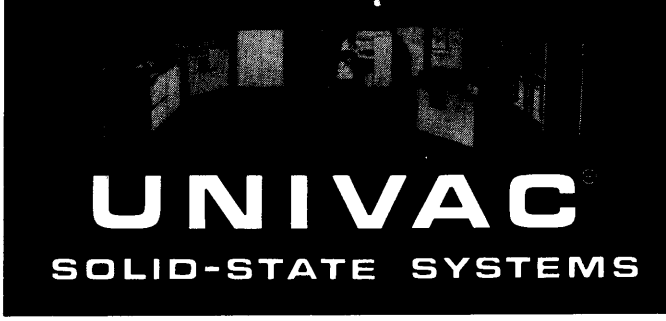


UNIVAC®
SOLID-STATE SYSTEMS

FORTRAN II

GENERAL
TECHNICAL
REFERENCE

ROUTINE BLOCK CHART
(ANNOTATED)

UP-3843.1

PRINTED IN U.S.A.

PREFACE

This release serves as a preliminary user document and supplement to the forthcoming FORTRAN II reference manual for UNIVAC Solid-State Systems (UP 3843). It contains a brief description of the FORTRAN II compiler, and a machine-generated annotated process chart of the compiler.

The chart, beginning on page 6, was produced as a by-product of a special-purpose compiler used in developing the FORTRAN II compiler, and is reproduced directly from a copy printed by the USS Printer. Standard charting techniques are generally followed, with the following alterations in symbology to accommodate these techniques to the characters available on the Printer:

The Operation Box (rectangle) is formed by lines of hyphens above and below, colons at left and right, and periods at corners.

The Decision Box (oblong) is formed by lines of hyphens above and below, and sets of parentheses arranged as (at left and at right.

```
(          )
 (        )
```

Connecting lines are indicated by rows of periods (horizontal), colons (verticle), and O's (at corners and as connectors).

Direction of flow is indicated by parentheses representing arrows. An arrow pointing to the right is indicated by), and arrow pointing left is indicated by (.

Entrances are indicated by (---IN---); exits, by EXIT; and remote connectors, by symbolic entries referring to subheadings in the accompanying annotation.

The reader should note that "missing" page numbers have been omitted in order to keep double pages facing each other.

Blank pages have been inserted where necessary to keep the first and second pages of double-page routines facing each other.

1. FORTRAN II Compiler Pass 1

The translator is divided into two major co-routines, 'SCAN' and 'GEN'.

SCAN has the duty of reading cards, condensing identifiers and constants into single entities and to feed items, in a convenient internal code, one at a time to GEN.

GEN has the duty of producing object code from these items. Control is passed between GEN and SCAN in a fashion such that each routine looks like a subroutine of the other.

The program begins by printing the title line, feeding a card, and going to the initialization routine, STEP Z1.

TABLE OF CONTENTS

A. Array Subscripting 34
B. Binary and Arithmetic Operators 32
C. Constant Scanner 14
D. Do Loop Control 40
E. Equivalence Processing 50
F. Function Calls 42
G. Generator Control 6
I. Assembler Structure 17
L. Linked Memory Subroutines 12
N. 'Get Next Character' Routine 10
P. Function and Subroutine Declarations 48
Q. Special Scanning Routines 16
S. Scanner Control 8
T. Symbol Table Search 11
U. Unary Operators and Special Generators 38
W. Input/Output (Read Punch Print) 46
X. Processing Format String 44
Z. Initialization and Termination 49

TABLE OF FORMATS

Information inside the compiler is treated in two principal formats, one for the symbol table entries in the scanner, and another for generator co-routine.

Symbol table equivalents are in the format

KM AAAA LLLL

where LLLL is a link to the next symbol, for searching

K equals 0: Simple Variable
M is 0: No memory assignment as yet AAAA is 0000
M is 1: Assigned AAAA in unique storage

M is 2: Equivalenced, not yet assigned. AAAA is link to other members of the equivalence class.

M is 3: Assigned AAAA in common.

M is 4: A formal parameter, whose subroutines are assigned AAAA, AAAA+1, and AAAA+2 in unique.

M is 5 The symbol is a 10 digit constant. If AAAA is 0, this constant has not been needed in object program yet, else it is assigned to location AAAA in unique.

K equals 3: Array

AAAA links to the dimension table entry, M is ignored. The dimension table has N+1 entries if there are N subscripts to this array.

```
AAAA+0:   3 M BBBB RRRR
AAAA+1:   0 0 TTTT SSSS
AAAA+2:   0 0 CCCC 0000
AAAA+3:   0 0 CCCC 0000  ETC
```

SSSS is link back to symbol table entry.

CCCC words, if present, are links to symbol table entries for constants (except for the last dimension).

TTTT is the total length of the array

M is 0: No memory assignment has been made as yet, BBBB is 0.

M is 1: The address BBBB is for A(1), i.e. the first cell of the array, in unique storage.

M is 2: Equivalenced array A(RRRR), BBBB is link to other elements in equivalence class.

M is 3: Same as M equal to 1 except common storage.

M is 4: Formal parameter, base address is stored in BBBB of unique storage.

K equals 5: Label

AAAA is the assignment in program storage.

M is 0: Unassigned as yet.

M is 1: Temporary assignment for Do Loops. AAAA links to an item in Llist,

```
AAAA+0: 02 TTTT XXXX
AAAA+1: SS SSSS LLLL
```

where XXXX is Llist link,
TTTT is temporary assignment of the label,
SS SSSS is like a permanent symbol table entry for labels, and LLLL is a link back to the symbol table entry.

M is 2: AAAA is the assignment for the label.

K equals 6: Function

M is 2: Assigned AAAA in program storage.

M is 5: Assigned AAAA, external reference.

M is 9: Special operator for scanner only.

K equals 7, 8, or 9 Operator, reserved word.

KM AAAA is code for operators.

In equivalence loops, a special meaning is given for K equal to 9, when M AAAA is a change in reference point of the equivalence loop, plus 50000.

Generator Code Formats

K T SSSS COOP

For operands, P is the sign, 0 plus, 5 minus

T is the type: 0 floating, 1 integer, 2 unspecified.

K equals 0: Simple variable, or a constant (if C is 5). SSSS is a link to the corresponding symbol table entry.

K equals 1: Computed result in rA.

K equals 2: Index Register 1 (do variable).

K equals 3: Array
SSSS links to dimension table entry when this array is sent from scan, and then after the subscript for the array is processed, SSSS links to an entry on the ARAS list. See routine A for the formats in ARAS.

K equals 4: Temp Storage
SSSS is the assignment in unique.

K equals 5: Label
Here SSSS is a link to the corresponding symbol table entry.

K equals 6: Function
SSSS is link to symbol table

K equals 7: Special
In the operand stack this is sometimes used for an array without a subscript.

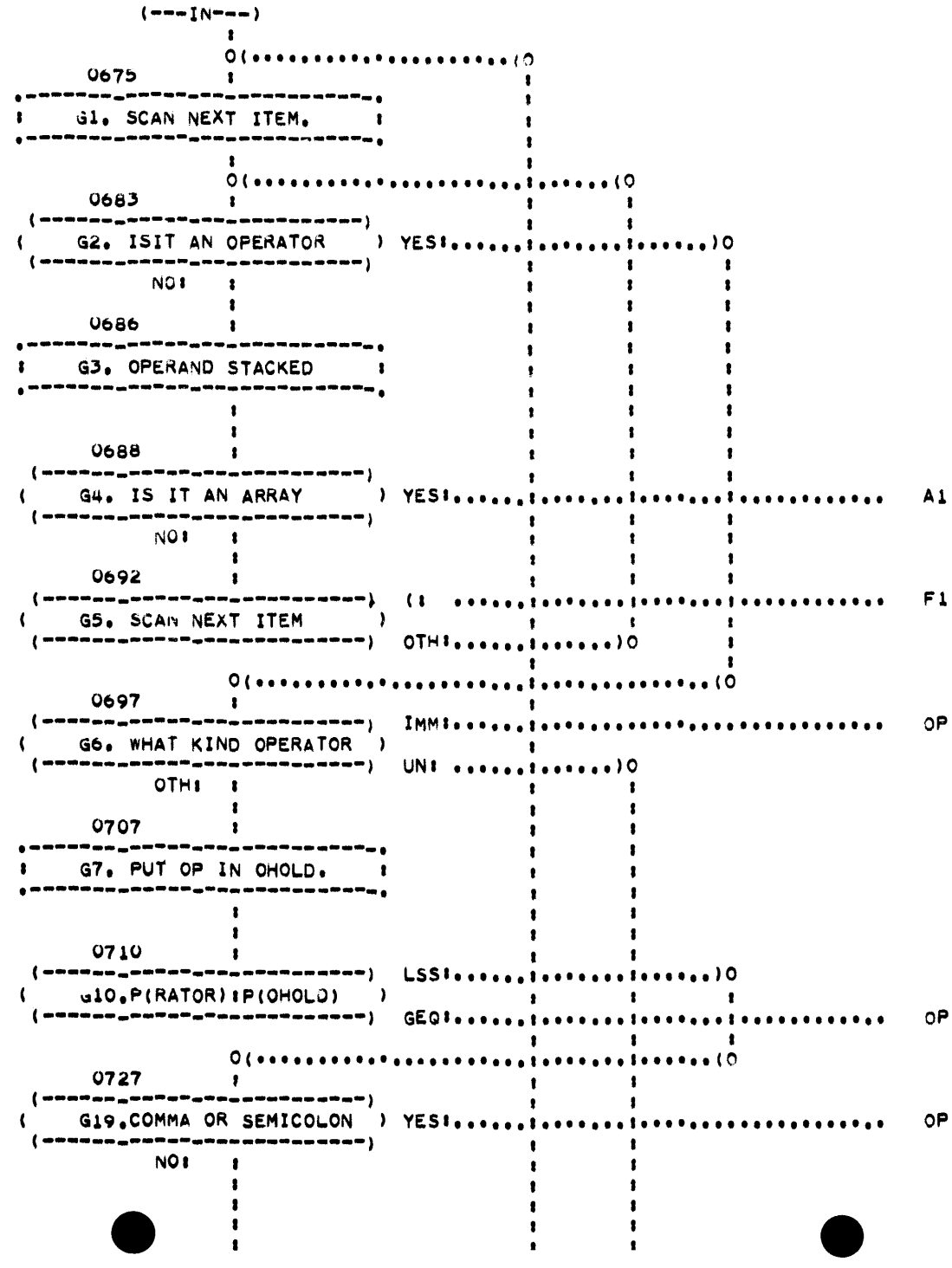
K equals 7, 8, or 9: Operator
KT SSSS is the same as the symbol table entry KM AAAA. KT is the priority of the operator. 99 means action for the operator immediately upon entry to GEN. 98 means the operator is a UNARY operator. Else T equal to 1, 3, 6, or 8 means immediate action before entering on the operator stack (see GEN control)

Reserved word codes which follow give the symbol table entries for all reserved identifiers and special characters, together with a symbolic reference corresponding to the assembly listing of

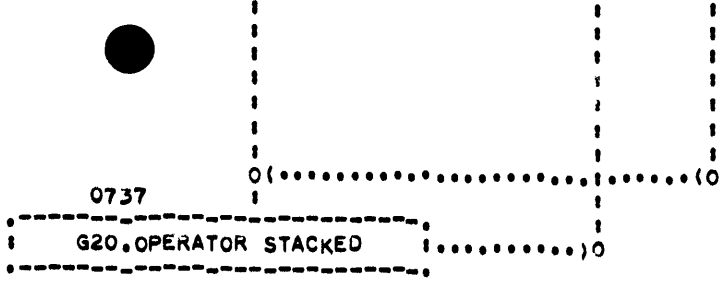
)))FORTRAN(((

Reserved word codes

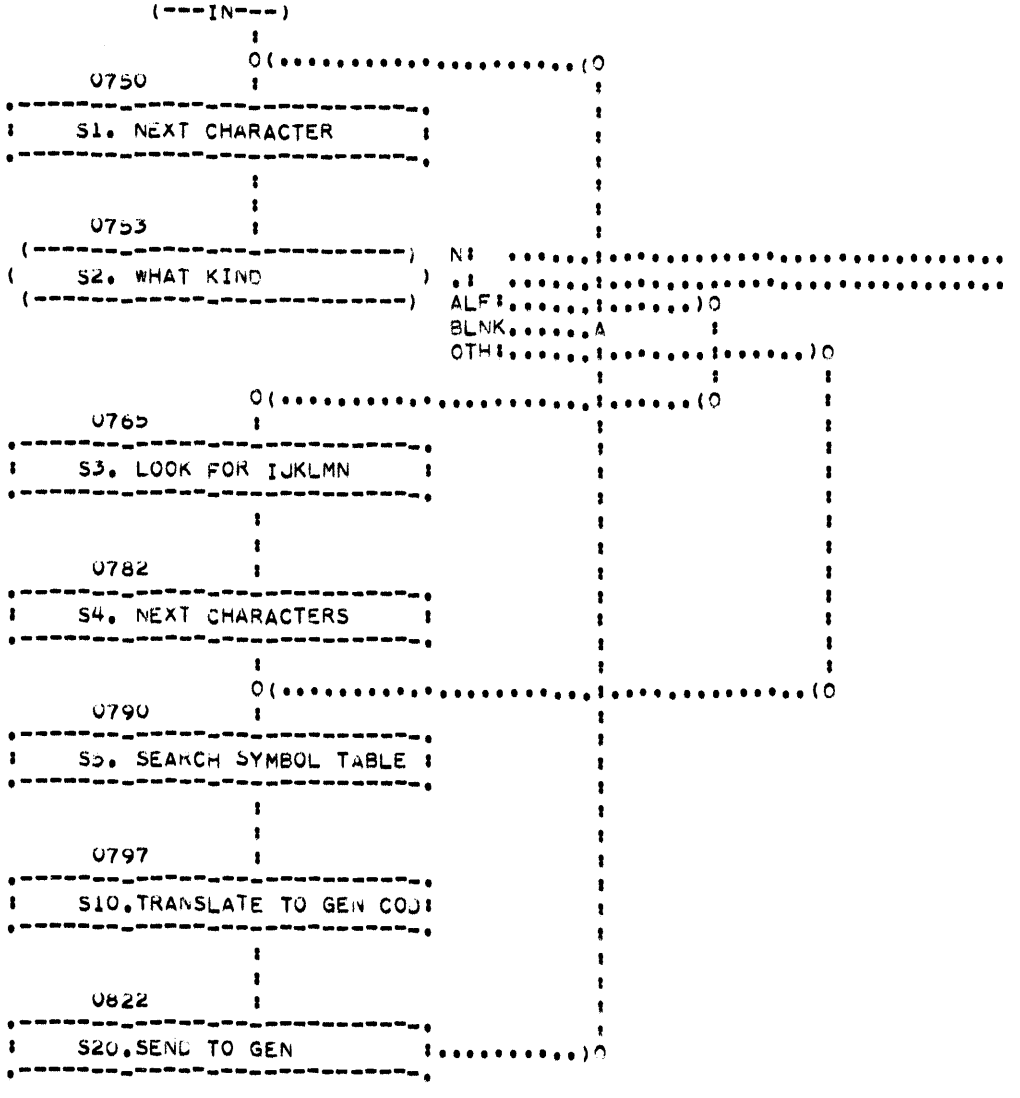
ITEM:	CODE:	SYMBOLIC:
&	9941050000	99 SIGN&
-	9941040000	99 SIGN-
/	8441150000	84 SIGN/
%	9941010000	99 SIGN%
*	9941110000	99 SIGN*
$	7341140000	73 SIGN$
:	7000000000	70 0000
'	7841130000	78 SIGN,
+	9941050000	99 SIGN&
	9941170000	99 SIGN#
(	9941010000	99 SIGN%
)	7000000000	70 0000
;	7341140000	73 SIGN$
NO	6941320000	69 WDNO
LIST	6941330000	69 WDLIS
CORE	6941370000	69 WDCOR
TRACE	6941360000	69 WDTRC
TO	6940500000	69 SCAN1
THROUGH	9941380000	99 WDTRU
GO	9941310000	99 WDGO
ASSIGN	9941300000	99 ASS1
IF	9941070000	99 WDIF
DO	9941000000	99 WDDO
CONTINUE	6940500000	69 SCAN1
PAUSE	9841410000	98 WDPOZ
STOP	9841420000	98 WDSTP
END	9941430000	99 WDEND
FUNCTION	9941440000	99 WDFUN
SUBROUTINE	9941450000	99 WDSUB
READ	9941460000	99 WDRED
PRINT	9941470000	99 WDPRT
FORMAT	9941490000	99 WDFMT
RETURN	9941500000	99 WDRTN
DIMENSION	9941510000	99 WDDIM
COMMON	9941520000	99 WDCOM
EQUIVLENCE	9941530000	99 WDEQU
SIN	9841200000	98 SINF
COS	9841210000	98 COSF
SQRT	9841190000	98 SQRTF
TAN	9841220000	98 TANF
ARCTAN	9841230000	98 ATANF
LN	9841240000	98 LNF
EXP	9841250000	98 EXPF
ABS	9841260000	98 ABSF
FLOAT	9841540000	98 FLOTF
FIX	9841550000	98 FIXF
PUNCH	9941590000	99 WDPCH
CALL	9941600000	99 WDCAL
NOT	9841610000	98 BCOMP
OR	7941630000	79 BOR
AND	8041620000	80 BLAND
CARDS	6941660000	69 WDPRG




```
*  G.  GENERATOR CONTROL
*      THIS ROUTINE CONTROLS THE
*      GENERATOR CO-ROUTINE.
*      THE NORMAL EXIT AT THE COMPLETION OF A GENER-
*      ATED ITEM IS TO G1, WHICH STARTS THE
*      PROCESSING OF THE NEXT ITEM. AT THE END OF
*      GENERATING CODE FOR CERTAIN OPERATORS, EXIT
*      OCCURS TO G10 RATHER THAN G1, SINCE WE MAY
*      WISH TO PERFORM SEVERAL OPERATIONS BEFORE
*      SCANNING ANOTHER ITEM.
*  G1. SCAN NEXT ITEM.
*      ACTIVATE THE SCANNER COROUTINE.
*      NORMALLY THIS MEANS WE ENTER STEP S1.
*  G2. ISIT AN OPERATOR
*      IF THE ITEM SCANNED IS AN OPERATOR, GO TO G6.
*  G3. OPERAND STACKED
*      PUT THE ITEM AT THE TOP OF THE OPERAND STACK.
*  G4. IS IT AN ARRAY
*      IF THE OPERAND IS A DIMENSIONED VARIABLE,
*      GO TO A1.
*  G5. SCAN NEXT ITEM
*      IF THE NEXT ITEM IS A LEFT PARENTHESIS, WE
*      TENTATIVELY HAVE A FUNCTION CALL SO WE GO
*      TO     STEP F1.
*      OTHERWISE WE GO BACK TO STEP G2.
*  G6. WHAT KIND OPERATOR
*      IF THE OPERATOR JUST SCANNED IS ONE THAT
*      REQUIRES IMMEDIATE ACTION (CODE 99), BRANCH
*      TO THE ROUTINE FOR THIS OP.
*      IF WE HAVE A UNARY OPERATOR (CODE 98) SUCH
*      AS LN OR ABS, GO TO G20.
*            OTHERWISE WE HAVE A BINARY OPERATOR
*      OR A DELIMITER WHOSE PRECEDENCE IS TO BE
*      TESTED.
*  G7. PUT OP IN OHOLD.
*      PUT THE OPERATOR JUST SCANNED INTO LOCATION
*      'OHOLD' BEFORE DECIDING WHAT TO DO WITH 'T.
*  G10.P(RATOR):P(OHOLD)
*      CHECK THE PRECEDENCE OF THE TOP OPERATOR ON
*      THE OPERATOR STACK AGAINST THE PRECEDENCE OF
*      THE OPERATOR IN 'OHOLD'.
*      IF IT IS LESS (E.G., IN A+B*C, + IS LESS
*      THAN *), WE MUST WAIT BEFORE OPERATING
*      FURTHER SO WE GO TO G19.
*      IF IT HAS GREATER PRECEDENCE OR
*      EQUAL PRECEDENCE,HOWEVER,THE OPERATOR ON TOP
*      OF THE STACK IS REMOVED AND WE BRANCH TO THE
*      APPROPRIATE ROUTINE FOR THIS OP.
*      PRECEDENCE IS 70 FOR VARIOUS KINDS OF LEFT
*      PARENTHESES,   73 FOR :    75 FOR EQUALS
*      78 FOR COMMA, 79 FOR OR, 80 FOR AND,
*      82 FOR PLUS AND MINUS, 84 FOR UNARY MINUS,
*      FOR MULTIPLY, AND FOR DIVIDE,   87 FOR POWER,
*      AND 93 FOR UNARY OPERATORS
*  G19.COMMA OR SEMICOLON
*      IF OHOLD HAS A PRECEDENCE WHOSE UNITS DIGIT
```



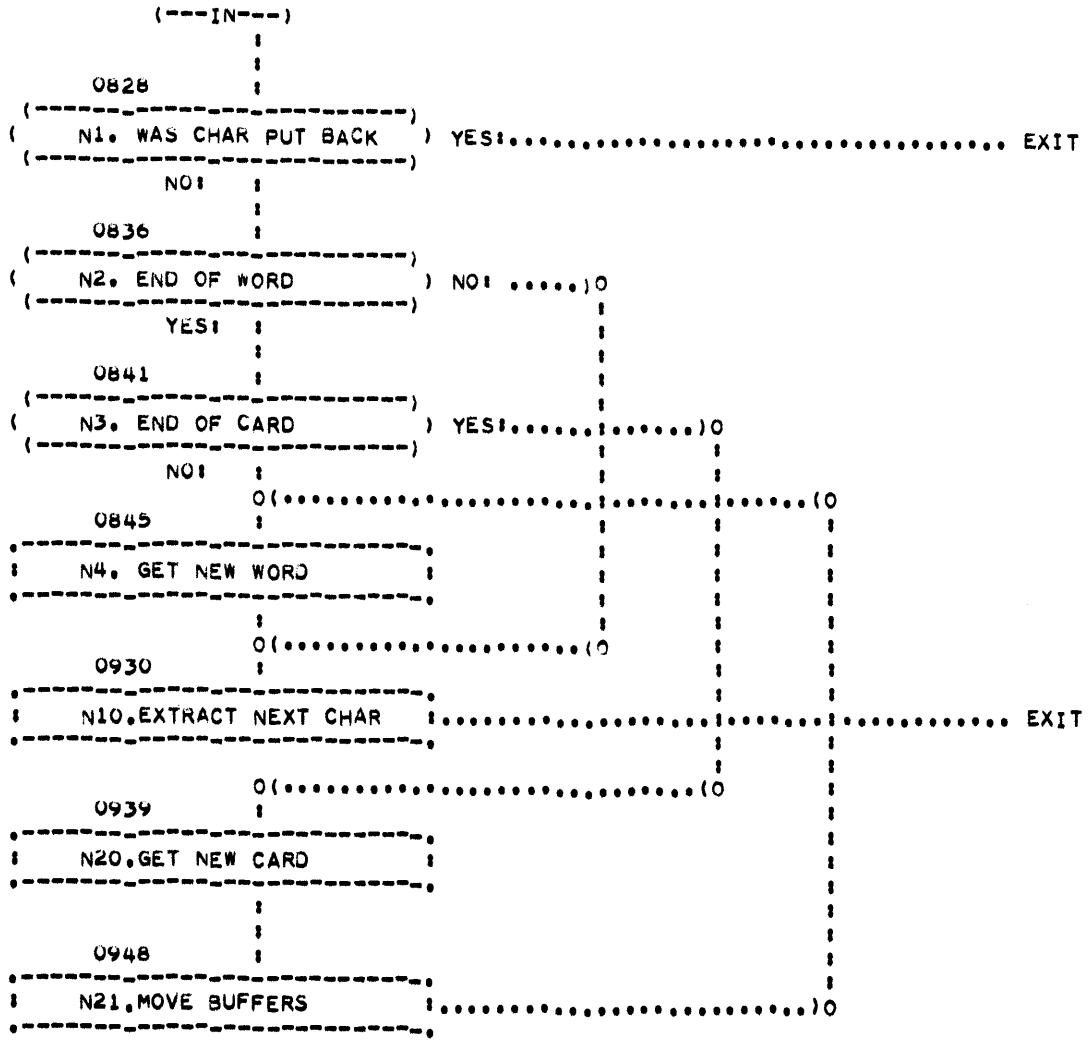

```
*
*
*
*
*
*
*       IS1,3,6,OR 8 IT MEANS WE ARE TO BRANCH TO
*       THIS  OP NOW THAT THE PRECEDENCE HAS BEEN
*       CHECKED.  AT PRESENT THIS IS USED ONLY FOR
*       SEMICOLON (EDN OF STATEMENT) OR COMMA AND THI
*       MEANS BRANCH TO THE ROUTINE SPECIFIED BY THE
*       CURRENT MODE.
*               OTHERWISE WE GO TO G20 TO PUT OHOLD
*       ON THE OPERATOR STACK
*   G20.OPERATOR STACKED
*       THE OPERATOR IS PUT ON TOP OF THE OPERATOR
*       STACK AND WE RETURN TO G1.
*   CODING DETAILS:
*       UPON ENTRY TO GET, REGISTER A CONTAINS THE
*       CURRENT ITEM AND REGISTER X CONTAINS THE
*       PREVIOUS ITEM. THESE ARE IN 'GENERATOR CODE'
*       WHICH IS EXPLAINED IN THE TABLE OF FORMATS
*       IN THE BEGINNING OF THE FLOWCHARTS.
*
```



