INTERNAL MAINTENANCE SPECIFICATIONS

1700 MASS STORAGE FORTRAN 3.1

under MSOS 4.0

TABLE OF CONTENTS

CHAPTER 1

TABLE OF CONTENTS

1.0 GENERAL DESCRIPTION OF COMPILER

1.1 Summary

1700 Mass Storage Fortran is oriented toward the users of medium-sized 1700's. Input is from punched cards or paper tape; output is on a printer or paper tape punch and is formatted accordingly. Card-image input and line-image output may be on magnetic tape. Intermediate scratch files are on the disc. 1700 Mass Storage Fortran is released in versions for 16K or 20K machines, and runs under the Mass Storage Operating System.

The compiler consists of four passes over the source code or its equivalent, accomplished in four phases, called A, B, C, and D/E. (The fourth pass is performed by either Phase D or Phase E, according to whether the user wants an assembly-language listing output.) Each phase consists of a "root" which is core-resident throughout the phase, and zero or more "local" subroutine groups which share the same core area and are read from disc as needed.

1.2 Phase A Tasks

These tasks read the source input, convert it to statements expressed in an internal code, and assign a statement number to the statement.

The statements are syntax checked and itemized into an output file. The loop structure table (LOOPT), equivalence table (IEQV), specification table (ISTAB), and much of the symbol table (SYMTAB) are built.

1.3 Phase B Tasks

PHASE B reads the output of PHASE A and generates pseudo-code from it. Pseudo code is similar to assembler input except that the index to be used in an indexed instruction is not specified and the addressing mode is not specified.

1.4 Phase C, D, E Tasks

Phases C and D/E are a two pass assembler. The output from Phase B is read. Index registers are optimally assigned. One word relative addressing is maximized. Relocatable binary output with futures and an assembly listing are produced.

1.5 Partial Compilation In Special Cases

The number of phases actually executed depends on the options selected and success of the compilation. If only a source listing is requested, only phase A is executed; this provides a complete syntax check without producing machine code which is not wanted. Similarly, only phase A will be executed if it detected a fatal error in the program; a fatal error, by definition, precludes the generation of correct machine code, so there is no reason to generate the code. A third abbreviation of the compilation process occurs with stacked compilations when the 'X' option is selected but the 'A', 'M', and 'P' options are not. Detection of a fatal error in a program clears the 'X' option for the stack, and therefore all following programs will be processed only by phase A.

1.6 Local Optimizations

Many local optimizations of the code generated have been implemented.

1. Index registers are optimally assigned.
2. Relative addressing is used where possible.
3 Storage is allocated to maximize relative addressing. For example, some arrays are put into the middle of code.
4. All simple FORTRAN provided functions are inserted in-line (e.g., IABS or AND).
5. A comprehensive analysis of IF statements is made. In the code generated there is cognizance of a transfer from the IF to the label of the next statement and if this statement is a GO TO.

2.0 Phase A Tasks and Subprograms

2.1 Summary

2.2 Overlay Programs

2.2.1 Flow of Control

2.2.1.1 Versions of GO routine

2.2.1.2 Versions of LOCAL routine

2.2.2 PHASEA Routine and Related Subroutines

2.2.2.1 Subroutine GNST
2.2.2.2 Subroutine TYPE
2.2.2.3 Subroutine PEQVS
2.2.2.4 Subroutine PLABEL
2.2.2.4.1 Subroutine RDLABL
2.2.2.5 Subroutine OPTIONS
2.2.2.6 Subroutine OUTENT
2.2.3.2 Subroutine ENDDO
2.2.3.3 Subroutine SAVEID

2.3 General Pass Routines

2.3.1.1 Subroutine SYMBOL
2.3.1.2 Subroutine STORE
2.3.1.3 Subroutine GETSYM
2.3.1.4 Subroutine CNVT
2.3.1.5 Subroutine SYMSCN

2.3.2 Field Processing Routines

2.3.2.1 Subroutine GETF
2.3.2.2 Subroutine GETC
2.3.2.3 Subroutine GPUT
2.3.2.4 Function IGETCF
2.3.2.5 Subroutine STCHAR

2.3.3 Special Field Analyzing Routines

2.3.3.1 Subroutine CONSUB
2.3.3.2 Subroutine CFIVOC
2.3.3.3 Subroutine CKIVC
2.3.3.4 Subroutine CKNAME

2.3.4 Diagnostic Handling Routine

2.3.4.1 Subroutine DIAG
2.3.4.2 Subroutine CONV
2.3.4.3 Subroutine PACK
2.3.4.4 Subroutine PUNT
2.3.4.5 Subroutine PRNTNM
2.3.4.6 Subroutine IOPR

2.4 Statement Specific Phase A Routines

2.4.1 DIMENSION Statements - Subroutine DIMPR
2.4.2 COMMON Statements - Subroutine COMNPR
2.4.3 TYPE Statements - Subroutine TYPEPR

2.0 PHASE A TASKS AND SUBPROGRAMS

2.1 Summary

The driver {FTN} is entered by the Job Processor control statements. The driver indirectly calls the Phase A main program {PHASEA}. The source lines are read in formatted ASCII, converted to statements in an internal code, given a sequential statement number, and stored in the buffer ISORS {routine GNST}. The statement type is computed {TYPE}. The statements are sent to their appropriate processors, syntax checked, itemized into an output buffer {see 7.5}, and output {routine OUTENT}. Upon encountering the first executable statement {or an END statement}, the EQUIVALENCE statements, which are in a tabulated form in core, are processed and the IEQV table is formed {routine PEQVS}. Control is returned to FTN through the routine GOA, which either proceeds to read in and execute Phase B, or restarts Phase A to read the next source program.

2.2 OVERLAY PROGRAMS

Each phase consists of a main section, or 'root', which is resident in core during the entire execution of its phase, and, if necessary, a number of subroutine groups, or 'local files', which are read in from the disc as needed. The local files overlay one another in core and are so programmed that they are 'read-only'... no program in a local file contains variables which are carried over from one call to the next, so the file may be overlaid by another and recalled later without being saved in the meantime. (Some programs which are in local files do have variables which must be preserved; these variables are placed in blank common, which is not overlaid during a phase.) In effect, the local files are an extension of the phase concept, except that a program in one local file can, with certain restrictions, call one in another local file of the same phase.

2.2.1 FLOW OF CONTROL

The compiler is entered by calling the relocatable program FTN from the system library. FTN in turn calls the file FORTAl from the library and transfers to the first instruction of the routine GOA. From this point on control is transferred by normal calling sequences from GOA to the main routine of phase A, PHASEA, and to various subroutines. When it is necessary to use a routine in a different local file, the subroutine LOCAL is called, giving file number and entry number within file. LOCAL reads the overlay portion of the specified file into the overlay area. The overlay portion of each file begins with a list of the addresses of its entry points; LOCAL transfers to the n'th address in the list for the new file. When the routine thus called returns to LOCAL, LOCAL reads the original calling file back in and returns control to the program which originally called it. (In the 20K compiler, most calls are from PHASEA, in the root, to local subroutines. In this case, if the desired subroutine is already in core, LOCAL is bypassed; if not, the routine

is read and called by LOCAL, but when it returns, its file is left in core. Inter-local calls... between IOSPR and ARITH, and between ARITH and MODMXR... are handled as in 16K.) The local-file mechanicm is generally transparent to the compiler programs.

At the end of phase A, the main routine (PHASEA) returns to GOA, which in turn returns to FTN. FTN reads in the first file of the next phase (FORTB1) and jumps to GOB. There are similar linkages from phase B to C, and from C to D or E.

In the event that compilation should be aborted after phases A or B (see section 1.5), a call is made to entry SKIPIT in subroutine GOA or GOB. SKIPIT causes an alternate return from GO to FTN, such that FTN re-initializes itself just as though phase D/E had just been completed, and reads FORTA1 to start compiling the next program (if any) on the input stack.

The choice between sub-phases D and E is based on the user-selected options; D is used unless an assembly-language listing is requested (A or M option). In the 16K compiler, FTN always reads FORTD1 at the start of phase D/E, and the phase D/E local routine decides whether to use the D or E local routines. (The D and E roots are identical.) In the 20K compiler, GOC tests the options and makes an alternate return to FTN such that FORTD1 or FORTE1 is read.

2.2.1.0.1 FLOWCHART OF PROGRAM FTN

2.2.1.1 VERSIONS OF 'GO' ROUTINE

In each phase, the subroutine GO is called by FTN; it calls the initialization entry of the I/O package, then the main routine of the phase; then performs some I/O clean-up and returns to FTN. In addition to this, each GO has special functions which will be detailed below.

GOA checks a flag set by FTN, to determine whether this is the start of a stacked compilation; if so, it STATUS'es the list device and sets a flag telling whether the device can accept standard carriage control; and it calls OPANAL to request options from the typewriter. If this is not the start of a stack, the first source statement is read and checked for being a MON; this saves rebuilding the phase A local files just to read a MON. (The statement, if not a MON, is left in the input work area for later processing by GNST.) Whether or not this is the start of a stack, the phase A local files must then be prepared. (A flag is tested to determine whether the files are still intact in the disc scratch area, as they would be if phase B had not been called; if so, LOCALS is called at entry LOCLZ2, rather than LOCLIZ, to just rebuild tables in core rather than rebuilding the scratch area.) PHASEA is then called. It will either return normally, or call SKIPIT; GOA's return address is adjusted accordingly. The WRITE entry in IOPR is called to force out the last scratch page, and GOA returns to FTN.

GOB clears the flag which indicates that the phase A local files are intact in the scratch disk area; interchanges the scratch file numbers so that the phase A output file becomes the phase B input file; in the 2.0A version, causes the phase B local files to be generated; calls phase B; and upon return, finishes the phase as GOA did.

GOC, in the 2.0A version, is similar to GOB. In the 2.0B version, after return from PHASEC (There is no SKIPIT entry), it chooses alternate returns to FTN depending on whether or not the A and M options are selected.

GOOD and GOE are identical to one another in function and are similar to GOB. However, after return from the main routine (PHASE6), there is no scratch page to be output; and, if the list device is a magnetic tape, an end of file is written.

The load-and-go file is built at the start of the scratch disk area, if the X option was specified, by the phase D/E routine NPUNCH. The compiler's scratch area "floats" on the growing load-and-go file; at the start of each compilation, GOA saves the address of the top of the file from the communication area in low core, and at the end of each compilation GOOD or GOE adds to that value the length of the current load-and-go output. (Also see discussion of disk allocation in description of subroutine IOPR.)

2.2.1.1.1 FLOWCHARTS OF SUBROUTINE GO

2.2.1.2 VERSIONS OF 'LOCAL' ROUTINE

References to non-resident subroutines (in phases A, B, D and E of the 2.0A compiler and phase A of the 2.0B pass through the subroutine LOCAL. Subroutines to be placed in "local files" are written so that any variables which must be preserved between calls are placed in blank common, so that the routine can be overlaid without first being saved. The non-resident routines are then grouped in files on scratch disk, and references to them are intercepted by dummy routines which call LOCAL. That routine reads the file containing the called routine, executes that routine, and returns control to the dummy's caller.

At the start of a phase containing local subroutines, the entry LOCLI3 is called; it transfers the local files from the library to scratch disk, so that they can be referenced directly rather than via GTFILE. Each file so transferred contains: (1) one word containing the address of ENDLOC, a dummy routine loaded at the end of each file so that the file length can be computed; (2) a list of words containing the address of each "entry point" of that file; and (3) the absolutized subroutines of the file.

A call to a local subroutine not presently in core goes through a dummy routine and becomes a

call to LOCAL with a file number and entry number. The present state of the overlay system is saved in a number of pushdown stacks. The desired file is read into the overlay area and the desired entry point is called. If that program causes a LOCAL call, the state saving and file reading is repeated; the number of levels of call allowed is limited by the size of the pushdown stacks. Upon return from a local file, the stacks are popped up one level. The calling file is read back in (except, in 2.0B, if the call was from the root), and the calling program is returned to.

A "flag" paramter is provided in LOCAL and has three quite different uses in different versions of the routine.

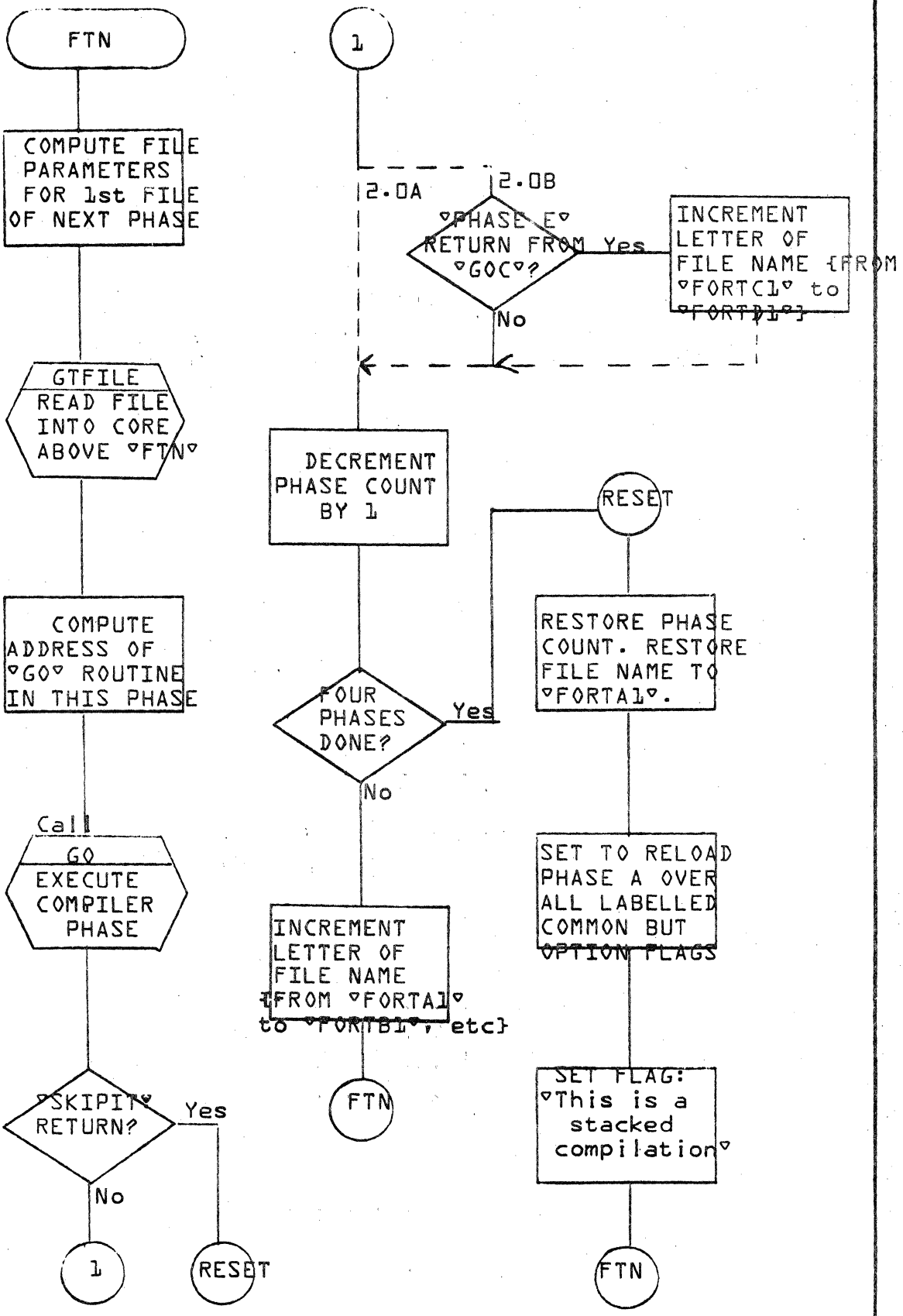
In 2.0A phase A, FLAG=1 means that the calling file has finished execution, and need not be read back in; for example, PHASEA in file A1 calls ARITH in file A2; just before returning, ARITH calls TREE in file A3; and just before returning, TREE may call MODMXR in file A6. There is no reason to reload TREE and ARITH after MODMXR is done, so the stacks are popped up three levels and control goes directly from MODMXR to PHASEA.

In 2.0A phase B, parameters must be passed to local routines; here, FLAG=1 means that the following word is a parameter address to be passed on.

