NORD PL

Program Documentation

NORSK DATA A.S

NORD PL

Program Documentation

REVISION RECORD

Revision	Notes
1/74	Original Printing

ND-60.059.01
January 1974

A/S NORSK DATA-ELEKTRONIKK
Lørenveien 57, Oslo 5 - Tlf.: 21 73 71

TABLE OF CONTENTS

+++
+

Chapters: Page

1 INTRODUCTION

1.1 General

The NORD PL compiler translates the NORD PL source language to MAC assembly source code. For the definition of the language, see the NORD PL User's Guide.

The compiler runs on a NORD-1 or NORD-10 computer, either under TSS, NORD-OPS or free-standing (using standard I/O).

1.2 Related Programs

a) The MAC assembler

b) TSS

c) NORD-OPS

d) Standard I/O

A MAC assembler is necessary to do the final convertion to binary or BRF format. In principle a very rudimentary version with no options can be used. However, if REAL variables are used in a NORD PL program, the floating point option should be included, and if BRF format is wanted, the BRF option should be used. As the user program can contain arbitrary sequences of MAC code, other options might be necessary in certain cases.

The debug facilities of MAC are supposed to be used; so the breakpoint option might be useful.

1.3 Operator Communication with the Compiler

The compiler uses the same principle as the MAC assembler. When the compiler is started, it receives input source text from the operator Teletype, so that the operator can write NORD PL commands and statements and then changes the input device number by the)9DEV command.

For NORD-OPS the device numbers are set by the control card.

1.4 Input/Output Format

The I/O is performed by means of call of INBT and OUTBT, with device number in T and character in A (standard call). If a negative A content is received, an error message will be given. ASCII code is used.

On input the parity bit is ignored. On output, even parity is given.

1.5 Programming Strategy

The compiler itself is written in NORD PL. The first version was hand-compiled to MAC code.

The variable parts (tables, buffers, variables, ..) are placed in the first part. The routines are read - only. The variables are placed in a global BASE field. The local variables of a subroutine are DISP-decleared, overlaying the first location of the BASE field. This means that the same B-register is used for as well local as global variables. When a new subroutine is called, the variables of the calling one will be saved on a subroutine stack.

The sequence of the subroutines is for the most hierarchal, so that a subroutine will be placed after its calls. Internal jumps in a subroutine will likewise be forward jumps as far as possible.

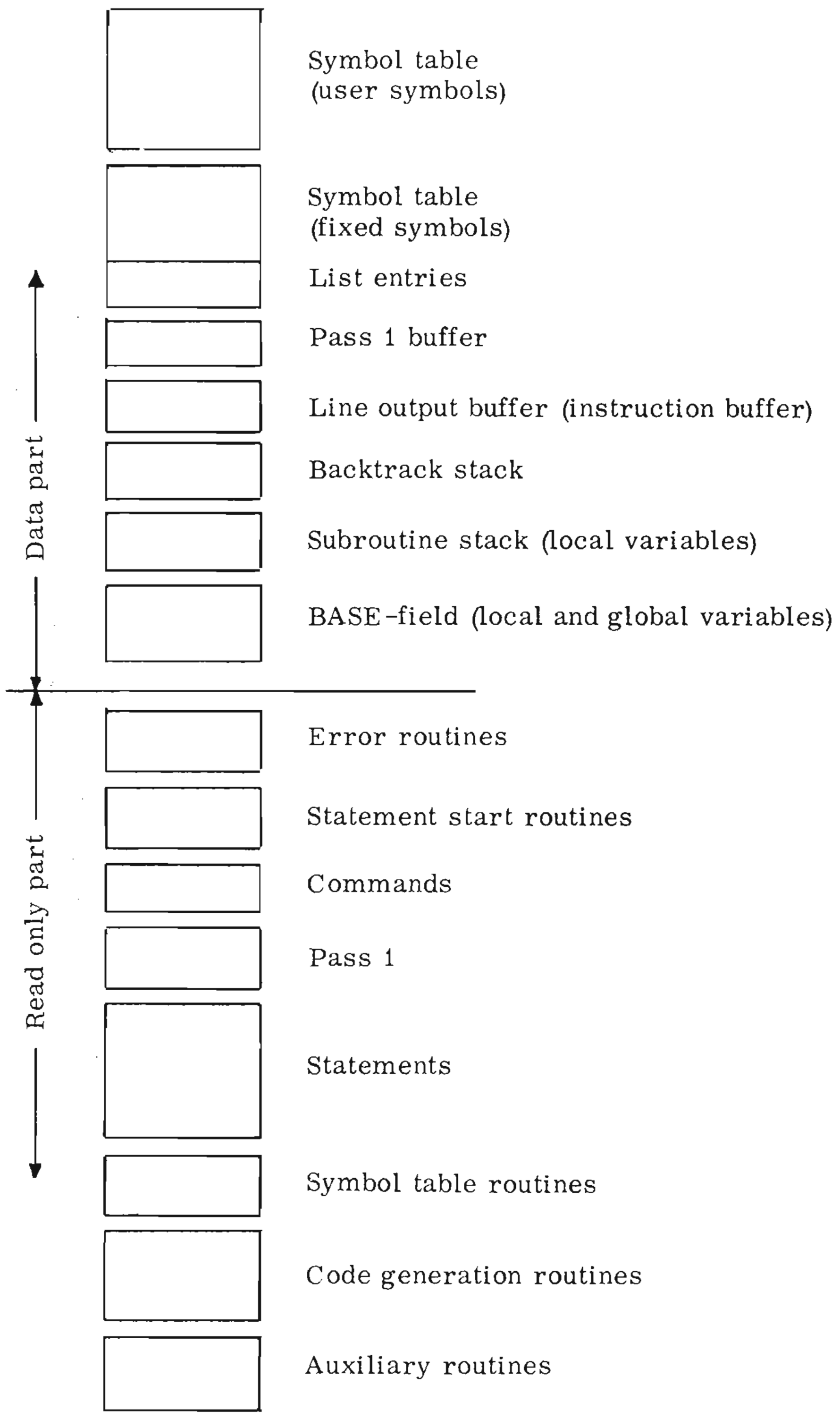
The compiler is to some degree table-oriented. The state-table techniques are used extensively.

Some of the main variables (TYPE, VALUE, ...) have a limited number of values (177_8). These values are defined equal to symbols beginning with the character "5". Subroutines referenced through global pointers begin with "3".

2 PROGRAM LOGIC

2.1 Layout

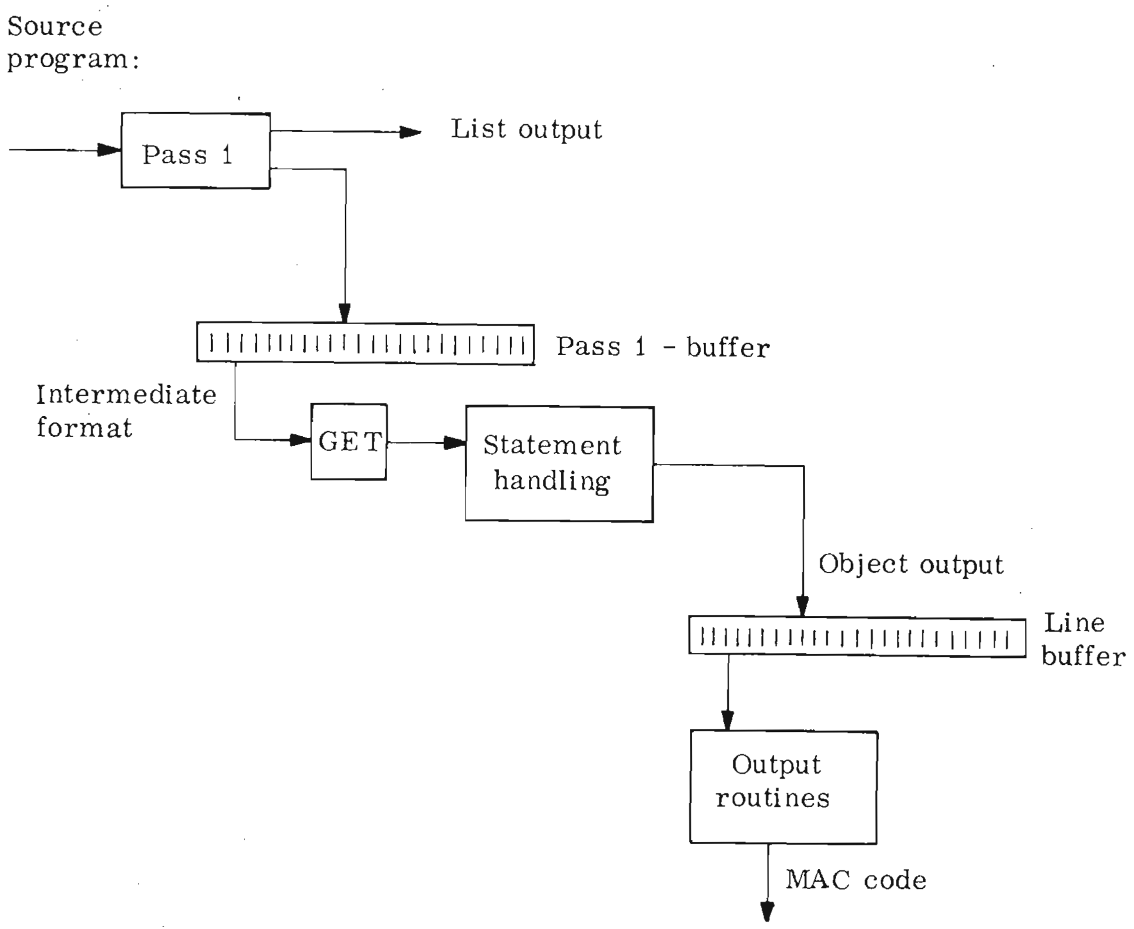
The compiler layout is like this



2.2 Information Flow

The compiler executes physically i one pass. However, each statement is preprocessed by a "Pass 1" routine.

Main information flow:



2.3 Tables

2.3.1 Main Symbol Table

The main symbol table is used for fixed symbols, user symbols, and IF-FOR nesting stack.

The main access method is linked hash-index. The three last bits in the last character of the symbol is used as index in a table, LISTIN, which contains the start of 8 linked lists in the table. An auxiliary table, LISTOUT, contains pointers to the end of the lists.

At the end of a subroutine the local symbols are removed, the elements being linked into the FREE list. When a new symbol is entered, an element is taken from the FREE list. If the list is empty, more place is allocated at the end of the active table.

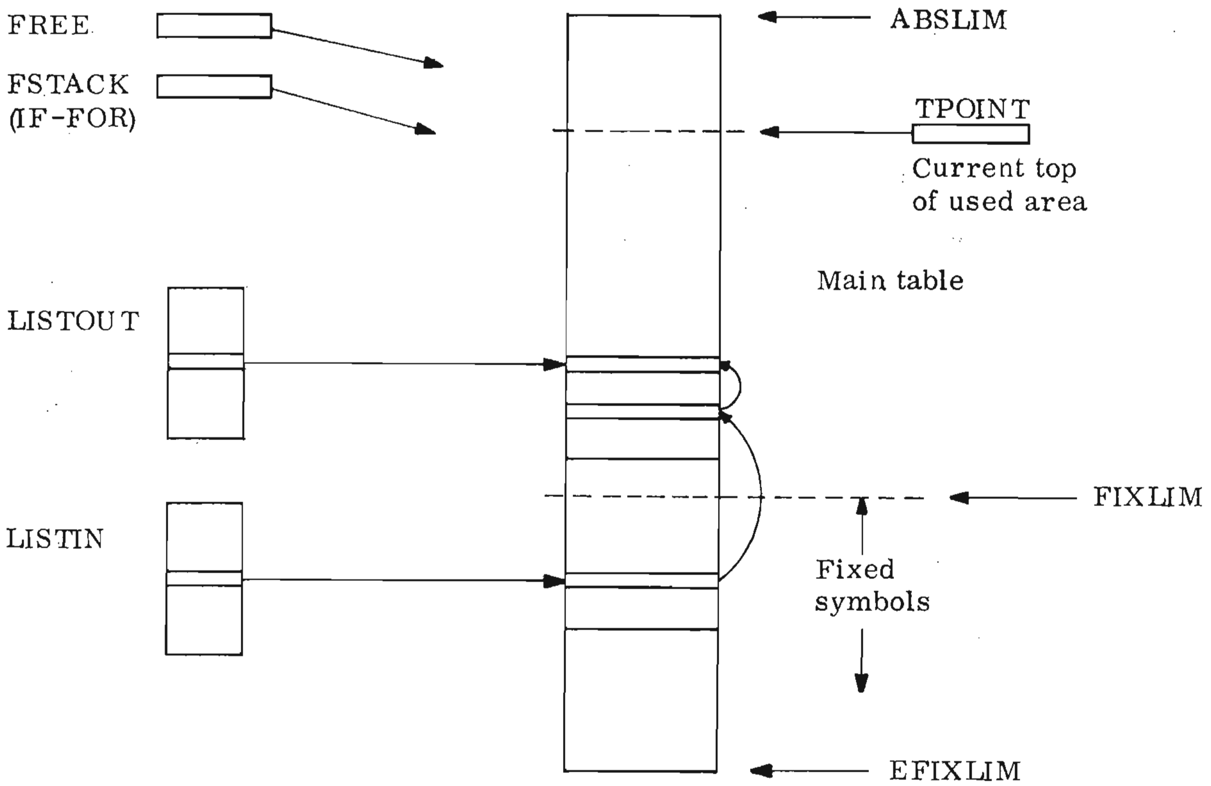
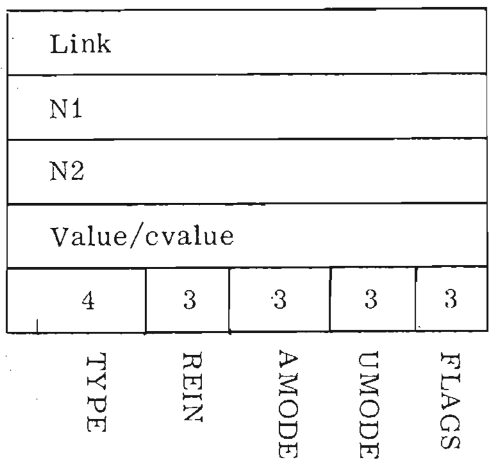


Table element:



"N1" - "N2" contains the symbol (5 characters).
"value/cvalue" contains VALUE, except for SYMBOL-defined symbols, where it contains CVALUE, and BASE-addressed items, where it contains TARI (table reference) for the BASE symbol.
"FLAGS" contains

BIT 0:	LIBRFLAG
BIT 1:	UDEFLAG
BIT 2:	LOCFLAG

2.3.2 Pass 1 Buffer

The Pass 1 buffer is an array, BUFFA, containing characters packed two by two. A byte pointer BUFP points to the current position in the buffer, and another pointer AVAILABLE tells the number of available characters.