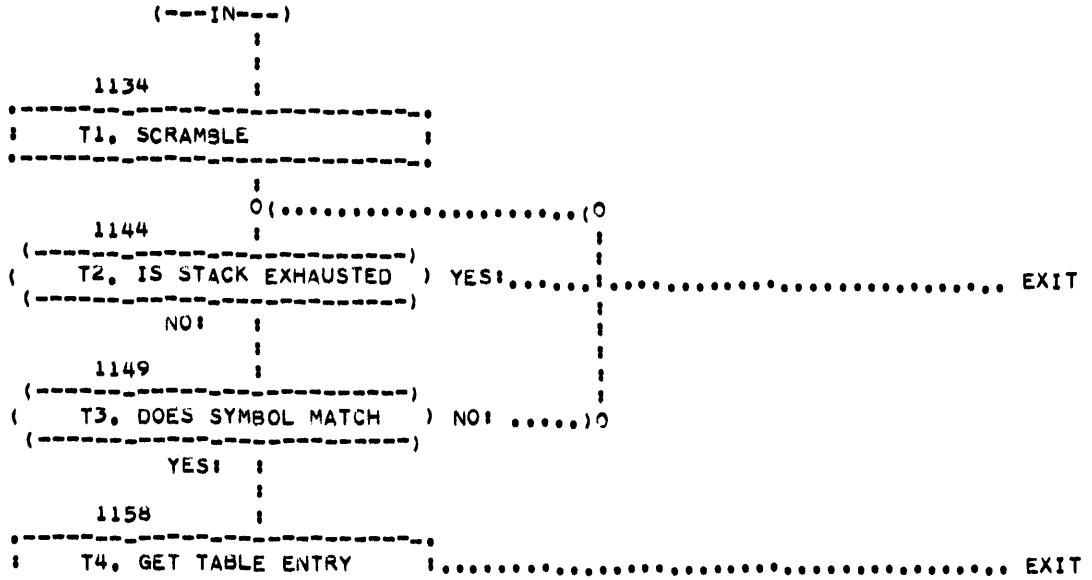

```
*   S.  SCANNER CONTROL
*       THIS ROUTINE CONTROLS THE SCANNER CO-ROUTINE.
*       NORMALLY ENTRY TO THE SCANNER IS TO STEP S1
*       WHICH BEGINS TO SCAN A NEW ITEM.
*   S1. NEXT CHARACTER
*       GET THE NEXT CHARACTER FORM THE INPUT CARD
*       (ROUTINE N).
*   S2. WHAT KIND
*       IF THE CHARACTER IS NUMERIC, IT IS THE
*       BEGINNING OF A CONSTANT, SO WE GO TO C1.
*       A DECIMAL POINT ALSO MEANS A CONSTANT, GO TO
*       STEP C2.
*       IF THE CHARACTER IS ALPHABETIC IT MEANS THE
*       FIRST LETTER OF AN IDENTIFIER, SO WE GO TO
*        S3.
*       IF THE CHARACTER IS BLANK,RETURN TO S1.
*       OTHERWISE WE HAVE A SPECIAL CHARACTER.  EACH
*       SPECIAL CHARACTER IS TREATED EXACTLY AS AN
*       IDENTIFIER TO LENGTH 1 AND WE GO TO STEP S5.
*   S3. LOOK FOR IJKLMN
*       IF THIS CHARACTER IS THE LETTER I THROUGH N,
*       RECORD FOR FUTURE REFERENCE THAT THIS
*       IDENTIFIER IS INTEGER TYPE. ALSO PREPARE TO
*       BUILD UP TO FIVE CHARACTERS OF EVERY IDEN-
*       TIFIER IN A COMPUTER WORD, IN THE FORM
*       ZZZZZNNNNN WITH LEADING BLANKS.
*   S4. NEXT CHARACTERS
*       SUCCESSIVELY GET CHARACTERS FROM THE CARD
*       (ROUTINE N) UNTIL THE FIRST NON-ALPHNUMERIC
*       CHARACTER APPEARS. IF THE TERMINAL CHARACTER
*       IS NONBLANK,'PUT IT BACK'ON THE CARD SO IT
*       WILL COME THROUGH AGAIN NEXT TIME.
*   S5. SEARCH SYMBOL TABLE
*       ACTIVATE ROUTINE T TO SEARCH FOR THIS IDENT-
*       IFIER OR SPECIAL CHARACTER IN THE SYMBOL
*       TABLE. IF NOT FOUND, IT IS ENTERED IN THE
*       TABLE AS A SIMPLE VARIABLE.  IF FOUND, THE
*       CODE FOUND IS USED IN STEP S10.
*   S10.TRANSLATE TO GEN CODE.
*       WE HAVE AN ITEM WHICH WE WANT TO SEND
*       TO THE GENERATOR, BUT IT IS IN SYMBOL TABLE
*       FORMAT RATHER THAN GENERATOR FORMAT.
*       SPECIFICATIONS OF THESE FORMATS ARE GIVEN AT
*       THE BEGINNING OF THE FLOWCHART LISTINGS.
*       THE CONVERSION IS MADE AT THIS POINT. IF THE
*       SPECIAL CODE 69 OCCURS HERE A BRANCH IS MADE
*       TO THE SPECIAL SCANNER OPERATOR WHICH NEVER
*       GETS TO THE GENERATOR CO-ROUTINE, SUCH AS
*       TRACE,LIST,CARDS,ETC. THE APOSTROPHE OPERATOR
*       (MEANING END OF CARD), ROUTINE Q, IS ONE OF
*       THESE SPECIAL SCANNER OPERATORS    , THE
*       OTHERS ARE MENTIONED IN STEP U29.
*   S20.SEND TO GEN
*       THE CODED ITEM IS SENT TO GEN, USUALLY
*       THIS IS TO STEP G1. UPON REENTRY, SCAN WILL
*
*
*
*
*
```

```
*
*
*
*
*
*
*
*
*
*
*
```

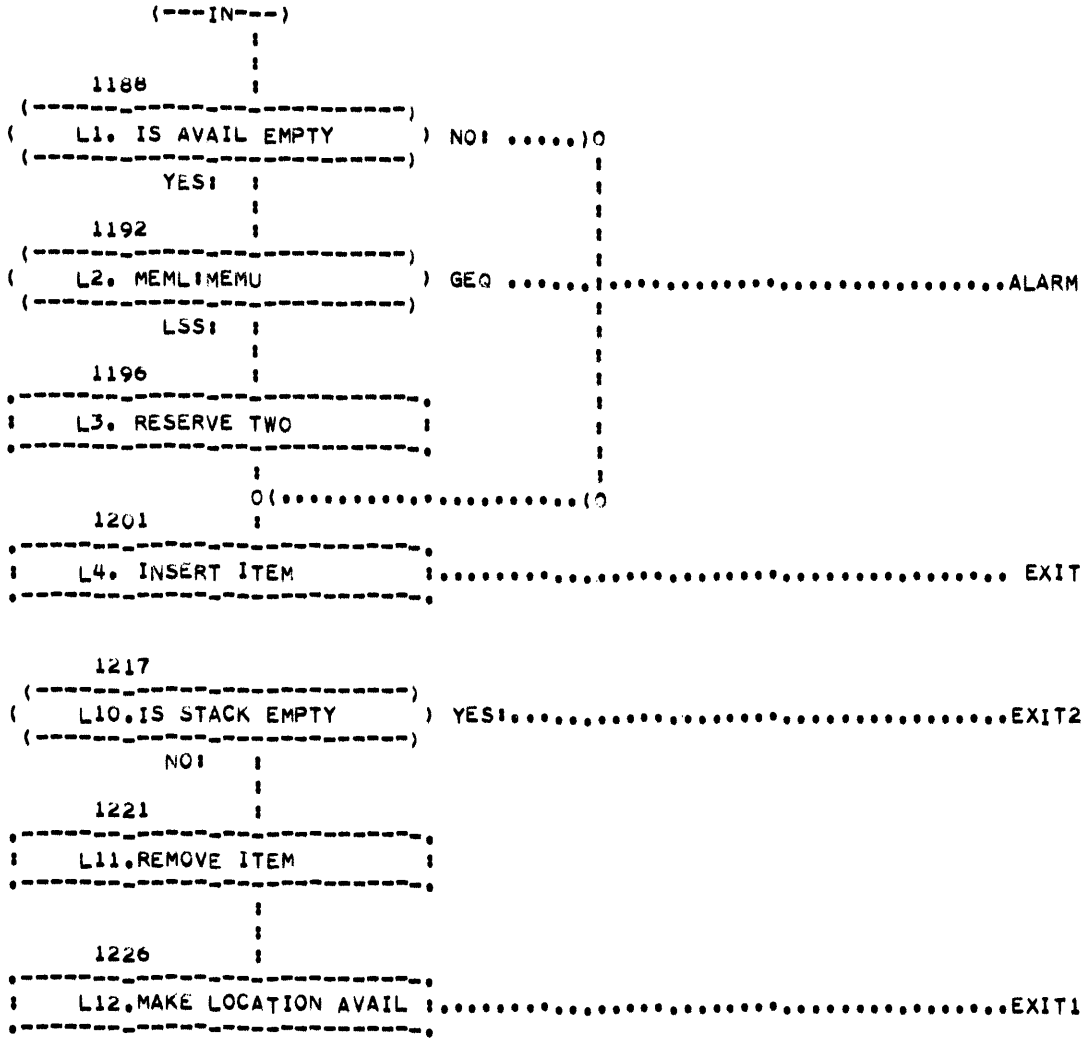
START UP AGAIN AT S1.



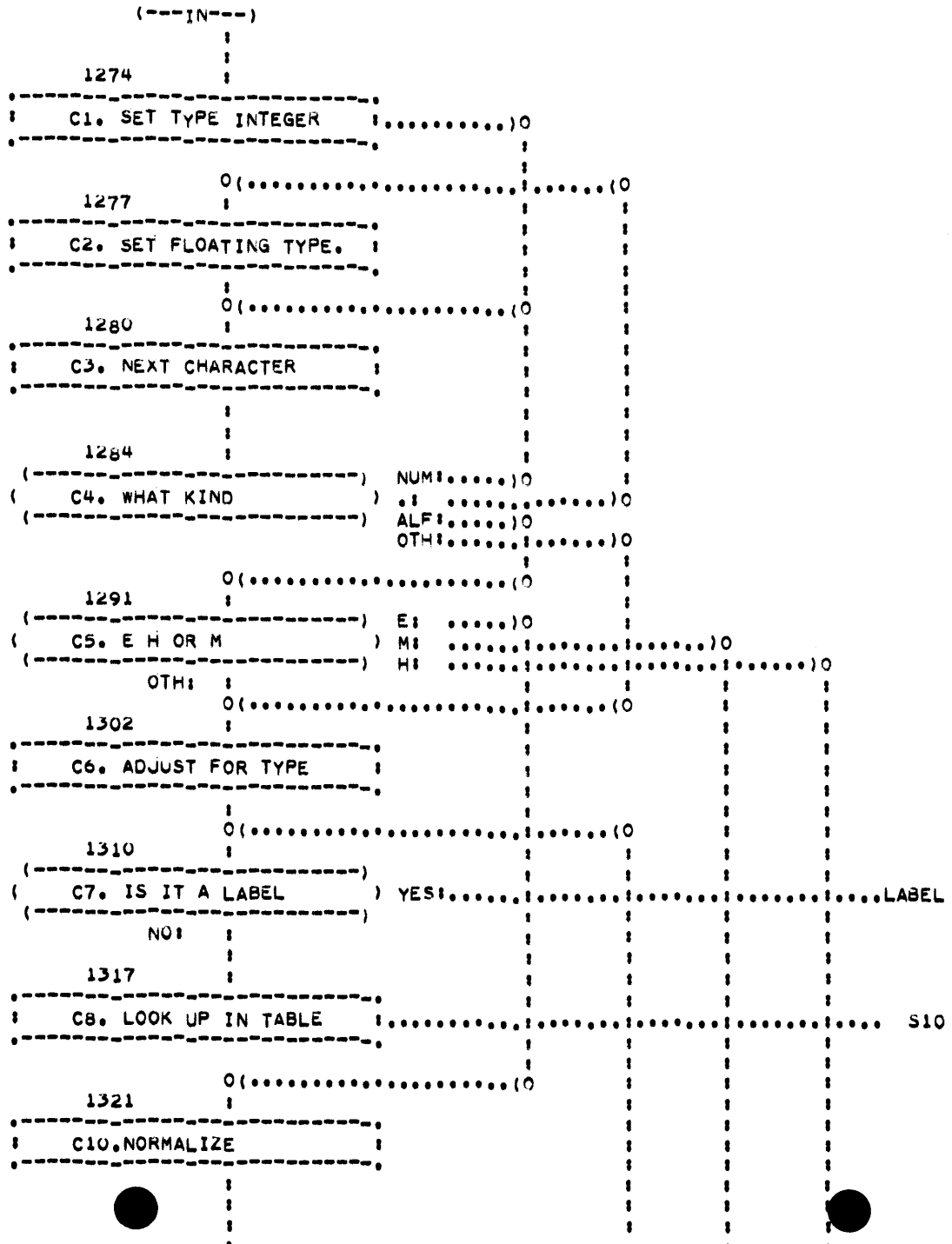

```
*   N.  'GET NEXT CHARACTER' ROUTINE
*   N1. WAS CHAR PUT BACK
*       IF A CHARACTER HAS BEEN 'PUT BACK' ONTHE CARD
*       RE-EMIT THIS CHARACTER AND EXIT.
*   N2. END OF WORD
*       IF WE ARE NOT AT THE END OF THE CURRENT
*       TEN-COLUMN PART OF THE CARD, GO TO STEP N10,
*       ELSE WE MUST BRING UP ANOTHER SECTION OF THE
*                CARD.
*   N3. END OF CARD
*       IF WE ARE AT THE END OF THIS CARED,   GO TO
*       STEP N20.
*   N4. GET NEW WORD
*       BRING UP THE NEW WORD. THIS MEANS USUALLY
*       THAT THE NEXT TEN ZONES AND NEXT TEN
*       NUMERICS ARE BROUGHT UP. SPECIAL ACTION IS
*       TAKEN ON THE 8TH WORD OF 80-COLUMN   CARDS
*       TO STOP AFTER COLUMN 72, AND ON THE FIRST
*       WORD TO START EITHER AT COLUMN 7 OR AT
*       COLUMN 1 IF THERE IS A LABEL.
*   N10.EXTRACT NEXT CHAR
*       REMOVE THE NEXT CHARACTER FROM THE CARD AND
*        EXIT.
*   N20.GET NEW CARD
*       UNLOAD HSR BUFFER IF IT HAS NOT ALREADY BEEN
*       UNLOADED. IF NO CARD IS CURRENTLY IN PROCESS,
*       GIVE 2223 ERROR HALT.
*   N21.MOVE BUFFERS
*       INITIATE READING NEXT CARD, AND TRANSFER
*       HSR INTERLACE TO PRINTER INTERLACE.
*       PRINT OUT THE CARD IMAGE, TOGETHER WITH
*       LEVEL + BAND.
*       RESET EMITTER AND GO TO N4.
```



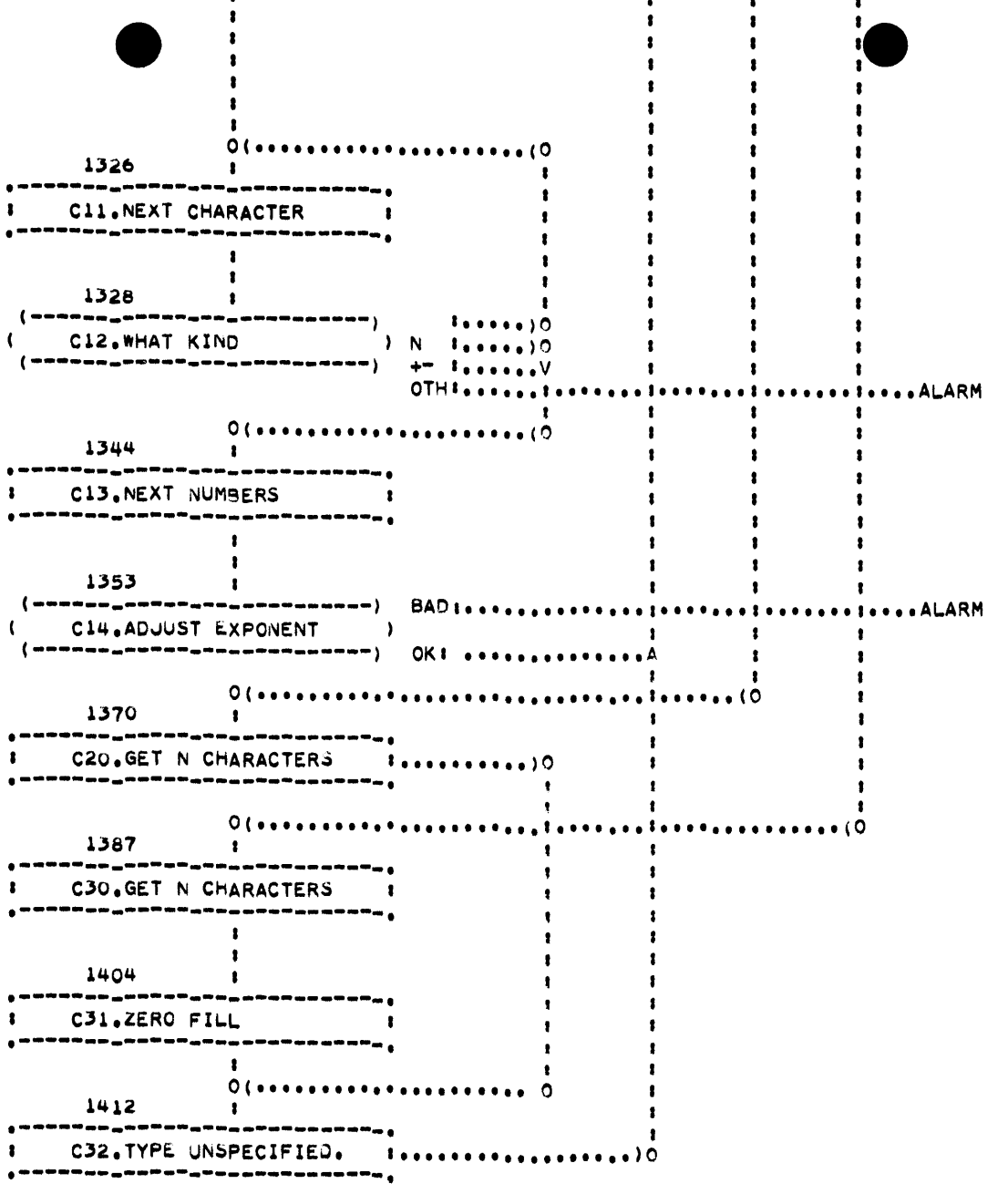

```
*   T.  SYMBOL TABLE SEARCH.
*       THIS SUBROUTINE IS USED TO LOOK UP IDENTIFIER
*       SPECIAL CHARACTERS, CONSTANTS, AND STATEMENT
*       NUMBERS (LABELS) IN THE BIG TABLE.  IF NOT
*       IN THE TABLE, THE ITEM IS ENTERED IN.
*   T1. SCRAMBLE
*       MULTIPLY ITEM BY 1010101010 AND THEN  ADD
*       (0 FOR CONSTANTS, LENGTH FOR IDENTIFIERS,
*       OR 99 FOR STATEMENT NUMBERS).  TAKE THE
*       RESULT MOD 100, GIVING THE STACK HEAD NUMBER
*       FOR THIS SYMBOL.
*   T2. IS STACK EXHAUSTED
*       IF THISSTACK HAS BEEN ENTIRELY PROCESSED,
*       INSERT THIS ITEM INTO THE TABLE ON THIS
*       STACK.  EXIT.
*   T3. DOES SYMBOL MATCH
*       COMPARE THE CURRENT ITEM IN THE STACK
*       AGAINST THE DESIRED SYMBOL.  IF THERE IS NO
*       MATCH, GO BACK TO  T2.
*   T4. GET TABLE ENTRY
*       GET THE CORRESPONDING ENTRY FOR THE SYMBOL WE
*       HAVE JUST FOUND.  EXIT.
*   CODING DETAILS:
*       AT INPUT, REGISTER A CONTAINS THE LENGTH OF
*       SYMBOL, REGISTER X CONTAINS THE CODE TO USE
*       IF NOT FOUND IN THE TABLE, RL CONTAINS THE
*       EXIT INSTRUCTION, TEMP2 CONTAINS THE SYMBOL.
*       AT EXIT, RL CONTAINS THE LOCATIONS OF THE
*       TABLE ENTRY IN ITS M ADDRESS, AND TEMP4 IS
*       THE EQUIVALENT OF THE SYMBOL.
*
```




```
*   L.  LINKED MEMORY SUBROUTINES.
*       THESE SUBROUTINES ARE USED IMPLICITLY IN MANY
*       PLACES OF THE PROGRAM, TO STORE AND RETRIEVE
*       INFORMATION FROM A POOLED MEMORY AREA.
*       THE FORMAT FOR POOLED MEMORY IS
*           STACK HEAD:  00 LINK 0000
*           AVAIL STACK  00 LINK 0000
*       OTHER ITEMS ARE IN TWO WORD FORMAT:
*             LINK      INFO1  LINK   1111112222
*             LINK+1         I N F O 2 1111111111
*       ZERO LINK INDICATES THE END. THE POOL IS
*       KEPT BETWEEN LOCATIONS MEML1 AND MEMU1
*       THE SYMBOL TABLE AND STACKS WORK DOWN FROM
*       MEMU1, DIMENSIONS AND EQUIVALENCE ENTRIES
*       ARE INSERTED UP FROM MEML1.
*       IN THIS SECTION, ENTRANCE L1 IS CALLED 'INS'
*       AND IT IS FOR INSERTING ITEMS, WHILE ENTRANCE
*       L10 IS FOR DELETING ITEMS FROM STACKS AND IT
*       IS CALLED 'REM'.
*   L1. IS AVAIL EMPTY
*       IF THE AVAIL STACK IS NOT EMPTY,REMOVE AN
*       ITEM AND GO TO L4.
*   L2. MEML:MEMU
*       IF THERE IS NO ROOM FOR ANOTHER ITEM, GIVE
*       THE I'M FULL ERROR ALARM.
*   L3. RESERVE TWO
*       DECREASE MEMU BY 2, WE WILL USE THESE TWO
*       LOCATIONS FOR THE NEW ITEM.
*   L4. INSERT ITEM
*       PUT THE NEW ITEM INTO THE MEMORY, FIX UP
*       LINKS PROPERLY.  EXIT.
*   CODING DETAILS FOR INS:
*       RB1 CONTAINS STACK HEAD LOCATION
*       RL  CONTAINS EXIT INSTRUCTION
*       RA  CONTAINS INFO2,RX CONTAINS INFO1
*       AT EXIT, RL IS NEW CONTENTS OF STACK HEAD,
*       RX IS INFO2.
*   L10.IS STACK EMPTY
*       IF STACK HAS NO ITEMS,GO TO EXIT2.
*   L11.REMOVE ITEM
*       REMOVE TOP ITEM OF STACK
*   L12.MAKE LOCATION AVAIL
*       PUT THE LOCATION JUST FREED ONTO THE AVAIL
*       STACK.  EXIT1.
*   CODING DETAILS FOR REM:
*       RB1 IS THE STACK HEAD LOCATION,
*       RX IS THE EMPTY EXIT (EXIT2),
*       RL IS THE ORDINARY EXIT1.
*       OUTPUT: RB1 IS THE LOCATION, RL IS INFO1.
*       INFO2 IS STILL IN MEMORY.
*
```




```
*   C.  CONSTANT SCANNER
*   C1. SET TYPE INTEGER
*       INITIALIZE  N  TO THE NUMBER JUST SCANNED,
*       SET TYPE INTEGER. GO TO C3.
*   C2. SET FLOATING TYPE.
*       SET  N TO FLOATING POINT TYPE.
*   C3. NEXT CHARACTER
*       GET THE NEXT NON-BLANK CHARACTER FROM THE
*       CARD (ROUTINE N).
*   C4. WHAT KIND
*       IF CHARACTER IS NUMERIC,SET N TO 10N+CHAR,
*       GO TO C3.
*       IF A DECIMAL POINT, GO TO C2.
*       IF ALPHABETIC, GO TO C5.
*       IF SPECIAL CHARACTER,PUT IT BACK ON THE CARD,
*       AND GO TO C6.
*   C5. E H OR M
*       IN A STATEMENT LABEL CONTEXT WE GO IMMEDIATEL
*       TO C6,  OTHERWISE WE GO TO C10 FOR AN E,
*       TO  C20 FOR AN M,
*       TO C30 FOR AN H,
*       OTHERWISE IT IS THE END OF THE CONSTANT
*       (PROBABLY SYNTACTICALLY INCORRECT) AND WE GO
*       TO STEP C6.
*   C6. ADJUST FOR TYPE
*       IF FLOATING POINT TYPE OCCURRED, CONVERT N TO
*       FLOATING POINT FORMAT, ELSE SET N TO 1000
*       TIMES N.
*   C7. IS IT A LABEL
*       IF LABEL CONTEXT,ENTER SPECIAL ROUTINE FOR
*       THIS CASE,DEPENDING ON THE SETTING OF THE
*       LABEL SWITCH. THE LABEL SWITCH IS AUTOMAT-
*            ICALLY SET OFF EVERY TIME GEN IS
*       ENTERED; GEN WILL SET IT WHENEVER A LABEL MAY
*       BE EXPECTED.
*   C8. LOOK UP IN TABLE
*       ACTIVATE ROUTINE T FOR THIS CONSTANT, THEN GO
*       TO S10 TO SEND A CONSTANT CODE TO GEN.
*   C10.NORMALIZE
*       INSERT A DECIMAL POINT IF NONE PRECEDED,
*       E.G. 2E5.
*   C11.NEXT CHARACTER
*       ACTIVATE ROUTINE N FOR THE NEXT CHARACTER.
*   C12.WHAT KIND
*       IF BLANK, RETURN TO C11.
*       IF NUMERIC,PUT BACK ON CARD,RECORD + SIGN.
*       TO C13.
*       IF PLUS OR MINUS, RECORD THE SIGN, TO C13.
*       OTHERWISE GIVE THE BAD CONSTANT ALARM.
*   C13.NEXT NUMBERS
*       CONTINUE ACTIVATING ROUTINE N UNTIL A NON-BLA
*       NK, NON-NUMERIC CHARACTER APPEARS.
*   C14.ADJUST EXPONENT
*       ADD THE EXPONENT TO THE FLOATING POINT
*       CONSTANT. IF OVERFLOW OR UNDERFLOW OCCURS,
*       GIVE THE BAD CONSTANT ALARM.
*       OTHERWISE RETURN TO C7.
*
*
```