In 2.0B phase A, almost all calls are from PHASEA in the root, to local routines; there FLAG=1 signals the case of a local-to-local call, where a different algorithm is needed to find the proper return address.

In the 3.0A and 3.1A compiler, routines STCHAR and GETC must be loaded at the same point in files FORTA1 and FORTA6 in order to have externals in SAVEID patched properly for both files. This is accomplished by a small BSS in DUMYA6 which is followed immediately by ERBPR to equal the length of DUMYA1.

2.2.1.2.1 Flowcharts of the "Local" Subroutines

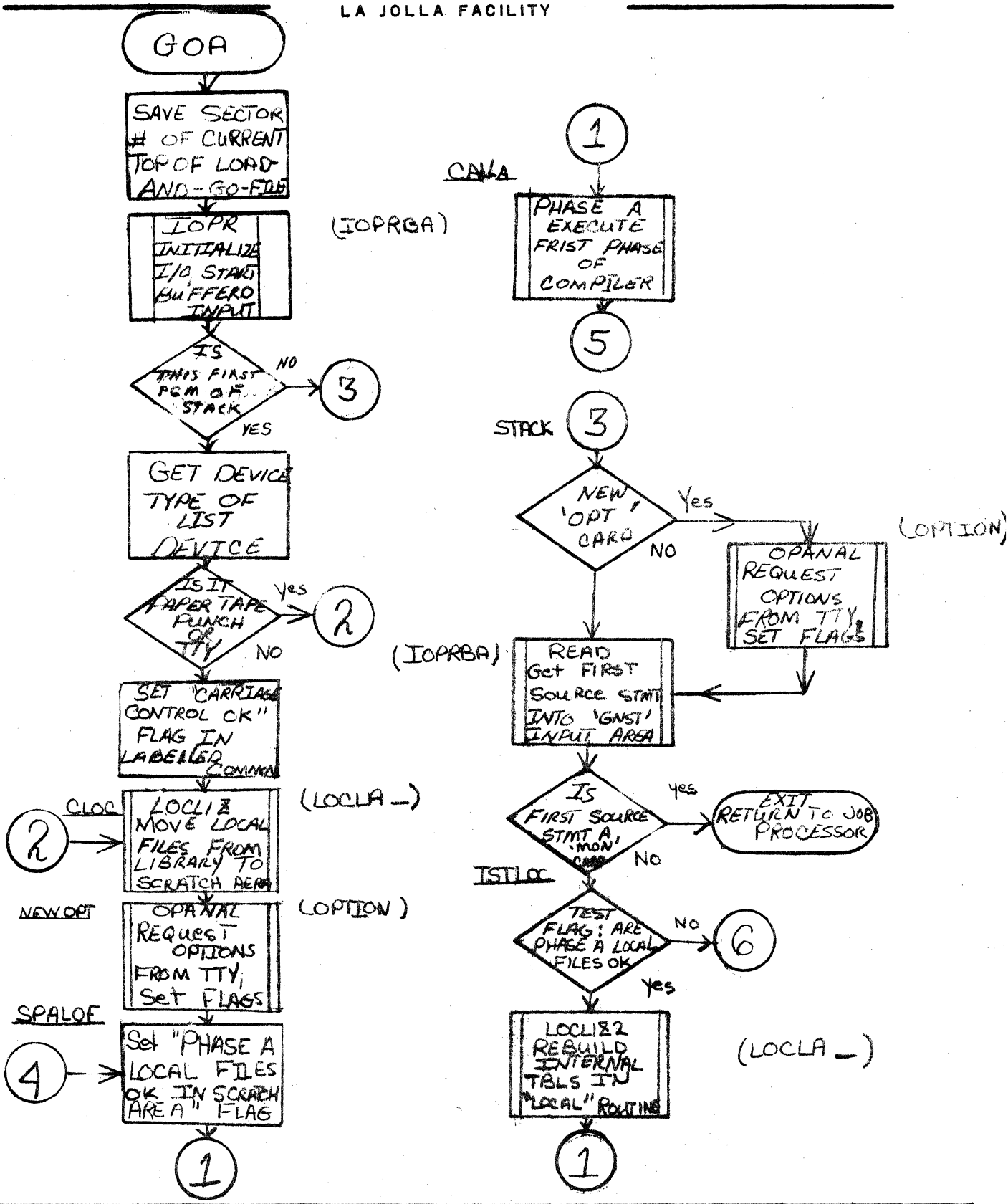


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
☐☒☐

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	MASS STORAGE FORTRAN	PAGE / OF /	
PROJECT NO.	PROJECT NAME	PROJECT MGR.	TASK NO.
REV.	APPROVED	DATE	ISSUE DATE
			NUMBER