2.3.3 Output Line Buffer

The output line buffer is an array IBUFA, containing characters packed two by two. A byte pointer, IBUFP, points to the current position.

2.3.4 Backtrack Stack

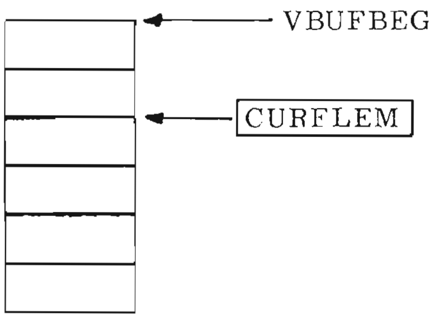
The backtrack stack is a ring buffer containing information on the last 30_8 items used. An element has the form:

N1
N2
value/cvalue
mix
TARI
IBUFP

The first four locations correspond to the last four of the main table element. It contains a snapshot of the main variables just before a new variable is fetched from the Pass 1 buffer (by the GET subroutine).

When the routine RESET is called, this situation is restored.

The backtrack stack resides in an array, VBUFBEG.



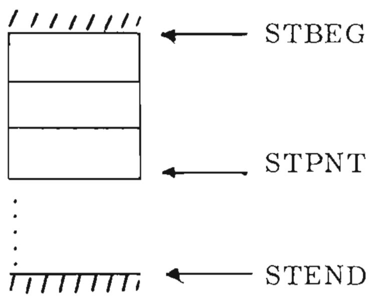
Pointing to the currently last element

(element no.)

The variable GETCOUNT tells how many elements are available at the moment.

2.3.5 Subroutine Stack

Each time the subroutine ENTER is called, eight locations are saved on a stack, in the array STBEG. A pointer STPNT points to the first free location.



An element looks like:

D register (return)
X register
The six first locations in the BASE field

By jump to the subroutine RETURN the saved X register is restored, the saved D register is used for return address, and the six locations are moved back to the BASE field.

2.4 The Variables in the BASE Field

V0-V7	- Area for local variables.
LINK	- Return address for the bottom subroutines.
INDEX	- X register for the bottom subroutines.
LINK2	- Return address for next to bottom subroutines.
SAVTAD	- Save locations for TAD.
IDEV	- Input device.
LDEV	- List device.
ODEV	- Object device.
ERDEV	- Error device.
ICOMDEV	- Input communication device.
OCOMDEV	- Output communication device.
ERBUSY	- Flag to prevent recursive call of the error routines.
ERSTNO	- Current line no. after last label.
ERNAME	- Last label name.
STADR	- Start of statement routine
LASTADR	- Last statement.
NSTATE	- Statement syntax state.
NOLDSTATE	- Former value of NSTATE.
BUFP	- Character counter of PASS1 buffer.
AVAILABLE	- Number of available elements.
CHAR	- Current character.
BCHAR	- Buffered character.
BYTE	- Last control byte put into PASS1 buffer.
ICRFLAG	- If set, carriage return is changed to space.
DECFLAG	- Decimal mode.
MACFLAG	- Processing assembly code.
FNUM	- IF-FOR label number.
ORLAB	- OR label number.
REG1	- Register of first expression in a relation.
RELOP	- Relation operator.
BITNO	- Bit number in bit test.
FMAX	- Last IF-FOR label number.
THENTYPE	- AND, OR. THEN or GO.
FTYPE	- Information to FI-OD.

FSTEP	- Step information.
FCONTROL	- Control variable definition.
REGISTER	- Primary register in an operation.
OPERATOR	- Used in GENERATE.
OPER2	- AD1, ADC.
DISPL	- Displacement value in DISP statement.
SAMO2	- Saved SAMODE.
PRESFLAG	- Set if the symbol is present in the table.
SAMODE	- Declared variables will get this value as their AMODE.
BTARI	- TARI for the BASE variable.
TPCHECK	- Check information, contents of the last used location in the main table.
FSTACK	- Start of the linked IF-FOR stack.
TPOINT	- Pointing to current top of table.
FREE	- Start of free list.
N1,N2	- Symbol, five characters.
VALUE	- Basic element identification.
CVALUE	- Constant value.
TYPE	- Main grouping of basic elements.
REIN	- Size of variable: 1, 2 or 3 locations.
AMODE	- Addressing mode.
VMODE	- Variable mode.
SLOCFLAG	- Inside subroutine indicator.
LOCFLAG	- Symbol with this flag set are killed at RBUS.
UDEFLAG	- Symbol not defined.
LIBRFLAG	- Include mode.
TARI	- Table reference.
CURELEM	- Current element in the backtrack stack.
GETCOUNT	- Number of available elements in the backtrack stack.
IBUF	- Pointer to instruction buffer start.
IBUFP	- Pointer (character relative) into instruction buffer.
STPNT	- Subroutine stack pointer.
PTARI	- For indirect reference through TARI.

3 ROUTINE LOGIC

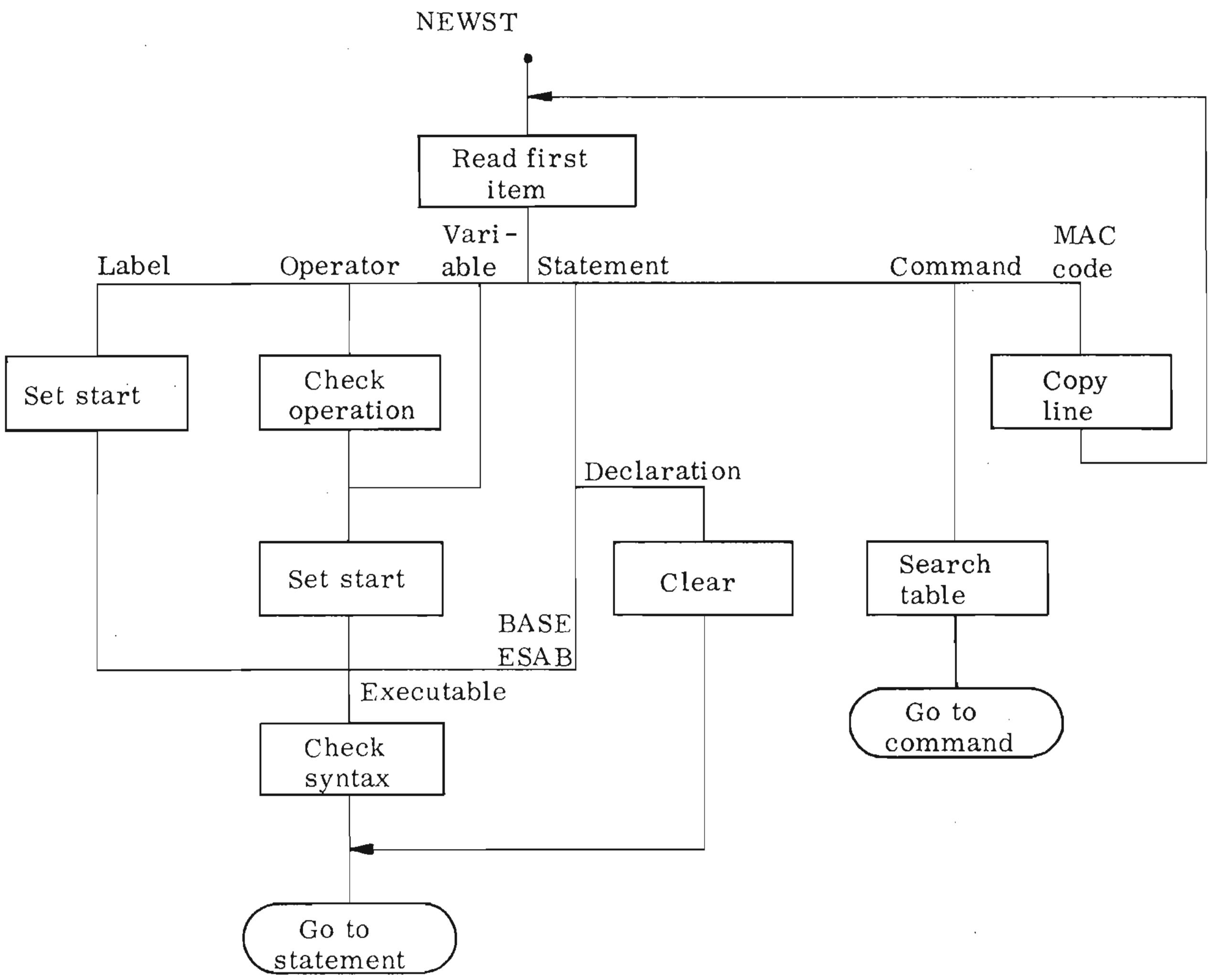
3.1 Compiler Entry Points

The entry point NPL is used for initial start. All tables are cleared, and the message "NORD PL version is written.

The entry point ONLINE is used for restart. The device numbers are set to the communication device. The symbol table is retained.

3.2 Error Handling

An error message is written, and for the most part control is returned to NEWST. Fatal errors exit to ONLINE.

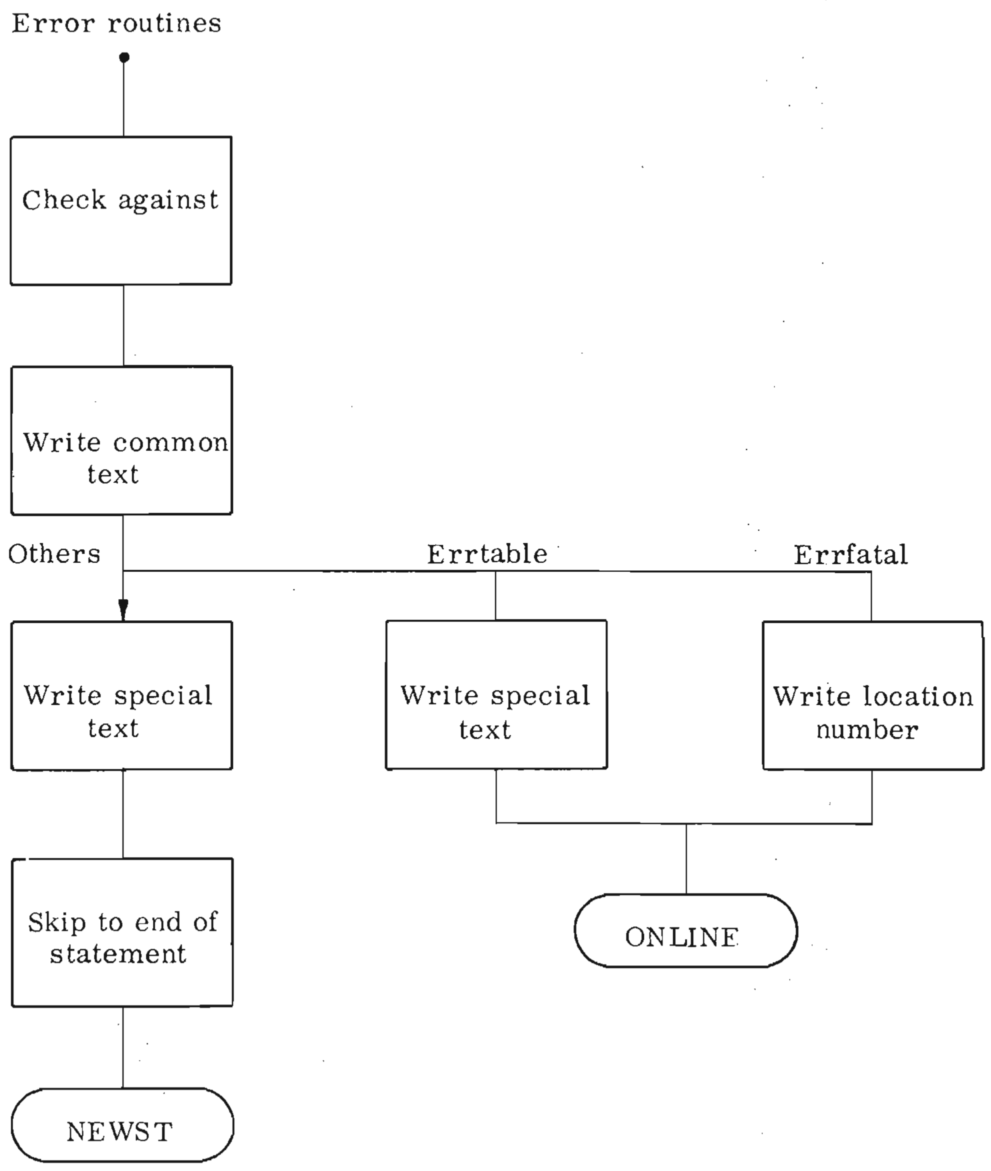


3.3 Statement Start

After each statement, the routine NEWST is entered, to determine which statement should come next. A state table, HUFF, is used for inter statement syntax check. A new state ≥ 8 means error. Before jumping to the statement routine, ENTER is called, with return address to NEWST.