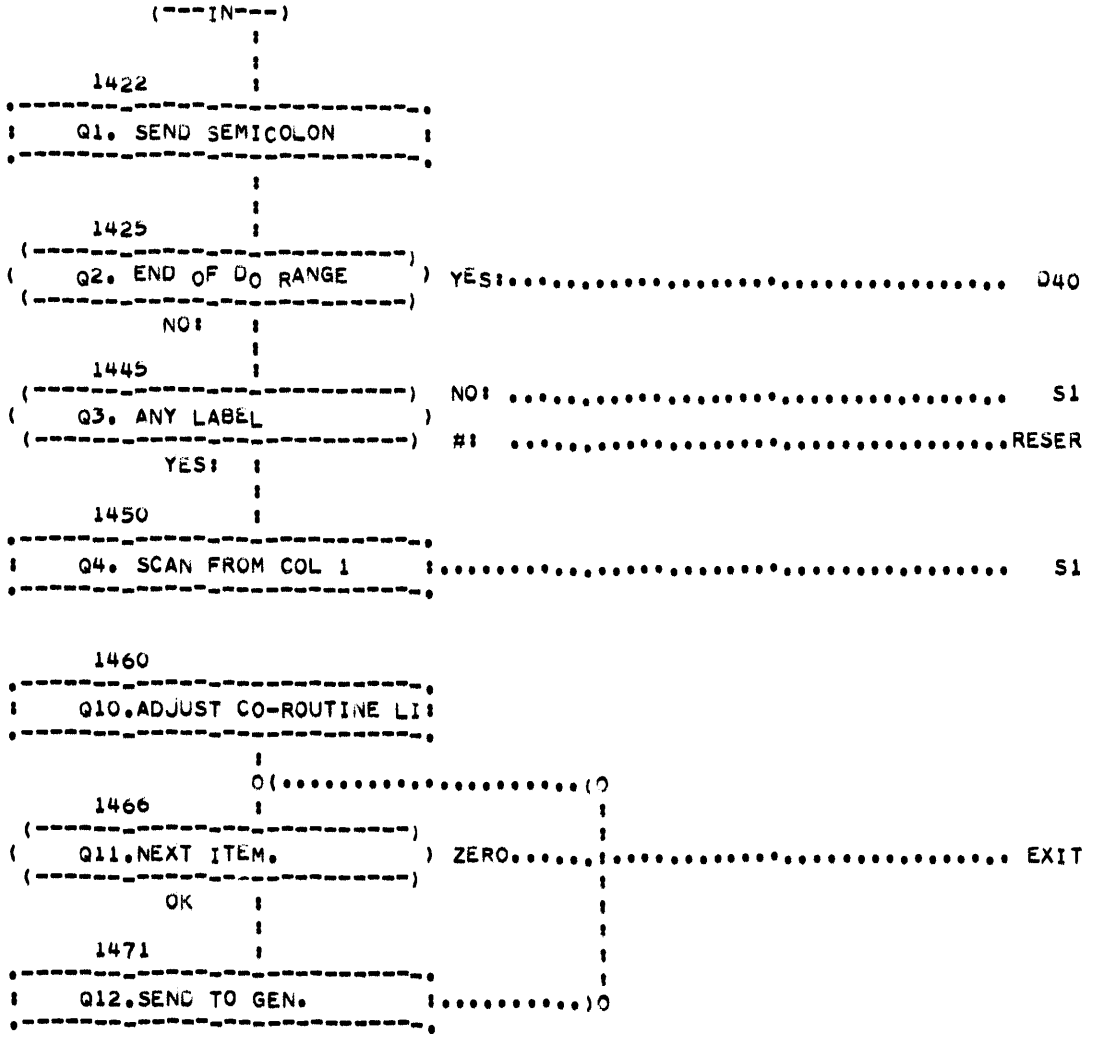


C20.GET N CHARACTERS
GET NEXT N CHARACTERS FROM CARD INCLUDING BLANKS AND BUILD MACHINE CODE CONSTANT. GO TO C32.

C30.GET N CHARACTERS
SET HOLLERITH SWITCH IN ROUTINE N; THIS SWITCH SIGNALS THAT ROUTINE TO TRANSMIT CHARACTERS IN CARD CODE ON 90-COLUMN SYSTEMS AND ALSO TO SUPPRESS A SPECIAL HIGH-SPEED SKIP OVER BLANK COLUMNS WHICH IT USUALLY HAS. GET THE NEXT N CHARACTERS FROM THE CARD, AND BUILD AN ALPHA CODE CONSTANT IN CARD CODE.

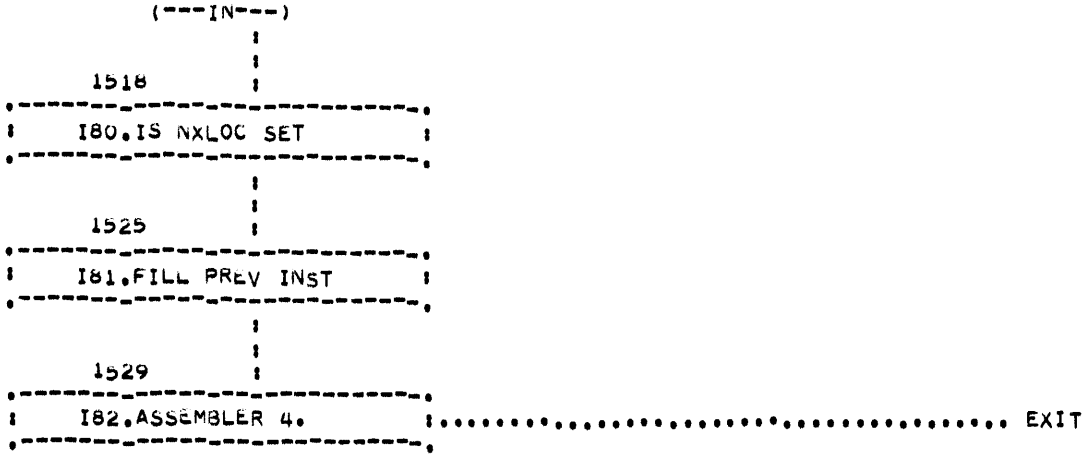
C31.ZERO FILL
IF N IS LESS THAN 5, ADD ZEROES TO FILL THE CONSTANT. RESTORE THE HOLLERITH SWITCH TO NORMAL. IF N IS MORE THAN 5, CRAZY CONSTANTS ARE GENERATED

C32.TYPE UNSPECIFIED.
SET THE TYPE OF THIS CONSTANT TO UNSPECIFIED.
GO TO C7.

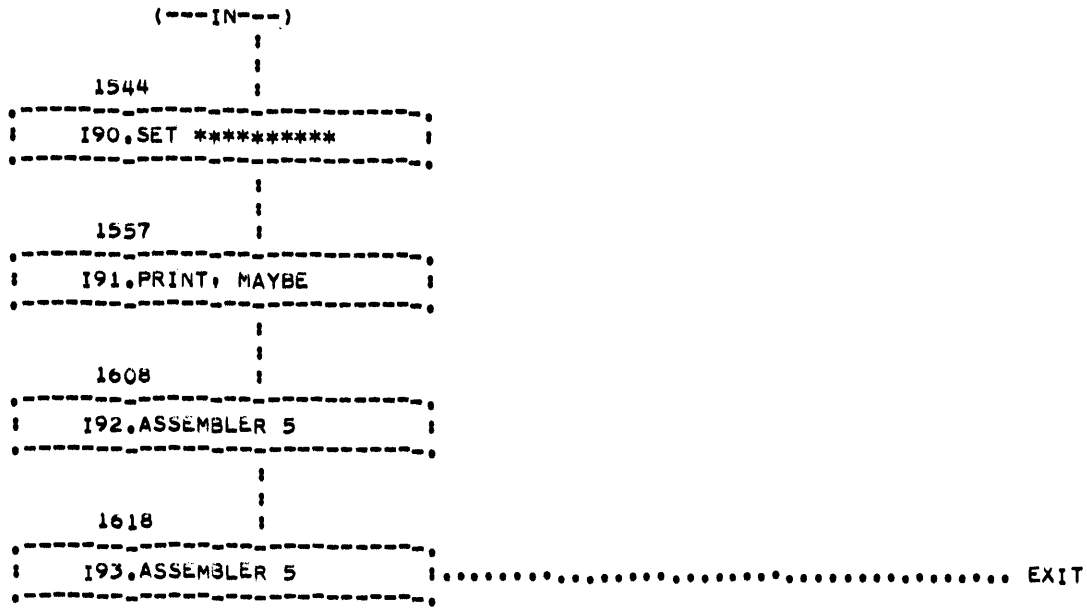



```
*  Q.  SPECIAL SCANNING ROUTINES
*      ENTRANCE TO Q1 OCCURS WHEN THE END OF CARD
*      ( WHICH IS DETECTED BY AN APOSTROPHE INSERTED
*      BY ROUTINE N) IS SENSED.  ENTRANCE Q10 IS
*      USED TO DIVERT NORMAL CONTROL OF SCAN, IN
*      ORDER TO EMIT A STRING OF INSERTED ITEMS
*      BEFORE RESUMING ORDINARY SCANNING.
*  Q1. SEND SEMICOLON
*      SEND SEMICOLON TO GEN ROUTINE, INDICATING
*      END OF THE STATEMENT.
*  Q2. END OF DO RANGE
*      IF THE CARD JUST COMPLETED IS THE END OF A
*      DO RANGE, GO TO D40.
*  Q3. ANY LABEL
*      IF COLS 1-5 OF THE NEXT CARD ARE BLANK, TO S1.
*      IF COL 1 IS NUMB SIGN GO TO SPECIAL RESERVED
*      WORD ENTERING PROCEDURE.
*  Q4. SCAN FROM COL 1
*      SET TO SCAN THIS CARD AT COLUMN 1 RATHER THAN
*      COLUMN 7, AND SET THE LABEL SWTICH (C7) TO
*      JUMP TO THE CHECKING ROUTINE MENTIONED IN THE
*      COMMENT JUST BEFORE STEP D40.  THEN
*      RETURN TO S1.
*  Q10.ADJUST CO-ROUTINE LINKS.
*      STORE CURRENT STARTING PLACE FOR SCAN CO-RTNE
*      IN EXIT OF THIS DIVRT SUBROUTINE. WE WILL
*      COME BACK TO THIS AFTER ALL SPECIAL ITEMS
*      HAVE BEEN INSERT IN THE PSEUDOCODE.
*  Q11.NEXT ITEM.
*      LOOK AT THE NEXT ITEM TAKEN FROM THE INSER-
*      TION TABLE. IF IT IS ZERO, WE ARE DONE
*      INSERTING AND SO WE EXIT TO RESTART THE SCAN
*            CO-ROUTINE.
*  Q12.SEND TO GEN.
*      SEND ITEM TO GEN, THEN RETURN TO Q11.
*  CODING DETAILS:  DIVT2 IS USED TO RE-INSERT THE
*      PREVIOUSLY SCANNED ITEM AT THE END OF THE
*      OTHER INSERTS. DIVT1 IS USED TO RESET GEN
*      TO ENTER AT G1. DIVRT IS THE NORMAL ENTRY.
*      REGISTER A CONTAINS THE STARTING T-TABLE
*      ENTRY. THIS ROUTINE IS ENTERED FROM GEN.
*
*
*
```

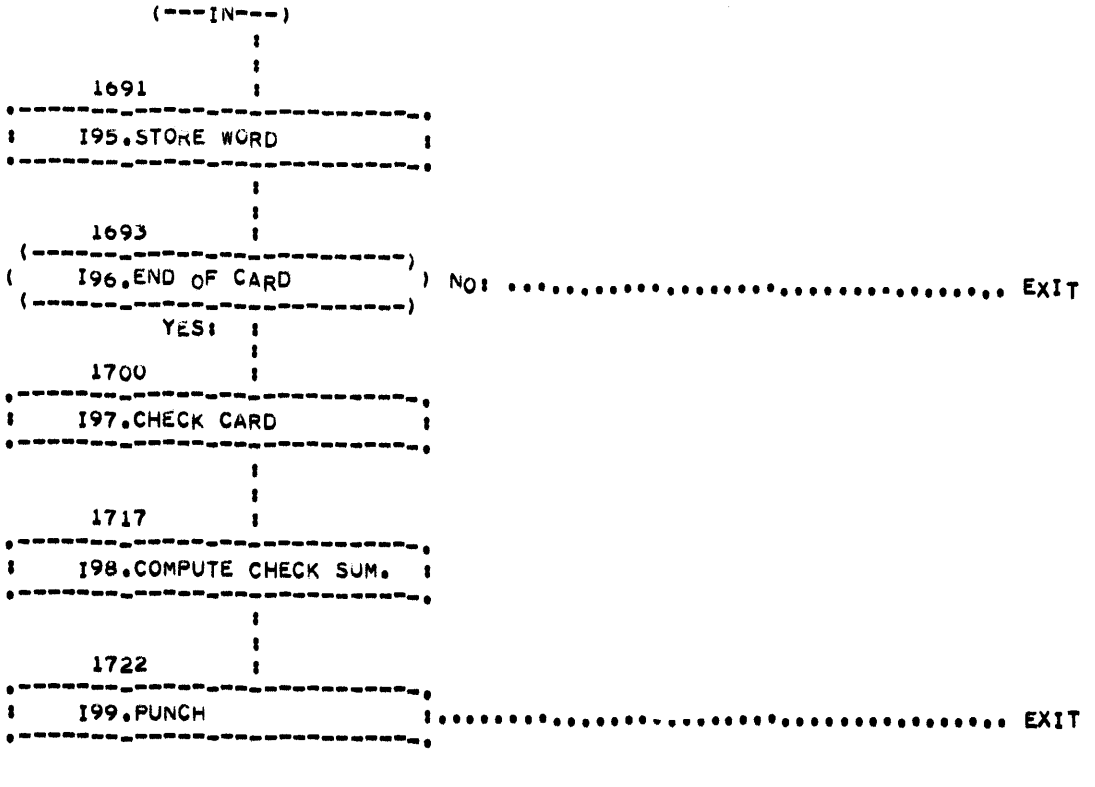
```
*    I.  ASSEMBLER STRUCTURE
*    TABLE OF CONTENTS
*        THIS SECTION IS A COMPLEX OF SUBROUTINES FOR
*        ASSEMBLING THE MACHINE LANGUAGE INSTRUCTIONS.
*        THE NAMES OF THESE VARIOUS LEVELS AND THEIR
*        FUNCTIONS ARE
*        I1.  ASM1  MACRO ASSEMBLER ... ASSEMBLES
*                    1 TO 5 INSTRUCTIONS AND/OR
*                    PSEUDO-INSTRUCTIONS.
*        I25. ASM2  ASSEMBLES ENCODED INSTRUCTIONS,
*                   FIXING UP THE ADDRESSES OF OPERAND
*        I30. ASM25  HALF ASMBLER,LIKE ASM2 EXCEPT IT
*                DEALS WITH ONE ADDRESS  M,C  ONLY.
*        I35. ASM28 SPECIAL ASSEMBLER FOR ADDRESSES OF
*                 SIMPLE VARIABLES AND CONSTANTS.
*        I50. ASIGN  FINDS ADDRESSES OF OPERANDS
*        I60. LSW    FINDS ADDRESSES OF STATEMENT LABEL
*        I70. CASIN  FINDS ADDRESSES OF CONSTANTS.
*        I80. ASM3     ASSEMBLES INSTRUCTIONS AND
*                    FIXES UP REFERENCES TO'NEXT INST.'
*        I90. ASM4    PROCESSES ASSEMBLED INSTRUCTIONS
*                AND LOCATIONS, IN OR OUT OF SEQUENCE,
*                AND PERHAPS LISTS THEM.
*        I95. ASM5  PUT ONE ITEM ON OUTPUT CARD.
*
```




```
*   I.  I80. ASSEMBLER 3
*       THIS SUBROUTINE ASSMELBES ABSOLUTE INSTRUC-
*       TIONS AND FIXES UP REFERENCES TO 'NEXT'.
*       A ONE-CYCLE DELAY IS KEPT,AN INSTRUCTION IS
*       NO PUT OUT UNTIL THE NEXT INSTRUCTION COMES
*       ALONG.
*   I80.IS NXLOC SET
*       IF NO PARTICULAR LOCATION FOR THE CURRENT
*       INSTRUCTION HAS BEEN CHOSEN, CHOOSE THE NEXT
*       LOCATION IN THE INTERLACE SEQUENCE.
*   I81.FILL PREV INST
*       FILL BLANK ADDRESSES IN PREVIOUS INSTRUCTION,
*       IF ANY, WITH THE LOCATION OF THIS ONE.
*   I82.ASSEMBLER 4.
*       ACTIVATE ROUTINE I91 TO OUTPUT THE PRECEDING
*       INSTRUCTION.  EXIT.
*   CODING DETAILS: RX IS 0RR0S000FF  WHERE RR ARE
*       RELOCATION DIGITS FOR M AND C, S IS SIGN, AND
*       FF ARE 0 OR 1 FOR NON-BLANK OR BLANK ADDRESS,
*       RESPECTIVELY.  RA IS THE INSTRUCTION, RL IS
*       THE EXIT. ASM31-ASM37 ARE SPECIAL ENTRANCES
*       FOR THE MOST COMMON CASES IN SETTING RX.
*
```




```
*   I.  I90. ASSEMBLER 4.
*       THIS SUBROUTINE PROCESSES ASSEMBLED INSTRUC-
*       TIONS AND LOCATIONS. ENTRY I90 IS USED FOR
*       OUT-OF-SEQUENCE LINES; I91 FOR THE PROGRAM
*       SEQUENCE.
*   I90.SET **********
*       SAVE COMMENT RESERVED FOR NEXT INSTRUCTION IN
*       PROGRAM SEQUENCE, AND INSERT THE COMMENT
*       **********.
*   I91.PRINT, MAYBE
*       IF LIST MODE IS ON, PRINT THE ASSEMBLED LINE
*       AND THE COMMENT.
*   I92.ASSEMBLER 5
*       PUT THE CONTROL WORD INTO THE OUTPUT
*       (ROUTINE I95)
*       AND ALSO STORE THE COMMENT FOR THE NEXT
*       INSTRUCTION LINE.
*   I93.ASSEMBLER 5
*       PUT THE INSTRUCTION WORD INTO THE OUTPUT
*       (ROUTINE I95).  EXIT.
*   CODING DETAILS:
*       ASM43,ASM44 PUT REGISTER A AS OUT-OF-SEQUENCE
*       LINE INTO NEXT LOCATION OF UNIQUE STORAGE
*       ASM42 PUTS TEMP2 AS OUT-OF-SEQUENCE INTO
*       LOC SPECIFIED BY 7 ADDRESS OF RA, RELOCATION
*       DIGIT FOR M BEING SPECIFIED IN REGISTER L.
*       ASM41,ASM4 HAVE CONTROL WORD IN REGISTER A,
*       INSTRUCTION WORD IN REGISTER X.
*
```



I. I95. ASSEMBLER 5
THIS SUBROUTINE IS THE SOLE COMMUNICATION BETWEEN THE COMPILER AND THE OUTPUT CARDS.

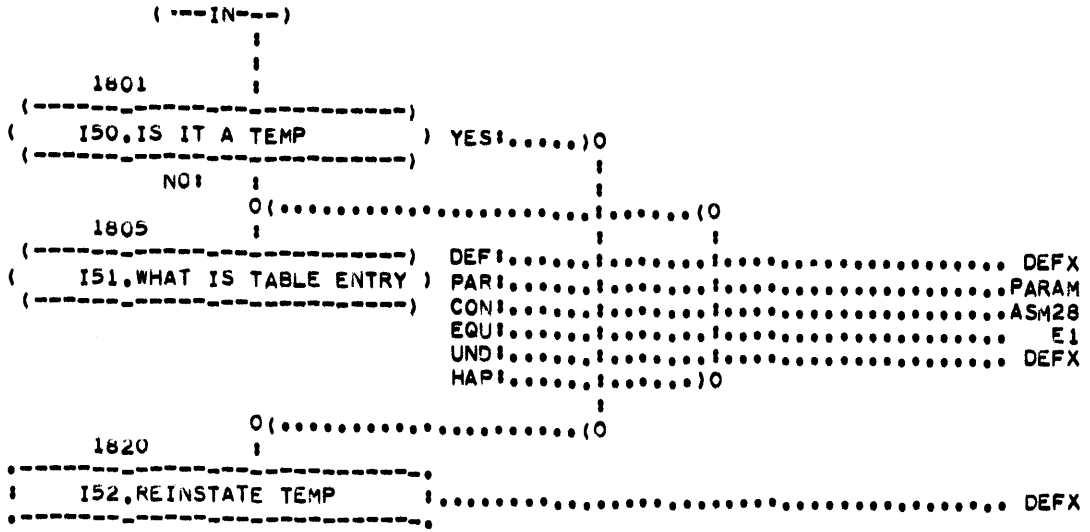
I95.STORE WORD
PUT THE OUTPUT WORD IN THE PUNCH INTERLACE.