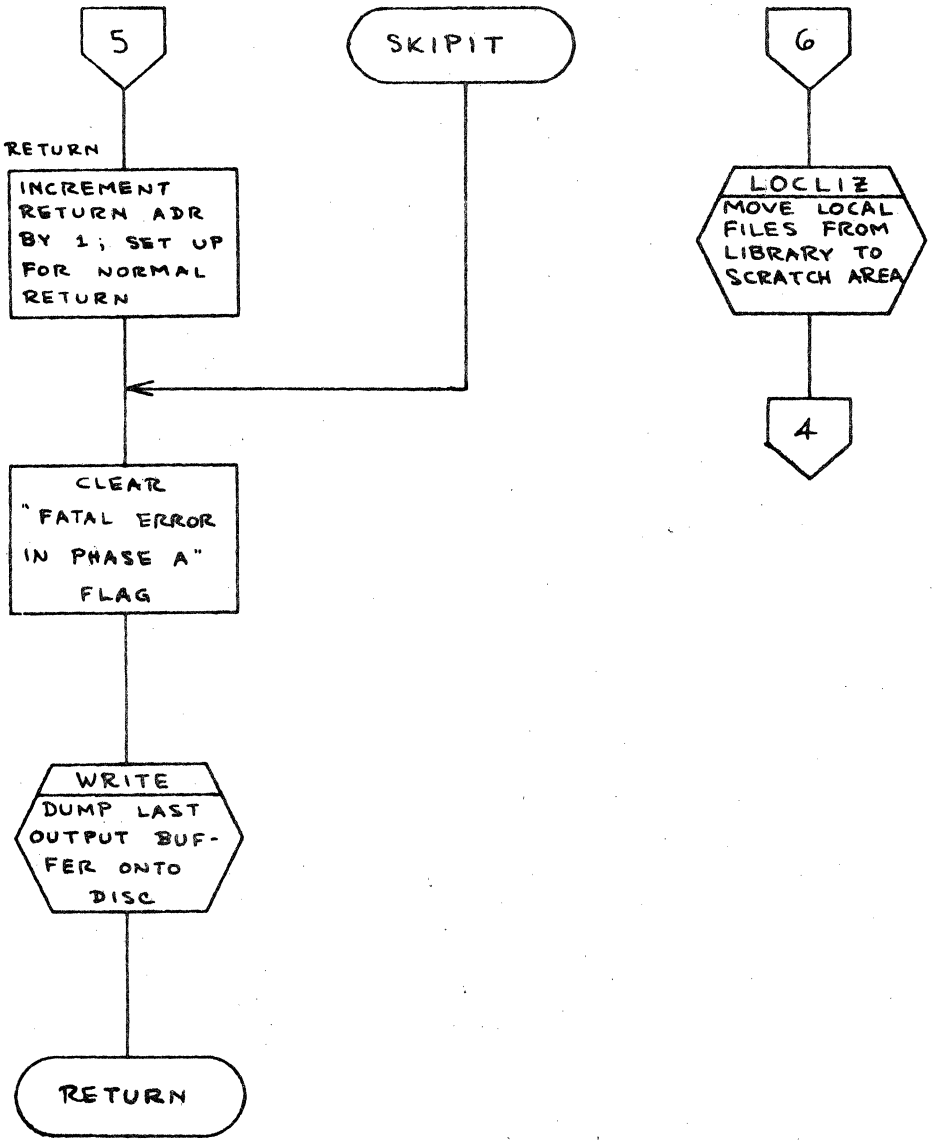
CONTROL DATA

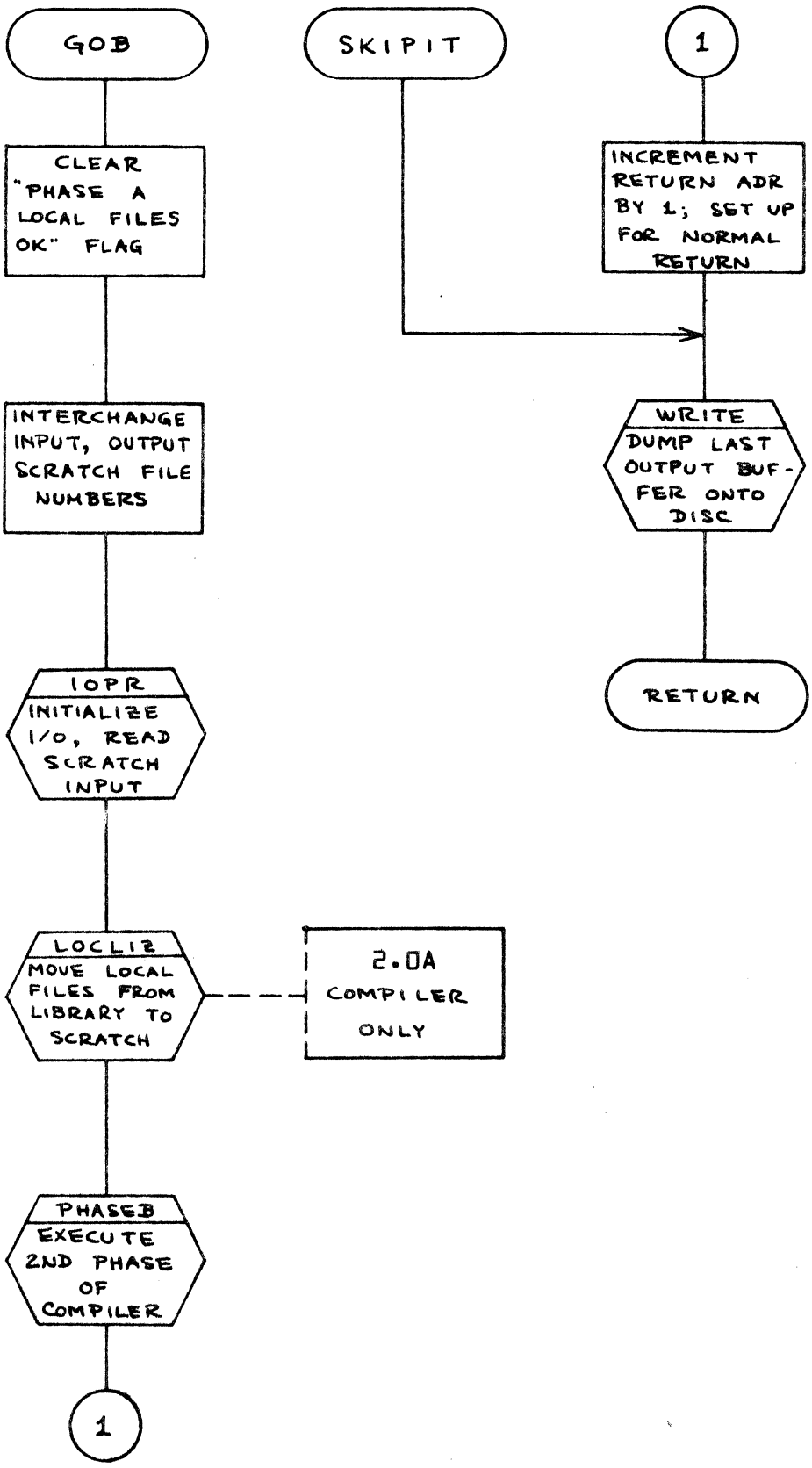
LA JOLLA FACILITY

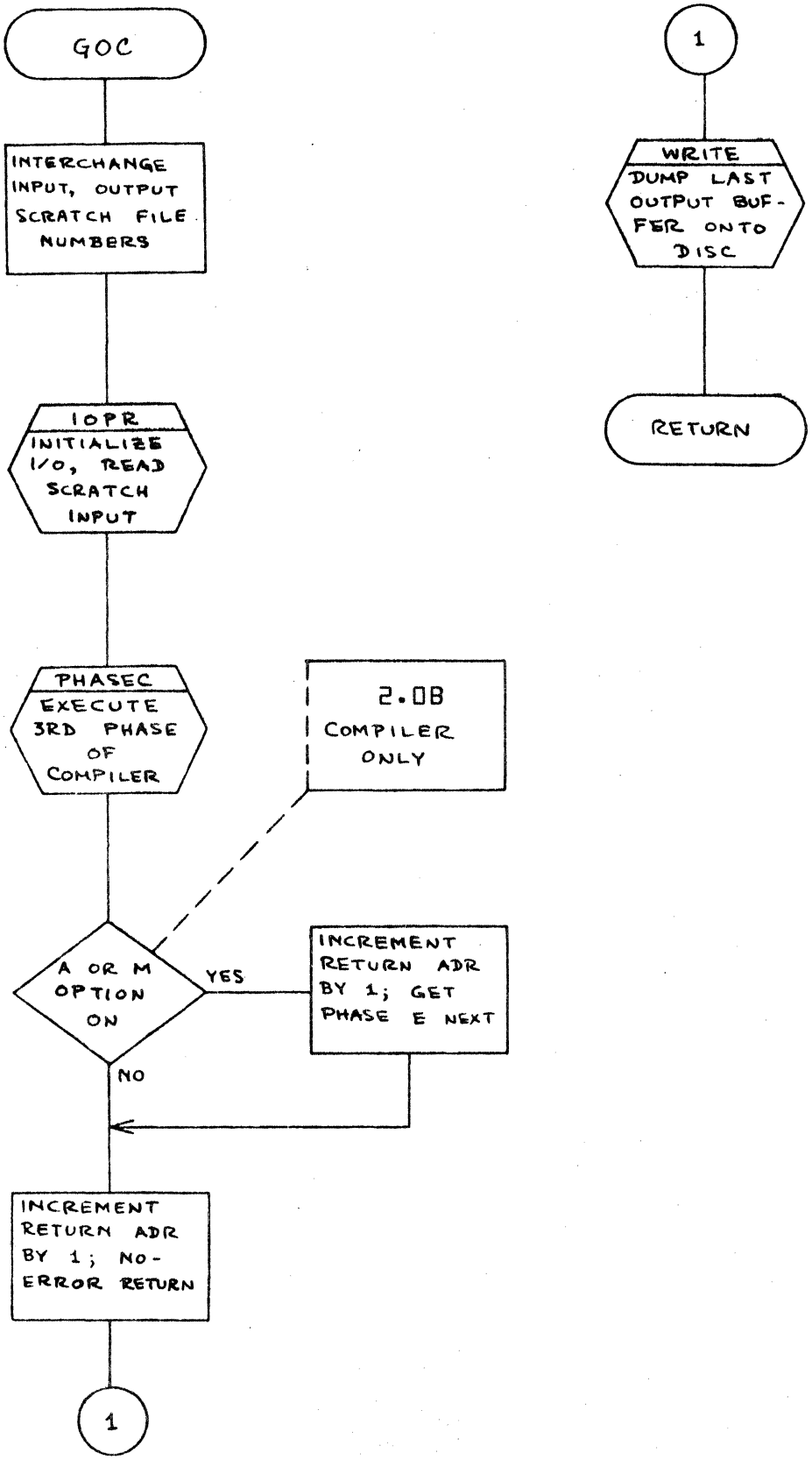


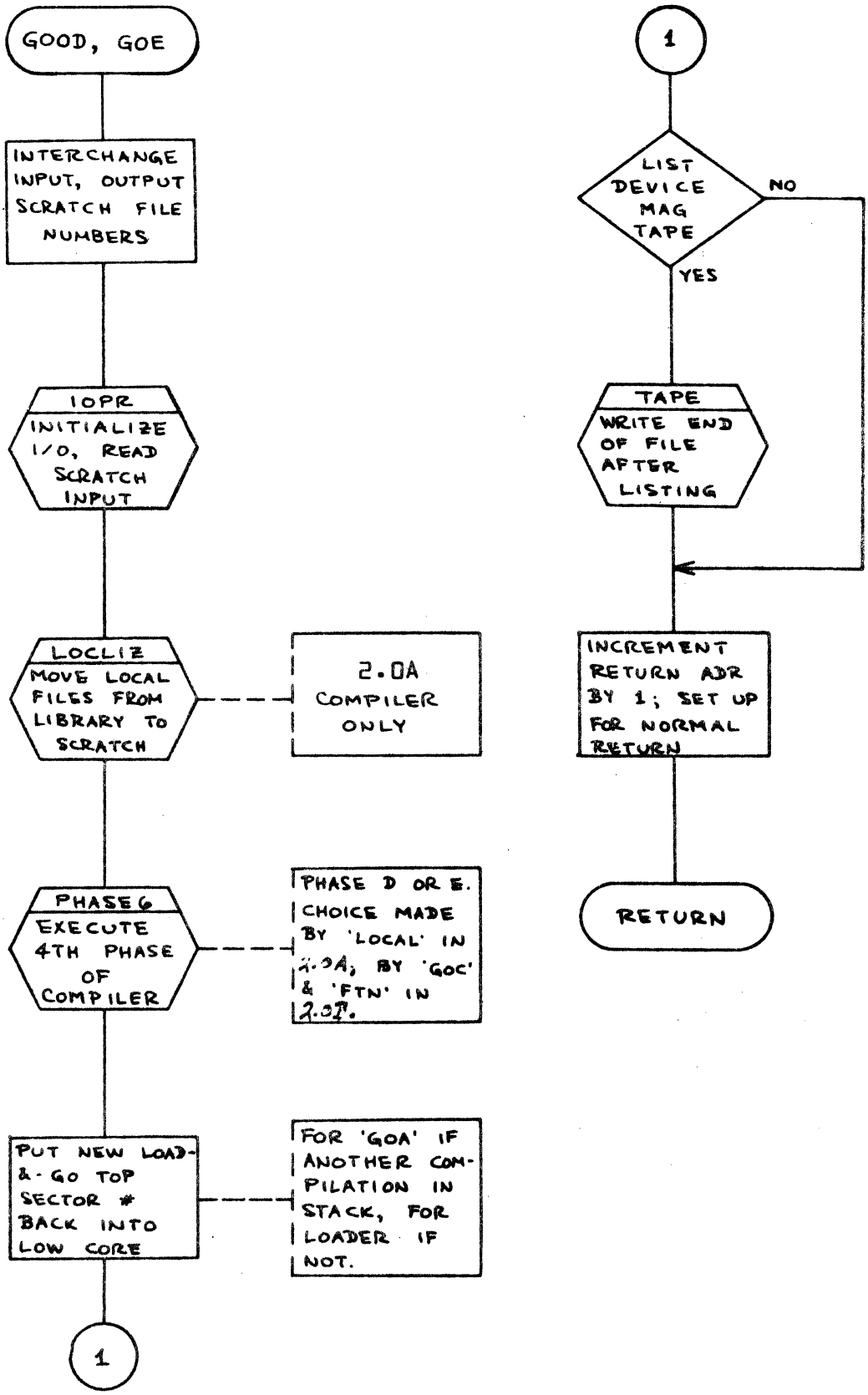
TITLE	GOA	DRG. NO.
		REVISION
DRAWN BY	PROJ.	DATE	SHEET / OF 2

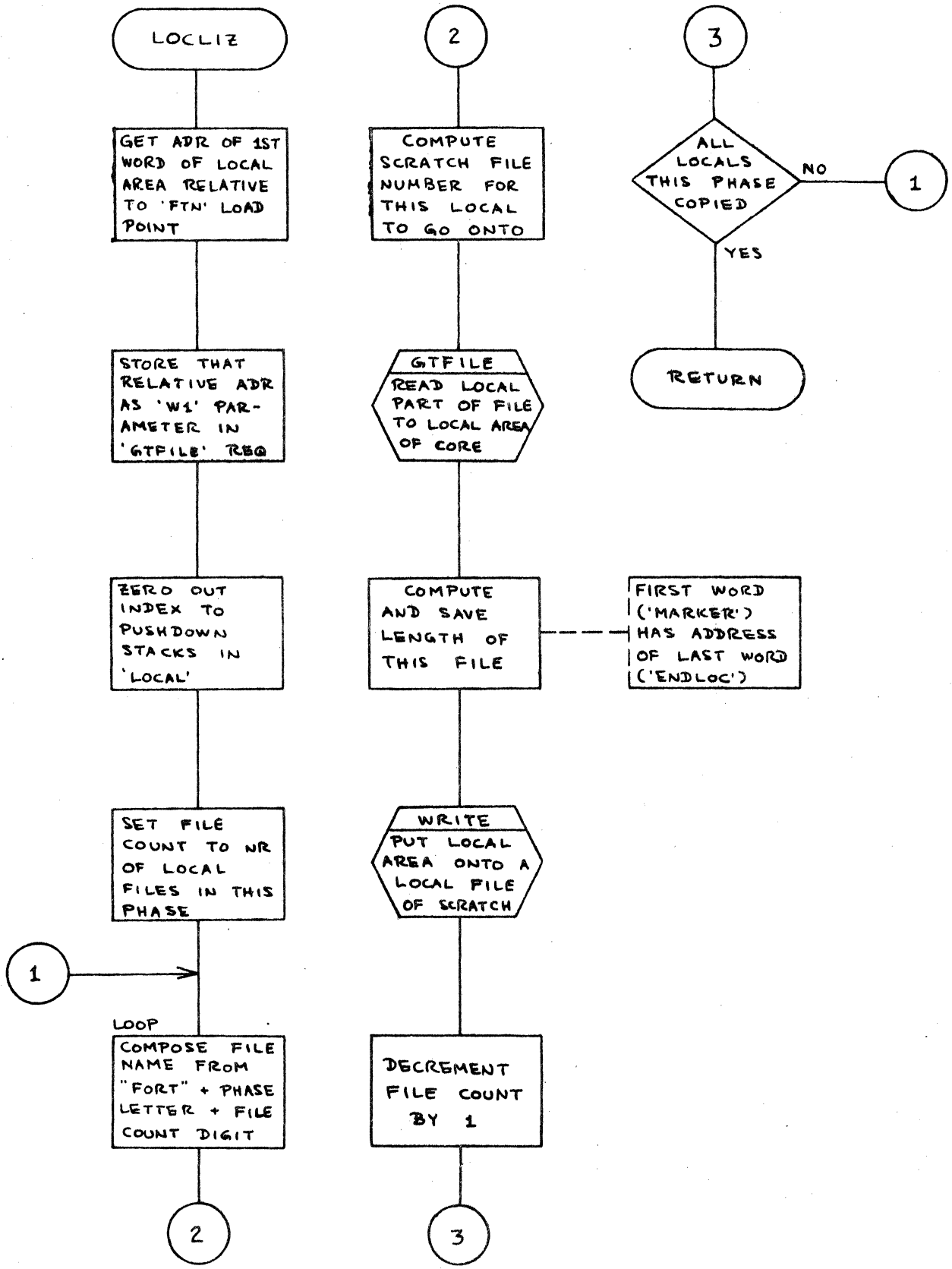
CSD 203

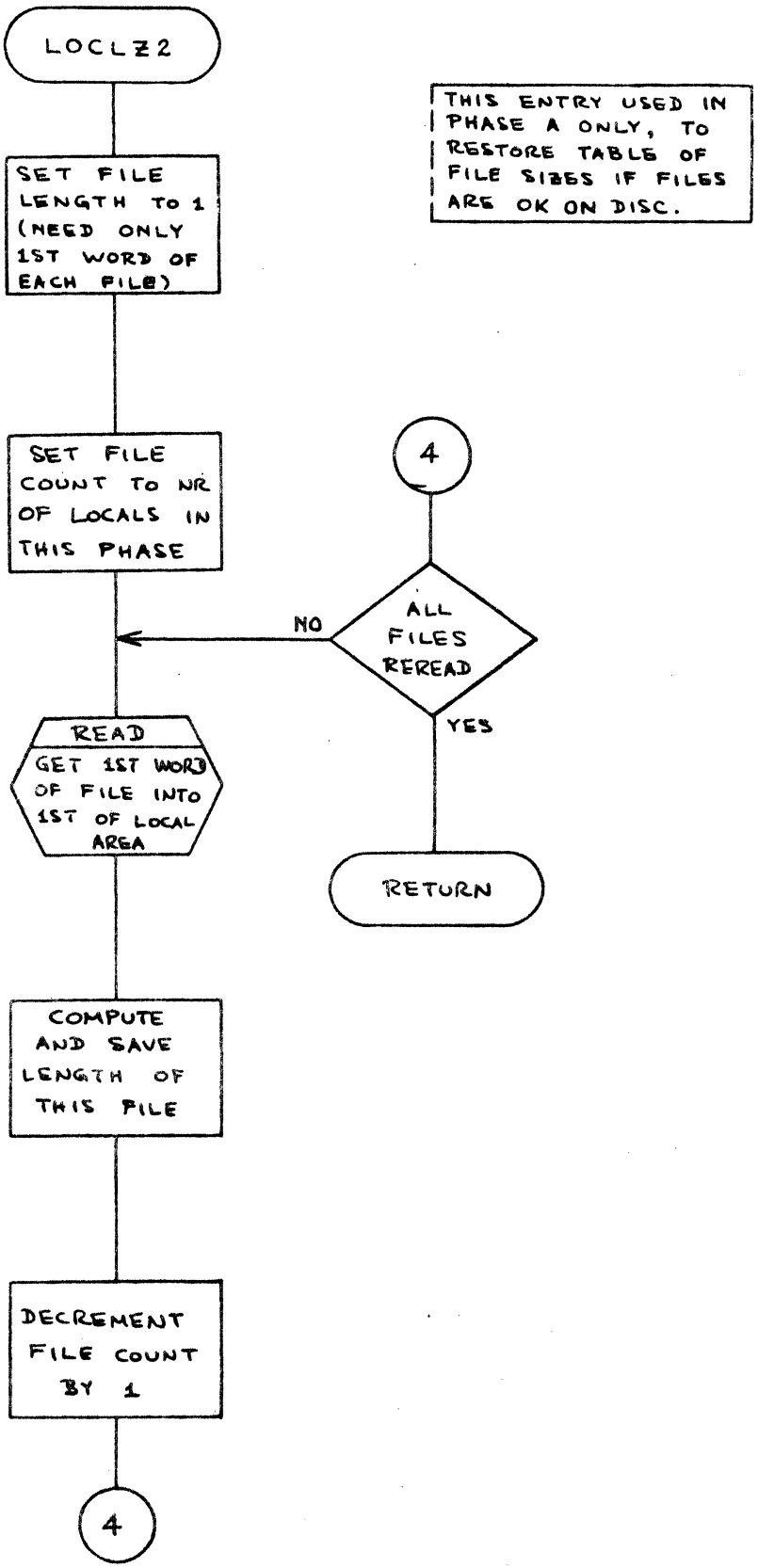

5
SKIPIT
6
RETURN
INCREMENT RETURN ADR BY 1; SET UP FOR NORMAL RETURN
LOCLIZ
MOVE LOCAL FILES FROM LIBRARY TO SCRATCH AREA
4
CLEAR "FATAL ERROR IN PHASE A" FLAG
WRITE
DUMP LAST OUTPUT BUFFER ONTO DISC
RETURN

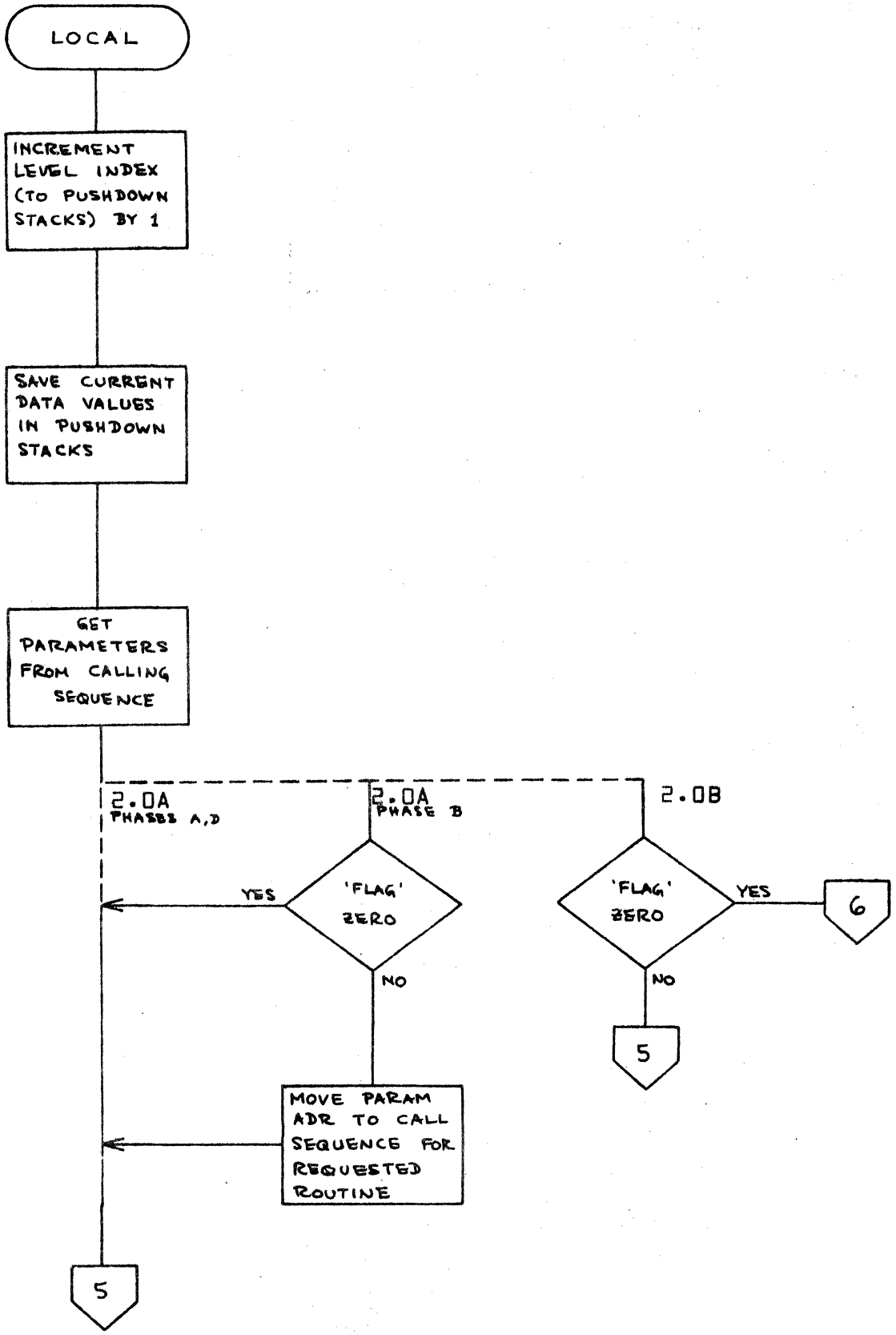

GOB
SKIPIT
1
CLEAR "PHASE A LOCAL FILES OK" FLAG
INCREMENT RETURN ADR BY 1; SET UP FOR NORMAL RETURN
INTERCHANGE INPUT, OUTPUT SCRATCH FILE NUMBERS
WRITE
DUMP LAST OUTPUT BUFFER ONTO DISC
IOPR
INITIALIZE I/O, READ SCRATCH INPUT
RETURN
LOCLIB
MOVE LOCAL FILES FROM LIBRARY TO SCRATCH
2.0A COMPILER ONLY
PHASEB
EXECUTE 2ND PHASE OF COMPILER
1