The HUFF table:

Input / Output		BASE	ESAB	DISP	PSID	SUBR	RBUS	Execut-able
At start	0	1	10	2	10	3	10	13
In BASE	1	11	0	11	11	11	11	13
In DISP	2	12	12	12	0	12	12	13
In SUBR	3	4	13	5	13	13	0	3
In local BASE	4	14	3	14	14	14	14	13
In local DISP	5	15	15	15	3	15	15	13


Error routines
Check against
Write common text
Others
Errtable
Errfatal
Write special text
Write special text
Write location number
Skip to end of statement
ONLINE
NEWST

3.4 Commands

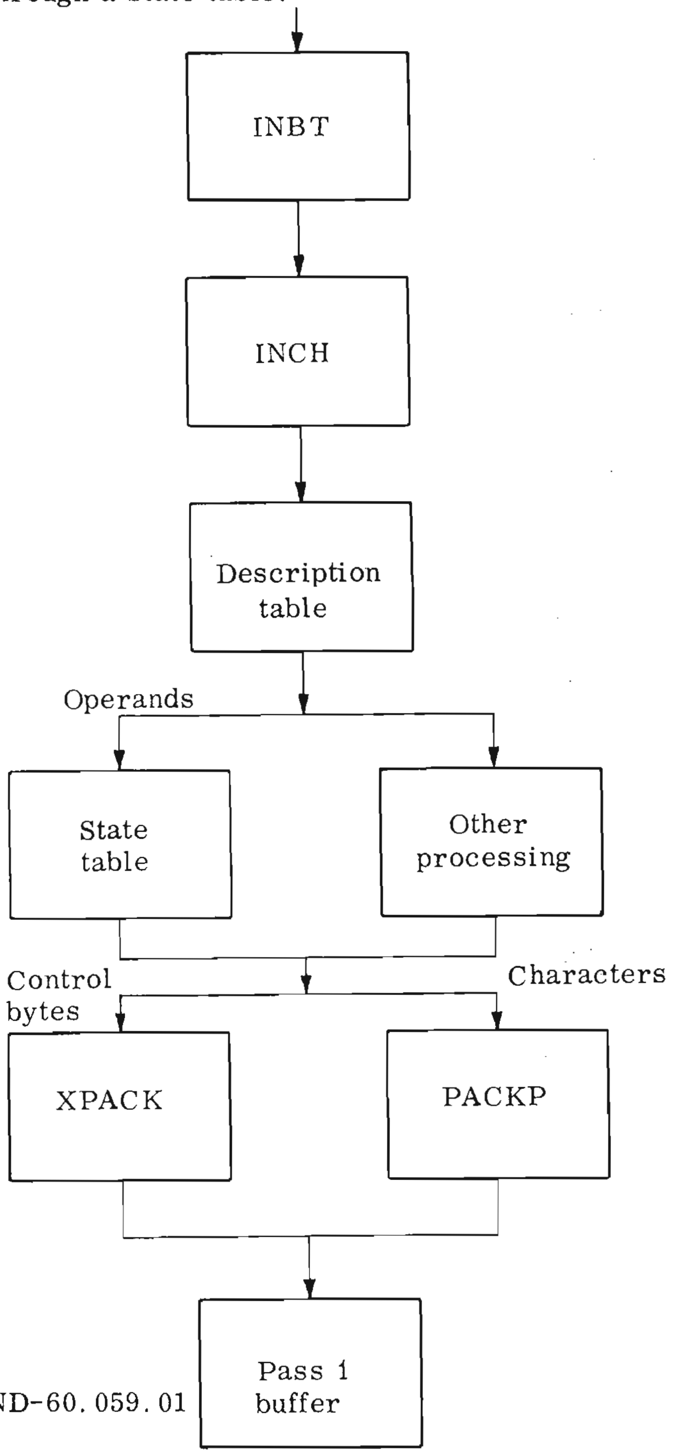
The command LIB:

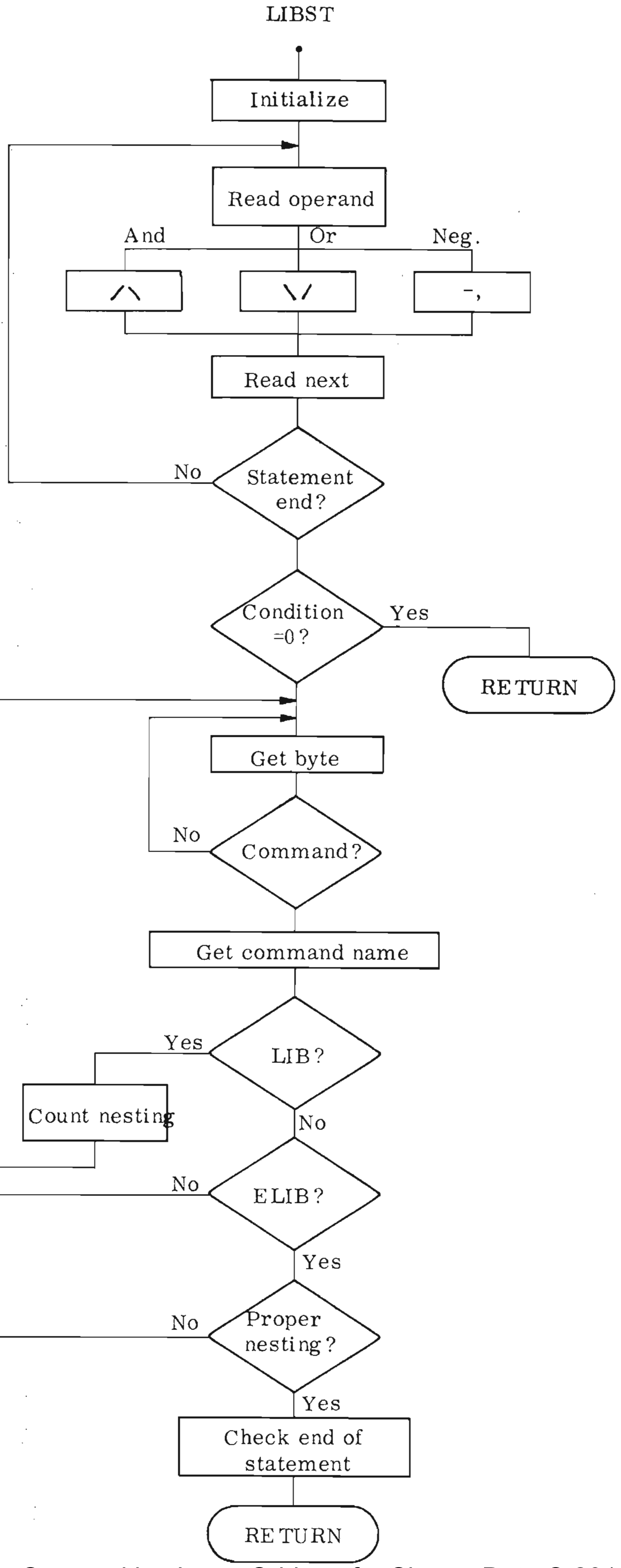
3.5 Pass1

The PASS1 subroutine is called from PICKP each time the Pass1 buffer has become empty. Then PASS1 processes information, putting it into the Pass1 buffer, until a comma or statement end (semicolon, carriage return) is found. The information is packed as 8 bits bytes. Each element consists of a control byte, which is a small number, with bit 7 set. Thereafter a character string (bit 7 never set), holding a symbol or a number. The operands (:=, +, :=:, · · ·) are reduced to control bytes only.

A descriptor table is used for the first classification. Symbols and constants are checked through a state table.

Main Information Flow:




LIBST
Initialize
Read operand
And
Or
Neg.
∧
∨
¬
Read next
No
Statement end?
Condition =0?
Yes
RETURN
Get byte
No
Command?
Get command name
Yes
LIB?
Count nesting
No
No
ELIB?
Yes
No
Proper nesting?
Yes
Check end of statement
RETURN

In the descriptor table the four leftmost bits contain a switch number. The rest contain either a control byte number or a relative label address. -1 indicates illegal character.

Octal	Char.	Switch	Byte	Address
40	Space	0		RNEXT
41	!	-1		
42	"	1	5 ref	
43	#	0		CHRS
44	$	-1		
45	%	0		PERCENT
46	&	0		OPND
47	'	0		STRING
50	(	1	5 lpar	
51	)	1	5 rpar	
52	-	0		STAR
53	+	1	5 plus	
54	,	1	5 comma	
55	-	2	5 minus	
56	.	1	5 dot	
57	/	2	5 div	
60-71	Digits	0		OPND
72	:	2	5 colon	
73	;	1	5 stend	
74	<	2	5 lst	
75	=	2	5 eql	
76	>	2	5 gre	
77	?	1	5 quest	
100	@	1	5 comnd	
101-132	Letters	0		OPND
133	[	-1		
134	\	2	5 byte	

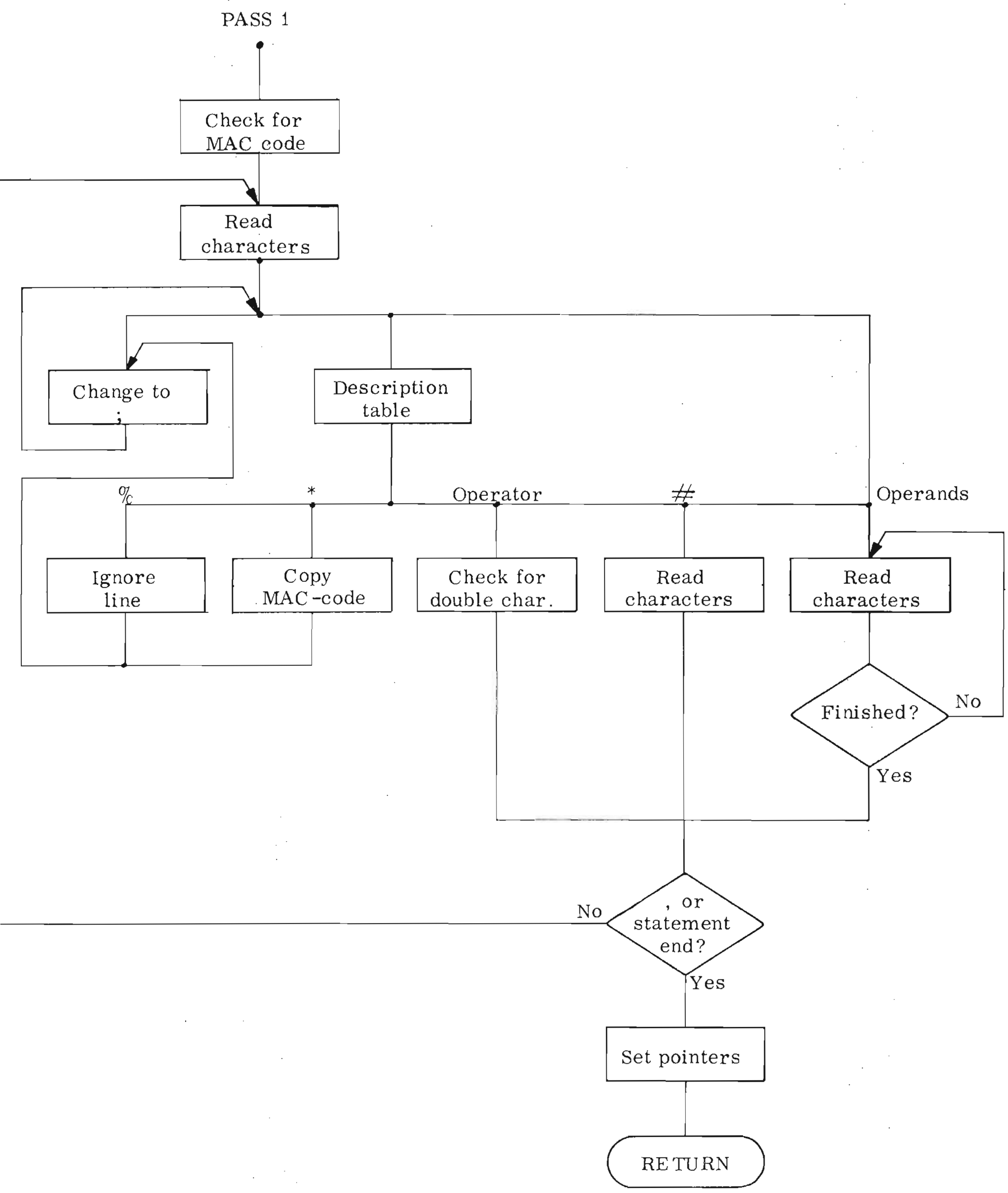
Switch values:	0:	Special processing
	1:	One character item
	2:	Possibly more than one character

Operand State Table:

Input / State	A-Z	Q-a	&	.	#	-	others
Start 0	copy char. 1	copy char. 2	octal 2	error 0	error 0	error 0	error 0
In symbol 1	read, check 1	read, check 1	finish, reset 1	finish, reset 0	finish, reset 0	finish, reset 0	finish, reset 0
In number 2	read, check 1	copy char. 2	error 0	copy char. 3	insert H 4	finish, reset 0	finish, reset 0
In fraction part 3	error 0	copy char. 3	error 0	error 0	insert H 4	finish, reset 0	finish, reset 0
After # 4	error 0	copy char. 5	error 0	error 0	error 0	copy char. 5	error 0
In exp. 5	error 0	copy char. 5	error 0	error 0	error 0	finish, reset 0	finish, reset 0

For the character constants (#, ## and ###···#) conceptually a state table exists, although it is coded slightly different.