I96.END OF CARD
IF THE CARD IS NOT FULL YET, EXIT.

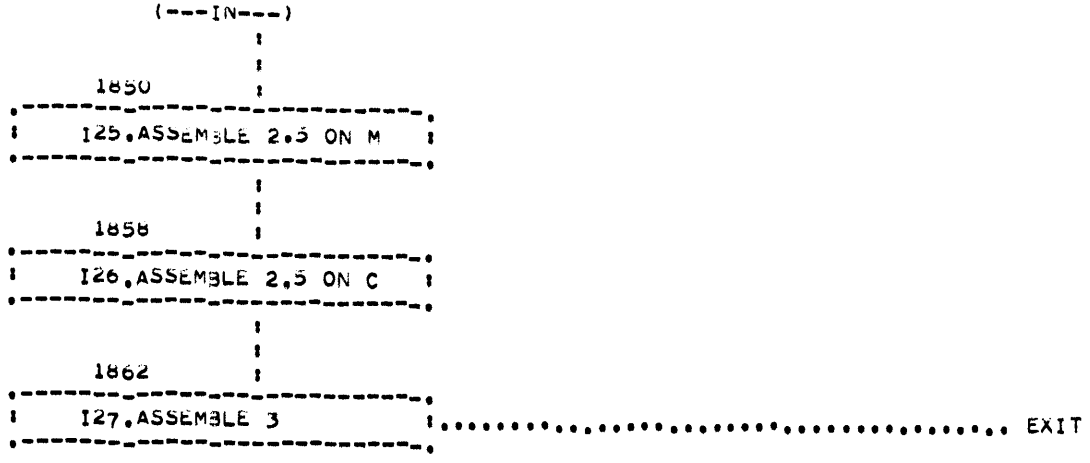
I97.CHECK CARD
UNLESS NO CARDS MODE IS IN EFFECT, UNLOAD THE BUFFER. THE THE 2ND READ STATEION IS NON-BLANK, SUM CHECK THE IMAGE AVAILABLE THERE. GIVE 1112 HALT IF THIS FAILS, AND DUMP HSR BUFFER.

I98.COMPUTE CHECK SUM.
COMPUTE SUM OF NUMERIC PORTIONS OF FIRST SEVEN WORDS, AND PLACE IN WORD 8 OF CARD.

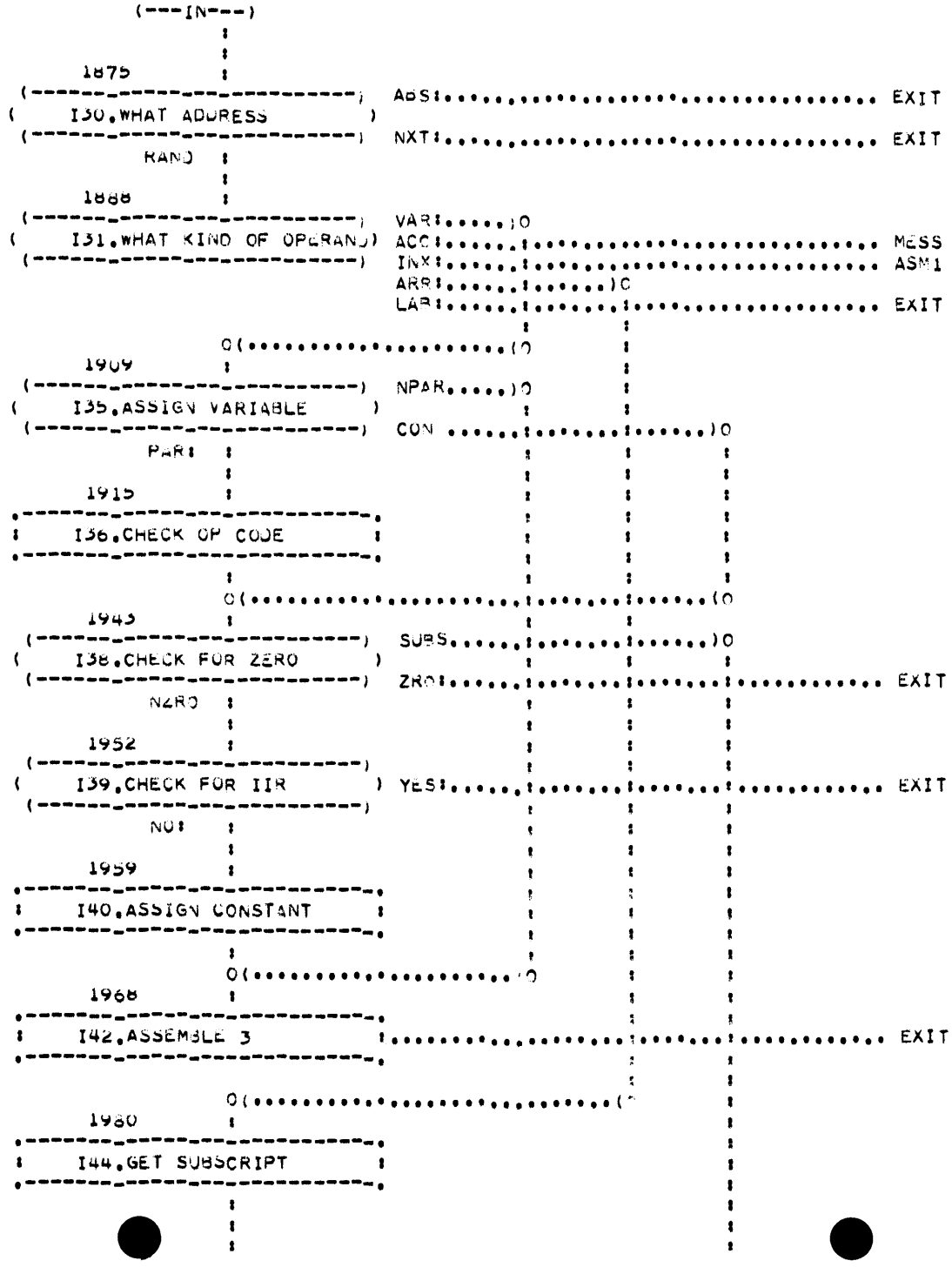
I99.PUNCH
PUNCH CARD, INCREASE SEQUENCE NUMBER, EXIT.



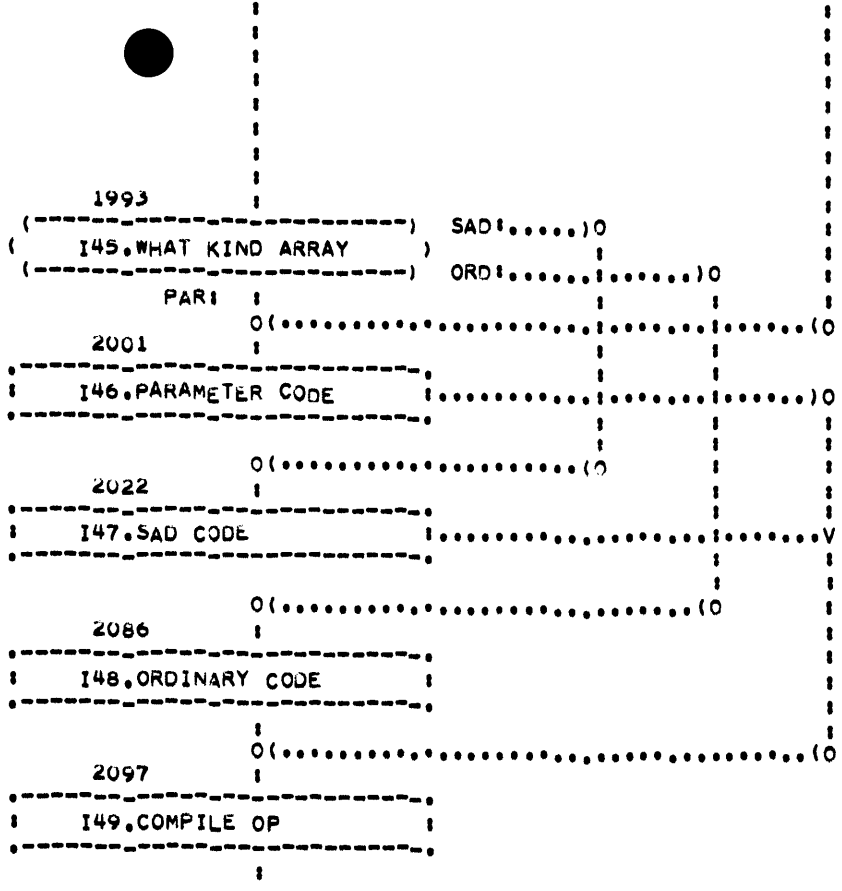

```
*   I.  I50.  ASIGN SUBROUTINE
*       THIS SUBROUTINE FINDS,OR MAKES,THE MEMORY
*       ASSIGNMENT FOR SIMPLE VARIABLES, ARRAYS, OR
*       TEMP STORAGES. IT IS NOT A TRUE SUBROUTINE,
*       FOR IF THE ITEM TURNS OUT TO BE A CONSTANT
*       OR HAPPY ARRY, IT JUMPS INTO THE MIDDLE OF
*       ASM28 ROUTINE.
*   I50.IS IT A TEMP
*       IF THE ITEM TO BE ASSIGNED IS A TEMP STORAGE,
*       GO TO I52.
*   I51.WHAT IS TABLE ENTRY
*       IF THE TABLE ENTRY INDICATES THIS ITEM IS
*       DEFINED IN UNIQUE OR COMMON,GOTO DEFX.
*       IF THE ITEM IS A PARAMETER,GO TO THE PARAMETE
*       EXIT. IF THE ITEM IS A CONSTANT, GO TO STEP
*       I38 IN ASM28,OR IF DOING A FUNCTION CALL GO
*       TO CASIN,STEP I70.
*       IF THE ITEM IS UNDEFINED AND EQUIVALENCED TO
*       OTHER ITEMS,GO TO E1.
*       IF THE ITEM IS UNDEFINED,NOT EQUIVALENCED,
*       ASSIGN IT IN UNIQUE STORAGE AND GO TO DEFX.
*       FINALLY IF THE ITEM IS A HAPPY ARRAY,ASSUME
*       WE HAVE BEEN CALLED BY ASM28,ADJUST OP CODE
*       FOR INDEXING IF NECESSARY,THEN CONVERT TO A
*       SIMPLE VARIABLE AND RECYCLE AT I51.
*   I52.REINSTATE TEMP
*       UNLESS PROCESSING A DO STATEMENT, THE TEMP
*       STORAGE LOCATION IS PUT BACK ON THE LIST OF
*       POTENTIAL TEMP STORAGES FOR FURTHER USE.
*       GO TO DEFX
*   CODING DETAILS:   RA IS THE OPERAND STACK ENTRY,
*       RL IS THE EXIT FOR A PARAMETER, RX IS THE
*       EXIT FOR A DEFINED NON-PARAMETER.
*
```



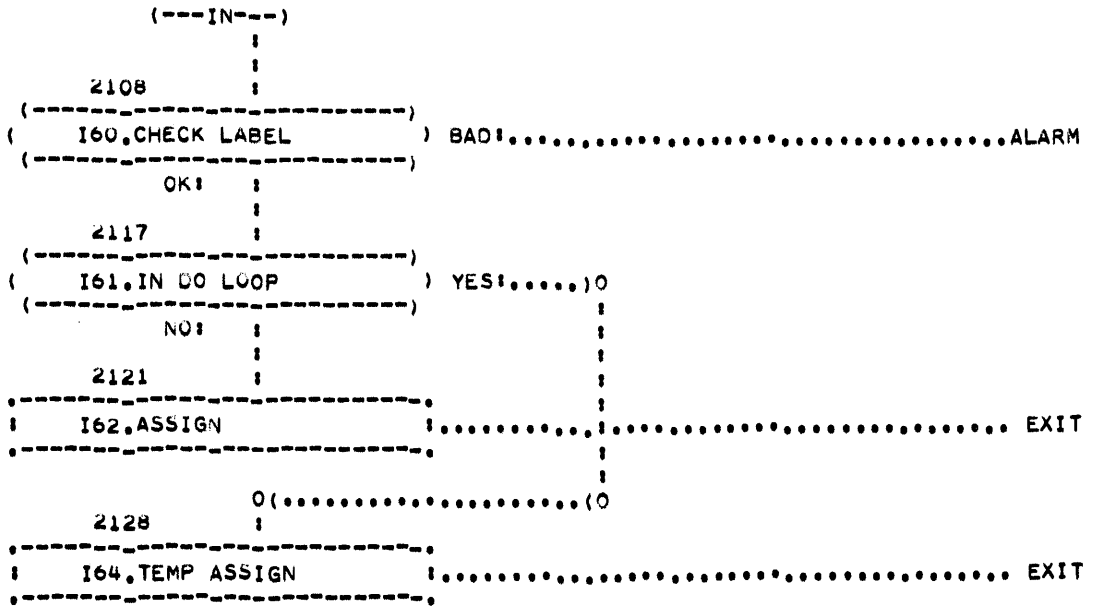

```
*   I.  I25. ASSEMBLER 2.
*       THIS SUBROUTINE ASSEMBLES MACHINE LANGUAGE
*       INSTRUCTIONS OF AN ALMOST SYMBOLIC NATURE;
*       THE OP-CODE IS THE TRUE OP BEFORE INDEXING,
*       AND THE ADDRESSES ARE EITHER ABSOLUTE,REFER
*       TO NEXT INSTRUCTION,OR REFER TO OPERANDS.
*       IN PARTICULAR,AN ARRAY OPERAND IS ALLOWED,
*       AND THIS MAY CAUSE MANY INSTRUCTIONS TO BE
*       GENERATED. IF THE OPERAND IS NOT A LABEL,
*       HOWEVER, THE ASSUMPTION IS MADE THAT IT GOES
*       IN M ADDRESS AND THAT C ADDRESS REFERS TO NXT
*   I25.ASSEMBLE 2.5 ON M
*       SEND THE M ADDRESS TO ASM2.5 FOR ASSEMBLY.
*       (IF IT IS AN OPERAND,WE WILL NEVER COME BACK
*       FROM ASM2.5,SEE THAT ROUTINE.)
*   I26.ASSEMBLE 2.5 ON C
*       SEND C ADDRESS TO ASM2.5 FOR ASSEMBLY.
*   I27.ASSEMBLE 3
*       SEND THE COMPILED INSTRUCTION TO ASM3 FOR
*       OUTPUT AND FINAL TOUCHES.  EXIT.
*   CODING DETAILS: ADDRESS 9999 MEANS NEXT. ADDRESS
*       99II MEANS OPERAND STACK + II. FOR EXAMPLE,
*       9901 IS THE TOP OF THE OPERAND STACK.  ADDRES
*       SES LESS THAN 9901 ARE ABSOLUTE.
*       AT INPUT RA IS A CODED INSTRUCTION,RL IS EXIT
*       LINE.
*
```



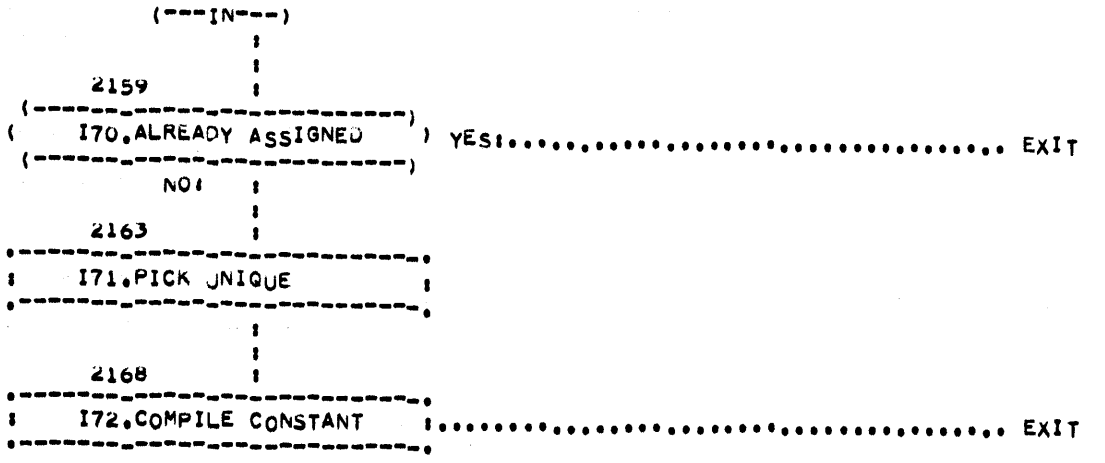
*
* I. I30. ASSEMBLERS 2.5 AND 2.8
* ASM2.5 DOES HALF THE JOB OF ASM2, Q.V.
* ASM2.8 IS USED FOR SIMPLE VARIABLES, TEMP
* STORAGES, AND SUBSCRIPTS.
* I30.WHAT ADDRESS
* IF THE ADDRESS TO BE ASSEMBLED IS ABSOLUTE,
* SET CORRESPONDING R-DIGIT ZERO AND EXIT.
* IF THE ADDRESS REFERS TO NEXT INSTRUCTION,
* TRANSMIT THIS INFORMATION AND TEMPORARILY SET
* THE ADDRESS ZERO. EXIT.
* OTHERWISE THE ADDRESS IS AN OPERAND
* AND FURTHER TESTS ARE NECESSARY.
* I31.WHAT KIND OF OPERAND
* FETCH THE OPERAND SPECIFIED AND CHECK TO SEE
* WHAT KIND IT IS.
* FOR A SIMPLE VARIABLE OR TEMP STORAGE, GO TO
* ASM28,STEP I35,AFTER WHICH WE EXIT FROM ASM2.
* FOR AN ACCUMULATOR SYMBOL THIS IS A BAD MESS.
* FOR AN INDEX VARIABLE,ASSUME WE WERE CALLED B
* BY ASSEMBLER 1 FOR A STORE OPERATION. TRANS-
* FER BACK TO ASM1 EMITTING THE INSTRUCTIONS
* TO LOAD RB1.
* FOR AN ARRAY VARIABLE GO TO STEP I44.
* FOR A LABEL,GO TO THE LABEL ASSIGN ROUTINE
* (I60) AND THEN EXIT.
* I35.ASSIGN VARIABLE
* GO TO ROUTINE I50 TO GET THE ASSIGNMENT FOR
* THIS SIMPLE VARIABLE.
* IF IT IS NOT A PARAMETER,GO TO STEP I42
* IF IT IS A CONSTANT,WE GET TO STEP I38.
* OTHERWISE IT'S A PARAMETER.
* I36.CHECK OP CODE
* FOR A SIMPLE VARIABLE PARAMETER, WE CHOOSE
* ONE OF THREE SUBROUTINES IN THE OBJECT CODE,
* DEPENDING WHETHER THE OP IS TO BE LDL, LDA,
* OR STL. FOR A STA WE DO ATL, STL. FOR OTHER
* OPERATIONS, WE DO LDL, OP RL.
* I38.CHECK FOR ZERO
* IF THE CONSTANT IS ZERO,AND IF THIS IS A
* ZERO SUBSCRIPT ON A PARAMETRIC ARRAY GO TO
* STEP I46. OTHERWISE FOR A ZERO CONSTANT, ADD
* ONE TO THE OP CODE AND SET 7 AND C TO NXT.
* GO TO ASM3 AND THEN EXIT FOR ASM2.
* I39.CHECK FOR IIR
* SEE IF THE OP IS LDA AND IF IT CAN BE CHANGED
* INTO IIR; IF SO, DO THIS AND EXIT.
* FROM ASM2 VIA ASM3.
* I40.ASSIGN CONSTANT
* USE THE CASIN ROUTINE (I70), DISPLAY THE
* COMMENT 'CONST'.
* I42.ASSEMBLE 3
* GIVE APPROPRIATE COMMENT, THEN EXIT FROM
* ASM2 VIA ASM3.
* I44.GET SUBSCRIPT
* IN AS42.5 WE HAVE AN ARRAY OPERAND.
* IF THE SUBSCRIPT IS NOT ALREADY IN REGISTER
* A, COMPILE CODE TO STORE A IN TEMP IF NEC-
* ESSARY, AND THEN TO LOAD A WITH THE SUBSCRIPT



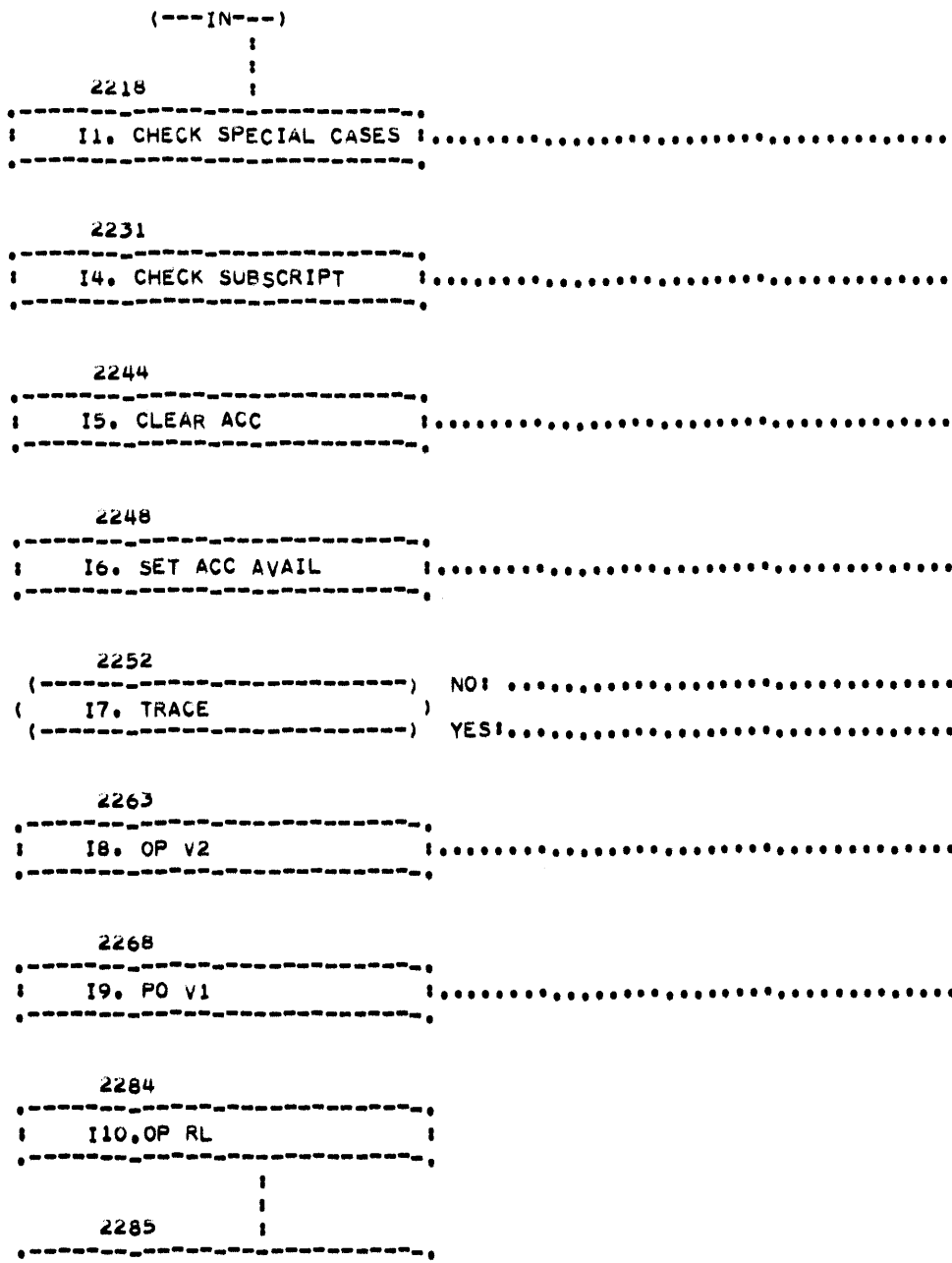

```
*
*
*
*
*
*
*
*
*       (USING ASM2.8, STEP I35).
*   I45.WHAT KIND ARRAY
*       (REFER TO STEP A24 WHERE THE VARIOUS CASES OF
*       ARRAY WERE DEFINED.) HAPPY ARRAYS DO NOT
*       COME THROUGH THIS PART,BUT WE MUST BRANCH
*       3 WAYS FOR THE OTHER TYPES OF ARRAYS:
*       SAD, GO TO I47.
*       ORDINARY, GO TO I48.
*             PARAMETRIC,GO TO NEXT STEP.
*   I46.PARAMETER CODE
*       IF CORE MODE IS ON, COMPILE
*           ADD 1F,LDX RA,LDA PAR,ADD RX RA.
*       ELSE COMPILE ADD PAR (OR LDA PAR IF SUBSCRIPT
*       IS ZERO), ADD NXT RA.  GO TO I49.
*   I47.SAD CODE
*       IF CORE MODE IS ON, COMPILE ADD 1F,
*       LDX RA, IIR ORELATIVE, ADD RX RA.
*       ELSE COMPILE ADD FUDGE, ADD NXT RA.
*       GO TO I49.
*   I48.ORDINARY CODE
*       COMPILE ADD NXT RA.
*   I49.COMPILE OP
*       NOW COMPILE THE ORIGINAL OP-CODE DESIRED FOR
*       THIS ARRAY OPERAND, PLUS 4 IF INDEXING IS
*       SPECIFIED. PUT NAME OF ARRAY AS COMMENT.
*   CODING DETAILS WILL BE OMITTED SINCE ASM2.5
*       AND ASM2.8 ARE ONLY FOR INTERNAL USE BY ASM2.
*
*
*
```