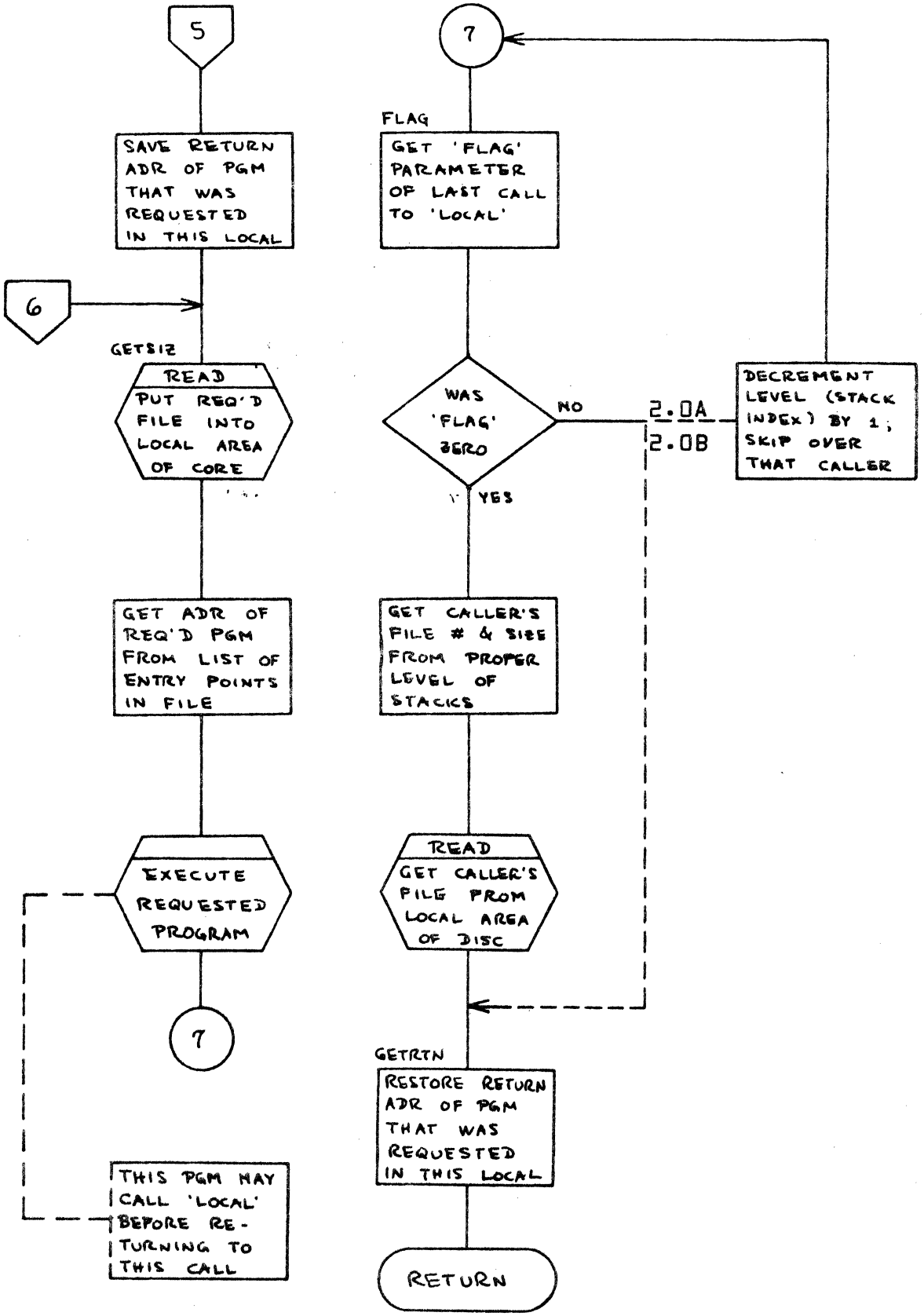

GOC
INTERCHANGE INPUT, OUTPUT SCRATCH FILE NUMBERS
IOPR
INITIALIZE I/O, READ SCRATCH INPUT
PHASEC
EXECUTE 3RD PHASE OF COMPILER
2.0B
COMPILER ONLY
A OR M OPTION ON
YES
INCREMENT RETURN ADR BY 1; GET PHASE E NEXT
NO
INCREMENT RETURN ADR BY 1; NO-ERROR RETURN
1
1
WRITE
DUMP LAST OUTPUT BUFFER ONTO DISC
RETURN


GOOD, GOE
INTERCHANGE INPUT, OUTPUT SCRATCH FILE NUMBERS
1OPR
INITIALIZE I/O, READ SCRATCH INPUT
LOCLIZ
MOVE LOCAL FILES FROM LIBRARY TO SCRATCH
2.0A COMPILER ONLY
PHASE6
EXECUTE 4TH PHASE OF COMPILER
PHASE D OR E. CHOICE MADE BY 'LOCAL' IN 2.0A, BY 'GOC' & 'FTN' IN 2.0B.
PUT NEW LOAD-&-GO TOP SECTOR # BACK INTO LOW CORE
FOR 'GOA' IF ANOTHER COMPILATION IN STACK, FOR LOADER IF NOT.
1
1
LIST DEVICE MAG TAPE
NO
YES
TAPE
WRITE END OF FILE AFTER LISTING
INCREMENT RETURN ADR BY 1; SET UP FOR NORMAL RETURN
RETURN


LOCLIZ
GET ADR OF 1ST WORD OF LOCAL AREA RELATIVE TO 'FTN' LOAD POINT
STORE THAT RELATIVE ADR AS 'W1' PARAMETER IN 'GTFILE' REQ
ZERO OUT INDEX TO PUSHDOWN STACKS IN 'LOCAL'
SET FILE COUNT TO NR OF LOCAL FILES IN THIS PHASE
1
LOOP
COMPOSE FILE NAME FROM "FORT" + PHASE LETTER + FILE COUNT DIGIT
2
2
COMPUTE SCRATCH FILE NUMBER FOR THIS LOCAL TO GO ONTO
GTFILE
READ LOCAL PART OF FILE TO LOCAL AREA OF CORE
COMPUTE AND SAVE LENGTH OF THIS FILE
FIRST WORD ('MARKER') HAS ADDRESS OF LAST WORD ('ENDLOC')
WRITE
PUT LOCAL AREA ONTO A LOCAL FILE OF SCRATCH
DECREMENT FILE COUNT BY 1
3
3
ALL LOCALS THIS PHASE COPIED
NO
1
YES
RETURN


LOCLZ2
THIS ENTRY USED IN PHASE A ONLY, TO RESTORE TABLE OF FILE SIZES IF FILES ARE OK ON DISC.
SET FILE LENGTH TO 1 (NEED ONLY 1ST WORD OF EACH FILE)
SET FILE COUNT TO NR OF LOCALS IN THIS PHASE
4
ALL FILES REREAD
NO
YES
READ
GET 1ST WORD OF FILE INTO 1ST OF LOCAL AREA
RETURN
COMPUTE AND SAVE LENGTH OF THIS FILE
DECREMENT FILE COUNT BY 1
4


LOCAL
INCREMENT LEVEL INDEX (TO PUSHDOWN STACKS) BY 1
SAVE CURRENT DATA VALUES IN PUSHDOWN STACKS
GET PARAMETERS FROM CALLING SEQUENCE
2.0A PHASES A,B
2.0A PHASE B
2.0B
'FLAG' ZERO
YES
NO
'FLAG' ZERO
YES
NO
6
5
MOVE PARAM ADR TO CALL SEQUENCE FOR REQUESTED ROUTINE
5


5
SAVE RETURN ADR OF PGM THAT WAS REQUESTED IN THIS LOCAL
6
GETSIZ
READ
PUT REQ'D FILE INTO LOCAL AREA OF CORE
GET ADR OF REQ'D PGM FROM LIST OF ENTRY POINTS IN FILE
EXECUTE REQUESTED PROGRAM
7
THIS PGM MAY CALL 'LOCAL' BEFORE RE-TURNING TO THIS CALL
7
FLAG
GET 'FLAG' PARAMETER OF LAST CALL TO 'LOCAL'
WAS 'FLAG' ZERO
NO
2.0A
2.0B
DECREMENT LEVEL (STACK INDEX) BY 1; SKIP OVER THAT CALLER
YES
GET CALLER'S FILE # & SIZE FROM PROPER LEVEL OF STACKS
READ
GET CALLER'S FILE FROM LOCAL AREA OF DISC
GETRTN
RESTORE RETURN ADR OF PGM THAT WAS REQUESTED IN THIS LOCAL
RETURN

2.2.2 PHASEA

Phase A operates as follows:

1. Perform PHASEA Initialization.
2. Read a statement.
3. Process the label.
4. Determine statement type.
5. Branch to process the statement type by either
   a. calling the particular processor if the statement type is not to be processed in PHASEA, or -
   b. branching to the in-line processing of the statement type in PHASEA.
6. Repeat the procedure from step 2 to step 5 until an END statement is reached. Upon the occurrence of an END statement, exit from PHASEA.

2.2.2.0.1 PHASEA Initialization

The initialization procedure for PHASEA operation is described in detail in the flow charts for PHASEA. The Phase A Block Data routine contains all data presets for labelled common.

2.2.2.0.2 Reading Statements

Each FORTRAN source statement is assigned a statement number by the PHASEA routine. ISTNO is the name of a register which contains the number of the statement about to be read and processed. ISTNO is set to zero by Initialization. It is then increased by 1 prior to reading each source statement. In this manner ISTNO contains the number of each source statement at the time it is to be read. A source statement is read by a call to the routine whose name is GNST. (See item 2.2.2.1)

2.2.2.0.3 Format of Input Entry

The GNST routine will read a source statement from a card and store it in its internal buffer {ISRS}. The characters which make up the source statement are recorded in the ASCII code as follows:

Character		ASCII Code
Digits:	0-9	$30 - 39
Letters:	A-Z	$41 - $5A

Special Characters

Dollar Sign	$	$24
Period	.	$2E
Plus	+	$2B
Minus	-	$2D
Equal Sign	=	$3D
Left Paren	(	$28
Right Paren	)	$29
Comma	,	$2C
Slash	/	$2F
Asterisk	*	$2A
Space		$20
Statement Terminator		$20
Single Quote	'	$27

The GNST routine will convert the code for each character of the statement from ASCII to Internal FORTRAN. The Internal FORTRAN code is as follows:

Character		Internal FORTRAN Code
Digit:	0-9	0 - 9
Letter:	A-Z	10 - 35

Special Characters

Dollar Sign	36
Period	37
Plus	38
Minus	39
Equal Sign	40
Left Paren	41
Right Paren	42
Comma	43
Slash	44
Asterisk	45
Space	46
Statement Terminator	47
Single Quote	48

Following the code conversions, the entire statement is transferred to the source buffer {ISORS}. The buffer which is internal to the GNST

routine and the source buffer are defined in the source language of the compiler by COMMON statements. The name of the source buffer is ISORS, and its length is 208. Upon return from GNST, ISORSX contains the number of words in the source statement.

2.2.2.0.4 Processing Label

The PHASEA subroutine will call the subroutine whose name is PLABEL. This subroutine will determine whether or not there is a label and, if so, process it.

2.2.2.0.5 Identifying Statement Type

The type of the source statement currently being processed is determined by the subroutine whose name is TYPE. Upon return from the subroutine TYPE, an identifying number giving statement type is recorded in the third word of the output buffer. (Refer to items 2.2.2.0.6.2 and 2.2.2.0.6.2.11).

There are 47 type numbers used to identify FORTRAN source statements:

Type Number	Statement
0	DIMENSION
1	COMMON
2	INTEGER
3	REAL
4	INTEGER FUNCTION
5	REAL FUNCTION
6	PROGRAM
7	SINGLE
8	BYTE
9	SIGNED BYTE
10	EXTERNAL
11	RELATIVE
12	EQUIVALENCE
13	BLOCK DATA
14	SUBROUTINE
15	FUNCTION
16	DATA
17	FORMAT
18	Replacement statement
19	Statement function
20	ASSIGN
21	CALL
22	RETURN

23	NOT USED
24	UNCONDITIONAL GO TO
25	COMPUTED GO TO
26	ASSIGNED GO TO
27	CONTINUE
28	STOP
29	STOP n
30	PAUSE
31	PAUSE n
32	END
33	END FILE
34	REWIND
35	BACKSPACE
36	READ {unformatted}
37	READ {formatted}
38	WRITE {unformatted}
39	WRITE {formatted}
40	BEGIN DO
41	END DO
42	ARITHMETIC IF
43	LOGICAL IF
44	ASSEM
45	OPEN
46	DOUBLE PRECISION
47	DOUBLE PRECISION FUNCTION

2.2.2.0.6 Branching to Process Statements

Branching to process a statement is accomplished by means of a Computed GO TO statement in which the index ITEMPX contains the type number +1 for the FORTRAN source statement. A jump is made to a particular statement number in the subroutine depending on the value in ITEMPX. After processing each statement, a test is made to see if the statement has a label. If the source statement does have a label, a call is made to the ENDDO subroutine. If this statement label terminates a DO loop, ENDDO will generate the necessary intermediate language for the DO loop termination procedure. Prior to calling ENDDO, the statement label is recorded in ILLABL.

2.2.2.0.6.1 Statements Not Processed in the PHASEA Routine

Each of these statements is processed by a call to its processor subroutine. The call is made from PHASEA subsequent to branching.

Source Statement	Subroutine called from PHASEA
DIMENSION	DIMPR
COMMON	COMNPR
SINGLE	TYPEPR
INTEGER	TYPEPR
INTEGER FUNCTION	SUBPPR
REAL FUNCTION	SUBPPR
REAL	TYPEPR
BYTE	BYE@PR
SIGNED BYTE	BYE@PR
EXTERNAL	EXRLPR
RELATIVE	EXRLPR
EQUIVALENCE	BYE@PR
FUNCTION	SUBPPR
SUBROUTINE	SUBPPR
DATA	DATAPR
FORMAT	CHECKF
ASSIGN	ASGNPR
CALL	ARITH
ENDFILE	ERBPR
REWIND	ERBPR
BACKSPACE	ERBPR
READ unformatted	IOSPR
READ formatted	IOSPR
WRITE unformatted	IOSPR
WRITE formatted	IOSPR
DO	BDOPR
ASSEM	ASEMPR
OPEN	IOSPR
DOUBLE PRECISION	TYPEPR
DOUBLE PRECISION FUNCTION	SUBPPR

2.2.2.0.6.2 Statements Processed in the PHASEA Routine

Several statement types are processed in the PHASEA subroutine. These are PROGRAM, BLOCK DATA, RETURN, GO TO, IF, CONTINUE, STOP, PAUSE, replacement statements, statement functions, and END.

2.2.2.0.6.2.1 Program

The PHASEA routine verifies the correct format of the statement, then records the name of the program in the symbol table as a program name. In the event of a format error, an indication is made by a call to the DIAG subroutine. The PHASEA subroutine then reads and processes the following source statement.

2.2.2.0.6.2.2 Block Data

The PHASEA routine verifies the correct format of the statement. In the event of an error, an indication is given by a call to the DIAG routine. The switch named ISUBP is set to a "1" to indicate the compiler will generate object code for COMMON, DATA, BYTE, EQUIVALENCE & DIMENSION statements only.

2.2.2.0.6.2.3 Return

A RETURN statement is legal only if the source program is either a FUNCTION or SUBROUTINE. If the RETURN statement is used legally, the intermediate language is passed on for processing by a subsequent phase.

If the return statement is used illegally, an error diagnostic is produced by a call to the DIAG subroutine. A STOP statement is substituted for the RETURN statement and it is passed on as intermediate language to be processed by a subsequent phase.

2.2.2.0.6.2.4 GO TO

The GO TO type is determined. Depending on the type, the statement is syntax checked and converted to list notation. Computed GO TO causes a call to ARITH.

2.2.2.0.6.2.5 IF

For both logical IF and Arithmetic IF statements, the expression in parentheses is processed by a call to the ARITH subroutine. After this call if the statement is an arithmetic IF, PHASEA will process the three statement labels following the arithmetic expression in the source statement. Statement labels following the expression must be separated from each other by commas, and the last one must be followed by an EOS. There must be exactly 3 statement labels in an Arithmetic IF statement. The statement labels are processed as follows:

PHASEA calls the RDLABL subroutine in order to extract the statement label from the source buffer. If the statement label does not appear in the symbol table, it is entered therein by a call to the STORE subroutine. Upon return from STORE, the entry in the symbol table is assigned the classification for a statement label by -

7 $\longrightarrow$ ICLASS(ISYMX)

An entry is made into the output when the pointer to the symbol table entry is recorded as part of the intermediate language by -

(ISYMX) + (ISYMP) $\longrightarrow$ IBUF2(ITEMP1)

(Upon return from ARITH and prior to processing the 1st statement label, the tally register IBUF2X was recorded in ITEMP1. IBUF2X was then increased by three for the three extra words of intermediate language generated in IBUF2).

After all the labels are processed, WORD 1 of the output buffer is set to the word length of the output buffer by -

(IBUF2X) - 1 $\longrightarrow$ IBUF2(1).

