	.	.	
MOV A, E	.	.	F F F D	.	.	
SAD, X X	.	.	F F F E	.	.	
(; 7) LDS 3, S	Ø Ø 2 7	.	.	.	.	Ø Ø 2 7 vs. Ø Ø 2 5
XOR Ø, B	Ø Ø Ø 2	.	.	.	.	JEZ, JNG Not taken
LDS 3, S	Ø Ø 2 7	.	.	.	✓	
CSL Ø, B	.	.	.	.	.	No Skip
GRTR: MOV E, A	F F F E	.	.	.	.	
ISZ, E E	.	.	F F F F	.	.	No Skip
LOOPG: SAD, X X	.	.	F F F F	.	.	JMP Comon
COMON: SUB, B E	.	3 Ø Ø 7	.	.	.	
MOV A, E	.	.	F F F E	.	.	
SAD, X X	.	.	F F F F	.	.	
(; 7) LDS 3, S	Ø Ø 2 7	.	.	.	.	Ø Ø 2 7 vs. Ø Ø 3 Ø
XOR Ø, B	Ø Ø 1 7	.	.	.	.	JEZ, JNG Not taken
LDS 3, S	Ø Ø 2 7	.	.	.	.	
CSL Ø, B	.	.	.	.	.	Skip taken
LESS: MOV E, A	F F F F	.	.	.	.	
ISZ, E E	.	.	Ø Ø Ø Ø	.	.	Skip taken
DSZ, B B	.	3 Ø Ø 6	.	.	.	
DONE: SUB, B						
S, INC	F F F F	Ø Ø Ø 5	Ø Ø Ø Ø	F F F F	1 Ø 2 C	Result in B-Reg
POP E	.	.	e e e e	.	1 Ø 2 D	
POP X	.	.	.	x x x x	1 Ø 2 E	
POP A	Ø Ø 2 7	.	.	.	1 0 2 F	
EXIT	Ø Ø 2 7	Ø Ø Ø 5	e e e e	x x x x	1 Ø 3 Ø	

RAYTHEON

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO. 49956

SPEC NO. 53959-GT-0760

SHEET 83 OF

REV

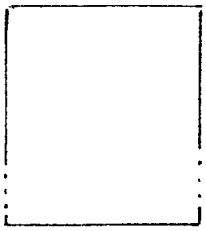
B. Intersection Routine (INSECT)

Entry Conditions:

A = 2 5 A B:

3 A 4 3
4 1 B E

B = 3 1 2 2:



17 Empty Locations

E = Ø Ø Ø Ø (NOP)

3 A 4 3:

1	0	0	0	①	0	0	1	①	0	1	0	0	0	0	KW1
1	1	0	0	1	1	0	0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	1	0	1	0	0	0	0	0	1	0	1	0	1	0

KWC/BV Lists

4 1 B E:

0	0	0	1	①	0	0	0	①	0	0	0	0	0	0	0	KW2
1	1	0	0	0	0	0	0	0	1	1	0	0	1	0	1
1	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

S = 1 1 Ø 4

These arrows show the pairs of BV words that must be ANDed. They correspond to the circled 1's in the keywords.

Appendix B — Intersection Algorithm -1 — (1 of 4)

Instruction	A	B	E	X	S	Remarks
INSECT: PSH X	2 5 A B	3 1 2 2	Ø Ø Ø Ø	x x x x	1 1 Ø 2	(1 1 Ø 2) = x x x x
PSH E	.	.	.	.	1 1 Ø 1	(1 1 Ø 1) = Ø Ø Ø Ø)NOP
PSH B	.	.	.	.	1 1 Ø Ø	(1 1 Ø Ø) = 3 1 2 2
(; 1) LDS, B A, INC	2 5 A C	3 A 4 3	.	.	.	B ← 1st List Addr
PSH B	.	.	.	.	1 Ø F F	(1 Ø F F) = 3 A 4 3
LDS, B A, INC	2 5 A D	4 1 B E	.	.	.	B ← 2nd List Addr
PSH B	.	.	.	.	1 Ø F E	(1 Ø F E) = 4 1 B E
LDS, B A, INC	2 5 A E	J U N K	.	.	.	B ← 3rd List Addr=Junk
PSH B	.	.	.	.	1 Ø F D	(1 Ø F D) = Junk
MOV S, X	.	.	.	1 Ø F D	.	
SUB, S=X'B'	.	.	.	.	1 Ø F 2	Make Room Temp Storage
(;2) CLR A	Ø Ø Ø Ø	.	.	.	.	Head of Double Words
COM A, B	.	F F F F	.	.	.	Proto KWI
(;3) EXQ X'F', S	.	.	.	.	.	(EA)=(1 1 Ø L) = NOP
JMP INSØ1	.	.	.	.	.	Avoid 3rd List
INSØ1: ISZ, X X	.	.	.	1 Ø F E	.	
LDS, E X, IND	.	.	1 8 8 Ø	1 Ø F F	.	((1 Ø F E))=(4 1 B E)= KW2
STD 4, S	.	.	.	.	.	(1 Ø F 6, F) =(Ø, 1 8 8 Ø)
AND, B E	.	1 8 8 Ø	.	.		
LDS, E X, IND	.	.	8 9 A Ø	1 1 Ø Ø	.	((1 Ø F F))=(3 A 4 3)= KW1
STD 2, S	.	.	.	.	.	(1 Ø F 4, 5)=(Ø, 8 9 A Ø)
AND, B E	.	Ø 8 8 Ø	.	.	.	JE2, B Not taken
(;6) +1 MOV B, E	.	.	Ø 8 8 Ø	.	.	
STD Ø, S	.	.	.	.	.	(1 Ø F 2, 3)=(Ø, Ø 8 8 Ø)
STS 8, S	.	.	.	.	.	(1 Ø F A) = Ø Ø Ø Ø
LDS, A = 16	Ø Ø 1 Ø	.	.	.	.	
STS 9, S	.	.	.	.	.	(1 Ø F B) = Ø Ø 1 Ø
INLOOP: LDS, X=1	.	.	.	Ø Ø Ø 1	.	
LDD Ø, S	Ø Ø Ø Ø	.	Ø 8 8 Ø	.	.	
INSHFT: JNG, E (= INSØ2)	.	.	.	.	.	
SAD =1	Ø Ø Ø Ø	.	1 1 Ø Ø	.	.	Not Taken