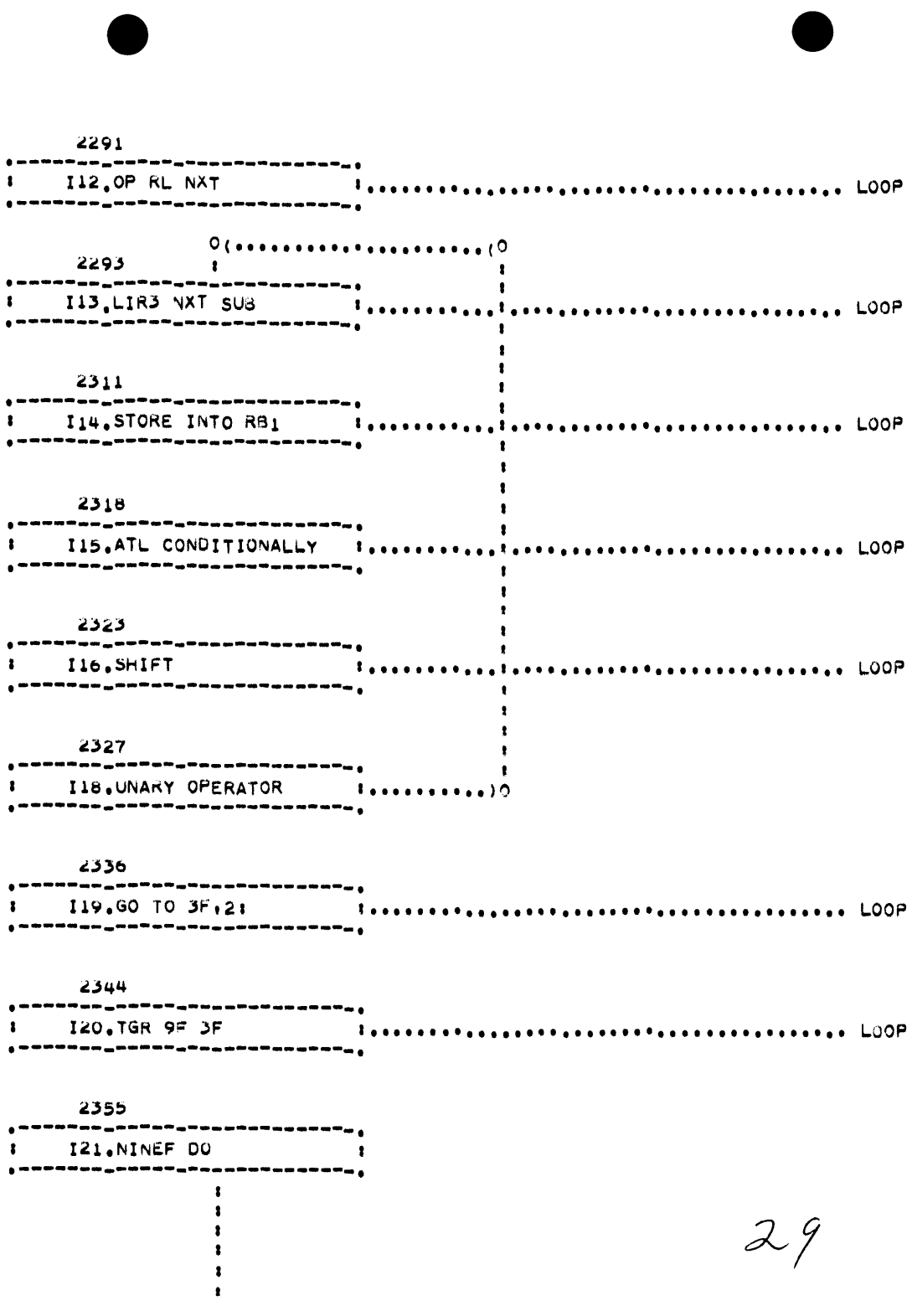

```
*   I.  I60. LSW  FOR ASSIGNING STATEMENT LABELS.
*       THIS ROUTINE HANDLES THE LOGIC FOR LABEL
*       ADDRESSES. THE PROBLEMS SOLVED ARE THOSE
*       OF FORWARD REFERENCES AND OF POTENTIAL GO TO
*       OUT OF DO LOOPS.
*   I60.CHECK LABEL
*       CHECK THAT THE OPERAND WHICH IS SUPPOSED TO
*       BE A LABEL IS ACTUALLY A STATEMENT NUMBER.
*       IF NOT, GIVE THE BAD LABEL ALARM.
*   I61.IN DO LOOP
*       IF WE ARE IN A DO LOOP GO TO STEP I64 UNLESS
*       WE WANT THE ABSOLUTE LOCATION OF THE LABEL
*   I62.ASSIGN
*       IF THE LABEL IS UNDEFINED,PICK LOCATION,
*       DEFINE IT, AND EXIT. IF THE LABEL IS TEMPO-
*       RARILY UNDEFINED (SEE BELOW),DO STEP I62 ON
*       THE AUXILIARY WORD. IF THE LABEL IS ALREADY
*       DEFINED, SIMPLY EXIT.
*   I64.TEMP ASSIGN
*       IN DO LOOP (AS OPPOSED TO DONT LOOP) WE MAKE
*       A TEMPO-A-Y ASS&GNMENT FOR THE LOCATION TO GO
*       TO,WHICH STORES RB1 BEFORE GOING TO THE
*       ACTUAL LOCATION. THE EXTRA INFORMATION IS
*       KEPT IN LLIST, IN THE FORM
*       SYMBOL TABLE ENTRY LLLL: 51AAAAXXXX
*                          AAAA:   02TTTT
*                          AAAA+1: XSSSSSLLLL
*       WHERE XSSSSS IS THE OLD SYMBOL TABLE ENTRY,
*       TTTT IS THE TEMPORARY ASSIGNMENT.
*       IN THIS STEP,WE CREATE THE LLIST ENTRY IF
*       NONE HAS BEEN MADE YET FOR THIS LABEL. OTHER-
*       WISE WE USE THE TEMPORARY ADDRESS. ALSO IF
*       THE  LABEL  HAD NO PERMANENT ADDRESS AND THE
*       LABEL HAS NOW OCCURRED IN COLS 1-5, WE SET
*       THE PERMANENT ADDRESS EQUAL TO THE TEMPORARY
*       ADDRESS.    EXIT.
*   CODING DETAILS:
*       ENTRANCE LSW IS USED FOR THE BRANCH ON DO
*       LOOP, ENTRANCE LSWOF IS USED FOR GETTING
*       ABSOLUTE LOCATIONS AS WITH A FORMAT OR
*       ASSIGN STTATEMENT.  RL IS THE EXIT LINE.
*       OUTPUT IS 02AAAA0000 IN REGISTER A.
*
```



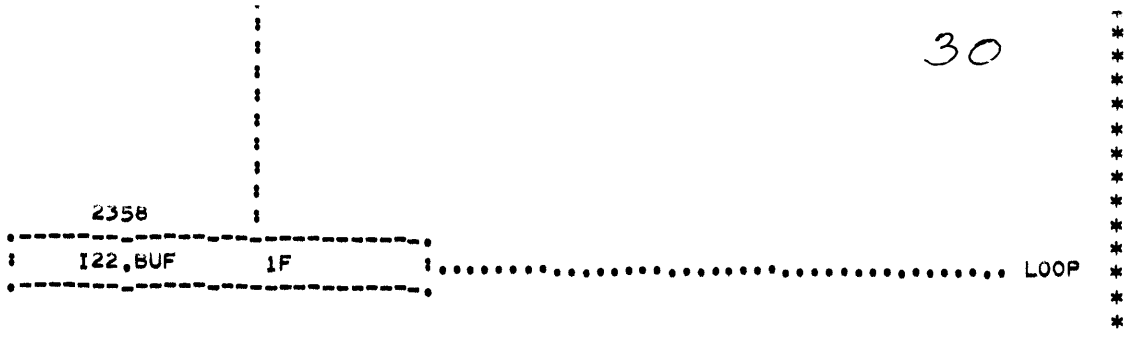

```
*   I.  I70. CASIN  ASSIGNING CONSTANTS.
*   I70.ALREADY ASSIGNED
*       IF THE CONSTANT HAS ALREADY BEEN ASSIGNED,
*       OUTPUT THE ASSIGNMENT. EXIT.
*   I71.PICK UNIQUE
*       PICK THE NEXT LOCATION IN UNIQUE STORAGE FOR
*       THIS CONSTANT
*   I72.COMPILE CONSTANT
*       OUTPUT THE CONSTANT OUT-OF-SEQUENCE USING ASS
*       EMBLER 4(I90).  EXIT.
*   CODING DETAILS:RX IS EXIT LINE, RB1 IS SYMBOL
*       TABLE REFERENCE. OUTPUT IS 01AAAA0000 IN RL.
*
*
*
*
*
*
*
*
*
```

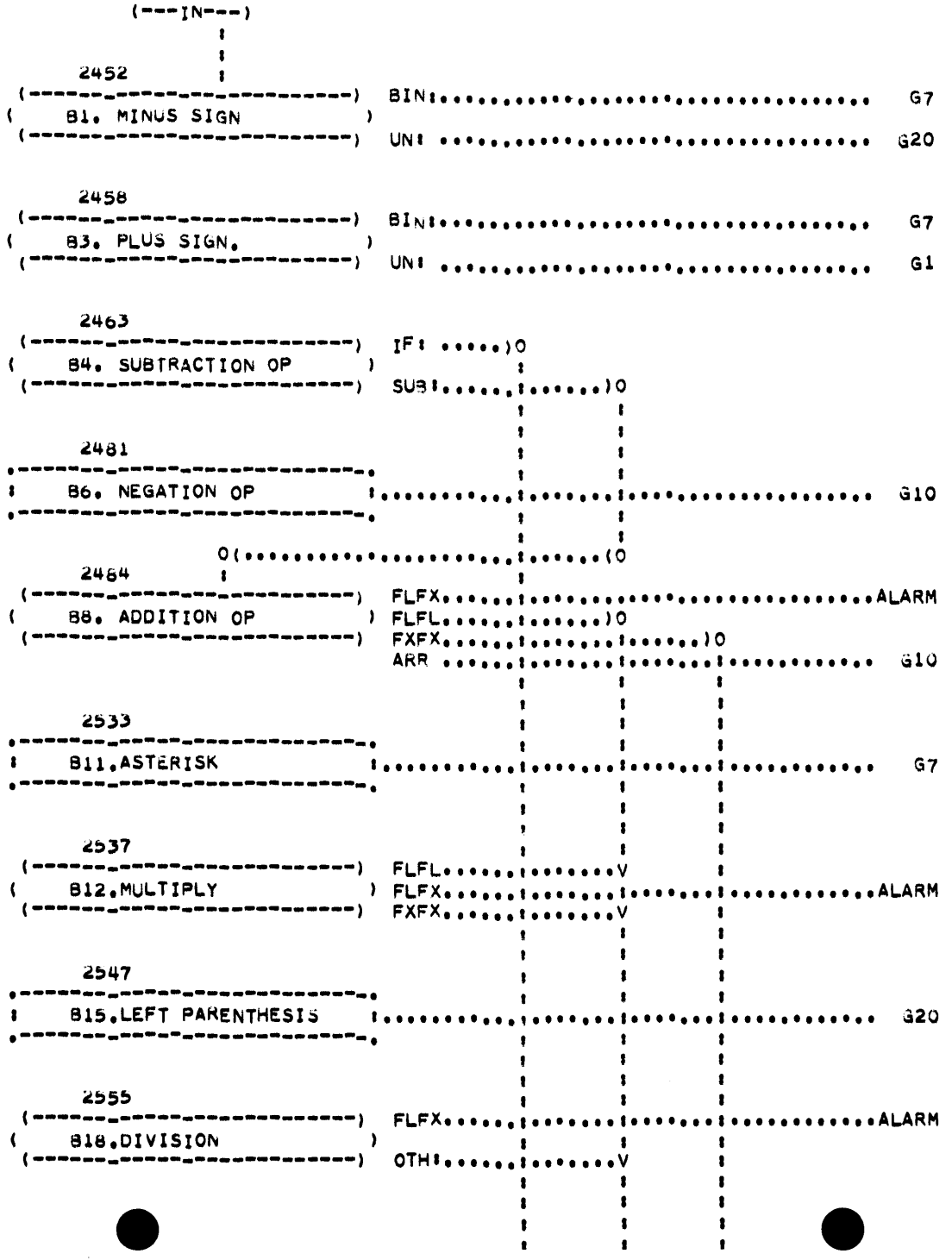


* I. I1. ASSEMBLR 1
* THIS IS A MACRO-ASSEMBLR
* WHICH IS GIVEN A LIST OF TWO-DIGIT INSTRUC-
* TION NUMBERS. THESE NUMBERS ARE EITHER
* REFERENCES TO A LIST OF STANDARD INSTRUCTIONS
* WHICH ARE PROCESSED BY ASSEMBLER 2, OR THEY A
* ARE REFERENCES TO PSEUDO-INSTRUCTIONS NUMBER
* I2 THRU I23. THE PSEUDO-INSTRUCTIONS ARE
* GIVEN HER IN THIS SECTION. THE PURPOSE OF
* ASM1 IS TO STEP THROUGH ALL 2-DIGIT
* CODES, AS AN INTERPRETIVE ROUTINE.
* 'LOOP' REPRESENTS THE PLACE TO RETURN TO
* STEP TO THE NEXT 2-DIGIT CODE.
* I1. CHECK SPECIAL CASES
* I1,I2,AND I3 ARE USED TO PROVIDE SLIGHTLY
* BETTER CODE FOR CERTAIN BINARY OPERATORS
* OR FOR IF-STATEMENTS WITH LABELS EQUAL, BY
* CHANGING THE ORDER OF OPERATION. LOOP.
* I4. CHECK SUBSCRIPT
* WHEN A BINARY OPERATION BETWEEN TWO ARRAY
* VARIABLES IS USED, A TEST IS MADE HERE TO
* SEE WHETHER EITHER SUBSCRIPT IS ALREADY IN
* THE ACCUMULATOR, FOR EFFICIENCY. LOOP.
* I5. CLEAR ACC
* IF THE ACCUMULATOR IN THE OBJECT PROGRAM
* IS IN USE, COMPILE THE INSTRUCTION
* 'STA TEMP.' LOOP.
* I6. SET ACC AVAIL
* THE ACCUMULATOR IS SET AVAILABLE, SINCE THE
* PREVIOUSLY COMPUTED RESULT IS TO BE USED
* NEXT. LOOP.
* I7. TRACE
* IF TRACE MODE IS NOT ON, EXIT FROM ASM1.
* OTHERWISE PREPARE THE INSTRUCTION LDX NAME
* PREPARATORY TO TRACING. LOOP.
* I8. OP V2
* EITHER OP V2 NXT OR
* LDL V2,OP RL IS COMPILED,
* 'WE ARE WORKING ON THE BINARY OPERATION
* V1 OP V2) THEN LOOP.
* I9. PO V1
* NOTE: V1 OP V2 EQUALS V2 PO V1
* EITHER PO V1 NXT OR
* LDL V1, PO RL IS COMPILED. LOOP.
* I10.OP RL
* I11.PO RL
* THIS PSEUDO OP IS USED TO SELECT ONE OF TWO
* ALTERNATIVES, WHICH ARE GIVEN AS PARAMETERS
* TO ASM1, DEPENDING ON WHICH OPERAND IS IN
* RL AND WHICH IS IN RA. LOOP.
* I12.OP RL NXT
* COMPILE OP RL NXT. LOOP.
* I13.LIR3 NXT SUB
* THIS COMPILES THE LINKAGE TO SUBROUTINES,
* INCLUDING THE CONTROL INFORMATION TO BRING A
* NEW SUBROUTINE IF THE SUBROUTINE HAS NOT

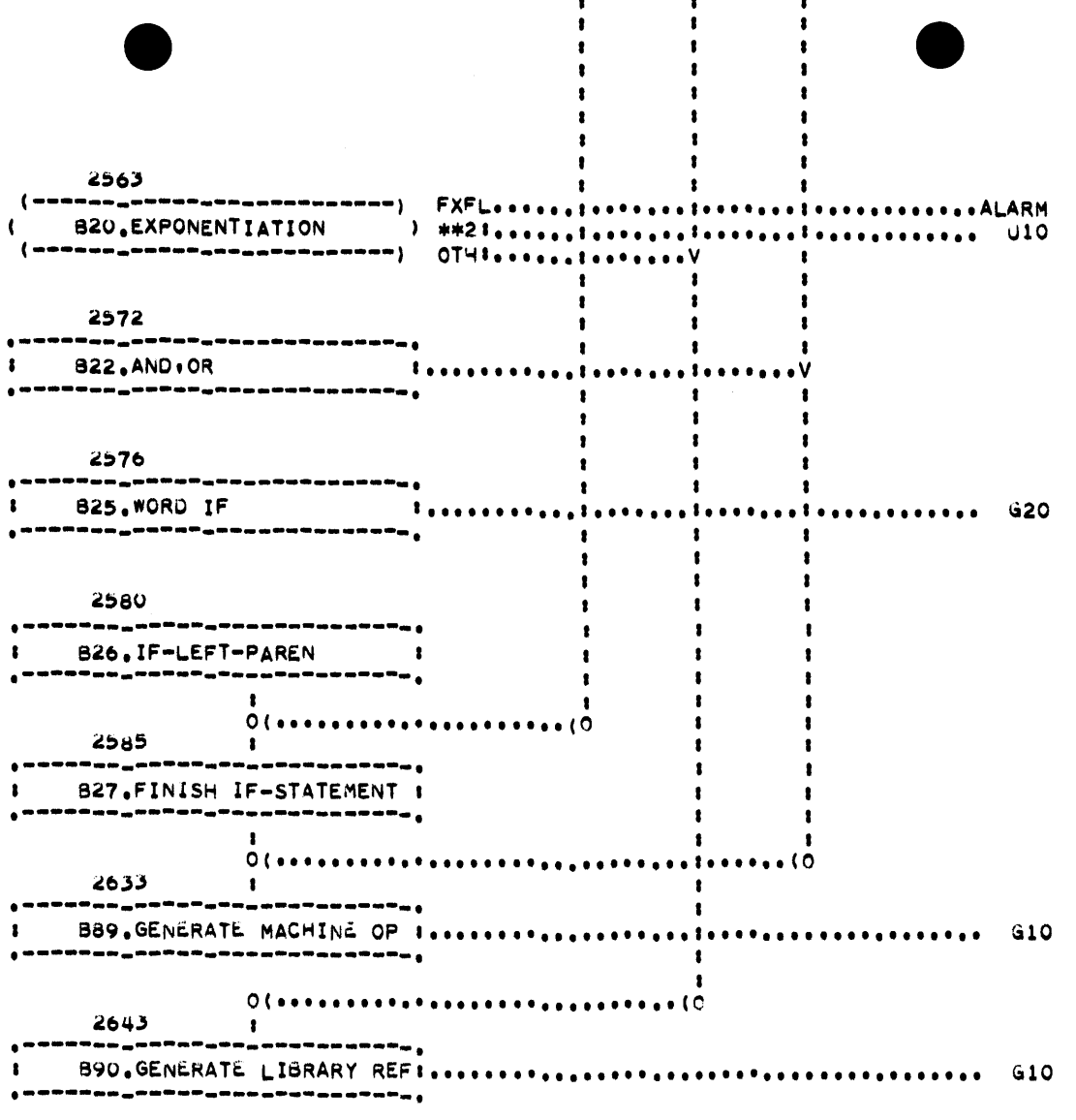



```
*       BEEN USED BEFORE.  LOOP.
*   I14.STORE INTO R91
*       COMPILE  LIR1 0000 NXT, WITH THE DO VARIABLE
*       AS COMMENT.   LOOP.
*   I15.ATL CONDITIONALLY
*       COMPILE  ATL 0000 NXT  UNLESS THE PRECEDING
*       INSTRUCTION IN SEQUENCE WAS AN LIR3 (IN WHICH
*       CASE THE ANSWER IS ALREADY IN RL).  LOOP.
*   I16.SHIFT
*       I16 AND I17 ARE USED FOR SHIFT COMMANDS
*       WHEN COMPILING CODE TO MULTIPLY BY POWERS
*       OF 10.   LOOP.
*   I18.UNARY OPERATOR
*       DEPENDING ON THE UNARY OPERATOR, THE
*       SUBROUTINE REFERENCE IS COMPILED USING I13.
*   I19.GO TO 3F,2
*       USED FOR MAKING FORWARD REFERENCES AT
*       THE BEGINNING OF DO LOOPS AND IN INPUT-
*       OUTPUT LISTS.   LOOP.
*   I20.TGR 9F 3F
*       THIS IS FOR THE TRANSFER INSTRUCTION FOR
*       EXITING FROM DO LOOPS.   LOOP.
*   I21.NINEF DO
*       THIS IS SIMPLY USED TO MARK THIS AS A DO
*       RATHER THAN A DONT LOOP.
*   I22.BUF      1F
*       USED AT BEGINNING OF FUNCTION OR SUBROUTINE.
*        LOOP.
```