The PHASEA subroutine will check for and give diagnostics upon occurrence of any of the following errors:

1. There are either fewer than or more than three statement labels following the expression in an Arithmetic IF statement.
2. A character other than a comma has occurred between two statement labels.
3. A character other than an EOS follows the last statement label.
4. Two commas have occurred consecutively in the source statement with no statement label between.

Each of the above errors will cause processing of the Arithmetic IF statement to be terminated.

The LOGICAL IF statement is broken into two statements by the PHASEA subroutine. Only the first of the two subsequent statements is assigned a statement number. There is an output entry generated for each statement. WORD 2 of the 1st output entry contains the statement number for the LOGICAL IF statement. A switch named LOGIF is at all times set to the zero position except -

1. it is set to a "1" when processing the 1st half of a Logical IF statement, and -
2. it is set to a "2" when processing the second half.

The second of the two statements may be one of the following:

REPLACEMENT
CALL
GO TO
STOP
PAUSE
RETURN
READ
WRITE
ENDFILE
REWIND
BACKSPACE

The switch named LOGIF is at all times set to zero except when processing a statement whose type number identifies the statement as a Logical IF. Upon receipt of a Logical IF statement, the LOGIF switch is set to a 1. The input entry containing the Logical IF is passed as intermediate language. If the state-

ment has a label, the label is recorded temporarily in ITEMP9, otherwise, (ITEMP9) = 0.

The second half of the logical IF statement is moved to the beginning of the ISORS buffer so that it appears to the PHASEA routine as an independent statement. PHASEA then branches to the point just after the call to GNST.

When the following statement is processed, it is recognized as the 2nd part of a Logical IF by the fact that - (LOGIF)=1.

The input entry for the 2nd part of the Logical IF is passed on as intermediate language for processing by a subsequent phase in the same manner as for the 1st part. Afterward, the switch LOGIF is set to a 2. Then if (ITEMP9) $\neq$ 0, the label in ITEMP9 is placed in ILLABL and a call is made to the ENDDO subroutine. If this statement label is associated with the end of a DO loop, ENDDO will generate the necessary intermediate language for termination of the DO loop procedure. PHASEA will then process the next statement in sequence. If (ITEMP9) = 0, PHASEA proceeds immediately to process the next statement in sequence.

The LOGIF switch will be reset to a zero when processing the next statement in sequence.

2.2.2.0.6.2.6 CONTINUE

A CONTINUE record is output. (See Output Format, Phase A, Chapter 7, section 7.5).

2.2.2.0.6.2.7 STOP

PHASEA determines whether or not the STOP statement is a

STOP n

in which "n" represents an octal integer, followed by a legal terminator (EOS character). One of the following three events will occur:

1. The statement is not a

   STOP n

   and the intermediate language will be passed

on for processing by a subsequent phase.

2. The statement is a

STOP n

where "n" is an octal integer followed by a legal terminator. The type number in word 3 of the output buffer is changed to type number for a STOP n by -

IBUF2(3)+1 → IBUF2(3).

3. The statement is a

STOP n

where either "n" is something other than an octal integer or an octal integer followed by something other than a legal terminator. An error diagnostic is produced by a call to the DIAG subroutine. The intermediate language for the statement is passed on.

2.2.2.0.6.2.8 PAUSE

PHASEA determines whether or not the PAUSE statement is a

PAUSE n

in which "n" represents an octal integer, followed by a legal terminator (EOS character). One of the following three events will occur:

1. The statement is not a

PAUSE n

and the intermediate language will be passed on for processing by a subsequent phase.

2. The statement is a

PAUSE n

wherein "n" is an octal integer followed by a legal terminator. The type number in word 3 of the output buffer is changed to the type number for a PAUSE n by -

IBUF2 (3)+1 —> IBUF2(3).

The octal integer is recorded as the last word of the output entry for the statement. The intermediate language for the statement will be passed on for processing by a subsequent phase.

3. The statement is a

PAUSE n

wherein either "n" is something other than an octal integer or an octal integer followed by something other than a legal terminator. An error diagnostic is produced by a call to the DIAG subroutine. The intermediate language for the statement is passed on.

2.2.2.0.6.2.9 Replacement Statements and Statement Functions

PHASEA reads the first field of the statement using GETF. If the field is not a variable, array, or unassigned name, an error is output and the statement ignored up to the equal sign.

If the field is a legal name, it is entered in SYMTAB, if not already there.

The field terminator is then checked. If the terminator is neither a left parenthesis nor an equal sign, an error is output and the statement ignored up to the equal sign.

If the terminator is left parenthesis and the name has not yet been classified as a variable or array and we have not yet encountered the first executable statement, the name is assumed a statement function. The initial part of the output entry is set up in IBUF2 accordingly, and the function parameters are entered in SYMTAB.

If a statement function is illegal at this point in the program or the name has previously been determined

a variable or array, PHASEA resets ISORSX so that the first field will be reprocessed and calls ARITH to process the information to the left of the equal sign (ISTOP =2).

After ascertaining in all cases that ISORSX is positioned to the field immediately following the equal sign PHASEA recalls ARITH to process the information in the statement to the right of the equal sign (ISTOP = 0).

Upon return from ARITH, PHASEA voids the SYMTAB entries for statement function parameters if it has been processing a statement function.

2.2.2.0.6.2.10 END

If the program being compiled is a main program, a check is made to see if a 'last executable statement' must be generated. If the type number of the previous statement (stored in LEST) is not one of the following, PHASEA will generate the 4 word output record for a STOP statement:

GO TO
STOP
Arithmetic IF

If any DO LOOP remains 'open', that is, if the label terminating any DO loop has not been encountered, a fatal diagnostic is generated. A CONTINUE statement is generated in IBUF2 and a call to ENDDO is made for each missing label.

An output record for an END statement is generated and output. Then CPLOOP is called to assign relative addresses for elements and dimensions in COMMON storage. If a fatal error has been detected in Phase A, or if no assembly or execute options (X,M,A,P) are set, compilation is terminated at this point by a call to SKIPIT. Otherwise, an exit is made from PHASEA.

2.2.2.0.6.3 Format of Phase A Output Records

An output record consists of the information necessary to convert the statement to pseudo-assembler instructions in Phase B. The format of the output

record for each statement type is listed in Chapter 7, section 7.5. The first 4 words are the same for all output records:

Word 1 contains the total number of words in the output entry, including words 1-4.

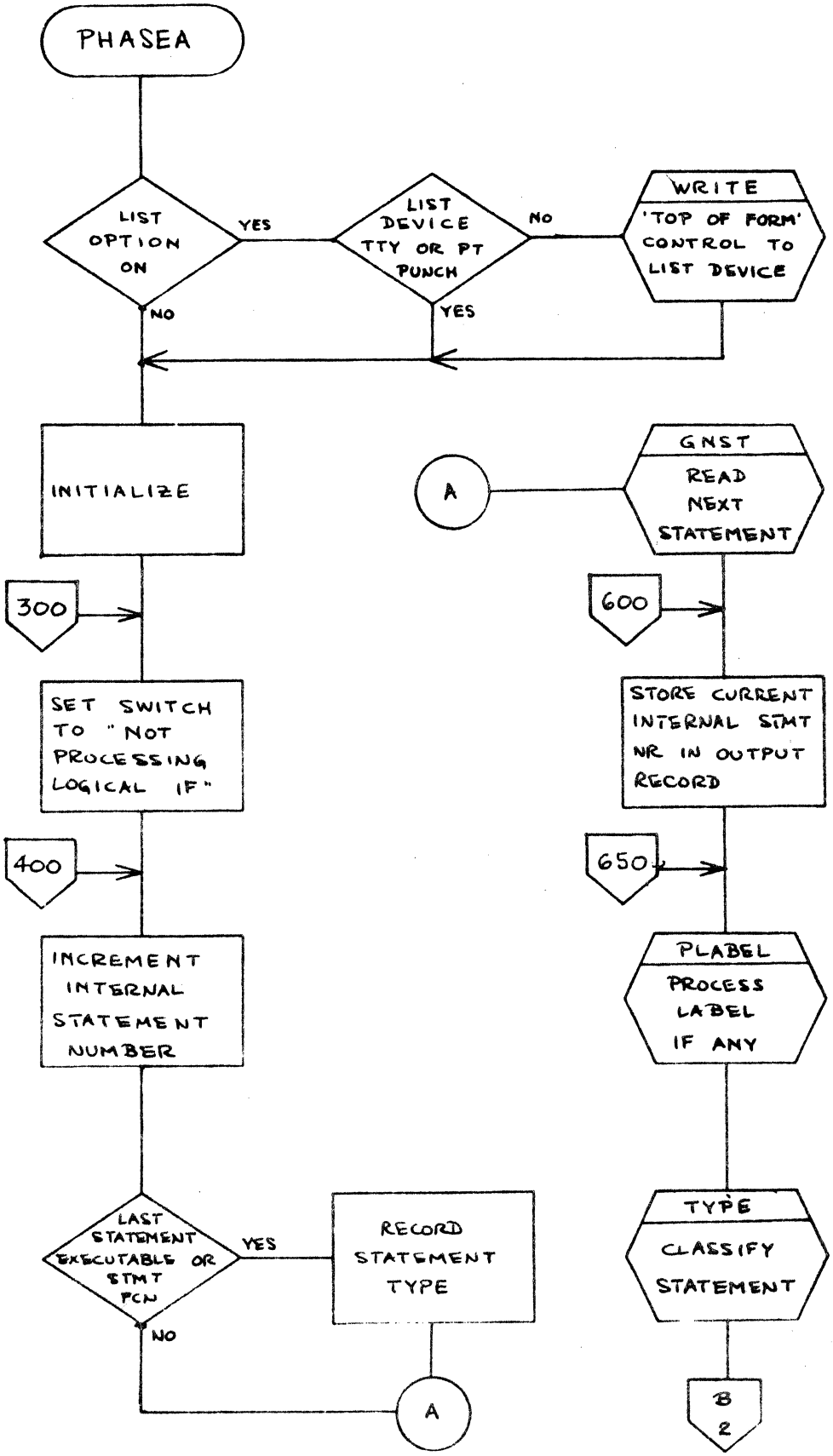
Word 2 contains the statement number for the source statement (ISTNO).

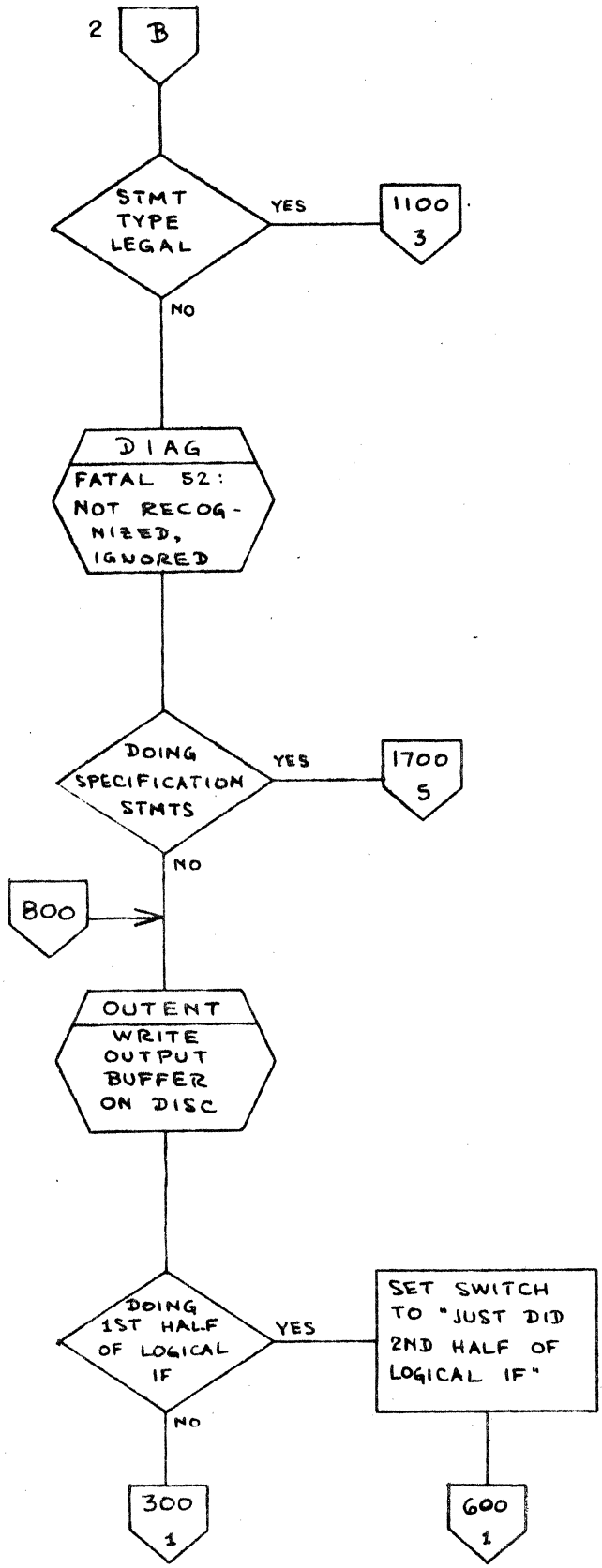
Word 3 contains the statement type.

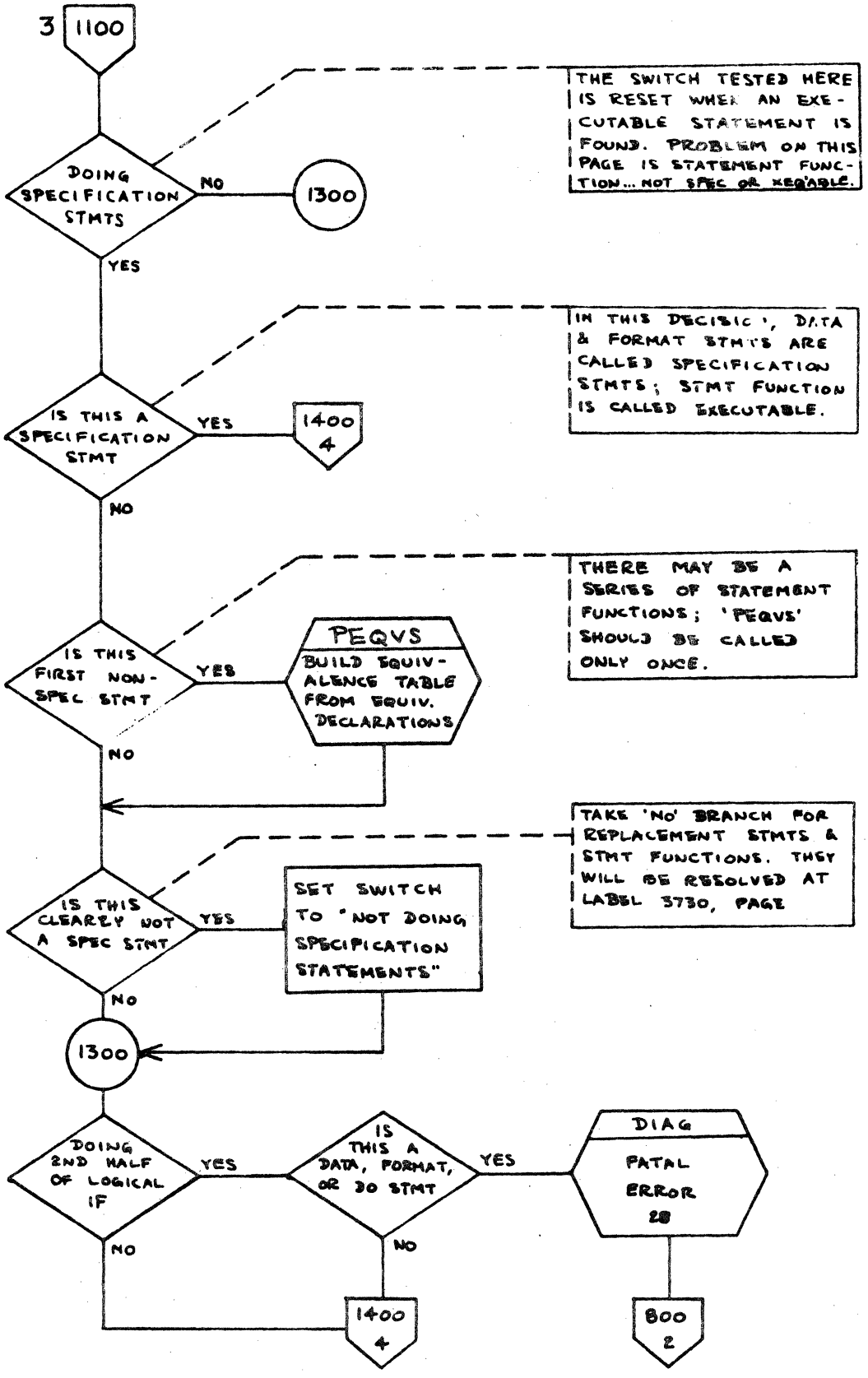
Word 4 is a switch used to signal the generated statement or statements which comprise the second half of a logical IF.