Input / State	#	Others
After # 0	1	copy char., read and copy next 0
After ## 1	output 5DCON and ### 2	output 5CH2 copy char. 0
After ## 2	finish 0	copy char. 2


PASS 1
Check for MAC code
Read characters
Change to ;
Description table
%
*
Operator
#
Operands
Ignore line
Copy MAC-code
Check for double char.
Read characters
Read characters
Finished?
No
Yes
, or statement end?
No
Yes
Set pointers
RETURN

INCH

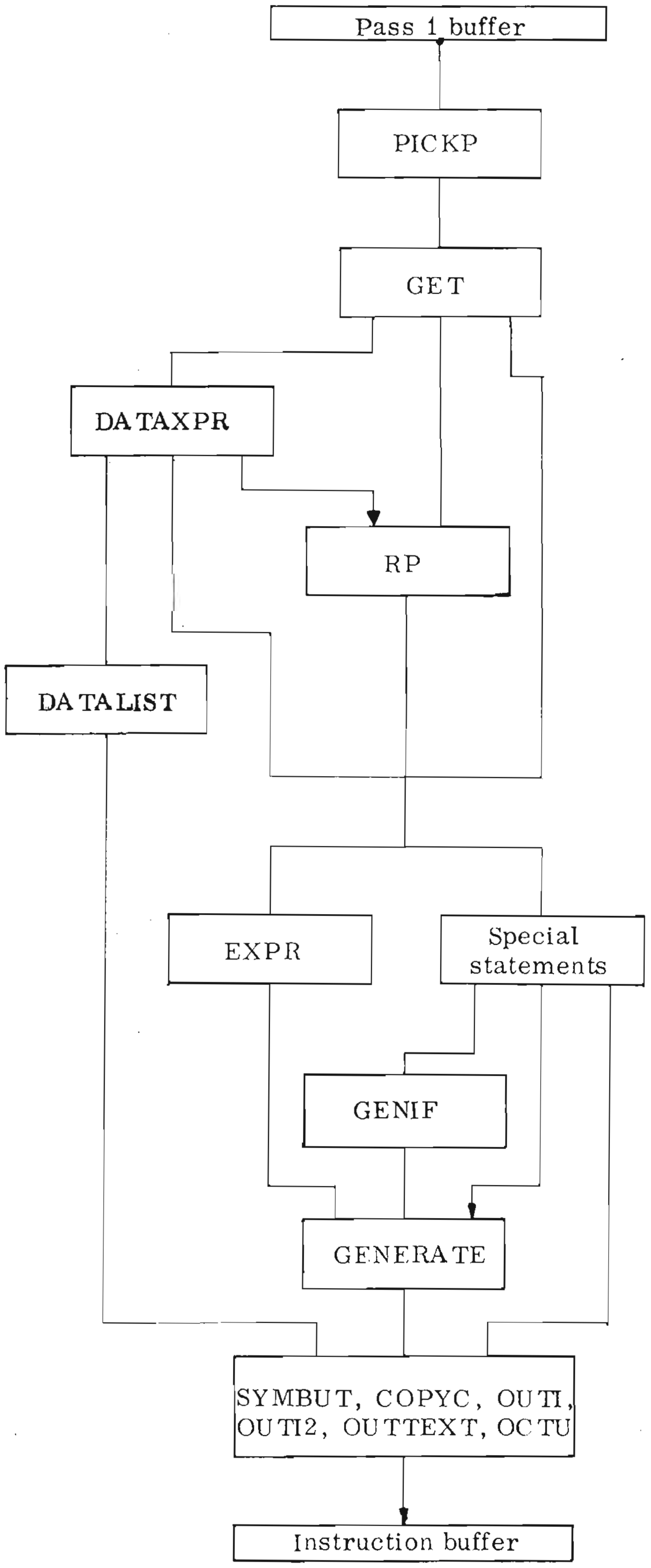
This subroutine is called from PASS1 to read a character. The parity bit is reset. The following characters are treated specially:

Tab	: To space
CR	: If online, a line feed is given
LF	: Ignored
Tape feed	: Ignored
End of medium (27)	: Go online

The text is copyed to the list device.

3.6 Executable Statements

Main information flow:



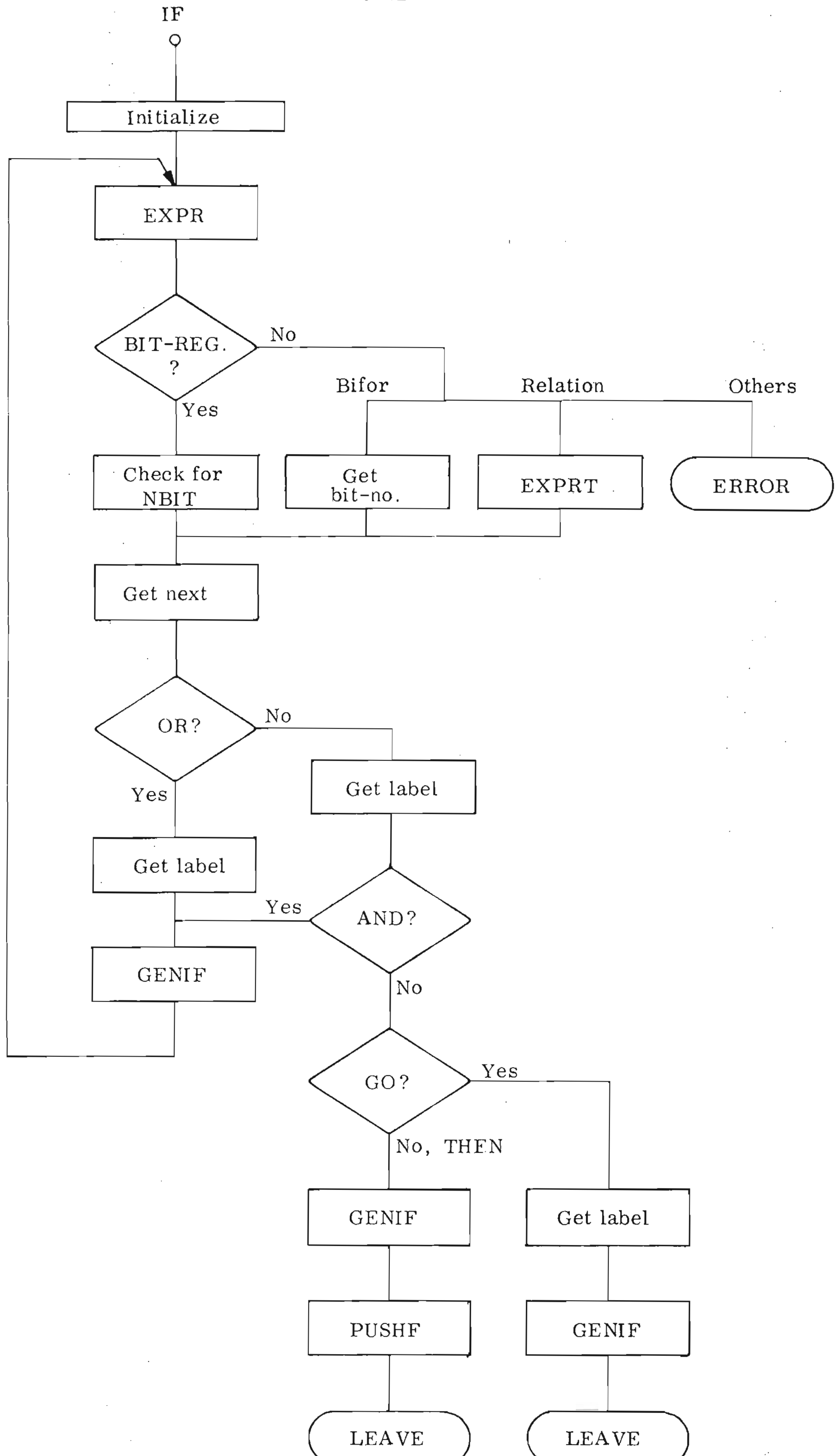
3.6.1 Arithmetic Statements

ARITS calls EXPR.

3.6.2 IF Statement

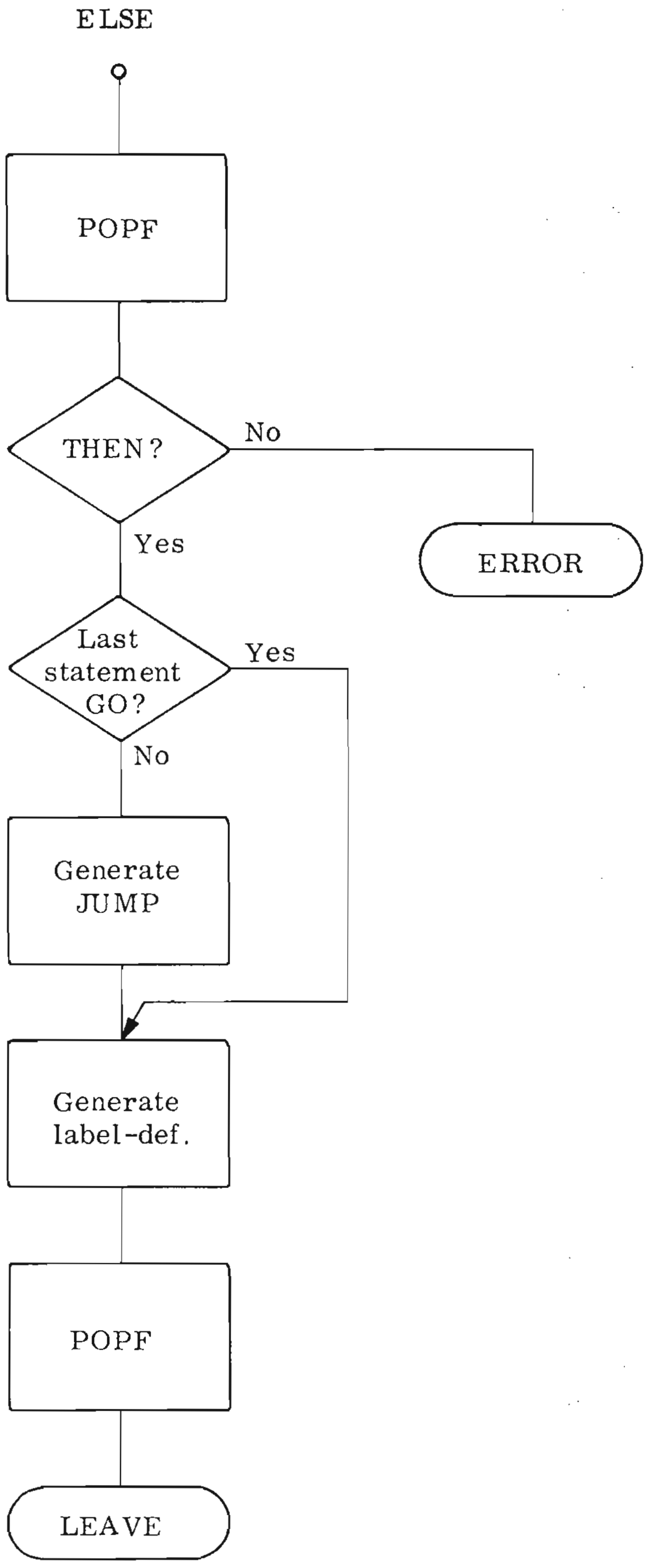
The conditional jumps go to "labels" starting with a comma, and thereafter four octal digits. The digits are converted from the number FMAX, which is incremented for each label generated. Example: ,0000 ,0001 ,0002. FMAX is also used by FOR statements. The nesting of IF and FOR is performed by the subroutines PUSHF and POPF, operating on a linked stack in the main table. FSTACK points to the first element.

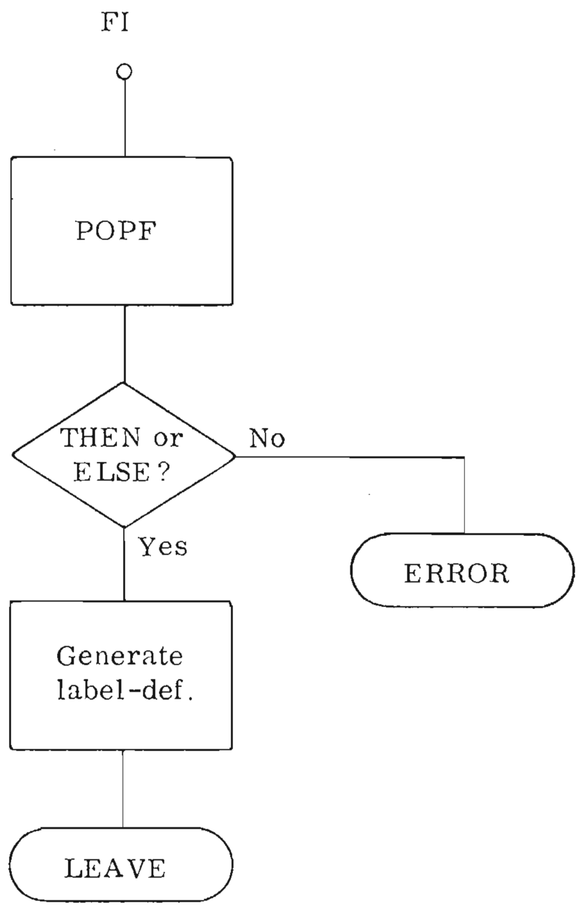
Each time a conditional jump is to be generated, the subroutine GENIF is called, outputting a conditional jump if possible, or a SKP followed by a JMP.


IF
Initialize
EXPR
BIT-REG. ?
No
Yes
Bifor
Relation
Others
Check for NBIT
Get bit-no.
EXPRT
ERROR
Get next
OR?
No
Yes
Get label
Get label
AND?
Yes
No
GENIF
GO?
Yes
No, THEN
GENIF
Get label
PUSHF
GENIF
LEAVE
LEAVE

3.6.3 ELSE - FI Statements

The ELSE - FI generate labels and jump to labels. If the last statement before the ELSE was an unconditional GO, no jump will be generated, i.e. ELSE will be dummy.