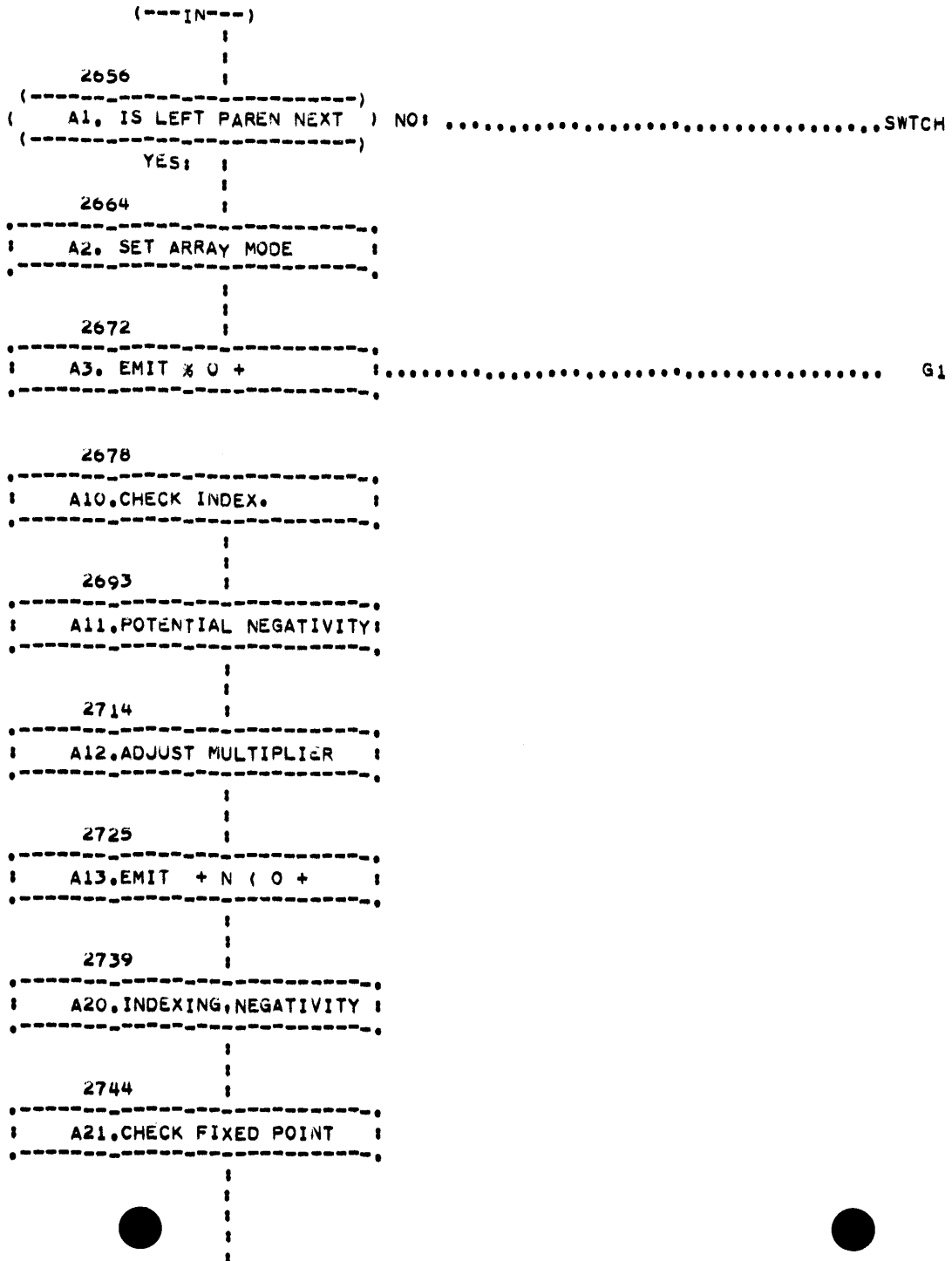

30
2358
I22,8UF
1F
LOOP




```
*  8.  ARITHMETIC OPERATORS
*      THIS SECTION CONTAINS THE GENERATORS FOR
*      ARITHMETIC OPERATORS, ENTERED FROM STEP G6
*      OR FROM STEP G10.  AN ODD NUMBERED STEP HERE
*      INDICATES AN ENTRY FROM STEP G6 (WHEN SYMBOL
*      IS FIRST SENSED) AND AN EVEN NUMBERED STEP
*      IN THIS SECTION INDICATES AN ENTRY FROM
*      G10 ( OFF THE OPERATOR STACK ).
*  B1. MINUS SIGN
*      CHECK IF THE PRECEDING ITEM WAS AN OPERAND OR
*      RIGHT PARENTHESIS. IF SO, A BINARY MINUS
*      OPERATOR IS SUBSTITUTED AND WE GO TO STEP G7.
*      IF NOT, A UNARY MINUS OPERATOR IS SUBSTITUTED
*      AND WE GO TO STEP G20.
*  B3. PLUS SIGN.
*      CHECK AS IN STEP B1 FOR UNARY OR BINARY.
*      ON BINARY PLUS, CHANGE TO THE BINARY ADD
*      OPERATOR AND GO TO STEP G7.
*      A UNARY PLUS IS IGNORED. GO TO G1.
*  B4. SUBTRACTION OP
*      CHECK IF THE OPERATOR IN OHOLD IS A RIGHT
*      PARENTHESIS AND IF THE TOP OF THE OPERATOR
*      STACK IS LEFT PARENTHESIS AFTER AN IF. IN
*      THIS CASE AND IF THE SIGNS OF THE TOP TWO
*      OPERANDS ARE EQUAL, SUBTRACTION IS NOT
*      CARRIED OUT, THE IF OPERATOR IS REMOVED
*      FROM THE STACK AND WE GO TO STEP  B27.
*      OTHERWISE NEGATE THE TOP OPERAND AND
*      CHANGE TO BINARY PLUS, STEP B8.
*  B6. NEGATION OP
*      CHANGE SIGN OF TOP OPERAND, EXIT TO G10.
*  B8. ADDITION OP
*      CHECK TYPES OF OPERANDS. IF THEY ARE MIXED
*      GIVE AN ERROR ALARM.
*      IF BOTH ARE FLOATING POINT,GO TO B90.
*      IF FIXED POINT,CHECK IF WE ARE ADDING A
*      CONSTANT IN AN ARRAY SUBSCRIPT. IF NOT,
*      GO TO B89. HOWEVER IF WE ARE ADDING  0 + V
*      THE ADDITION IS SUPPRESSED.
*      IN THE ARRAY SUBSCRIPT CASE, RECORD IF THE
*      CONSTANT IS GREATER THAN +1. MULTIPLY THE
*      CONSTANT BY THE APPROPRIATE DIMENSION
*      AND ADD THIS TO THE BASE. EXIT TO G10.
* B11.ASTERISK
*      CHECK FOR SECOND ASTERISK AND CHANGE TO A
*      MULTIPLY OR POWER OPERATOR; GO TO G7.
* B12.MULTIPLY
*      IF FLOAT-FLOAT, GO TO B90.
*      IF MIXED TYPE, GIVE ERROR ALARM.
*      IF FIX-FIX, SET 0*V EQUAL TO 0. IN OTHER
*      CASES, WE MAY CHECK FOR POSSIBILITY OF
*      ADD RA  AND/OR  SHIFT COMMNDS TO IMPLEMENT
*      MULTIPLICATION, ELSE GO TO B90.
* B15.LEFT PARENTHESIS
*      PUT A LEFT PARENTHESIS OPERATOR ON THE STACK,
*      STEP G20. WHEN IT COMES OFF THE STACK IT WILL
*      NECESSARILY BE FORCED OFF BY ITS MATCHING
```




```
*
*
*
*
*
*
*
*
*       RIGHT PARENTHESIS AND IN THIS CASE
*       WE WILL SIMPLY GO TO STEP G1.
*    818.DIVISION
*       GIVE ERROR ALARM IF MIXED TYPE, OTHERWISE
*       GO TO B90.
*    820.EXPONENTIATION
*       GIVE ERROR ALARM IF FIX**FLOAT
*       IF RAISING TO THE SECOND POWER, GO TO THE
*       UNARY SQUARING OPERATOR,STEP U10.
*       OTHERWISE GO TO B90.
*    822.AND,OR
*       FOR BOOLEAN AND,OR WE SET THE TYPE OF THE
*       RESULT TO UNSPECIFIED, THEN GO TO B89.
*    825.WORD IF
*       CHECK THAT A LEFT PARENTHESIS FOLLOWS,ELSE
*       GIVE AN ERROR ALARM, PUT A SPECIAL IF-LEFT-
*       PARENTHESIS ON THE STACK,AT STEP G20.
*    826.IF-LEFT-PAREN
*       AT THIS POINT WE HVE PROCESSED THE EXPRESSION
*       IN AN IF-STATEMENT AND MUST COMPARE IT
*       AGAINST ZERO.THEREFORE THE CONSTANT ZERO IS
*       PUT ON TOP OF THE OPERAND STACK.
*    827.FINISH IF-STATEMENT
*       SET UP IF MODE, THEN PROCESS THE STATEMENT
*       NUMBERS. CHECK THAT THERE ARE EXACTLY THREE,
*       THEN CHOOSE THE BEST CODING SEQUENCE BASED
*       ON EQUALITIES BETWEEN THESE.
*    889.GENERATE MACHINE OP
*       GENERATE CODING FOR THE MACHINE OPS
*       ADD,SUB,ERS, OR BUF, USING ASSEMBLER
*       1 (ROUTINE I), AND USING ONE OF 16 TABLE
*       ENTRIES DEPENDING ON WHETHER THE OPERANDS
*       ARE      0 SIMPLE VARIABLES, ETC.
*                1 IN THE ACCUMULATOR
*                2 INDEX REGISTER 1
*                3 ARRAY VARIABLES
*       EXIT TO G10.
*    890.GENERATE LIBRARY REF
*       GENERATE A REFERENCE TO A BINARY LIBRARY
*       SUBROUTINE. THERE ARE 8 CASES DEPENDING
*       ON WHETHER EITHER OPERAND IS NEGATED, AND
*       DEPENDING WHICH OPERAND WAS MOST CONVENIENT
*       TO PLACE IN REGISTER L.  THESE CASES ARE
*       SELECTED BY REFERENCING A TABLE ENTRY
*       AS IN STEP B89 AND GOING TO ASSEMBLER 1
*       (ROUTINE I). EXIT TO G10.
*
```




```
*   A.  PROCESSING OF ARRAY SUBSCRIPTS
*       WHEN A DIMENSIONED VARIABLE IS SENT FROM THE
*       SCANNER, ENTRY IS MADE TO A1.  A COMMA
*       BETWEEN SUBSCRIPTS CAUSES ENTRY TO A10.
*   A1. IS LEFT PAREN NEXT
*       SCAN NEXT ITEM. IF IT IS NOT A LEFT
*       PARENTHESIS,GO TO THE UNDIMENSIONED ARRAY
*        SWTCH. THIS SWITCH IS NORMALLY SET TO THE
*       'MISSING LEFT PARENTHESIS' ALARM WHICH
*       INSERTS A LEFT PARENTHESIS INTO THE
*       STATEMENT AND RETURNS HERE.
*   A2. SET ARRAY MODE
*       THE MODE STACK RECEIVES FOUR NEW ENTRIES:
*       2 0000  ARRAY MODE (A10 FOR COMMA,
*                           MISSING RIGHT PAREN FOR $)
*       2 9999  BASE CALCULATION
*       2 9998  CURRENT PRODUCT OF DIMENSIONS
*       2 9997  REFERENCE TO DIMENSION LIST
*   A3. EMIT % 0 +
*       FOR CONVENIENCE, THE CHARACTER ( 0 + ARE
*       INSERTED. THIS LEFT PARENTHESIS IS A SPECIAL
*       ONE WHICH SENDS CONTROL TO STEP A20 WHEN
*       THE MATCHING RIGHT PARENTHESIS COMES ALONG.
*       EXIT TO G1.
*   A10.CHECK INDEX.
*       IF THIS IS THE FIRST SUBSCRIPT AND ITS
*       CURRENT VALUE IS RB1 CODE, INDEXING IS SET
*       UP AND THE SUBSCRIPT IS REPLACED BY ZERO.
*       THIS OCCURS ONLY IF THE FIRST SUBSCRIPT IS
*       DOVAR & CONSTANT, WHERE THE CONSTANT IS
*       GREATER THAN -30, AND IF WE ARE NOT CALLING
*       A FUNCTION.
*       THE IMPORTANT ASSUMPTION IS MADE HARE THAT
*       NEITHER UNIQUE NOR COMMON STORAGE WILL BE
*       ASSIGNED TO CORE LOCATIONS B000 - B029.
*       WITH THIS CONVENTION, THE NUMBER OF SAD
*       ARRAYS(SEE SECTION A24) IS GREATLY REDUCED.
*   A11.POTENTIAL NEGATIVITY
*       IF ANY CONSTANTS GREATER THAN 1 OCCURRED
*       DURING THE LAST SUBSCRIPT ALONG WITH
*       ANYTHING OTHER THAN DOVAR, THIS ARRAY IS
*       MARKED AS HAVING A POTENTIALLY NEGATIVE
*       SUBSCRIPT.
*   A12.ADJUST MULTIPLIER
*       IF THERE ARE NO MORE DIMENSIONS, THE EXTRA
*       SUBSCRIPT ALARM IS GIVEN, ELSE IT IS
*       MULTIPLIED TO GIVE THE CURRENT PRODUCT OF
*       DIMENSIONS.
*   A13.EMIT  + N ( 0 +
*       FOR CONVENIENCE, THE COMMA IS TRANSFORMED
*       INTO THE CHARACTERS +N(0+    THIS LEFT
*       PARENTHESIS IS LIKE A MULTIPLICATION SYMBOL,
*       ONLY THE CHECK AT STEP A11 IS MADE FIRST.
*   A20.INDEXING,NEGATIVITY
*       WE HAVE NOW SCANNED THE ENTIRE SUBSCRIPT
*       OF THE ARRAY. STEPS A10 AND A11 ARE PERFORMED
```

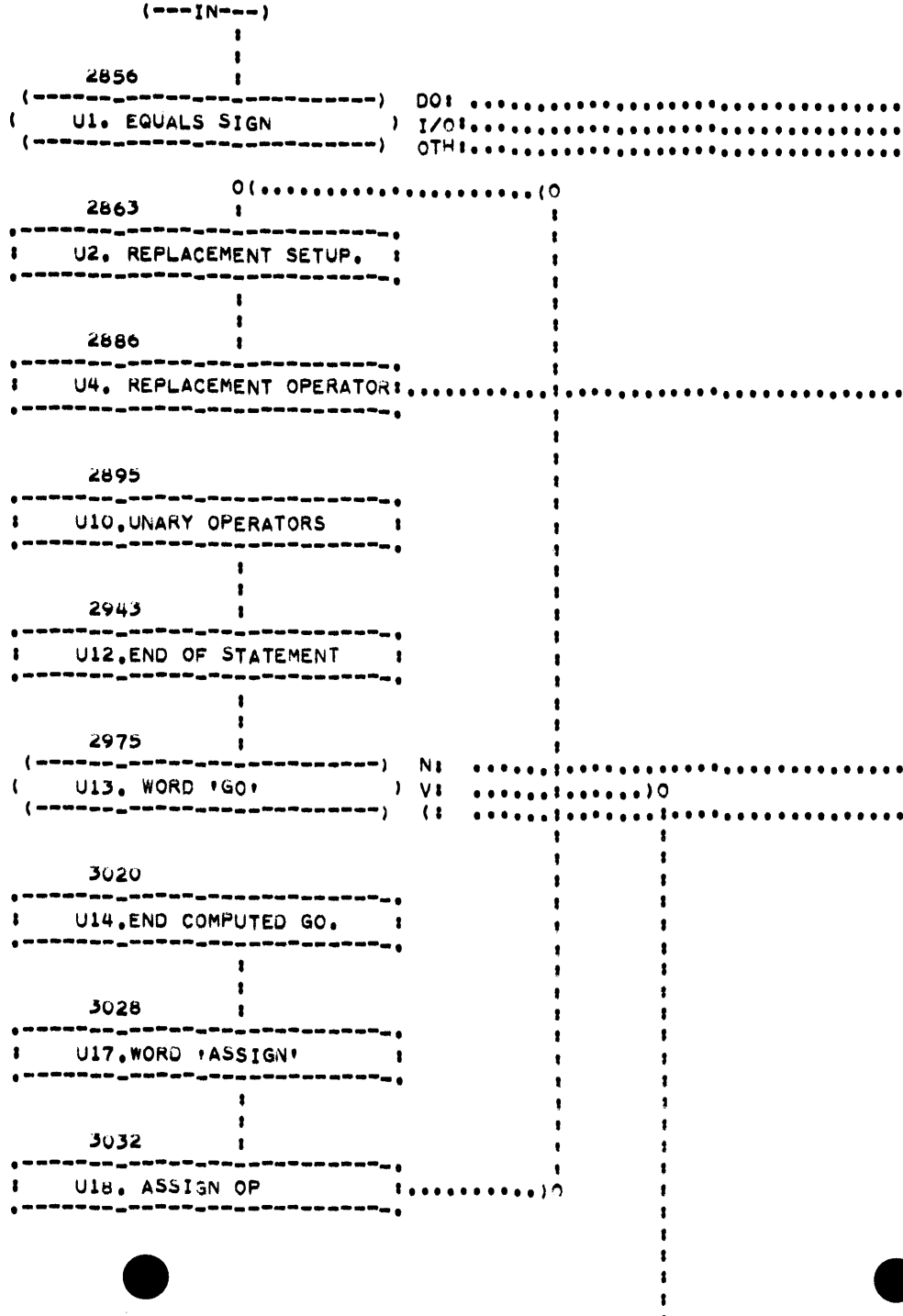
```
                    :
                    :
        2747        :
 .------------------------------.
 :   A22.COMPUTE SUBSCRIPT      :
 .------------------------------.
                    :
                    :
        2756        :
  (-----------------------------)
 (   A23.EQUIVALENCE DECL.       ) YES:..................................  E30
  (-----------------------------)
            NO:     :
                    :
        2760        :
  (-----------------------------)  PAR:.....)O
 (   A24.WHAT TYPE ARRAY         ) SAD:.....:......)O
  (-----------------------------)  HAP:.....:........:......)O
           ORD:     :                       :        :      :
                    :                       :        :      :
        2785        :                       :        :      :
 .------------------------------.           :        :      :
 :   A25.CODE 3SLLLLI000        :...........:........:......:......)O
 .------------------------------.           :        :      :      :
                                            :        :      :      :
                    O(......................:......(O       :      :
        2790        :                       :               :      :
 .------------------------------.           :               :      :
 :   A26.CODE 2BBBBBS000        :...........:...............:......V
 .------------------------------.           :               :      :
                                            :               :      :
                    O(...................(O                 :      :
        2795        :                                       :      :
 .------------------------------.                           :      :
 :   A27.CODE 1BBBBBPPPP        :...........................:......V
 .------------------------------.                           :      :
                                                            :      :
                    O(...................................(O        :
        2805        :                                              :
 .------------------------------.                                  :
 :   A28.CODE AS SIMPLE VAR.    :                                  :
 .------------------------------.                                  :
                    :                                              :
                    O(...........................................(O
        2814        :
 .------------------------------.
 :   A29.MOVE SUBSCRIPT         :....................................  G1
 .------------------------------.
```

35

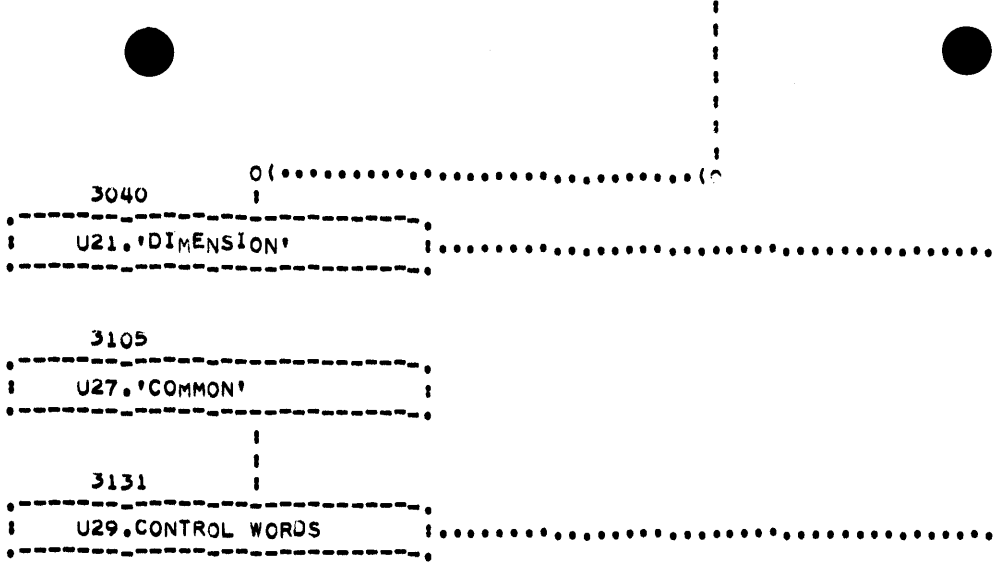
```
    THEY ARE REDUNDANT UNLESS THE ARRAY IS
    SINGLY SUBSCRIPTED.
A21.CHECK FIXED POINT
    IF SUBSCRIPT IS FLOATING, GIVE ERROR ALARM.
A22.COMPUTE SUBSCRIPT
    ITSELF, PRODUCE CODE TO LOAD IT WITH TRUE
    SIGN INTO THE ACCUMULATOR.
A23.EQUIVALENCE DECL.
    IF WE ARE IN AN EQUIVALENCE DECLARATION,
    EXIT TO THE EQUIVALENCE ROUTINE E30.
A24.WHAT TYPE ARRAY
    THERE ARE FOUR KINDS OF ARRAYS, AND WE DECIDE
    NOW WHAT KIND THIS IS.
    IF THE ARRAY IS PARAMETRIC, GO TO A27.
    CONSTANTS IN THE SUBSCRIPT ARE ADDED TO THE
    BASE. IF THE BASE HAS THEREBY BECOME NEGATIVE
    OR TOO LARGE OR IF THE SUBSCRIPT IS
    POTENTIALLY NEGATIVE, THIS IS CALLED A SAD
    ARRAY, AND WE GO TO A26.
    EXAMPLE: A(J+2), WHERE J MAY BE NEGATIVE
    IF THE SUBSCRIPT IS NOW ZERO, THIS INDICATES
    THAT IT WAS ALL CONSTANT EXCEPT PERHAPS FOR
    INDEX REGISTER MODIFICATION, SO GO TO A28.
             ELSE IT IS AN ORDINARY ARRAY:
A25.CODE 3SLLLLI000
    FOR AN ORDINARY ARRAY, THE CODE 3SLLLLI000
    IS SET UP,WHERE  S  IS THE STORAGE TYPE,
    LLLL IS THE BASE LOCATION, AND  I  IS 0 OR 4
    FOR INDEXING.  TO A29.
A26.CODE 2BBBBBS000
    FOR A SAD ARRAY,THE CODE 2BBBBBS000 IS SET
    UP, WHERE  S  IS THE STORAGE TYPE, BBBB IS
    THE BASE LOCATION PLUS 50000, PLUS 40000 IF
    INDEXING.  TO A29.
A27.CODE 1BBBBBPPPP
    FOR A PARAMETRIC ARRAY THE CODE 1BBBBBPPPP
    IS SET UP, WHERE BBBBB IS THE BASE LOCATION
    PLUS 50000, PLUS 40000 IF INDEXING, AND
    PPPP IS THE LOCATION OF THE PARAMETER. TO A29
A28.CODE AS SIMPLE VAR.
    THIS ARRAY IS CHANGED TO LOOK ALMOST LIKE
    A SIMPLE VARIABLE.
A29.MOVE SUBSCRIPT
    THE STATUS OF THINGS IS CHANGED TO:
    SAD:         PAR:         HAP:         ORD:
           OPERAND STACK ENTRY:
     3TAAAA0000 3TAAAA0000 0TAAAA0000 3TAAAA0000
           LOCATIONS AAAA AND AAAA+1:
     000000CCCC 000000CCCC 090I00CCCC 000000CCCC
     2BBBBBS000 1BBBBBPPPP 090I000000 3SLLLLI000
           LOCATIONS CCCC AND CCCC+1:
     SUBSCRIPT  SUBSCRIPT  0SLLLL**** SUBSCRIPT
     ZZZZZNNNNN ZZZZZNNNNN ZZZZZNNNNN ZZZZZNNNNN
    WHERE ZZZZZNNNNN IS THE ARRAY NAME, AND
```

```
*
*
*
*
*
*
*
*
*
*
```

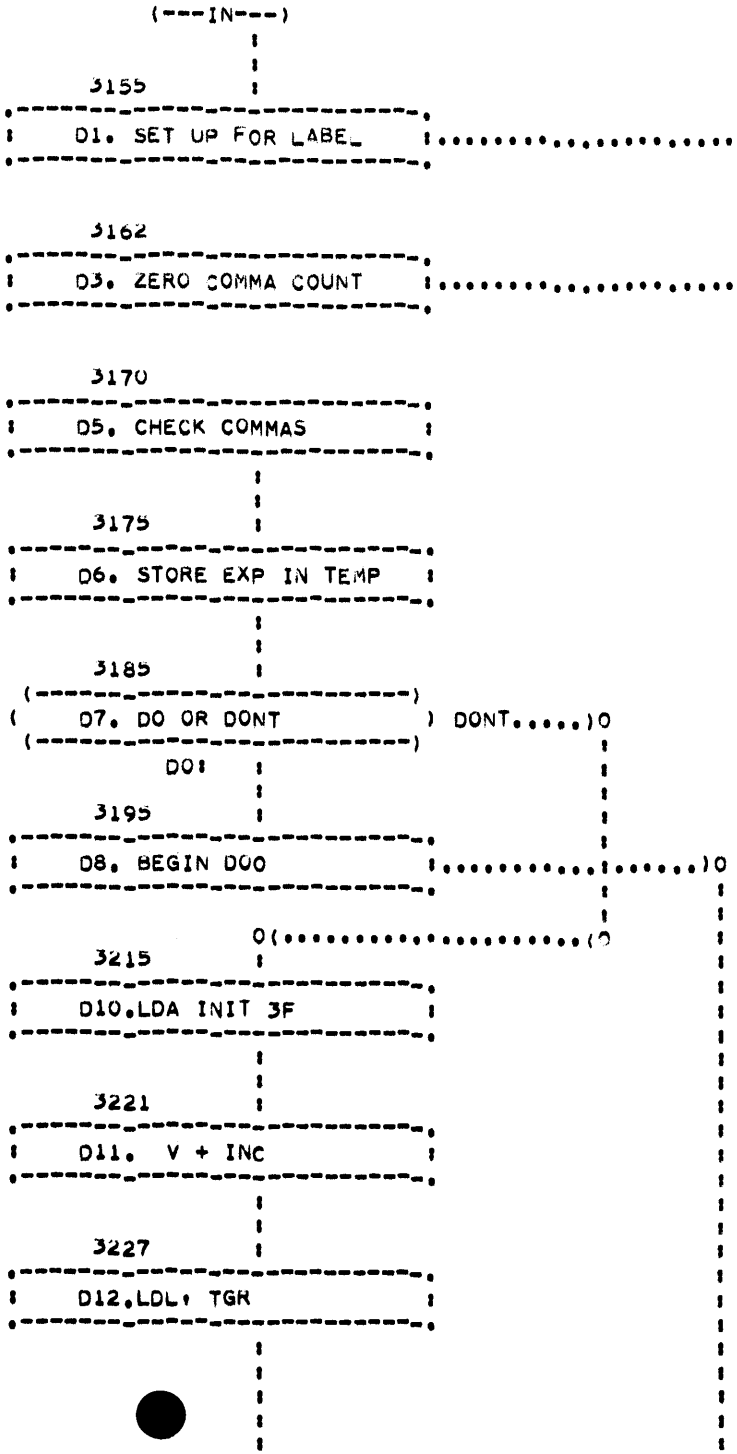
T INDICATES THE TYPE. EXIT TO G1.