Word 5. If word 2 is negative, the absolute value of word 2 is the statement number of the statement, and word 5 contains the SYMTAB entry for the statement label of that statement. If word 2 is positive, the coded information appropriate to the statement type begins in word 5.

Output record information is built in IBUF2, and is written on the disk by a call to the OUTENT routine. The starting address for the write operation is specified as the only argument of the subroutine call.

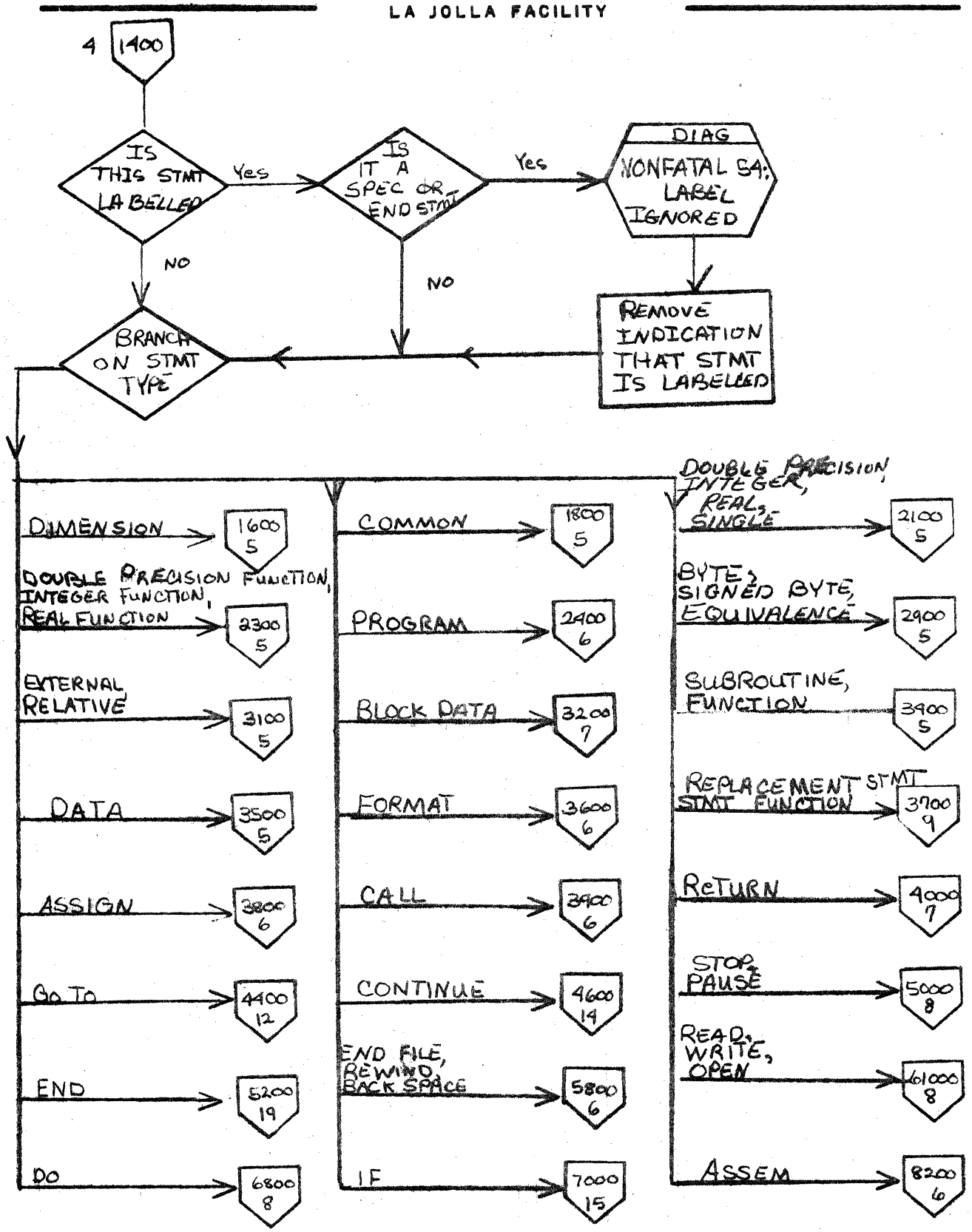

PHASEA
LIST OPTION ON
YES
NO
LIST DEVICE TTY OR PT PUNCH
NO
YES
WRITE
'TOP OF FORM' CONTROL TO LIST DEVICE
INITIALIZE
300
SET SWITCH TO "NOT PROCESSING LOGICAL IF"
400
INCREMENT INTERNAL STATEMENT NUMBER
LAST STATEMENT EXECUTABLE OR STMT FCN
YES
NO
RECORD STATEMENT TYPE
A
A
GNST
READ NEXT STATEMENT
600
STORE CURRENT INTERNAL STMT NR IN OUTPUT RECORD
650
PLABEL
PROCESS LABEL IF ANY
TYPE
CLASSIFY STATEMENT
B
2


2 B
STMT TYPE LEGAL
YES
1100 3
NO
DIAG
FATAL 52: NOT RECOG-NIZED, IGNORED
DOING SPECIFICATION STMTS
YES
1700 5
NO
800
OUTENT
WRITE OUTPUT BUFFER ON DISC
DOING 1ST HALF OF LOGICAL IF
YES
SET SWITCH TO "JUST DID 2ND HALF OF LOGICAL IF"
NO
300 1
600 1


3 1100
DOING SPECIFICATION STMTS
NO
1300
YES
THE SWITCH TESTED HERE IS RESET WHEN AN EXECUTABLE STATEMENT IS FOUND. PROBLEM ON THIS PAGE IS STATEMENT FUNCTION... NOT SPEC OR XEQ'ABLE.
IS THIS A SPECIFICATION STMT
YES
1400 4
IN THIS DECISION, DATA & FORMAT STMTS ARE CALLED SPECIFICATION STMTS; STMT FUNCTION IS CALLED EXECUTABLE.
NO
IS THIS FIRST NON-SPEC STMT
YES
PEQVS
BUILD EQUIVALENCE TABLE FROM EQUIV. DECLARATIONS
THERE MAY BE A SERIES OF STATEMENT FUNCTIONS; 'PEQVS' SHOULD BE CALLED ONLY ONCE.
NO
IS THIS CLEARLY NOT A SPEC STMT
YES
SET SWITCH TO "NOT DOING SPECIFICATION STATEMENTS"
TAKE 'NO' BRANCH FOR REPLACEMENT STMTS & STMT FUNCTIONS. THEY WILL BE RESOLVED AT LABEL 3730, PAGE
NO
1300
DOING 2ND HALF OF LOGICAL IF
YES
IS THIS A DATA, FORMAT, OR DO STMT
YES
DIAG
FATAL ERROR 28
NO
NO
1400 4
800 2