FI
POPF
THEN or ELSE ?
No
Yes
ERROR
Generate label-def.
LEAVE

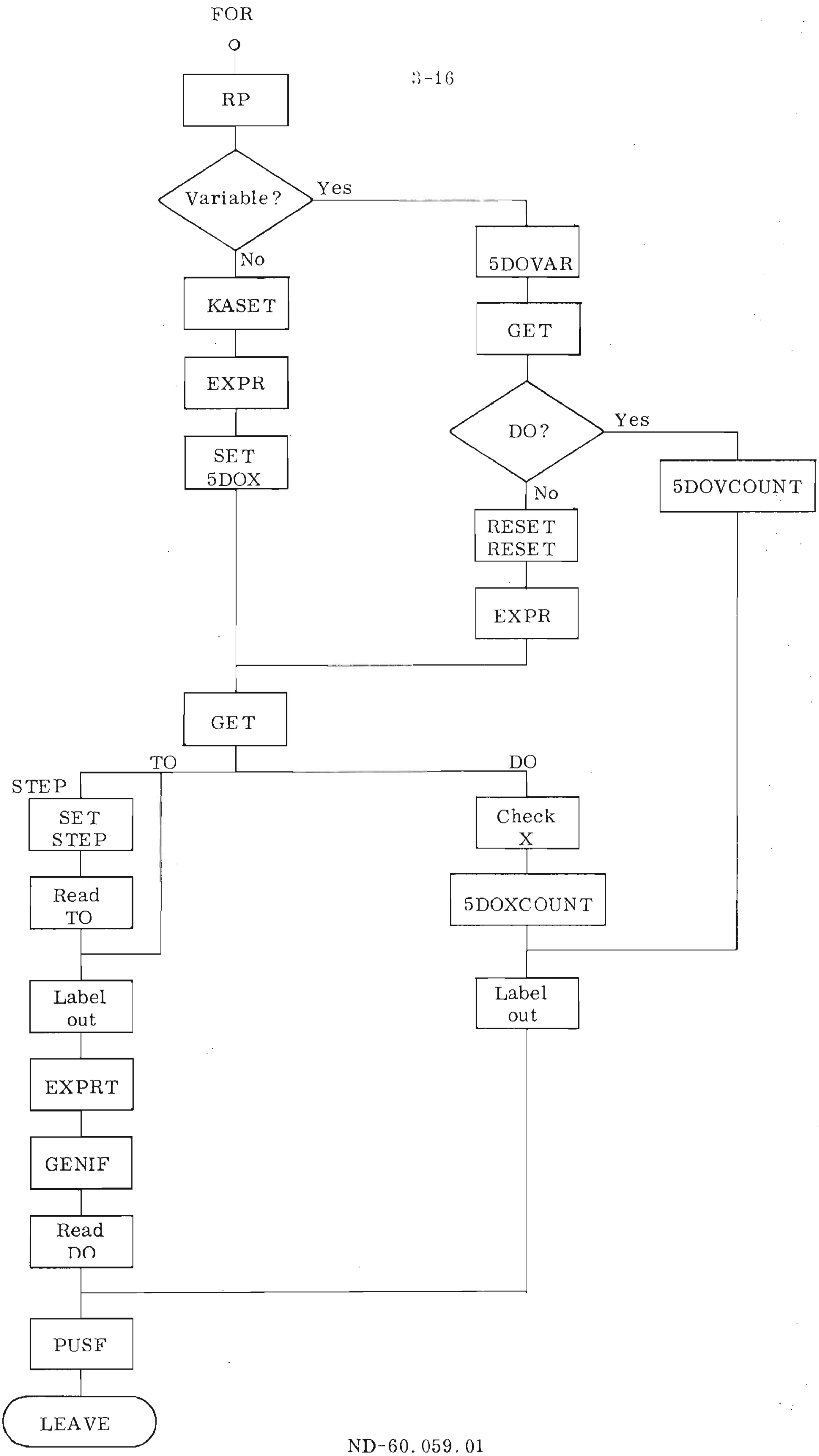
3.6.4 FOR Statement

The FOR statement generates many different sets of code. Normally, it generates a label for return jump and a jump till after the corresponding DO.

The variable FTYPE has the values

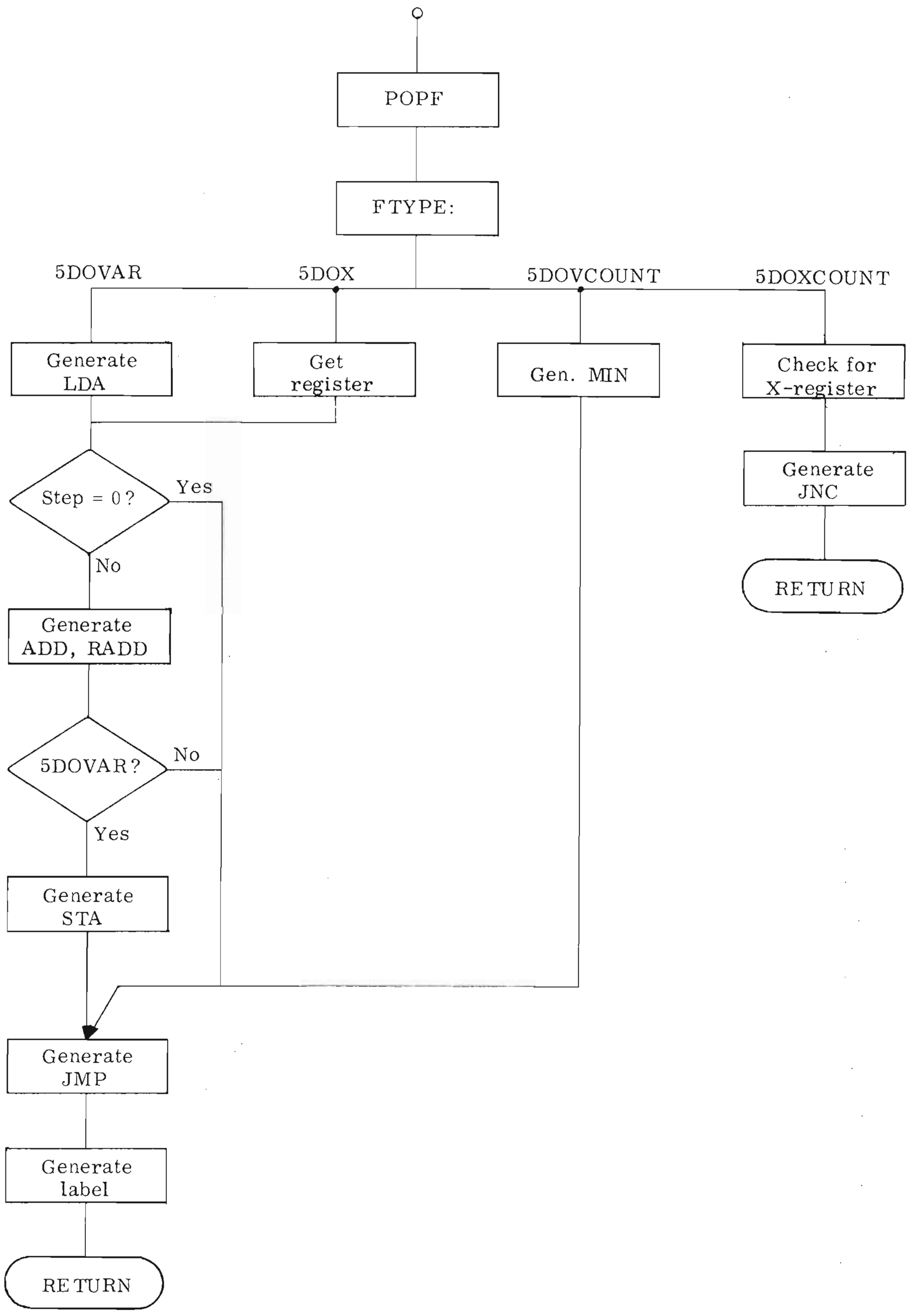
5DOX	-	Register as stepping value
5DOVAR	-	Variable as stepping value
5DOXCOUNT	-	X register as counting value
5DOVCOUNT	-	Variable as counting value

The subroutine PUSHF pushes some information into the same nesting stack as IF - FI use. The information is retrieved at OD.


FOR
RP
Variable?
Yes
No
5DOVAR
KASET
GET
EXPR
DO?
Yes
SET 5DOX
No
5DOVCOUNT
RESET RESET
EXPR
GET
TO
DO
STEP
SET STEP
Check X
Read TO
5DOXCOUNT
Label out
Label out
EXPRT
GENIF
Read DO
PUSF
LEAVE

3.6.5 OD Statement

The OD statement generates a jump back to FOR. In case of a FOR - TO construction, the stepping value is modified, else a count and test for zero are generated (JNC or MIN).


POPF
FTYPE:
5DOVAR
5DOX
5DOVCOUNT
5DOXCOUNT
Generate LDA
Get register
Gen. MIN
Check for X-register
Generate JNC
RETURN
Step = 0?
Yes
No
Generate ADD, RADD
5DOVAR?
No
Yes
Generate STA
Generate JMP
Generate label
RETURN

3.6.6 CALL, GO, and LABEL Statements

The CALL statement generates a JPL to the subroutine. If the entry-point is not defined, it is assumed to be global (generating JPL I (ENTR)).

The GO statement generates a JMP to a label. If the label is not defined, it is assumed to be local (generating JMP LABEL).

The LABEL statement defines a label, generating LABEL = ✱.
The undefined -flag is zeroed in the symbol table entry, and so is the library flag for conditional compiling.

CALL, GO, and IF statements call a common subroutine, CALGO. This subroutine reads a label reference. It checks for a preceding FAR, then flagging an indirect jump, but without setting the symbol permanently external.

3.6.7 Executable Expressions

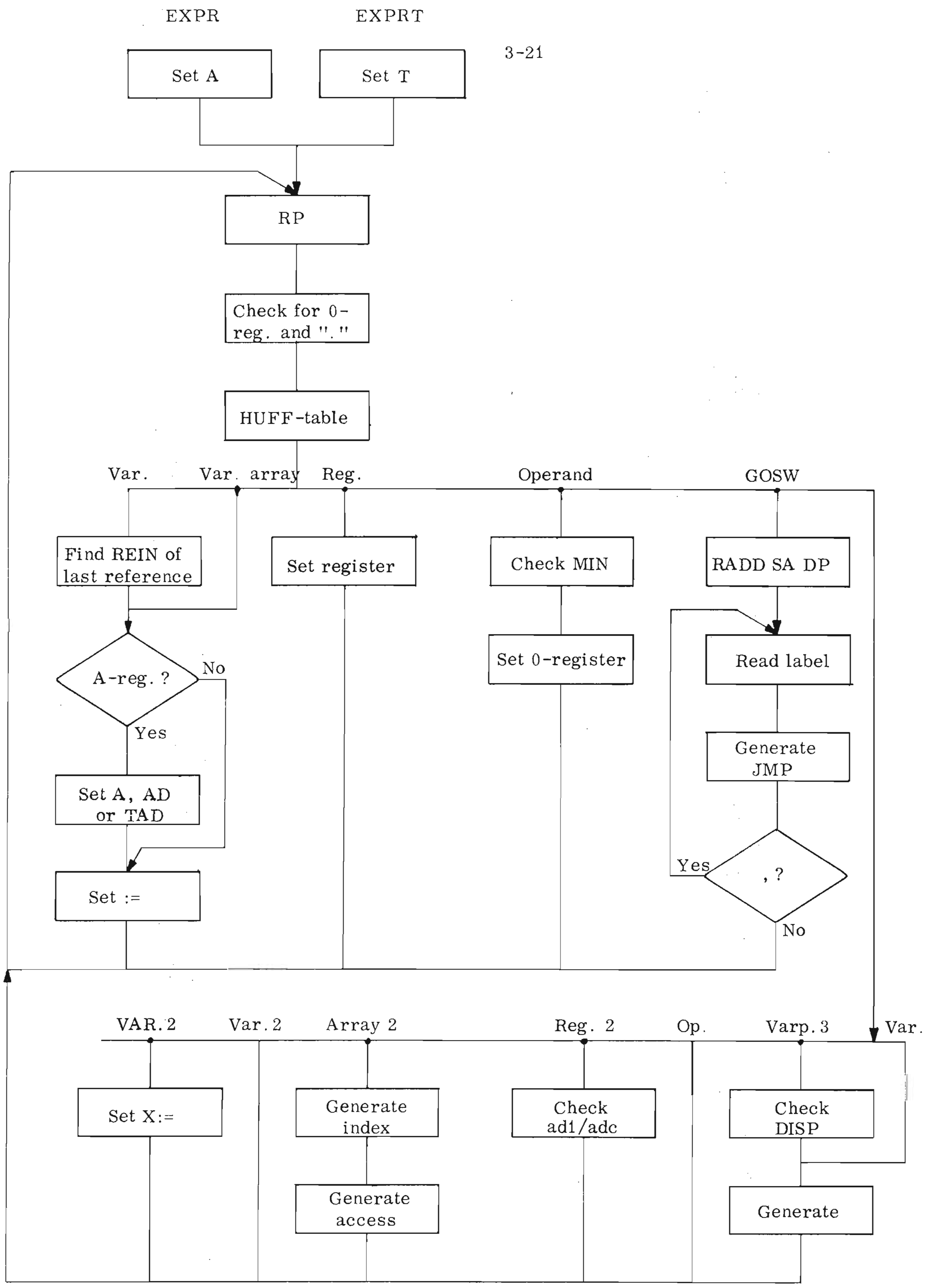
This subroutine is called from Arithmetical statements, and IF and FOR statements. It has two entrypoints:

EXPR	-	setting A, AD or TAD as default register
EXPRT	-	setting T as default register

The subroutine checks the syntax, normally generating an instruction each time an operator is found (calling GENERATE).

The main program logic is controlled by a state label.