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

SPEC. NO. 53959-GT-0760
SHEET 85 OF
REV

Instruction	A	B	E	X	S	Remarks
ISZ X X	.	.	.	Ø Ø Ø 2	.	
INSHFT: JNG, E (=INSØ2)	.	.	.	.	.	Not Taken
SAD =1	Ø Ø Ø Ø	.	2 2 Ø Ø	.	.	
ISZ, X X	.	.	.	Ø Ø Ø 3	.	
INSHFT: JNG etc	.	.	.	.	.	Not Taken
SAD =1	Ø Ø Ø Ø	.	4 4 Ø Ø	.	.	
ISZ, X X	.	.	.	Ø Ø Ø 4	.	
INSHFT: JNG etc	.	.	.	.	.	Not Taken
SAD = 1	Ø Ø Ø Ø	.	8 8 Ø Ø	.	.	
ISZ, X X	.	.	.	Ø Ø Ø 5	.	
INSHFT: JNG, E (= INSHFT)	.	.	.	.	.	Taken
INSØ2: SAD =1	Ø Ø Ø 1	.	1 Ø Ø Ø	.	.	
STD Ø, S	.	.	.	.	.	(1 Ø 2, 3)=(1, 1 Ø Ø Ø)
(; 11) LDD 2, S	Ø Ø Ø Ø	.	8 9 A Ø	.	.	
SAD, A X	Ø Ø 1 1	.	3 4 Ø Ø	.	.	
STD 2, S	.	.	.	.	.	(1 ØF 4, 5)=(11, 3 4 Ø Ø)
JSB PARITY	Ø Ø Ø 2	.	.	.	.	1's Bit Count of A-Reg
ADD X'D', S	3 A 4 5	.	.	.	.	
LDS, A A, INC	8 8 4 4	.	.	.	.	BV word for List 1
STS X'A', S	.	.	.	.	.	(1 Ø F C)= 8 8 4 4
(;12) LDD 4, S	Ø Ø Ø Ø	Ø 8 8 Ø	1 8 8 Ø	Ø Ø Ø 5	1 Ø F 2	
SAD, A X	Ø Ø Ø 3	.	1 Ø Ø Ø	.	.	
STD 4, S	.	.	.	.	.	(1 Ø F 6, 7)=(3, 1 Ø Ø Ø)
JSB PARITY	Ø Ø Ø 2	.	.	.	.	1's Bit Count of A-Reg
ADD X'C', S	4 1 C Ø	.	.	.	.	
LDS, A A, INC	8 8 Ø 4	.	.	.	.	
AND X'A', S	8 8 Ø 4	.	.	.	.	8 8 Ø ∧ 8 8 4 4
(;13) EXQ X'F', S	.	.	.	.	.	A NOP
JMP INSØ3	.	.	.	.	.	Avoid 3rd List
INSØ3: JNZ INSØ4	.	.	.	.	.	Taken