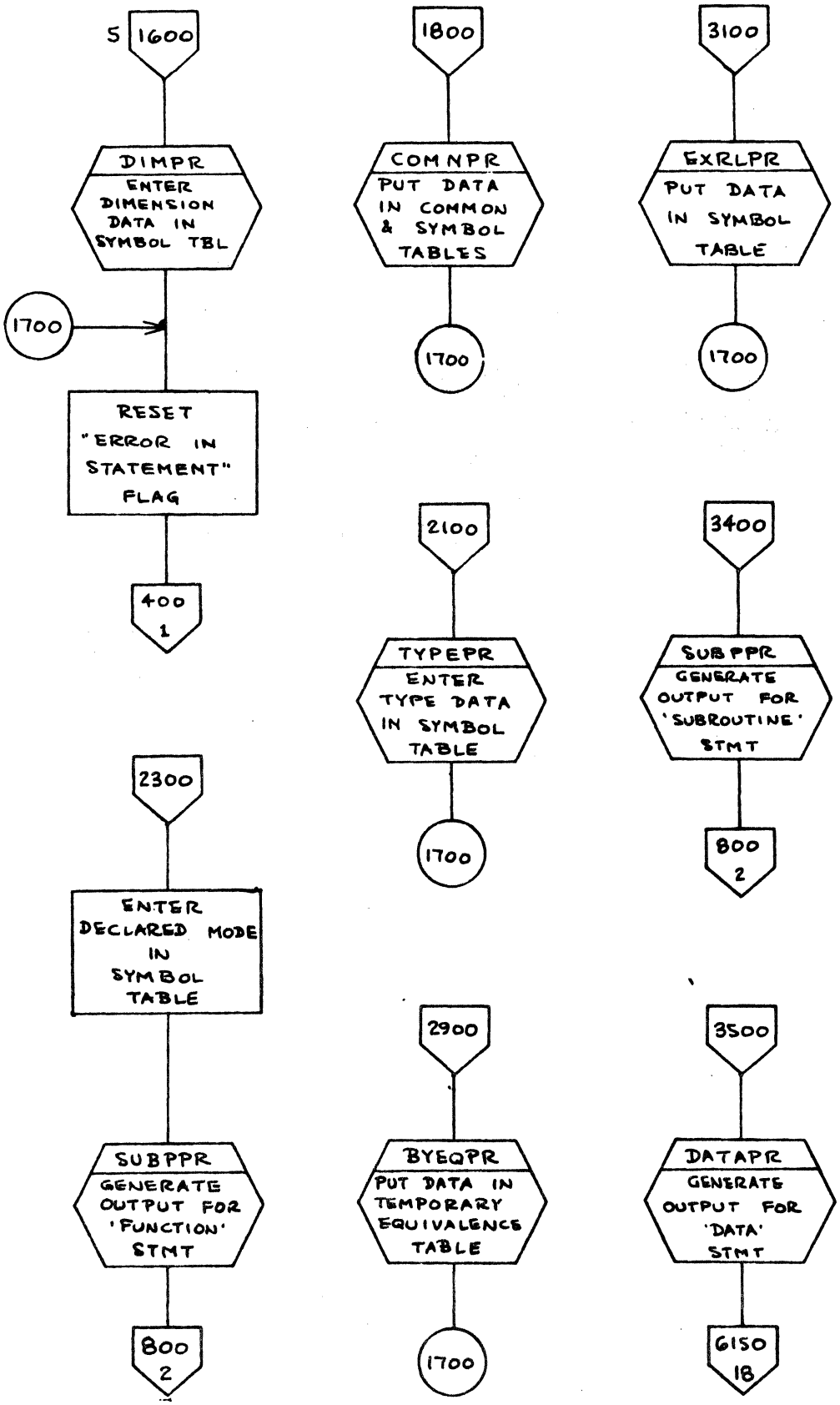
CONTROL DATA

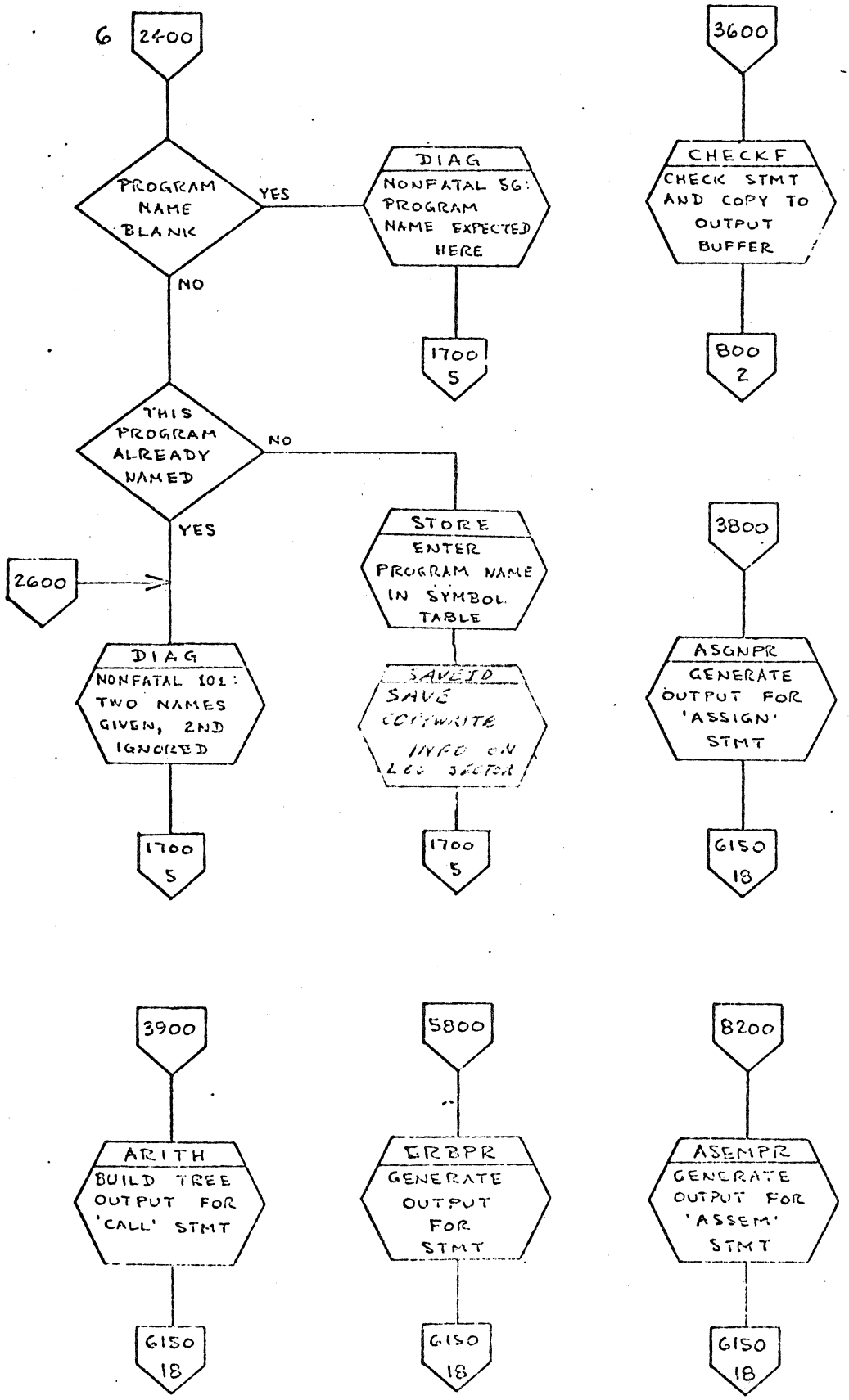
LA JOLLA FACILITY

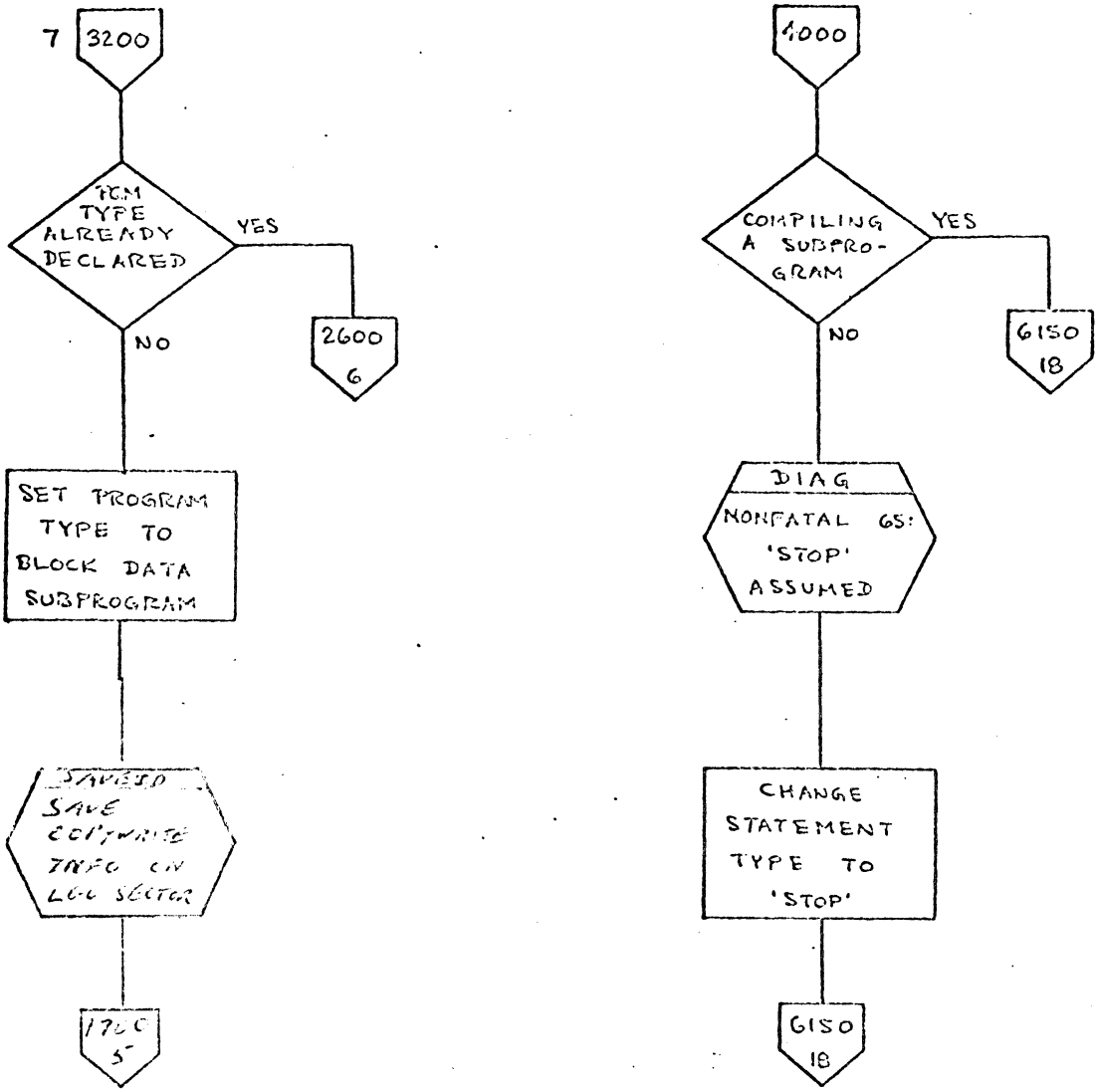


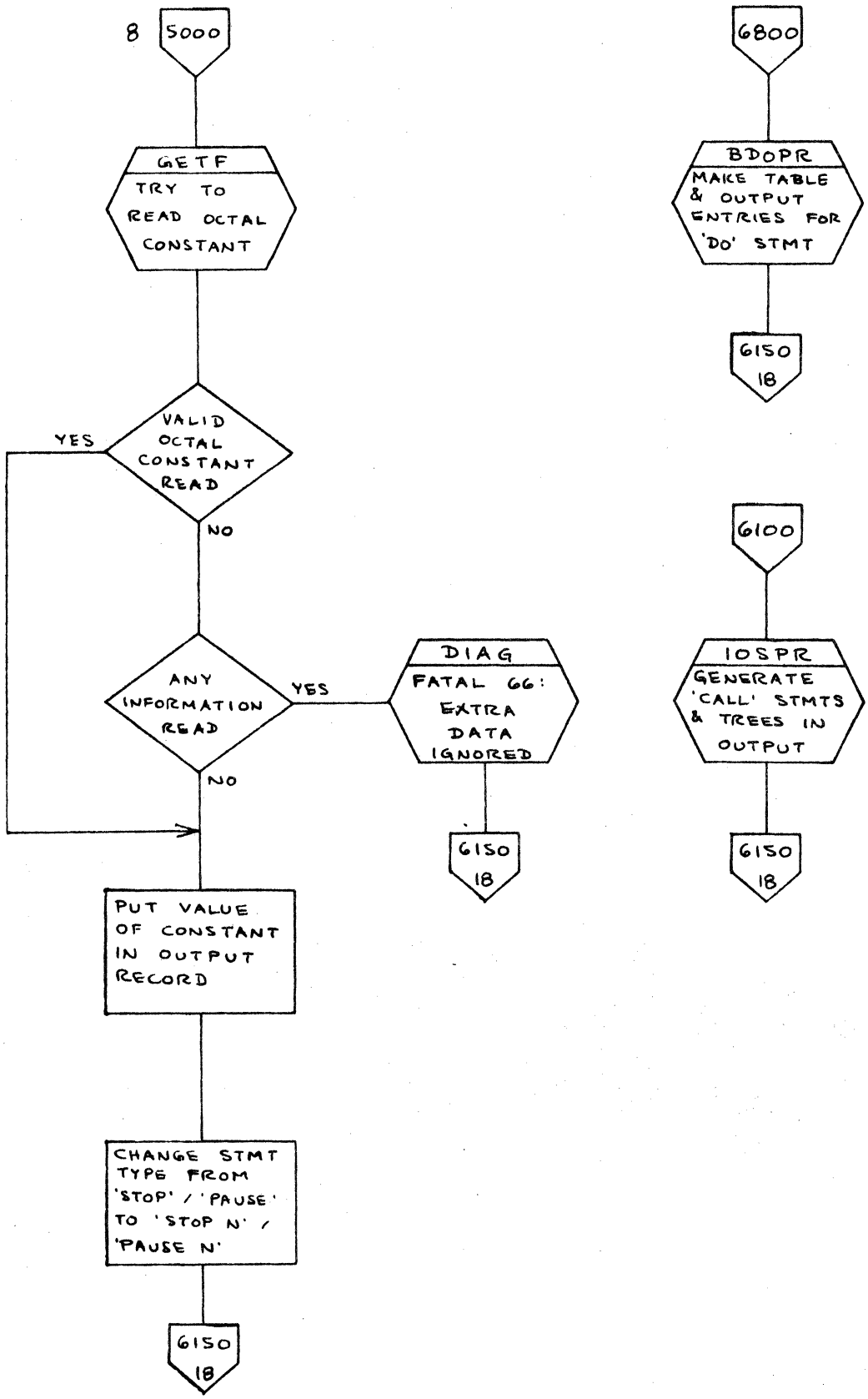
TITLE	PHASEA		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET OF

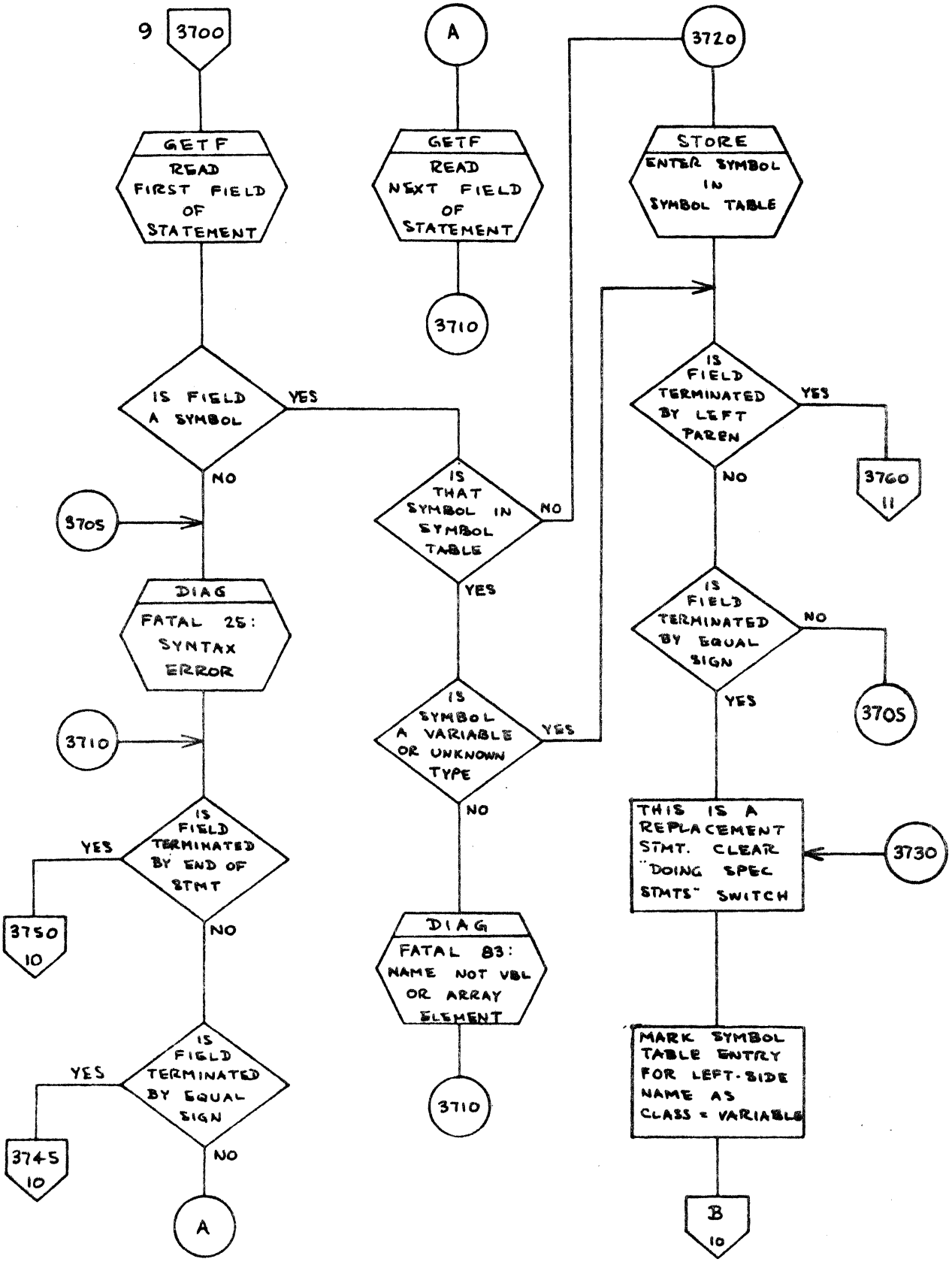
CSD 208

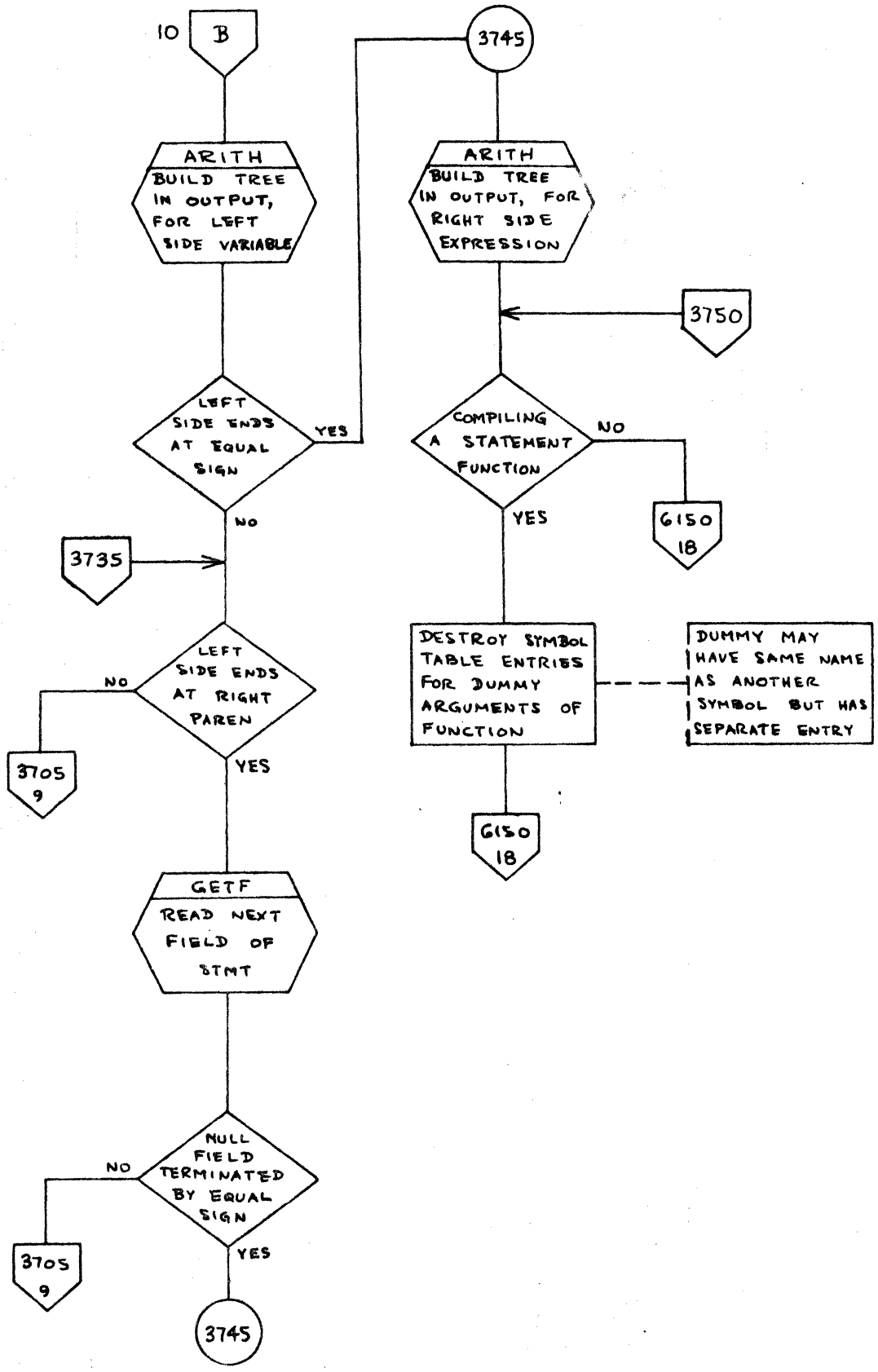

5 1600
DIMPR
ENTER DIMENSION DATA IN SYMBOL TBL
1700
RESET "ERROR IN STATEMENT" FLAG
400 1
1800
COMNPR
PUT DATA IN COMMON & SYMBOL TABLES
1700
3100
EXRLPR
PUT DATA IN SYMBOL TABLE
1700
2100
TYPEPR
ENTER TYPE DATA IN SYMBOL TABLE
1700
3400
SUBPPR
GENERATE OUTPUT FOR 'SUBROUTINE' STMT
800 2
2300
ENTER DECLARED MODE IN SYMBOL TABLE
SUBPPR
GENERATE OUTPUT FOR 'FUNCTION' STMT
800 2
2900
BYEQPR
PUT DATA IN TEMPORARY EQUIVALENCE TABLE
1700
3500
DATAPR
GENERATE OUTPUT FOR 'DATA' STMT
6150 18