Input / State	Const./ variable	Const./ variable "."	Register	Array	Arithm. operator	GOSW	Others
Start 0	Set register 2	Look for last ref. set reg. 2	Set register 1	Set register 2	Check for MIN 2	Set A register generate jumps 0	Set register return 0
After operand 1					Set operator 2	Generate jumps 0	Return 0
After operator 2	Generate 1	Set x:= generate 3	Check for AD1/ADC generate 1	Generate index generate 2	Generate 1	Generate, generate jumps 1	Generate return 1
After "." 3	Generate 1	Generate 3					


EXPR
EXPRT
Set A
Set T
RP
Check for 0-reg. and "."
HUFF-table
Var.
Var. array
Reg.
Operand
GOSW
Find REIN of last reference
Set register
Check MIN
RADD SA DP
A-reg. ?
No
Yes
Set 0-register
Read label
Generate JMP
Set A, AD or TAD
Set :=
Yes
, ?
No
VAR.2
Var.2
Array 2
Reg. 2
Op.
Varp.3
Var.
Set X:=
Generate index
Check ad1/adc
Check DISP
Generate access
Generate

3.7 Declaration Statements

3.7.1 BASE - DISP - SUBR

These statements control the addressing mode (AMODE) of the enclosed variables. The variable SAMODE tells which value AMODE should have when a variable is declared.

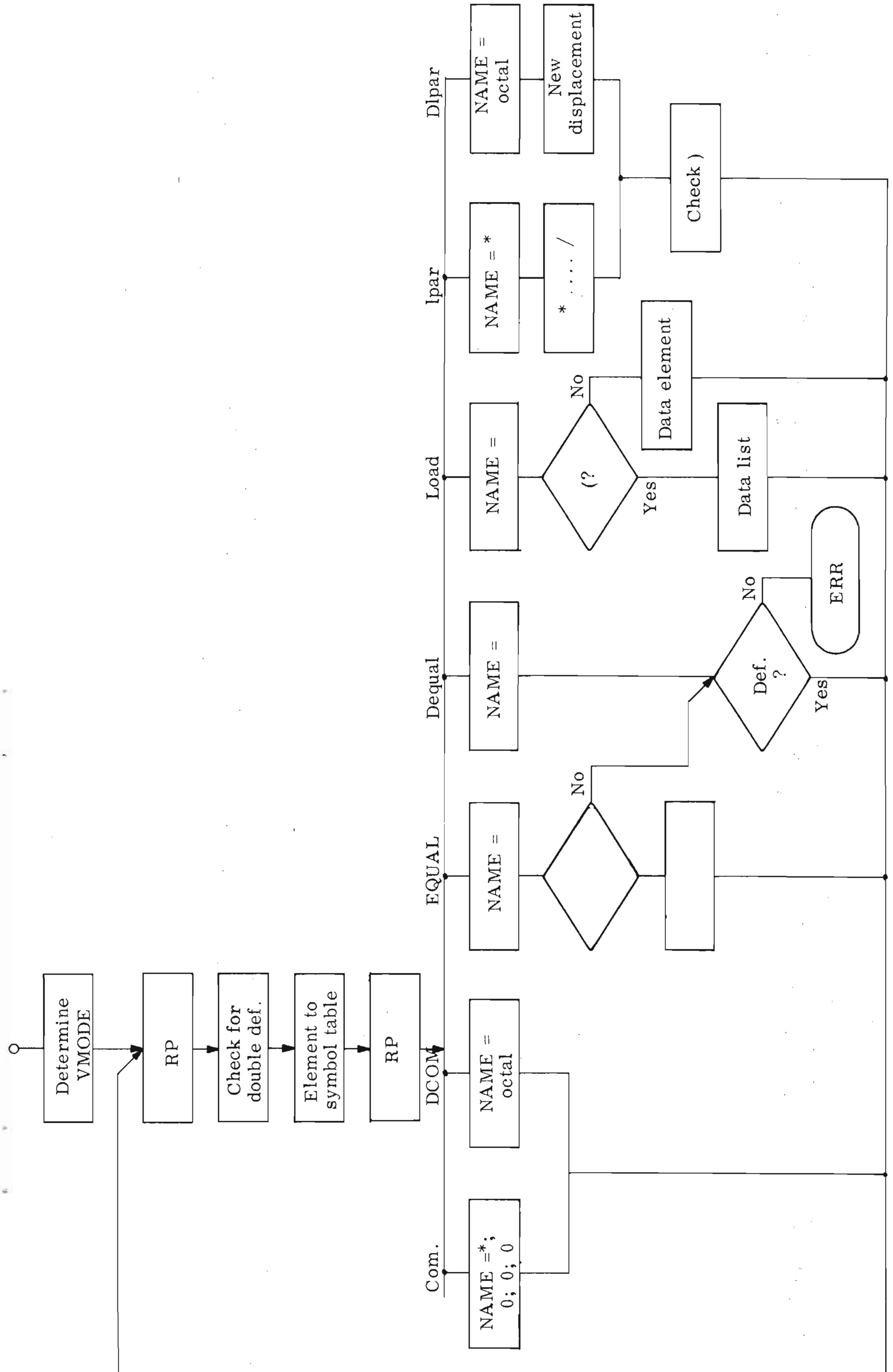
In RBUS all local variables are removed, calling PCLEAR.

3.7.2 Declaration of Variables: INTEGER, DOUBLE, and REAL

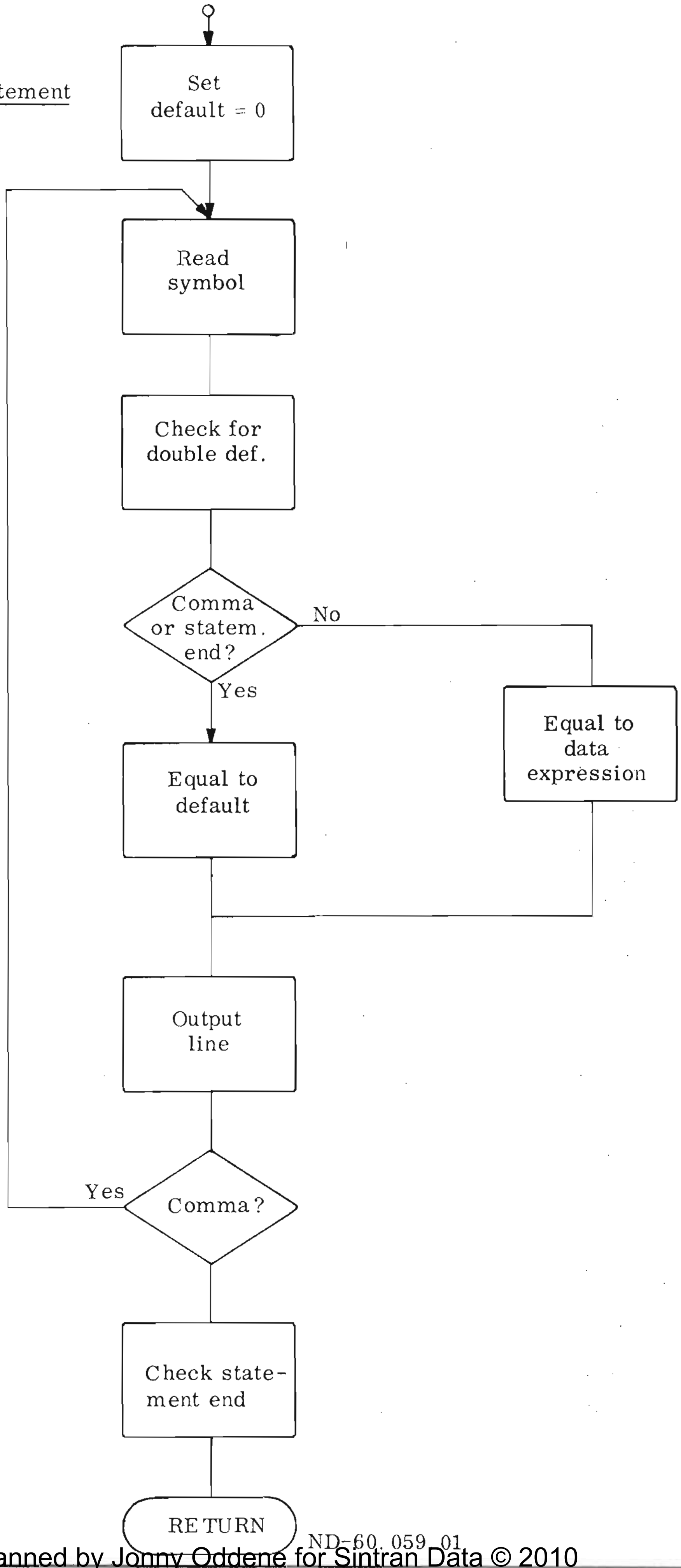
The routine checks for ARRAY and/or POINTER, adjusting the variable's mode (VMODE). Then the list of variables is processed. The variables are checked for double definition, and the possible initialisations are processed. The processing is different for DISP variables.

The following occurrences are used in a switch:

Comma	:	,
Statement end		
Equal sign	:	=
Load	:	:=
Left bracket	:	(


Determine VMODE
RP
Check for double def.
Element to symbol table
RP
Com.
DCOM
EQUAL
Dequal
Load
Ipar
Dlpar
NAME =*; 0; 0; 0
NAME = octal
NAME =
NAME =
NAME =
NAME = *
NAME = octal
No
Def. ?
Yes
No
ERR
(?
Yes
No
Data list
Data element
* . . . . /
New displacement
Check)

3.7.3 SYMBOL Statement



3.7.4 Data Expressions

The subroutine DATAXPR is called when a data expression is supposed to occur in an executable expression (called from RP) or in a declaration. The subroutine can treat:

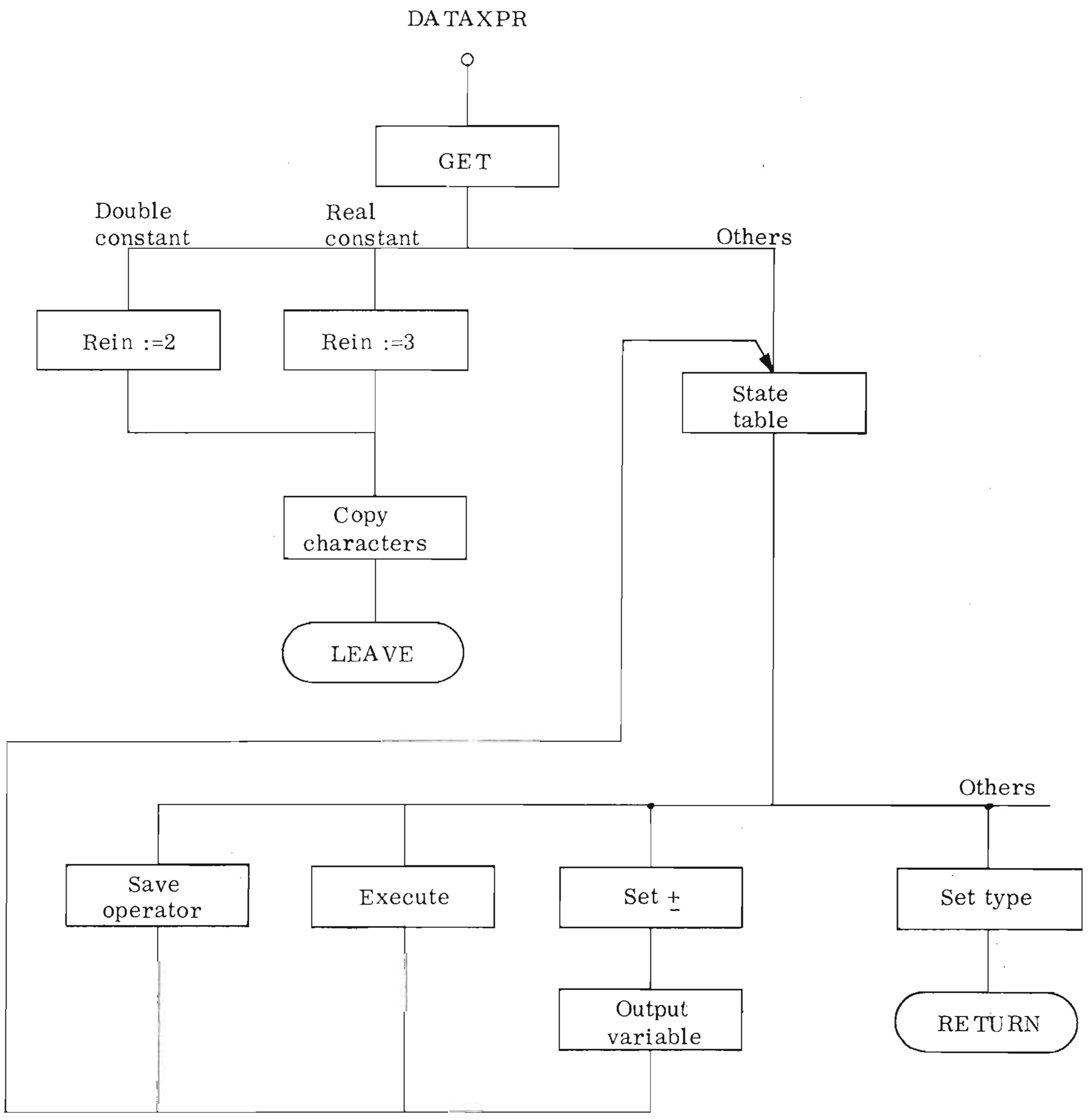
a) Double constants

b) Real constants

c) Integer expressions - where the operators can occur: +, -, * and \

The syntax is checked by a state table:

Input / State	Operator	Integer constant or symb.	Variable	Others
Start 0	Store operator 1	Execute 2	Output value + variable 3	╳
After operator 1	╳	Execute 2	Output value + variable 3	╳
After constant 2	Store operator 1	╳	╳	Return, type=5 const. 0
After variable 3	Store operator 4	╳	╳	Return, type=5 const.
After var. and operator 4	╳	Output ± and value 3	Output ± and variable 3	╳


DATAXPR
GET
Double constant
Real constant
Others
Rein :=2
Rein :=3
State table
Copy characters
LEAVE
Others
Save operator
Execute
Set ±
Set type
Output variable
RETURN

3.8 Element Fetching Routines

3.8.1 RP

The main purpose of this routine is to fetch an operand in an executable expression. The operand can be a variable, constant, register or data expression (in quotes). The routine determines the value of TYPE - the main grouping of elements, returning if also in the A-register. If a pointer enclosed by quotes ("point") is found, the addressing mode is modified to direct addressing.