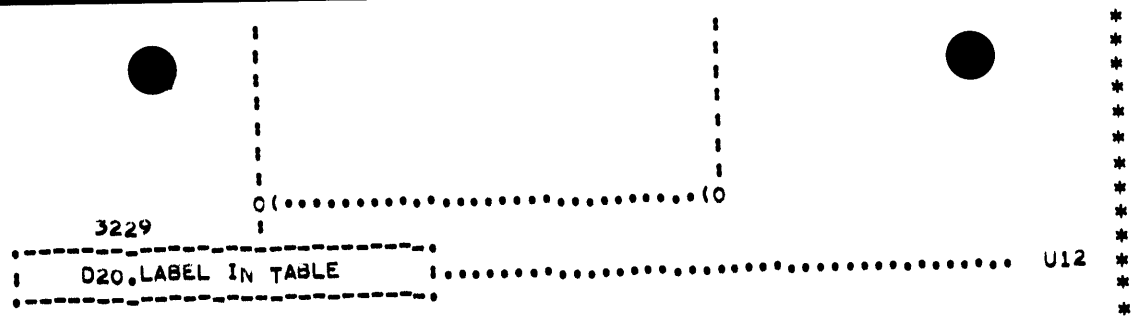
* U. UNARY OPERATORS AND SPECIAL GENERATORS
* COMPARE WITH THE INTRODUCTORY REMARKS OF
* SECTION B; ODD-NUMBERED STEPS INDICATE ENTRY
* FROM G6; EVEN NUMBERED, FROM G10.
* U1. EQUALS SIGN
* THIS IS A SWITCH WHICH IS SET IN SEVERAL
* PLACES. IF THIS EQUALS SIGN OCCURS IN A DO
* STATEMENT, GO TO D3. IF IT IS IN AN INPUT-
* OUTPUT STATEMENT, GO TO W17. OTHERWISE THIS
* IS A PLAIN OLD EQUALS SIGN, AND WE PUT A
* REPLACEMENT OPERATOR ON THE STACK, G20.
* U2. REPLACEMENT SETUP.
* IN A MULTIPLE ASSIGNMENT STATEMENT WE ENTER
* AT STEP U2 THE FIRST REPLACEMENT OPERATOR,
* STEP U4 SUCCEEDING TIMES. CHECK TYPES, AND
* IF DIFFERENCE IS PRESENT PUT OUT THE CODE TO
* FIX OR FLOAT. IF THE TYPES ARE THE SAME,
* DECIDE WHETHER TO PUT THE RIGHT-HAND SIDE IN
* REGISTER A OR NOT. REGISTER L IS SELECTED IF
* THERE IS A MULTIPLE ASSIGNMENT STATEMENT, OR
* IF THE LEFT-HAND PART IS NOT A SIMPLE VARIABL
* OR IF TRACE MODE IS ON.
* THE CODING TO PUT THE RIGHT-HAND SIDE, WITH
* TRUE SIGN, INTO THE SELECTED RIGISTER, IS
* ACCOMPLISHED BY SELECTING A
* TABLE ENTRY AND ACTIVATING ASM1(ROUTINE I).
* U4. REPLACEMENT OPERATOR
* PUT OUT CODING TO STORE A OR L IN THE
* APPROPRIATE LOCATION AND POSSIBLY TO
* CAUSE TRACING, USING ASSEMBLER 1(ROUTINE I).
* REMOVE OPERAND FROM STACK. EXIT TO G10.
* U10. UNARY OPERATORS
* IN THE CASE OF FIX, EXP, SIN, COS, TAN, ATAN, LN,
* SQRT, CHECK THAT THE ARGUMENT IS FLOATING
* POINT. SQUARING, THE 'NOT' OPERATOR, AND
* FLOAT PLUS THE ONES MENTIONED EARLIER ARE
* THAN CALLED FROM THE LIBRARY SUBROUTINES,
* USING A TABLE ENTRY AND ACTIVATING ASSEMBLER
* 1. THERE ARE TWO CASES, DEPENDING WHETHER THE
* ARGUMENT IS NEGATED OR NOT. IN THE CASE OF
* ABS, A SPECIAL TABLE ENTRY FOR AN OPEN
* SUBROUTINE IS USED.
* U12. END OF STATEMENT
* AT THE END OF MOST STATEMENTS WE CHECK THAT
* THE OPERATOR AND OPERAND STACKS ARE EMPTY,
* ELSE GIVE THE ERROR 'MISSING RIGHT PARENTHE-
* SIS' OR 'MISSING OPERAND' OR 'EXTRA OPERAND'.
* U13. WORD 'GO'
* SET LABEL CONTEXT ON, AND SCAN THE NEXT ITEM
* (ROUTINE S). THE WORD TO IS IGNORED BY
* FORTRAN. IF THE NEXT ITEM IS A LABEL, PUT IT
* IN A BLANK ADDRESS OF THE PRECEDING
* INSTRUCTION OR ELSE CREATE A JMP INSTRUCTION,
* THEN GO TO G1.
* IF THE NEXT ITEM IS A VARIABLE, COMPILE CODE
* TO STORE RB1 IF WE ARE IN A DO LOOP, THEN

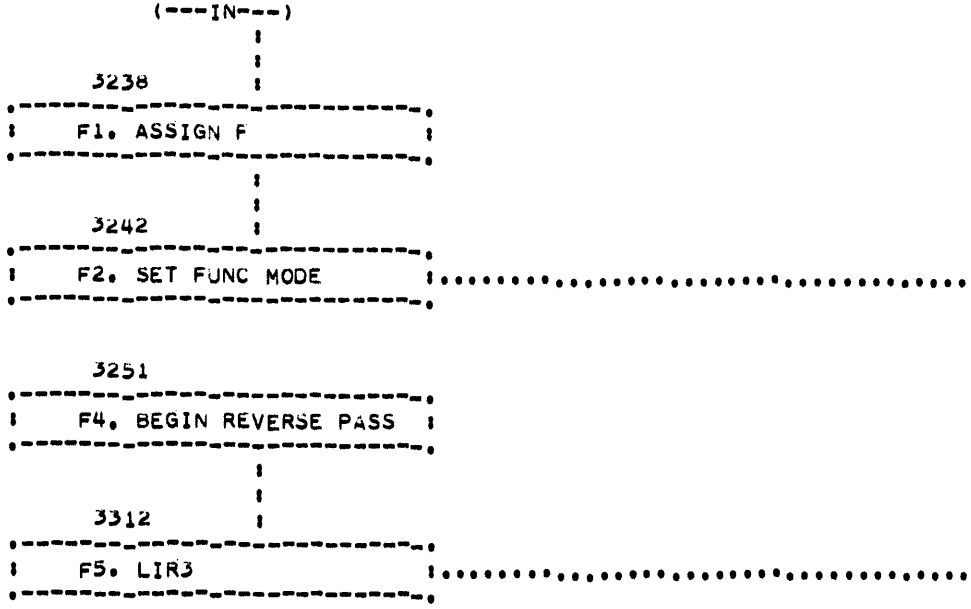



```
*
*
*
*
*
*
*
*
*       CODE TO JUMP TO THE VARIABLE ITSELF.  U21
*       FINALLY IF IT IS A LEFT PARENTHESIS,
*       WE SET UP GO MODE, COMPILE EACH LABEL
*       OUT OF SEQUENCE, THEN WHEN THE RIGHT
*       PARENTHESIS COMES ALONG WE RETURN TO G1 TO
*       PROCESS THE EXPRESSION.
*   U14.END COMPUTED GO.
*       COMPILE CODE TO GET THE EXPRESSION WITH
*       TRUE SIGN IS REGISTER A, THEN
*       ADD NXT RA,  JMP TO THE TABLE.
*   U17.WORD 'ASSIGN'
*       SET LABEL CONTEXT,AND PLACE THE ASSIGN
*       OPERATOR ON THE STACK. THE WORD 'TO'
*       IS IGNORED BY FORTRAN.
*   U18. ASSIGN OP
*       CREATE A CONSTANT FOR THE ABSOLUTE LOCATION
*       OF THE LABEL (USE I62), THEN INTERCHANGE
*       OERANDS AND TREAT ANALOGOUS TO REPLACEMENT
*       AT STEP U2.
*   U21.'DIMENSION'
*       WHEN A DIMENSION DECLARATION APPEARS THE REST
*       OF THE COMPILER IS RIGGED UP TO HANDLE THIS
*       STATEMENT PROPERLY BY STTING UP DIMENSION
*       MODE. WHEN A NAME COMES ALONG, A SECOND MODE
*       IS SET UP, AND THIS MODE CREATES THE
*       TABLE ENTRIES FOR AN ARRAY VARIABLE.
*       AT THE END, EXIT TO G1.  NO STORAGE
*       ASSIGNMENTS ARE MADE YET, THEY ARE MADE
*       WHEN THE ARRAY IS FIRST REFERENCED.
*   U27.'COMMON'
*       SET UP COMMON MODE, MARK EACH IDENTIFIER
*       THAT COMES ALONG AS COMMON AND ALLOCATE
*       THE STORAGE FOR IT.
*   U29.CONTROL WORDS
*       THE WORDS NO,TRACE,LISTYCORE,CARDS REALLY
*       NEVER GET PAST THE SCANNER, THEY ARE
*       DETECTED AT STEP S10. THEY MERELY SET
*       INTERNAL SWITCHES INSIDE THE COMPILER.
*       AND RUN OFF TO G1.
*
```

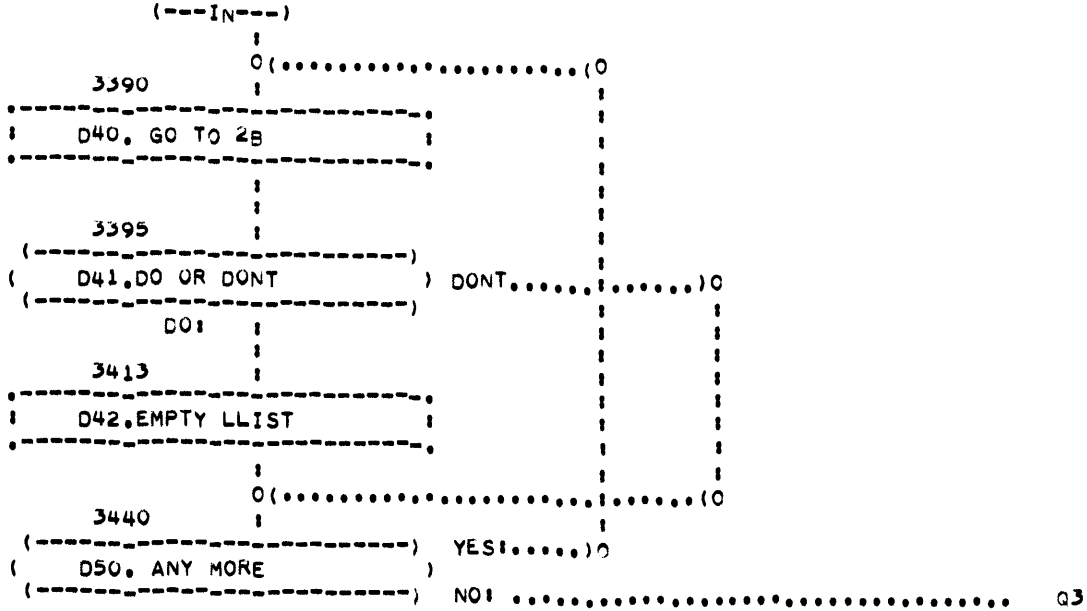



```
*   D.  DO LOOP CONTROL
*       WHEN THE WORD DO OR THROUGH IS SENSED, ENTRY
*       IS MADE TO STEP D1.
*   D1. SET UP FOR LABEL
*       DO MODE IS SET UP. A SWITCH IS SET SO THAT
*       WHEN THE NEXT EQUAL SIGN OCCURS,CONTROL GOES
*       TO STEP D3. SEMI-LABEL CONTEXT IS SET UP
*       SO THAT THE LABEL FOLLOWING COMES IN AS A
*       CONSTANT,YET STEP C5 GOES IMMEDIATELY TO C6
*       IN THE CONSTANT SCANNER. GO TO G1.
*   D3. ZERO COMMA COUNT
*       THE FACT THAT A COMMA MAY HAVE OCCURRED
*       BEFORE THE CONTROLLED VARIABLE IS FORGOTTEN.
*       AT THE END OF THIS STATEMENT,CONTROL WILL
*       PASS TO STEP D5. GO TO G1.
*   D5. CHECK COMMAS
*       IF LESS THAN TWO COMMAS HAVE OCCURRED,
*       INSERT  ',1'  IN THE PSEUDOCODE.
*   D6. STORE EXP IN TEMP
*       COMPILE CODE TO STORE REGISTER A IF THERE IS
*       A COMPUTED RESULT THERE. SET A SWITCH SO THAT
*       THE TEMP STORAGES USED TO HOLD COMPUTED
*       RESULTS ARE MADE PERMANENT STORAGES
*       (SEE STEP I52).
*   D7. DO OR DONT
*       THIS IS A DONT LOOP UNLESS:
*       A) THE WORD THROUGH WAS NOT USED
*       B) NO DO IS IN PROGRESS
*       C) BOTH THE STARTING VALUE AND INCREMENT
*       ARE CONSTANTS.
*       IN CASE OF A DONT LOOP,GO TO STEP D10.
*   D8. BEGIN DOO
*       SET THINGS UP FOR PUTTING VARIABLE IN AN
*       INDEX REGISTER. SET SWITCH FOR SPECIAL
*       HANDLING OF LABELS. COMPILE LIR1 N 3F,
*       2 IIR1 M,  LDL V, TGR 9F.  GO TO STEP D20.
*   D10.LDA INIT 3F
*       COMPILE LDA WITH INITIAL VALUE.
*   D11.  V + INC
*       ARTIFICIALLY INSERT  +V  INTO THE PROGRAM,
*       THUS RUNNING THROUGH THE ORDINARY ADD
*       GENERATOR TO CREATE CODE TO PUT THE SUM OF
*       V + INC IN REGISTER A.
*   D12.LDL, TGR
*       COMPILE  3 LDL FIN, TGR 9F, STA V
*   D20.LABEL IN TABLE
*       PUT THE LABEL NUMBER,TOGETHER WITH THE PER-
*       TINENT ADDRESSES FOR LINKING UP CONTROL
*       (9F,2B) INTO THE DO STACK. EXIT TO U12.
```

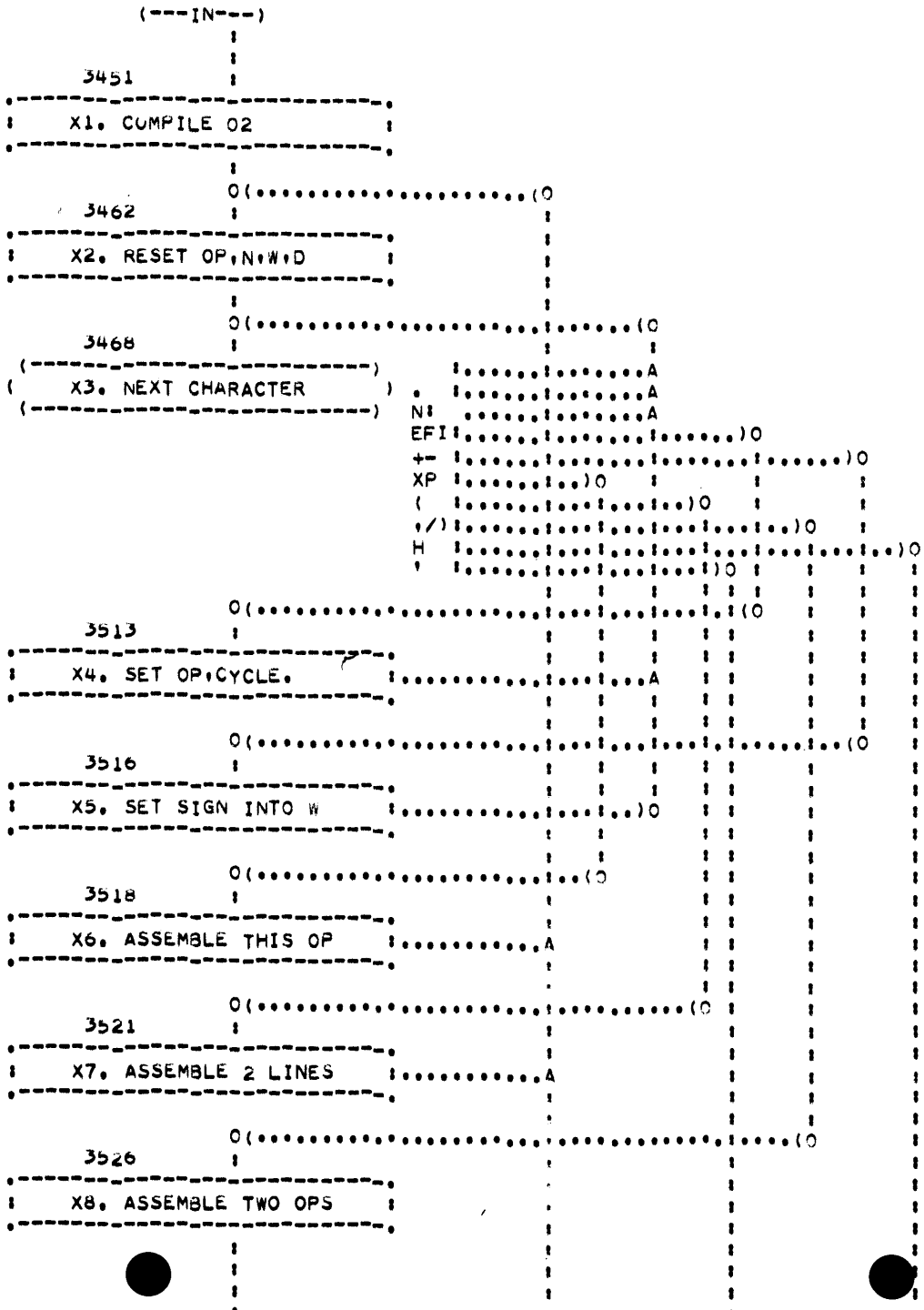

3229
D20.LABEL IN TABLE
U12



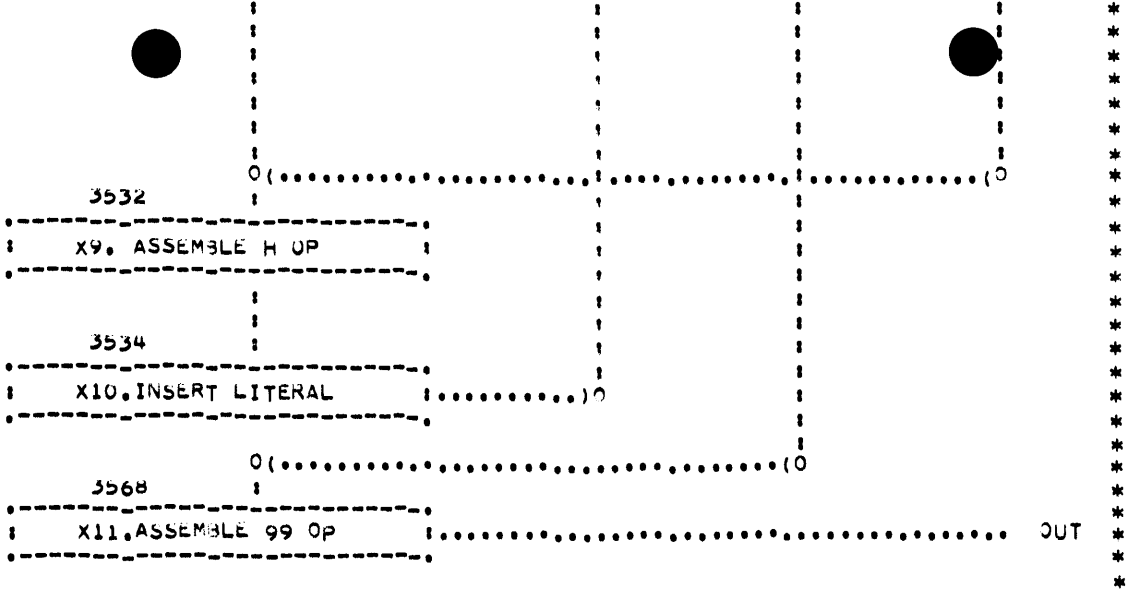

```
*   F.  FUNCTION CALLS
*       TRANSFER IS MADE TO STEP F1 IF WE HAVE AN
*       UNDIMENSIONED IDENTIFIER FOLLOWED BY A LEFT
*       PARENTHESIS,NOT OCCURRING IN A DIMENSION DEC.
*   F1. ASSIGN F
*       IF THIS IS A NEW FUNCTION DEFINE IT. IF IT IS
*       A CONSTANT OR SIMPLE VARIABLE,TREAT AS
*       IMPLIED MULTIPLICATION.
*   F2. SET FUNC MODE
*       SET UP FUNCTION MODE,AND ALSO PUT A SPECIAL
*       LEFT PARENTHESIS OPERATOR ON THE STACK.
*       AS WE PASS OVER THE LIST OF PARAMETERS,
*       CODE IS COMPILED TO COMPUTE THEM  AND STORE
*       THEM IN TEMP, IF THE PARAMETER IS A CONSTANT
*       OR INDEX REGISTER. AS THE RIGHT PARENTHESIS
*       CLOSING THE FUNCTION CALL OCCURS, TRANSFER
*       WILL GO TO STEP F4. GO NOW TO STEP G1.
*   F4. BEGIN REVERSE PASS
*       BEGIN NOW A RIGHT-TO-LEFT PASS OVER THE
*       PARAMETERS. RESERVE THE UNIQUE STORAGE FOR
*       THEM, THEN PROCESS EACH PARAMETER IN TURN.
*       THE TYPES OF CODE PRODUCED ARE:
*       FOR SIMPLE VARIABLE PARAMETER-PARAMETER
*             IIR HHHH, ERS PARAM, STA LIST
*             AND LIST IS MARKED AS TEMP STORAGE.
*       FOR A LABEL (I-O SUBROUTINES ONLY), CODE
*                   00 LLLL 0000 (OUT OF SEQUENCE).
*       FOR AN ARRAY,  IIR AO, STA LIST.
*       FOR A SIMPLE VARIABLE OR TEMP STORAGE,
*                 00 LLLL 0000 (OUT-OF-SEQUENCE).
*   F5. LIR3
*       AFTER ALL PARAMETERS HAVE BEEN PROCESSED,
*       COMPILE THE INSTRUCTION
*              LIR3 U(I)FUNCT, AND THE
*       NEXT INSTRUCTION GOES TO LOCATION U(I).
*       THE PARAMETERS HAVE BEEN LISTED IN U(I+1),
*       U(I+2), ETC.
*       IF THIS CALL IS NOT IN A CALL STATEMENT, TREA
*       THE RESULT AS A COMPUTED QUANTITY IN
*       REGISTER A.  GO TO G1.
*       NOTE: IF A CALL STATEMENT IS GIVEN WITH
*       NO PARAMETERS, NO REFERENCE TO UNIQUE
*       STORAGE IS MADE.
*
```