Instruction	A	B	E	X	S	Remarks
INSØ4: MOV A, E			8 8 Ø 4			A=Non-Zero Result Word
ISZ 8, S						(1 Ø F A)=Ø Ø Ø 1
LDS 8, S	Ø Ø Ø 1					
ADD X'E', S	3 1 2 3					Result Destination Addr
STS, E A, INC	3 1 2 4					(3 1 2 3)= 8 8 Ø 4 *
INSØ5: LDS 9, S	Ø Ø 1 Ø					Tests for Done
SUB, A X	Ø Ø Ø B					
JE2 INDUN2						Not Taken
STS 9, S						(1 Ø F B) = Ø Ø Ø B
LDD Ø, S	Ø Ø Ø 1		1 Ø Ø Ø			
JNZ, E (=INLOOP)						Taken
INLOOP: LDS, X =1				Ø Ø Ø 1		Omitting Shift Loop
INSØ2: SAD =1	Ø Ø 1 1		Ø Ø Ø Ø	Ø Ø Ø 4		Detail
STD Ø, S						(1 Ø F 2, 3) = (11, Ø)
(;11) LDD 2, S	Ø Ø 1 1		3 4 Ø Ø			
SAD, A X	Ø 1 1 3		4 Ø Ø Ø			
STD 2, S						(1 Ø F 4, 5)=(Ø 1 1 4, 4 Ø Ø Ø)
JSB PARITY	Ø Ø Ø 4					
ADD X'D', S	3 A 4 F					
LDS, A A, INC	Ø Ø Ø 1					
STS X'A', S						
(;12) LDD 4, 5	Ø Ø Ø 3		1 Ø Ø Ø			(1 Ø F C)= Ø Ø Ø 1
SAD, A X	Ø Ø 3 1		Ø Ø Ø Ø			
STD 4, 5						
JSB PARITY	Ø Ø Ø 3					(1 Ø F 6, 7) = (31, Ø)
ADD X'C', S	4 1 C 1					
LDS, A A, INC	C Ø Ø Ø					
AND X'A', S	Ø Ø Ø Ø					
(;13) EXQ X'F', S						Ø Ø Ø 1^C Ø Ø Ø
JMP INSØ3						A NOP
INSØ3: JNZ INSØ4						Avoid 3rd List
						Not Taken

Appendix B — Intersection Algorithm -1 (continued) — (4 of 4)

Instruction	A	B	E	X	S	Remarks
(;16) LDS Ø, S	Ø Ø 1 1	.	.	.	.	Head of DWI
XOR = 1	Ø Ø 1 Ø	.	.	.	.	
STS Ø, S	.	.	.	.	.	(1 Ø F 2, 3)=(1 Ø, Ø)
INSØ5: LDS 9, S	Ø Ø Ø 8	.	.	.	.	
SUB, A X	Ø Ø Ø 7	.	.	.	.	
JE2 INDUN2	.	.	.	.	.	Not Taken
STS 9, S	.	.	.	.	.	(1 Ø F B)= Ø Ø Ø F
LDD Ø, S	Ø Ø 1 Ø	.	Ø Ø Ø Ø	.	.	
JNZ, E (=INLOOP)	.	.	.	.	.	Not Taken
(;19) SAD 9, S	Ø 8 Ø Ø	Ø 8 8 Ø	Ø Ø Ø Ø	Ø Ø Ø 4	1 Ø F 2	Complete Shifts
INDUN1: STS X'E', SI	.	.	.	.	.	(EA)=(3 1 2 2)= Ø 8 Ø Ø *
ADD, S = X'1 Ø'	.	.	.	.	1 1 Ø 2	
POP X	.	.	.	x x x x	1 1 Ø 3	
EXIT	.	.	.	.	1 1 Ø 4	

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO. 49956

SPEC NO. 53959-GT-0760

SHEET 88 OF

REV

Appendix C

Parity - - Counting One's Bits

Entry Conditions: A-Reg = 8 Ø Ø 1
S-Reg = 2 Ø 1 4

Instruction	A	B	E	X	S	Remarks
PARITY:						
JEZ DONE	8 Ø Ø 1	b b b b	e e e e	.	2 Ø 1 3	Not Taken
PSH B	.	.	.	.	2 Ø 1 2	(2 Ø 1 2)=b b b b
PSH E	.	.	.	.	2 Ø 1 1	(2 0 1 1)=e e e e
CLR B	.	Ø Ø Ø Ø	.	.	.	
PLOOP:						
MOV A, E	.	.	8 Ø Ø 1	.	.	
DSZ, E E	.	.	8 Ø Ø Ø	.	.	No Skip
NOP	.	.	.	.	.	
AND, A E	8 Ø Ø Ø	.	.	.	.	
ISZ, B B	.	Ø Ø Ø 1	.	.	.	Taken
JNZ PLOOP	.	.	.	.	.	
PLOOP:						
MOV A, E	.	.	8 Ø Ø Ø	.	.	
DSZ, E E	.	.	7 F F F	.	.	No Skip
NOP	.	.	.	.	.	
AND, A E	Ø Ø Ø Ø	.	.	.	.	
ISZ, B B	.	Ø Ø Ø 2	.	.	.	
JNZ PLOOP	.	.	.	.	.	Not Taken
(;3) MOV B, A	Ø Ø Ø 2	.	.	.	.	
POP E	.	.	e e e e	.	2 Ø 1 2	
POP B	.	b b b b	.	.	2 Ø 1 3	
DONE: EXIT	Ø Ø Ø 2	b b b b	e e e e	.	2 Ø 1 4	

RAYTHEON COMPANY
LEXINGTON, MASS. 02173

CODE IDENT NO. 49956

SPEC NO. 53959-GT-0760

SHEET 89 OF 89

REV

PRINTED IN U.S.A.