The current level of the backtracking stack is recorded at the beginning and inserted at the end, so that a RESET after RP will reset to the situation when RP was called.

A special entry point, RP CHECK, assumes the expected type to be in the A-register at entry, giving error if the resulting type is not the same.

The result is placed in the output buffer.

See the flowchart on page 3-25.

3.8.2 GET

GET is the routine which gets an element from the pass 1 buffer. If the element is a symbol, it is looked up in the symbol table. If the type is found to be a MAC mnemonic, the symbol is prefixed by a comma and looked up anew, to avoid collisions wit MAC symbols.

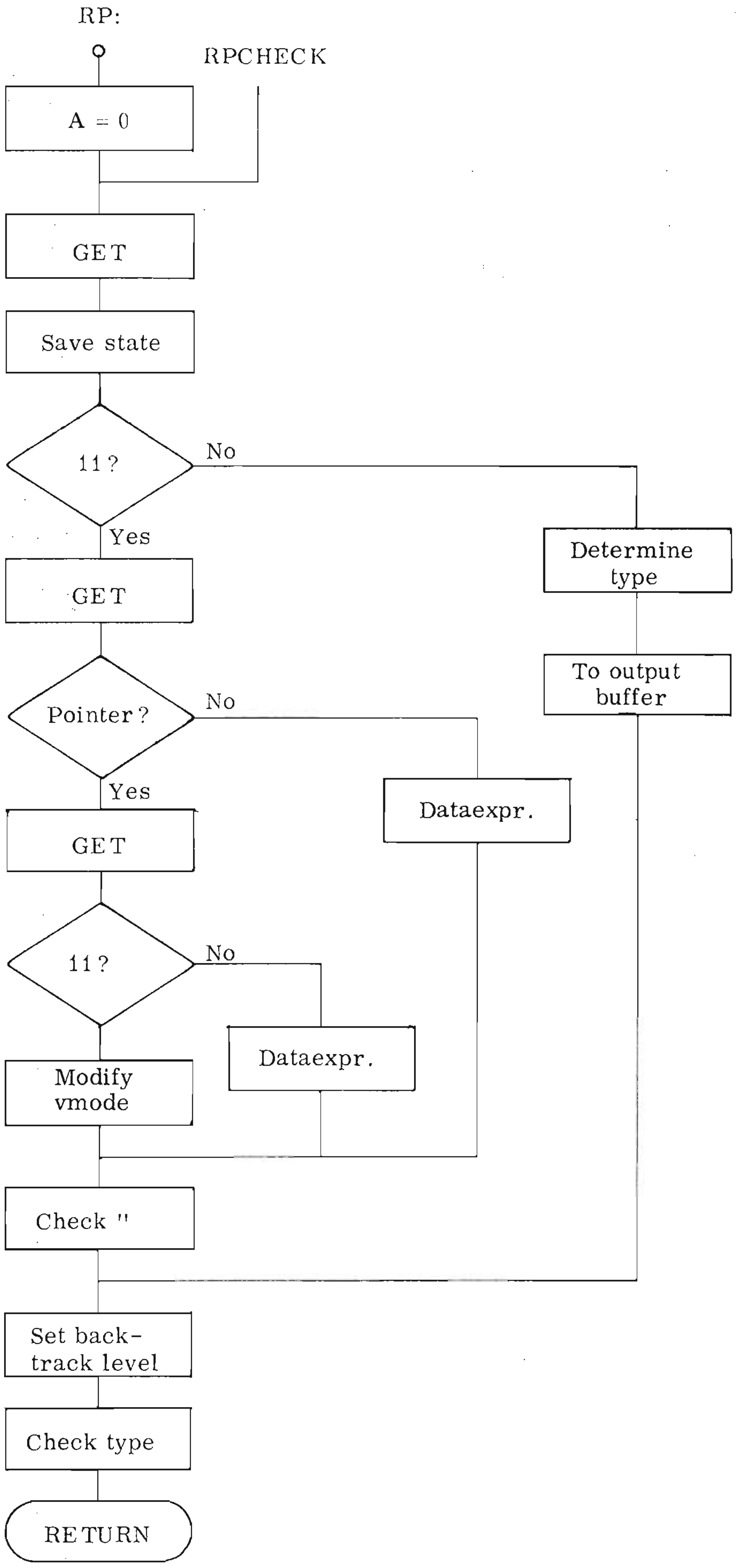
Octal and decimal constants are assembled and put into CVALUE; the same for # and # # character constants. Double and real constants and strings remain in the pass 1 buffer and may later be fetched by the COPYC routine.

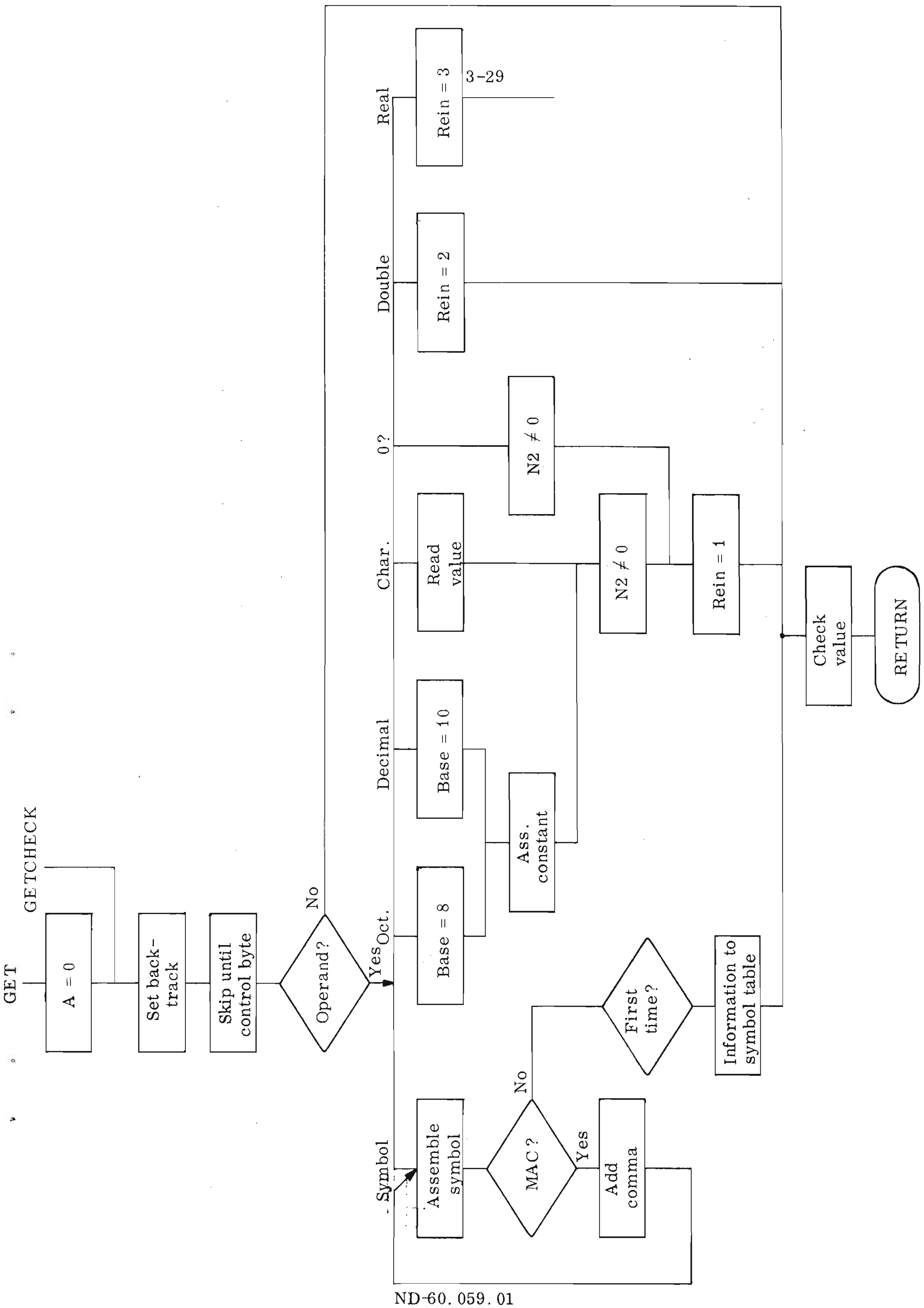
The entry point GETCHECK is used when a certain value is expected, being placed in the A-register. If non correspondence, an error message is given.

For each call of GET another element is added to the backtrack stack. For each RESET an element is removed, restoring the situation before GET.

If a constant = 0, N2 is set $\neq 0$, to flag that it may be interpreted as the zero register.

See the flowchart on page 3-26.


RP:
RPCHECK
A = 0
GET
Save state
11?
No
Yes
Determine type
To output buffer
GET
Pointer?
No
Yes
Dataexpr.
GET
11?
No
Dataexpr.
Modify vmode
Check "
Set back-track level
Check type
RETURN


GET
GETCHECK
A = 0
Set back-track
Skip until control byte
Operand?
No
Yes
Symbol
Oct.
Decimal
Char.
0?
Double
Real
Base = 8
Base = 10
Ass. constant
Read value
N2 ≠ 0
N2 ≠ 0
Rein = 1
Rein = 2
Rein = 3
Check value
RETURN
Assemble symbol
MAC?
Yes
No
Add comma
First time?
Information to symbol table

3.9 Table Routines

3.9.1 CLTAB

This routine resets the whole symbol table. It links the fixed symbols; therefore it must be called when the compiler is started (called from MCLEAR).

3.9.2 SEARCH

The symbol table is searched for a symbol, which is placed in N1 and N2. The last 3 bits in the symbol is used as an hash index.

If the symbol is not found, an entry for it is allocated in the table, where it gets the type 5 udef.

3.9.3 ALLOCATE

If the free-list is non-empty, an entry is fetched from it; else it is taken from the top of the symbol table. The location TPCHECK contains the value of the last word in the symbol table. If it has been changed, the table has been destroyed (overlap from user program.

3.9.4 PCLEAR

This subroutine is called at RBUS, removing all local variables from the symbol table, generating)KILL for them. It scans all the lists, looking for entries with LOCFLAG set. If a local symbol is undefined, an error message is written.

3.9.5 CODE - DECODE

CODE puts information into a table element. TARI points to the element.

If it is a constant (SYMBOL), CVALUE will be saved. For variables and arrays CVALUE contains TARI for the BASE variable in case of base addressing.

DECODE unpacks the information the same way.

3.9.6 PUSHF - POPF

These routines operate on the IF-FOR-stack, making the proper nestings. The IF-FOR-stack is a linked list in the main symbol table. If POPF is called and the list is empty, an error message is given.

An element has the form:

link
FTYPE
FNUM
FCONTROL
FSTEP

3.9.7 PUSHVAR - RESET

These routines operate on the backtrack stack (see Section 2.3.4).

They use CODE - DECODE for putting information in and out.

PUSHVAR is called from the beginning of GET, recording the state before GET was called. RESET is called several places in the compiler.

3.10 Code Generating

3.10.1 GENIF - Generate a conditional Jump

The subroutine is called at IF and FOR statements, generating a conditional jump if possible, otherwise a skip and jump.

The information is transferred to GENIF through the following global variables:

RELOP	- relational operator
BITNO	- used at bit-skip
REG1	- the register to the left of the relational operator
REGISTER	- the register to the right of the relational operator
THENTYPE	- 5THEN, 5OR, 5AND, 5GO
FNUM	- label number
ORLAB	- label number in case of OR
TARI	

The array AXTAB is used to determine if conditional jumps can be used.

	Bit 1:	Bit 0:	
Operator	X-reg.	A-reg.	
>	0	0	
>=	1	1	JXN, JAN
=	0	1	JAF
><	1	1	JXZ, JAZ
<=	0	0	
<	0	1	JAP

If AXTAB is equal to zero for some operators (> and <=), the source and destination register must be swapped, and a different operator must be used. The proper operator is found in the array MODREL.

See the flowchart on next page.