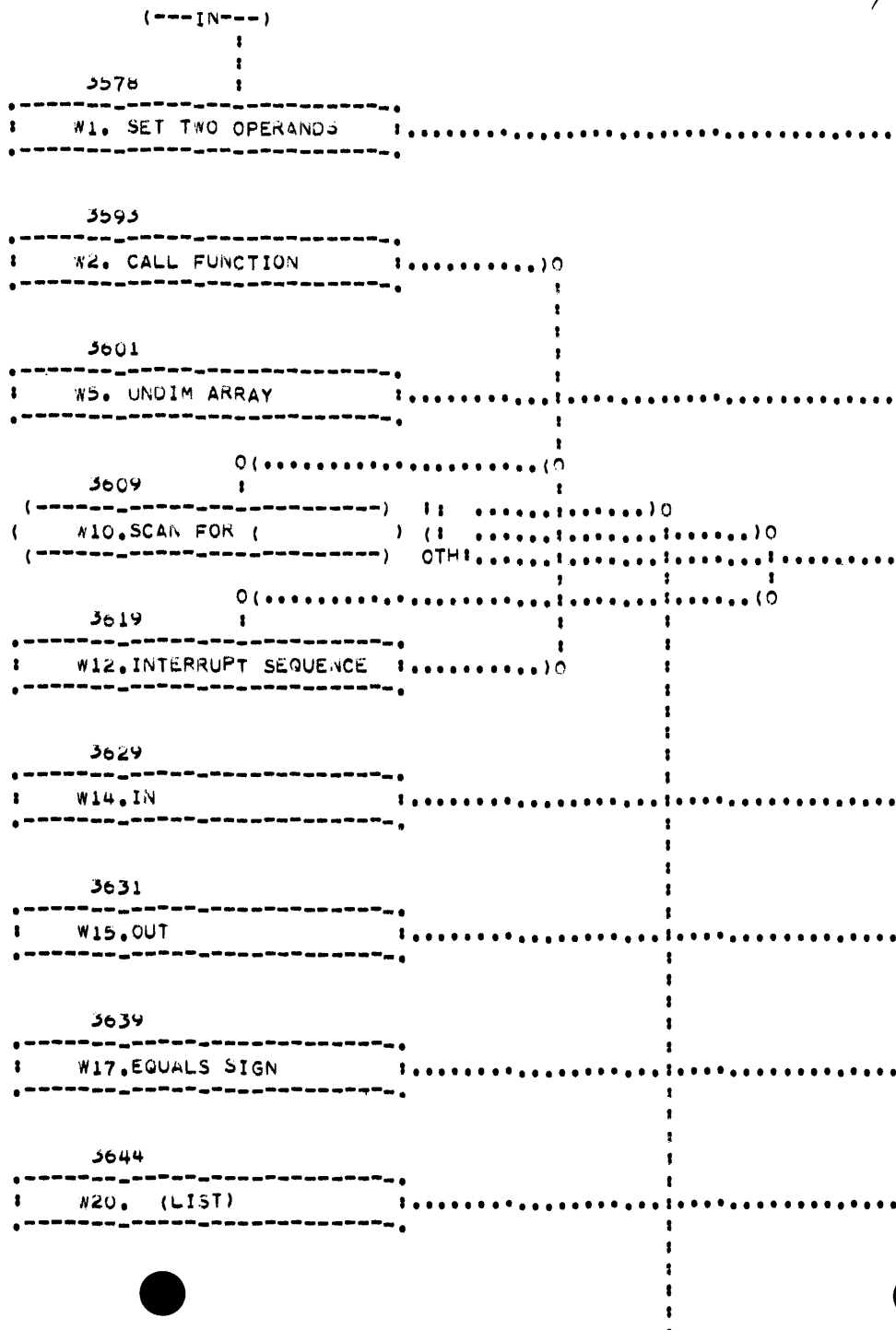

```
*   D.  D40. CLOSE OF DO RANGE.
*       AS EACH STATEMENT LABEL IS SCANNED IT IS
*       CHECKED AGAINST THE TOP OF THE DO STACK
*       TO SEE WHETHER THIS STATEMENT IS THE END OF
*       THE DO RANGE. IF IT IS, THE NEXT APOSTROPHE
*       OPERATOR (END OF STATEMENT) SENDS CONTROL
*       TO STEP D40.
*   D40. GO TO 2B
*       EFFECTIVELY COMPILE GO TO THE INCREMENTATION
*       PHASE AT THE BEGINNING OF THE DO LOOP CODING,
*       AND SET THE NEXT INSTRUCTION LOCATION TO BE
*       9F, THE ADDRESS FOR EXHAUSTION OF THE DO.
*   D41.DO OR DONT
*       IF THE LOOP JUST ENDED WAS A DONT LOOP,
*       SKIP TO STEP D50.
*   D42.EMPTY LLIST
*       TURN OFF THE VARIOUS INDICATORS WHICH ARE
*       SET DIFFERENTLY WHILE WE ARE IN A DO LOOP.
*       THEN FOR ALL LABELS WHICH WERE GIVEN
*       TEMPORARY ASSIGNMENTS, WE HAVE AN LLIST
*       ENTRY AND WE NOW OUTPUT THE INSTRUCTIONS
*             T     IIR1 0
*                   STA  V     P
*       WHERE V IS THE DO VARAIBLE, T IS THE TEM-
*       PORARY ASSIGNMENT, P IS THE PERMANENT
*       ASSIGNMENT. THE TEMP ASSIGNMENT IS THEN
*       FORGOTTEN.
*   D50. ANY MORE
*       IF ANOTHER DO LOOP ENDS ON THIS
*       STATEMENT,RETURN TO STEP D40. ELSE
*       GO TO Q3.
*
```



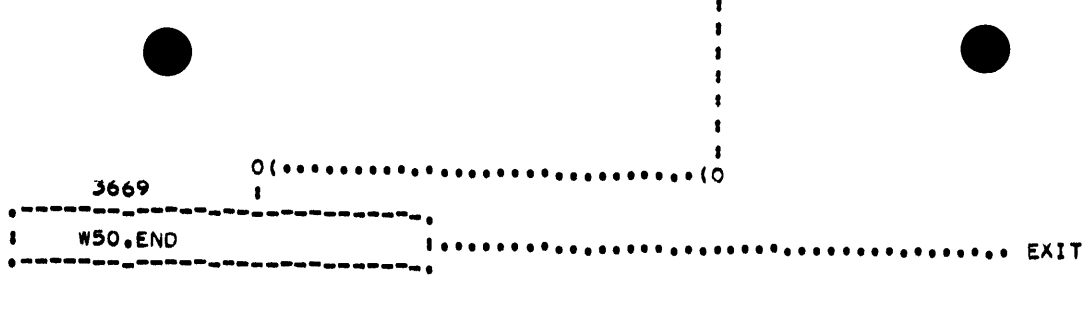

```
*   X.  PROCESSING FORMAT STRING
*   X1. COMPILE 02
*       THE INSTRUCTION 02 MMMM CCCC IS COMPILED
*       WHERE MMMM IS THE STARTING LOCATION OF THE
*       FORMAT CODE. WITH THIS TRICK, A FORMAT LABEL
*       IS LIKE ANY STATEMENT LABEL.
*       NOW WE TRANSLATE THE FORMAT INTO A SPECIAL
*       PSEUDOCODE. THIS CODE GENERATES INSTRUCTIONS
*       OF THE FORM OP NNN WWW DD, CORRESPONDING
*       TO FORMAT SPECIFICATION 'NNN E WWW.DD'.
*       OPCODES 0-10 CORRESPOND RESPECTIVELY TO
*       ()PIEFXAHM/
*   X2. RESET OP,N,W,D
*       CLEAR OP, N, W, AND D TO ZERO
*   X3. NEXT CHARACTER
*       GET THE NEXT CHARACTER FROM THE FORMAT LIST.
*       IF IT IS BLANK, DO X3 AGAIN.
*       IF IT IS A DECIMAL POINT, CYCLE N,W,D LEFT 1
*       AND RETURN TO X3
*       IF IT IS NUMERIC, SET D TO 10*D PLUS CHAR  X3
*       IF IT IS ALPHABETIC OR SPECIAL CHARACTER,
*       LOOK IT UP IN A TABLE TO SEE WHAT TO DO.
*       AN E F I A OR M MEANS GO TO X4.
*       A PLUS OR MINUS MEANS GO TO X5.
*       AN X OR P MEANS GO TO X6.
*       A LEFT PARENTHESIS MEANS GO TO X7.
*       COMMA SLASH AND RIGHT PARENTHESIS GO TO X8.
*       THE LETTER H MEANS GO TO X9.
*       AN APOSTROPHE MEANS WE GO TO X11.
*   X4. SET OP,CYCLE.
*       SET OP TO THE APPROPRIATE NUMBER, AND CYCLE
*       N,W, AND D LEFT 1. RETURN TO X3.
*   X5. SET SIGN INTO W
*       SET W TO 0 OR 1 (PLUS OR MINUS),RETURN TO X3.
*   X6. ASSEMBLE THIS OP
*       MOVE D TO N, THEN ASSEMBLE
*       OPNNNWWWDD INTO THE FORMAT CODE.RETURN TO X2.
*   X7. ASSEMBLE 2 LINES
*       MOVE D TO N AND ASSEMBLE, THEN INSERT A WORD
*       OF ZEROES INTO THE FORMAT CODE. THIS WORD
*       IS USED AS A SCRATCH PAD BY THE FORMAT
*       PROCESSING PACKAGE. RETURN TO X2.
*   X8. ASSEMBLE TWO OPS
*       IF DECIMAL POINT HAS NOT APPEARED, CYCLE
*       N,W,D LEFT 1. IF PREVIOUS OP IS WAITING
*       ASSEMBLE IT, AND CLEAR W. IF CURRENT IS NOT
*       A COMMA, ASSEMBLE IT TOO.
*       NOTE THAT ON  N/  THE COUNT N COMES OUT IN W.
*   X9. ASSEMBLE H OP
*       MOVE D TO N AND ASSEMBLE.
*   X10.INSERT LITERAL
*       OUTPUT 5 CHARACTERS OF THE LITERAL AT A TIME
*       UNTIL THE H LITERAL IS COMPLETED.
*       THE ROUTINE FOR H LITERALS IN THE CONSTANT
*       CONDENSER IS USED, WITH ZERO FILL AT THE
*       RIGHT. RETURN TO X2.
*   X11.ASSEMBLE 99 OP
*
*
```



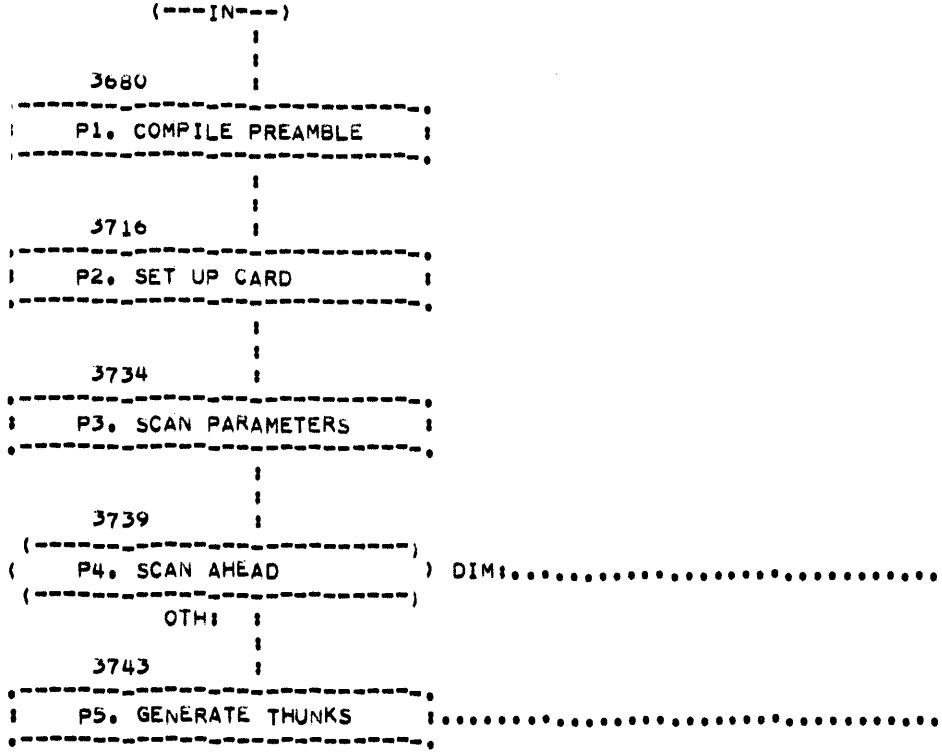
THE APOSTROPHE SIGNALS THE END OF THE STATEMENT. ASSEMBLE A TERMINATION LINE AND GO OUT.




```
*   W.  INPUT-OUTPUT (READ,PUNCH,PRINT)
*       THIS SECTION IS WITHOUT DOUBT THE CLIMAX OF
*       THE COMPILER. AT LEAST 95% OF THE CODING OF
*       THIS COMPILER PROGRAM CAN BE ACTIVE WHILE
*       PROCESSING A SINGLE I/O LIST.
*   W1. SET TWO OPERANDS
*       SET UP TWO OPERANDS, ONE FOR THE EDITING
*       SUBROUTINE AND ONE FOR THE DRIVER SUBROUTINE
*       (I-O DEVICE).  SET UP TO EXPECT A LABEL.
*       GO TO G1, WE WILL RETURN TO STEP W2 WHEN
*       THE COMMA IS SENSED.
*   W2. CALL FUNCTION
*       USE THE FUNCTION CALL ROUTINE (ROUTINE F) TO
*       CREATE INITIAL ENTRY TO THE I/O SUBROUTINE.
*       THEN SET UP I/O MODE.  IF AN UNDIMENSIONED
*       ARRAY VARIABLE OCCURS WE WILL GO TO STEP
*       W5. ON A COMMA WE GO TO STEP W10.
*       AT THE END OF THE STATEMENT, WE GO TO STEP
*       W50, NOW WE GO TO THE COMMA ROUTINE, STEP W10.
*   W5. UNDIM ARRAY
*       AN UNDIMENSIONED ARAY A IS CONVERTED INTO
*       (A(*I*), *I* IS 1,N)
*       USING ROUTINE Q10     WHERE *I* IS A DUMMY
*       VARIABLE AND N IS THE PRODUCT OF THE ARRAY
*       DIMENSIONS.
*   W10.SCAN FOR (
*       SCAN NEXT ITEM (CO-ROUTINE S). IF IT IS THE
*       END OF THE STATEMENT, GO TO STEP W50. IF IT IS
*       A LEFT PARENTHESIS, GO TO STEP W12.
*       OTHERWISE INSERT AN  IN  OR  OUT  OPERATOR
*       ON THE STACK THEN GO TO G2.  IN IS STEP W14,
*       OUT IS STEP W15.
*   W12.INTERRUPT SEQUENCE
*       CREATEA BREAK IN THE INSTRUCTION SEQUENCE,
*       FOR WHICH CODE WILL BE INSERTED LATER. PUT A
*       SPECIAL LEFT PARENTHESIS ON THE STACK.  THIS
*       SPECIAL LEFT PARENTHESIS IS STEP W20.
*       GO TO STEP  W10 AGAIN.
*   W14.IN
*       COMPILE LIR3     SUB, STL V. GO TO G10.
*   W15.OUT
*       COMPILE LDA V, LIR3     SUB. GO TO G10.
*       SUB IS ONE OF THREE ENTRIES, DEPENDING ON
*       THE TYPE (FLOAT,FIX,UNSPECIFIED) OF V.
*   W17.EQUALS SIGN
*       AN EQUALS SIGN HAS APPEARED, SO WE PULL THE
*       SPURIOUS IN OR OUT OPERATOR OFF THE STACK.
*       WE NOW COURAGEOUSLY JUMP INTO THE MIDDLE
*       OF THE THROUGH ROUTINE, STEP D1.
*   W20.  (LIST)
*       THE RIGHT PARENTHESIS MATCHING A LEFT HAS BEE
*       ENCOUNTERED. IF AN IMPLIED DO LOOP OCCURRED
*       INSIDE, WE USE PARTS OF ROUTINE D TO CREATE
*       CODING FOR THE DONT LOOP CONTROL.  FINALLY
*       THE INTERRUPTIONS FROM STEP W12 ARE ALL
*       LINKED TOGETHER PROPERLY.  GO TO G1.
```



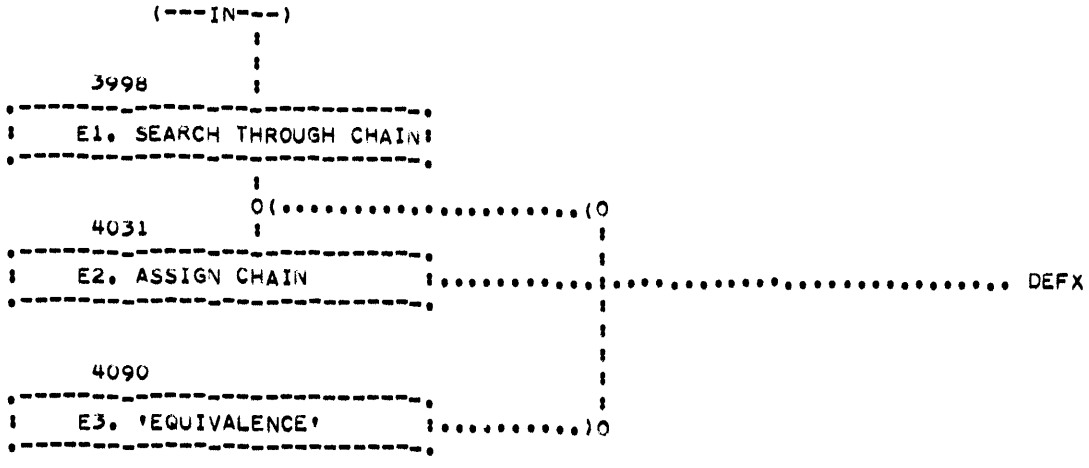

```
W50.END
    COMPILE LIR3    SUB, THE ENDING SUBROUTINE.
    AND THEN EXIT.
```




```
*   P.  FUNCTION AND SUBROUTINE DECLARATIONS
*   P1. COMPILE PREAMBLE
*       COMPILE IIR1 0000, BUF    1F, LIR1 0000,
*       (AND LDA 0001, ATL  1F FUNCTION)
*   P2. SET UP CARD
*       SET UP THE NAME OF THE FUNCTION INTO THE
*       OUTPUT CARDS, INITIALIZE OTHER THINGS LIKE
*       THE MEANING OF RETURN.  A MAIN PROGRAM
*       IS DISTINGUISED FROM A SUBPROGRAM
*       ONLY BY DEFAULT.
*   P3. SCAN PARAMETERS
*       SCAN UNTIL THE END OF THE STATEMENT,
*       COLLECTING ALL PARAMETERS ON THE OPERAND
*       STACK.  WE GET TO STEP P4 AT THE END
*       OF THE STATEMENT.
*   P4. SCAN AHEAD
*       IF THE NEXT ITEM SCANNED IS ANOTHER END OF
*       STATEMENT OR DIMENSION, GO TO G2.
*   P5. GENERATE THUNKS
*       ELSE WE ASSUME ALL DIMENSIONED PARAMETERS
*       HAVE BEEN NAMED, AND WE COMPILE CODE TO TRANS
*       FER FROM THE PARAMETER LIST TO UNIQUE STORAGE
*       TEN INSTRUCTIONS FOR NON-DIMENSIONED
*       PARAMETERS AND TWO FOR DIMENSIONED ONES.
*       THEN OFF TO G2.
```

```
        (---IN---)
             !
             O(....................(O
    3797     !                     !
.-------------------------.        !
!  Z1. SET UP HEADER TABLE !       !
.-------------------------.        !
             !                     !
             !                     !
    3807     !                     !
.-------------------------.        !
!  Z2. CLEAR SYMBOL TABLE  !       !
.-------------------------.        !
             !                     !
             !                     !
    3832     !                     !
.-------------------------.        !
!  Z3. INITIALIZE COUNTERS !..........!...........................  G1
.-------------------------.        !
                                   !
                                   !
    3876                           !
.-------------------------.        !
!  Z50.END IS SENSED       !       !
.-------------------------.        !
             !                     !
             !                     !
    3884     !                     !
.-------------------------.        !
!  Z51.PUNCH HEADERS       !       !
.-------------------------.        !
             !                     !
             !                     !
    3929     !                     !
 (-----------------------)  EMP!.....!............................ STOP
(   Z52.READ NEXT CARD    ) LOAD.....!............................PASS2
 (-----------------------)  OTH!.....)O
```

```
*  Z.   INITIALIZATION AND TERMINATION
*       Z1 IS ENTERED AT THE BEGINNING OF EACH
*       PROGRAM AND SUBPROGRAM.
*  Z1.  SET UP HEADER TABLE
*       THE HEADER CARD INFORMATION IS KEPT IN A 50-
*       POSITION CIRCULAR TABLE. IF MORE THAN 50
*       TOTAL ITEMS ARE PUT IN, A FLAG IS SET
*       SO THAT LOAD-AND-GO OPERATION IS DISALLOWED.
*  Z2.  CLEAR SYMBOL TABLE
*       ALL SYMBOLS EXCEPT RESERVED WORDS ARE
*       REMOVED FROM THE SYMBOL TABLE.
*  Z3.  INITIALIZE COUNTERS
*       VARIOUS THINGS ARE RESET, E.G. SUBROUTINE
*       PACKAGE REQUESTS, STORAGE ALLOCATION REQUESTS
*       COUNTERS ARE SET UP TO INDICATE A MAIN
*       PROGRAM, THESE WILL BE EFFECTIVE UNLESS
*       A FUNCTION OR SUBROUTINE DECLARATION FOLLOWS.
*       START COMPILING BY TROTTING FORTH TO G1.
*  Z50.END IS SENSED
*       AN END CARD MEANS WE SIMULATE A RETURN
*       STATEMENT (I.E. GO TO EXIT).
*  Z51.PUNCH HEADERS
*       PUNCH AND PRINT HEADER INFORMATION.
*  Z52.READ NEXT CARD
*       IF NO MORE INPUT CARDS ARE IN THE BUFFER,
*       PUNCH OUT SEVERAL BLANK CARDS AND STOP.
*       IF THE NEXT CARD IS THE BEGINNING OF PASS2,
*       TRANSFER TO THE SECOND PASS UNLESS AN ERROR
*       OCCURRED IN THE PRECEDING PROGRAMS.
*       OTHERWISE WE GO TO Z1 TO PROCESS ANOTHER
*
```




```
*   E.  EQUIVALENCE DECLARATIONS.
*       IT IS ALMOST IMPOSSIBLE TO EXPLAIN HOW THE
*       PROCESSING OF EQUIVALENCE DECLARATIONS
*       WORKS IN THIS COMPILER.
*                       EQUIVALENCE CLASSES ARE KEPT
*       IN CIRCULARLY-LINKED CHAINS.  IT IS EASY TO
*       MERGE TWO CHAINS INTO ONE. WHEN AN ITEM
*       OF A CHAIN IS FIRST REFERENCED AFTER AN
*       EQUIVALENCE DECLARATION, WE GO TO E1. FORMATS
*       OF THE CHAIN ENTRIES APPEAR IN THE TABLE
*       OF FORMATS.
*   E1. SEARCH THROUGH CHAIN
*       TRAVERSE THE CHAIN ONCE TO SEE HOW MUCH
*       UNIQUE STORAGE IS TO BE RESERVED.
*   E2. ASSIGN CHAIN
*       TRAVERSE THE CHAIN AGAIN, ASSIGNING EVERY
*       VARIABLE IN THE CHAIN RELATIVE TO THE OTHERS.
*       GO TO DEFX.
*   E3. 'EQUIVALENCE'
*       ON THE EQUIVALENCE DECLARATION, VARIOUS
*       MODES ARE SET UP.  AT THE END OF EACH
*       EQUIVALENCE, A CHECK IS MADE TO SEE IF
*       ANY OF THE ITEMS WAS PREVIOUSLY DEFINED.
*       IF SO, THE ENTIRE CHAIN IS THEN DEFINED,
*       AS IN STEP  E2.
*
```

FORTRAN II

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

UP-3843.1