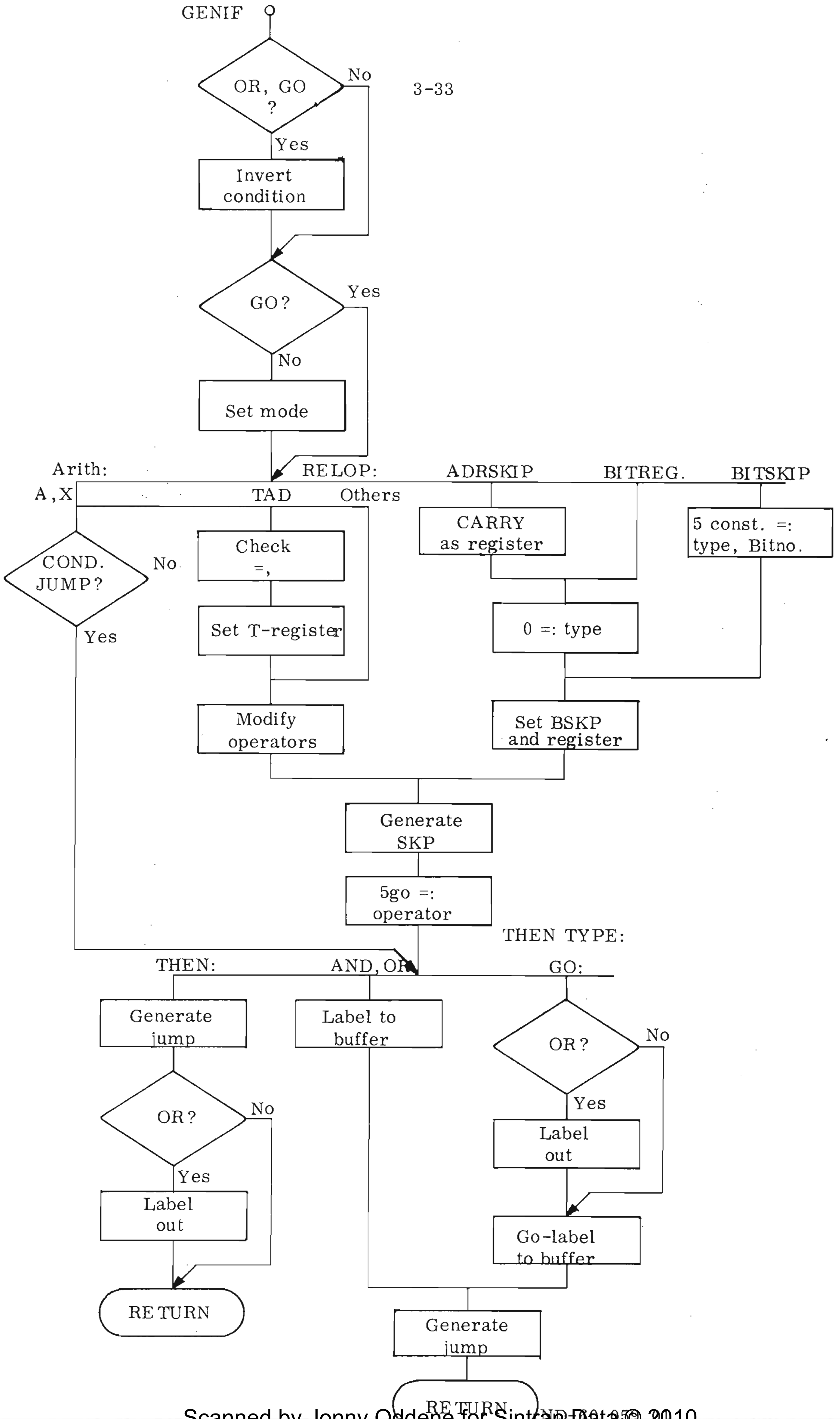
3.10.2 GENERATE - generate an Instruction

This routine is called each time an instruction is to be generated. It is called from several places in the compiler.

The information is transferred to GENERATE through the following global variables:

REGISTER	- primary register
OPERATOR	- additional information - ADC, AD1, CM1, CM2
TYPE	
REIN	- REAL, INTEGER, DOUBLE
VALUE	
CVALUE	
AMODE	- addressing mode: BASE, PSID
VMODE	- variable mode: variable, array, pointer

The instruction is built by putting parts of the instruction into fixed places in an instruction buffer, IBUFA.


GENIF
OR, GO ?
No
Yes
Invert condition
GO?
Yes
No
Set mode
Arith:
A,X
RELOP:
TAD
Others
ADRSKIP
BITREG.
BITSKIP
COND. JUMP?
No
Yes
Check =,
Set T-register
Modify operators
CARRY as register
0 =: type
5 const. =: type, Bitno.
Set BSKP and register
Generate SKP
5go =: operator
THEN TYPE:
THEN:
AND, OR
GO:
Generate jump
OR?
No
Yes
Label out
RETURN
Label to buffer
OR?
No
Yes
Label out
Go-label to buffer
Generate jump
RETURN

The format is like this:

0	1	2	3	4	5	6	7	10
L D	A	, X	I	, B		(	V A	R I A
C O	P Y		S A		D D		A D	1
S K	P		S T		D A		F Q	L
B S	E T		O N	E	D D		0 0	0 1 7 0
A A	X						0 0	0 0 1
B S	K P		Z R	O	S S	K		
S A	D		Z I	N	S H	R	1 0	

The characters are inserted, 2 or 4 at a time, using the displacements T1, T2, ··· or DT1, DT2, ··· .

The routine GENERATE determines which type of instruction it is to make. If TYPE for instance is a constant, it will either generate a memory reference instruction, an argument instruction, or a register operation (only for values -1, 0, or 1). In the latter case the array OPT1 is used, using the value and the operator as indices.

Value / Operator	-1	0	1
:= :=	COPY CM1	COPY	COPY AD1
+ +	RADD CM1	RADD	RADD AD1
- -	RADD AD1	RADD	RADD CM1

For memory reference instructions the array MODTAB is used to determine the addressing mode of the instruction. AMODE and VMODE are used as indices in the array. An array element contains some bit flags:

Bit 0:	,B
Bit 1:	I
Bit 2:	,X
Bit 3:	(
Bit 4:	Base modification

A negative value means error.

MODTAB:

AMODE / VMODE	Local	Global	Base	Disp.	X disp.
Variable	-	I (	,B - BAS	,B	,X
Array	I ,X (	I ,X (	,X ,B-BAS	,X ,B	Error
Pointer	I	Error	I ,B - BAS	I ,B	Error
Array pointer	Error	Error	,X I, B-BAS	,X I, B	Error

See the flowchart on next page.

3.11 Object Output

The code generating routines (GENERATE, declaration processors) put characters into the buffer IBUFA, either direct by using displacements, or by using subroutines (OUTI, OUTI2, OCTU, · · ·). When the necessary information is ready, the whole line is output to the object device by calling LINUT. Trailing spaces are ignored. Then the buffer is cleared (filled with spaces).

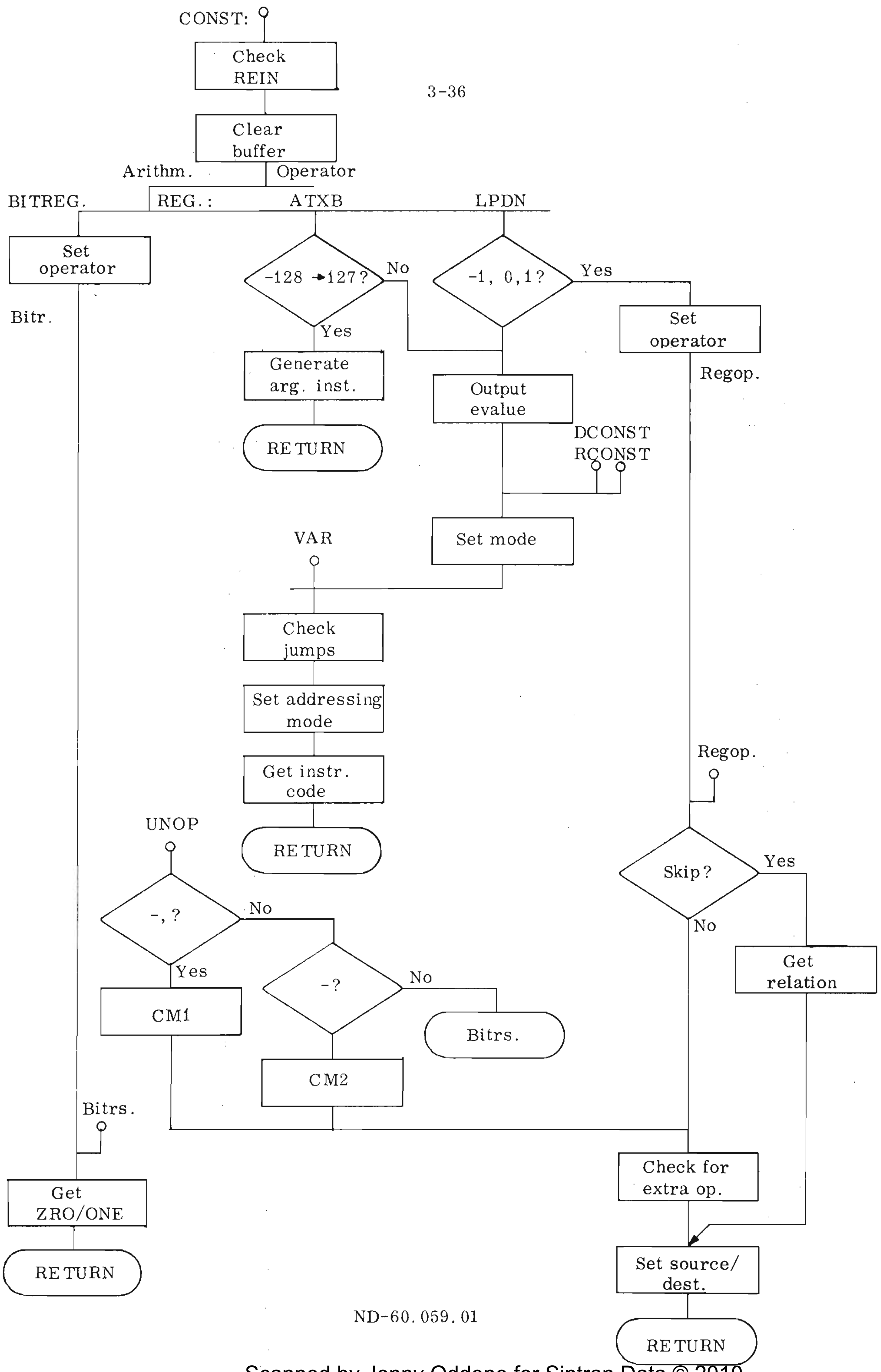
The following entry points exist:

LINUT	-	output line and clear buffer
CLIBUF	-	clear instruction buffer
CLINST	-	clear instruction buffer and position the buffer pointer to T7 (operand place)
OUTCH	-	output one character with even parity

3.12 Auxiliary Routines

3.12.1 ENTER - LEAVE

The routines operate on the subroutine stack (see Section 2.3.5).


CONST:
Check REIN
Clear buffer
Arithm.
Operator
BITREG.
REG.:
ATXB
LPDN
Set operator
-128 → 127 ?
No
Yes
-1, 0, 1 ?
Yes
Set operator
Bitr.
Generate arg. inst.
RETURN
Output evalue
Regop.
DCONST
RCONST
Set mode
VAR
Check jumps
Set addressing mode
Get instr. code
RETURN
Regop.
UNOP
Skip ?
Yes
No
-, ?
No
Yes
-?
No
CM1
Bitrs.
Get relation
CM2
Bitrs.
Check for extra op.
Get ZRO/ONE
RETURN
Set source/ dest.
RETURN

3.12.2 Symbol Output Routines

FNAME	- A number in A is converted to octal and prefixed by a comma, making an internal label for IF-FOR. The instruction buffer is cleared beforehand.
SYMBUT	- The symbol in N1 - N2 is placed in the instruction buffer.
LABUT	- A label is assumed to be in the instruction buffer. The characters =* is then inserted, making a label definition, then outputting the line.
COPYC	- An element is copied from the pass 1 buffer to the instruction buffer, typically a string.

3.12.3 Text and Number Routines

OUTTEXT	- The A-register points to a standard MAC-string to be copied to the instruction buffer.
OCTU	- A number in the A-register is converted to octal and put into the instruction buffer.

3.12.4 Character Input/Output Routines

OUTI	- One character to the instruction buffer.
OUTI2	- Two characters to the instruction buffer.
PACKP	- One byte to the pass 1 buffer.
XPACK	- One control byte (bit 7 set) to the pass 1 buffer.
LOADBYTE	- T points to string start, X contains the byte count. One character is fetched to the A-register, like the instruction LBYT in NORD-10.
PICKP	- One character is fetched from pass 1 buffer. If the buffer is empty, PASS1 is called.
REBUF	- The character fetch pointer in the PASS1 buffer is backspaced.

3.12.5 Searching Routines

Two routines are used to fine the element number of wanted element in an array. The location after the call points to the array, and the value is in the A-register. The index is returned in the A-register. The search is terminated when the match is found or when a negative number has occurred. Typically the arrays to be searched are terminated with -1.

There are two entry points:

SRCHARR	-	Search for A equal to element.
SRCHINT	-	Search for A between the leftmost byte and the rightmost byte of the element.

Name searching routine, GETNAME.

The location after the call points to the start of an array, the elements of which containing three words.

Value
Name 2
Name 3

The routine searches for match between the A-register and the value in the first word of the element. If match is found, the name is returned in AD. If not (negative value is found), error is reported.

4 MAINTENANCE

4.1 Generating

As the compiler compiles itself, some assembly-version must exist, either the original hand-compiled (NPS) or a version which has been compiled. If a binary version is provided, generating a new compiler version is like this:

- Compile the compiler (NPC)
- Assemble the object code

Three symbols are undefined:

EXX	-	Exit from the compiler at @ EOF (MCALL 0 for TSS)
,INBT	-	Standard INBT
,OUTB	-	Standard OUTBT

There are two main entry points:

NPL	-	Start address
ONLIN	-	Restart address

4.2 Modifying the Compiler

The present version generated NORD-1 code. It will probably be useful to generate special NORD-10 and NORD-20 code. Then, mainly the routine GENERATE must be changed, because this routine contains the checking for legal operations and code generating. To obtain magnitude skip in NORD-10 the routine GENIF must also be modified.

As the MAC assembler may be equipped with more standard symbols, these should be added to the MAC-symbol table of NORD-PL to avoid collisions.

The I/O interface is performed by standard INBT/OUTBT. It can easily be used for co-routine interface to MAC, or for implementing macros.

A/S NORSK DATA-ELEKTRONIKK
Lørenveien 57, Oslo 5 - Tlf. 21 73 71

COMMENT AND EVALUATION SHEET

ND-60.059.01

NORD PL
PROGRAM DOCUMENTATION

In order for this manual to develop to the point where it best suits your needs, we must have your comments, corrections, suggestions for additions, etc. Please write down your comments on this pre-addressed form and post it. Please be specific wherever possible.

FROM: ______________________________

____

____

– we make bits for the future

NORSK DATA A.S BOX 4 LINDEBERG GÅRD OSLO 10 NORWAY PHONE: 39 16 01 TELEX: 18661