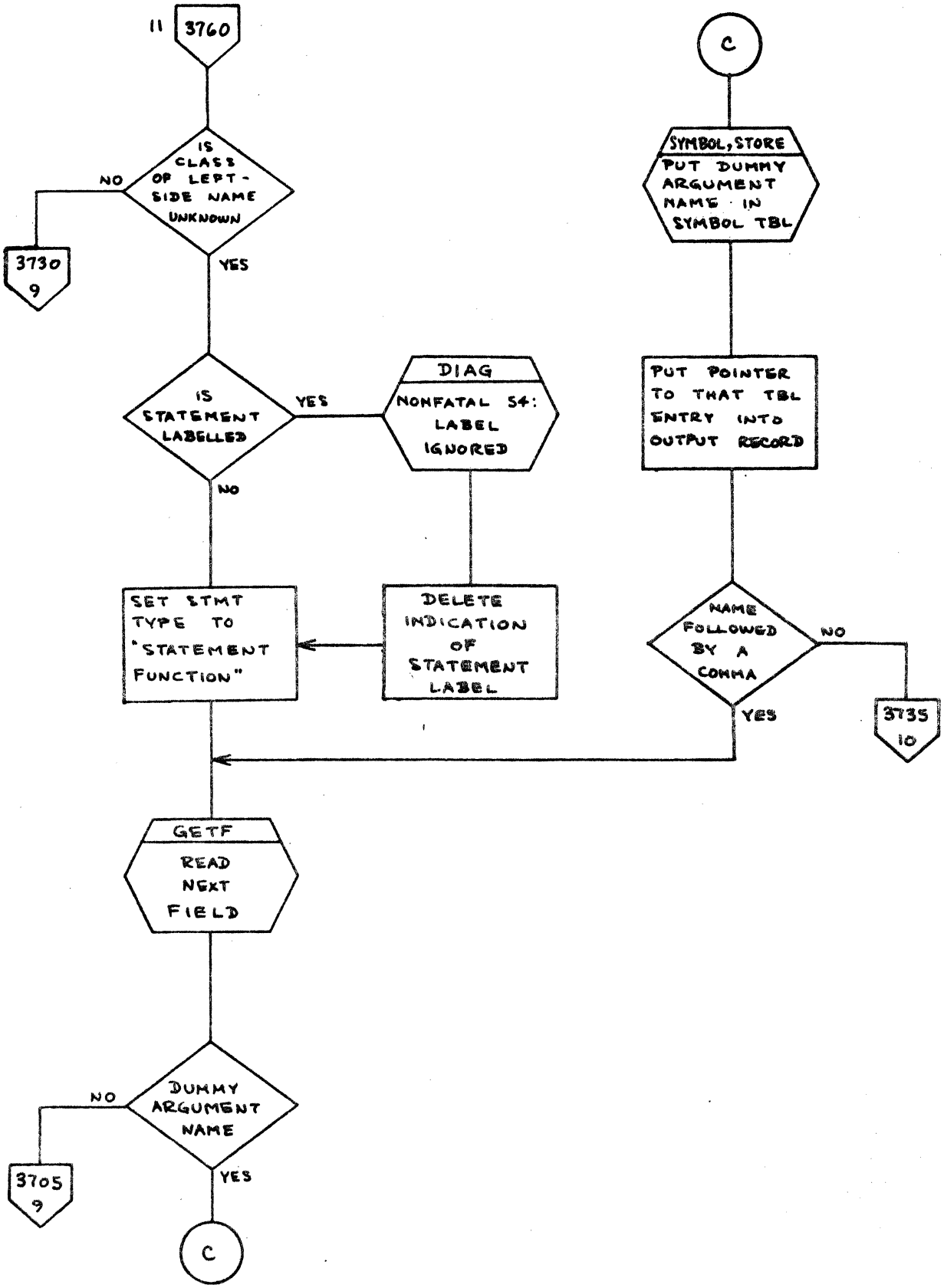

6
2400
PROGRAM NAME BLANK
YES
DIAG
NONFATAL 56: PROGRAM NAME EXPECTED HERE
1700 5
NO
THIS PROGRAM ALREADY NAMED
NO
YES
2600
DIAG
NONFATAL 101: TWO NAMES GIVEN, 2ND IGNORED
1700 5
STORE
ENTER PROGRAM NAME IN SYMBOL TABLE
SAVEID
SAVE
1700 5
3600
CHECKF
CHECK STMT AND COPY TO OUTPUT BUFFER
800 2
3800
ASGNPR
GENERATE OUTPUT FOR 'ASSIGN' STMT
6150 18
3900
ARITH
BUILD TREE OUTPUT FOR 'CALL' STMT
6150 18
5800
ERRPR
GENERATE OUTPUT FOR STMT
6150 18
8200
ASEMPR
GENERATE OUTPUT FOR 'ASSEM' STMT
6150 18

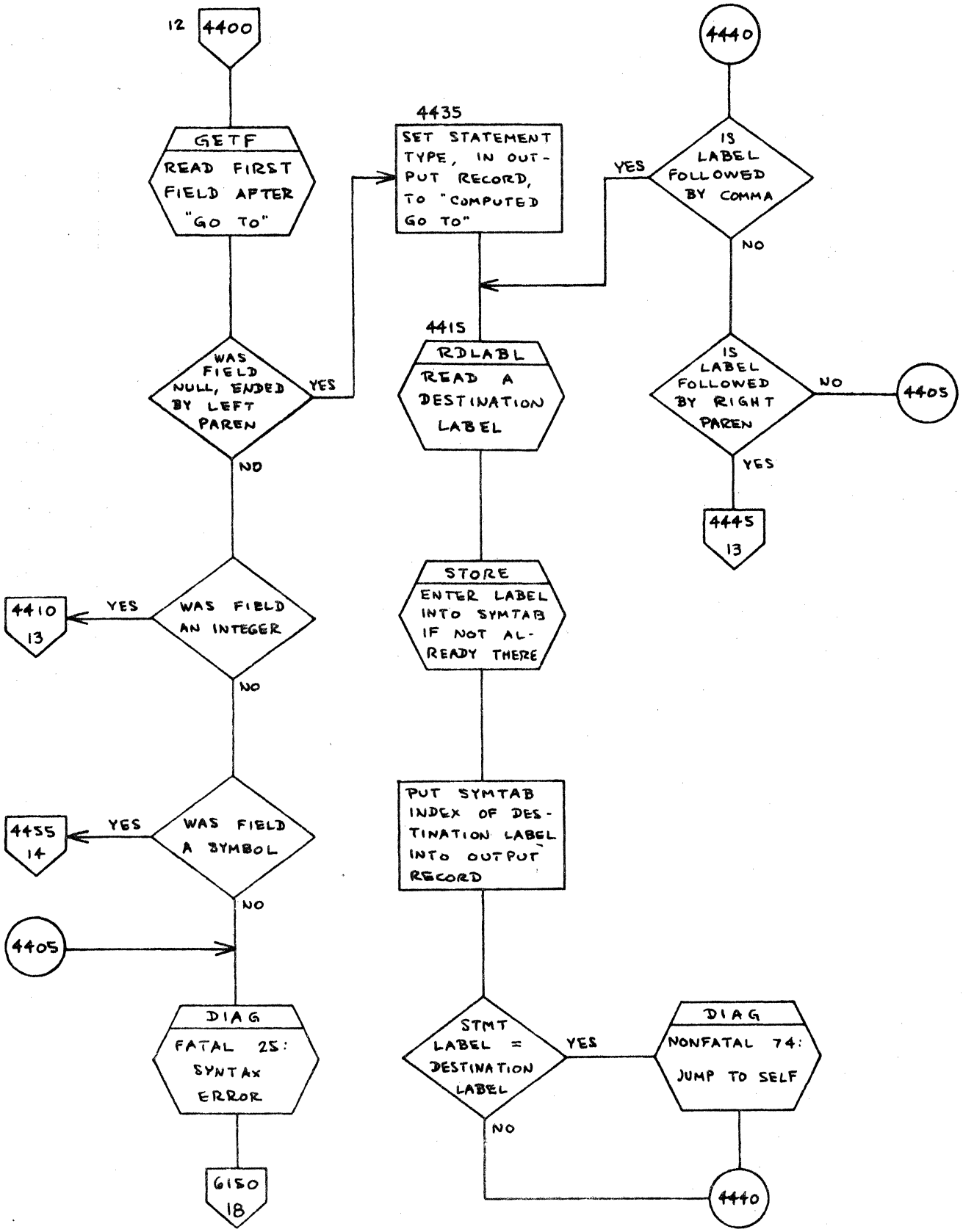

7 3200
PGM TYPE ALREADY DECLARED
YES
NO
2600 6
SET PROGRAM TYPE TO BLOCK DATA SUBPROGRAM
SAVE COMPILE INFO ON LGO SECTOR
1700 5
4000
COMPILING A SUBPRO-GRAM
YES
NO
6150 18
DIAG
NONFATAL 65: 'STOP' ASSUMED
CHANGE STATEMENT TYPE TO 'STOP'
6150 18

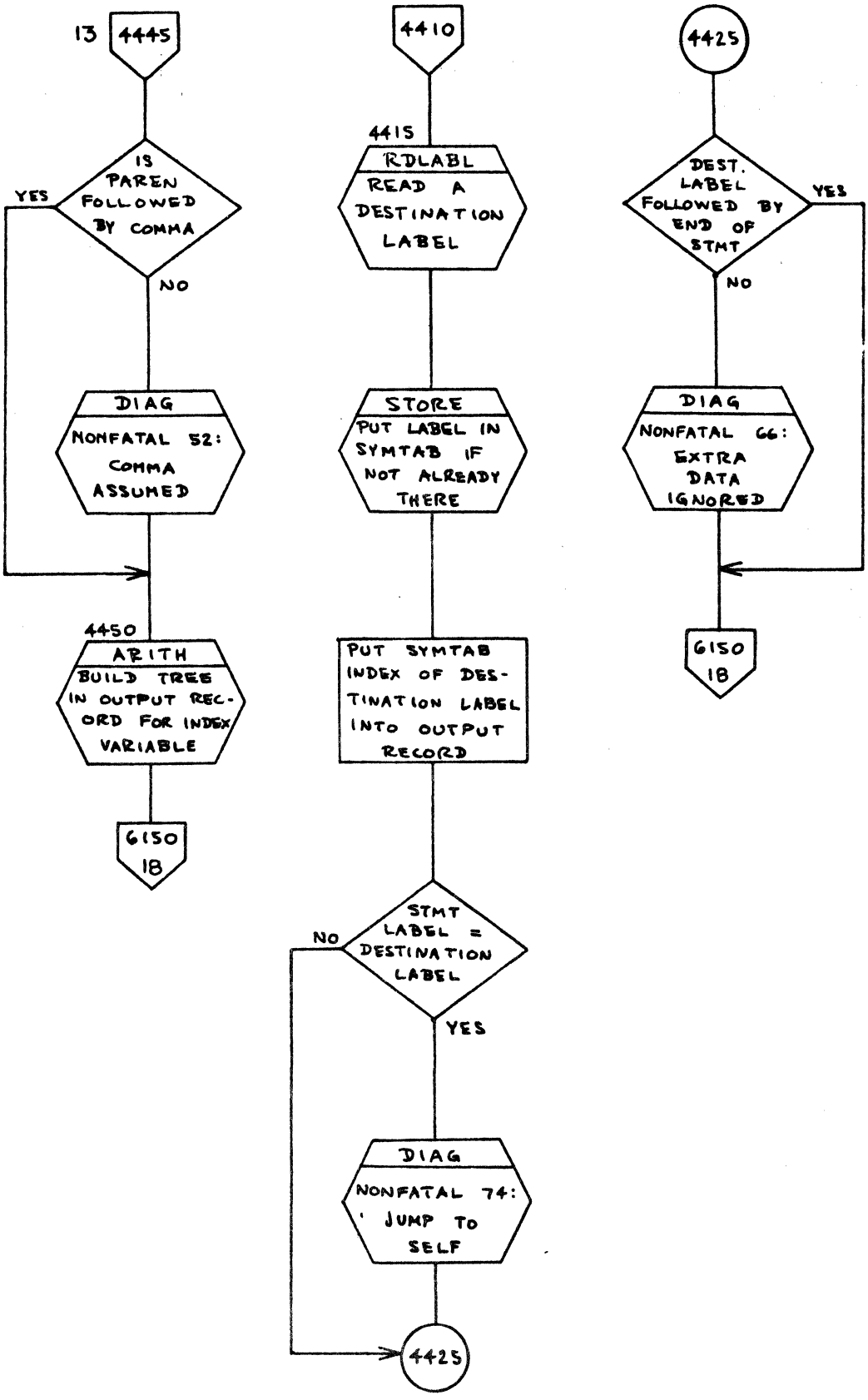

8
5000
GETF
TRY TO READ OCTAL CONSTANT
VALID OCTAL CONSTANT READ
YES
NO
ANY INFORMATION READ
YES
NO
DIAG
FATAL 66: EXTRA DATA IGNORED
6150 18
PUT VALUE OF CONSTANT IN OUTPUT RECORD
CHANGE STMT TYPE FROM 'STOP' / 'PAUSE' TO 'STOP N' / 'PAUSE N'
6150 18
6800
BDOPR
MAKE TABLE & OUTPUT ENTRIES FOR 'DO' STMT
6150 18
6100
IOSPR
GENERATE 'CALL' STMTS & TREES IN OUTPUT
6150 18

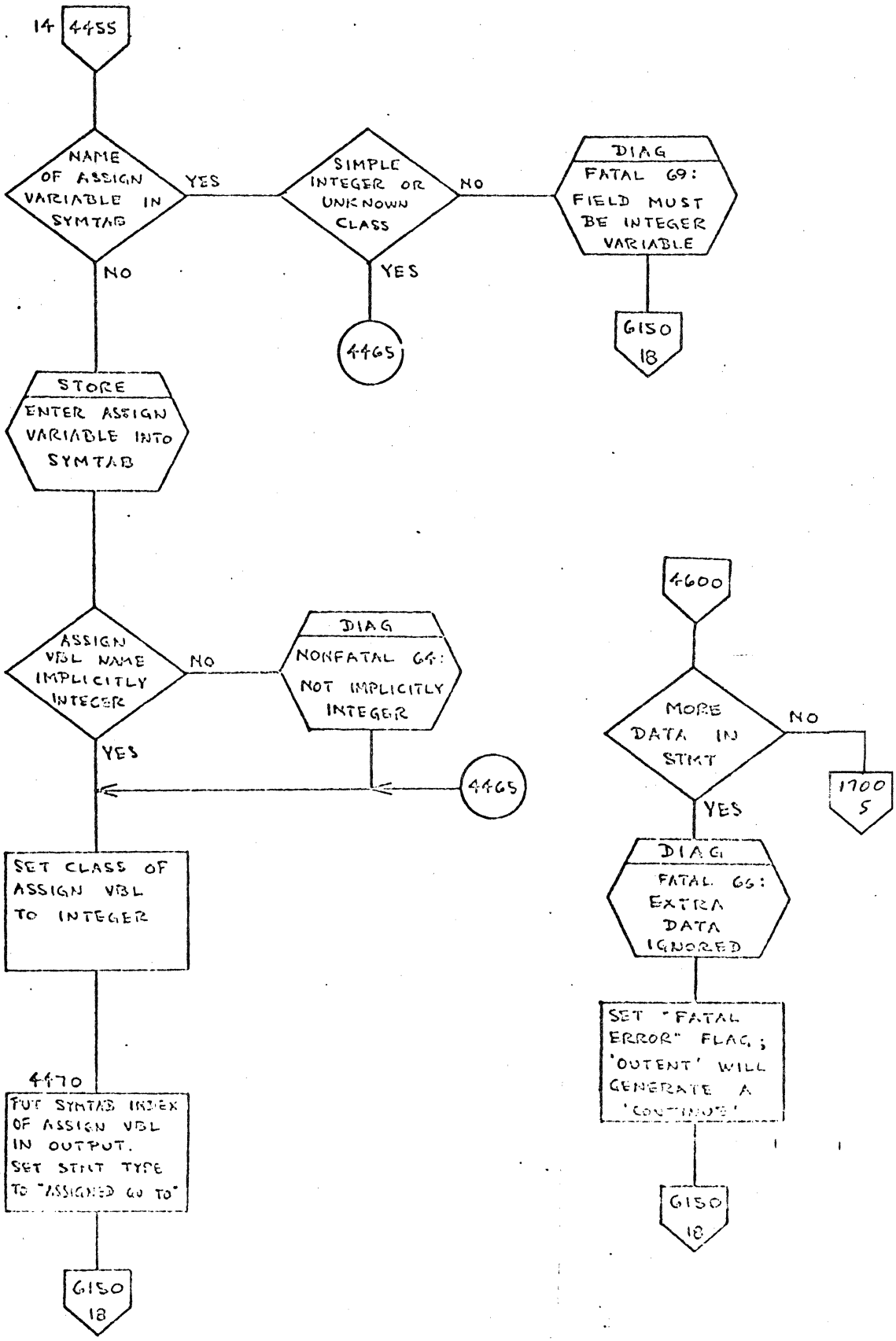

9
3700
GETF
READ FIRST FIELD OF STATEMENT
IS FIELD A SYMBOL
YES
NO
3705
DIAG
FATAL 25: SYNTAX ERROR
3710
IS FIELD TERMINATED BY END OF STMT
YES
3750 10
NO
IS FIELD TERMINATED BY EQUAL SIGN
YES
3745 10
NO
A
A
GETF
READ NEXT FIELD OF STATEMENT
3710
IS THAT SYMBOL IN SYMBOL TABLE
NO
YES
IS SYMBOL A VARIABLE OR UNKNOWN TYPE
YES
NO
DIAG
FATAL 83: NAME NOT VBL OR ARRAY ELEMENT
3710
3720
STORE
ENTER SYMBOL IN SYMBOL TABLE
IS FIELD TERMINATED BY LEFT PAREN
YES
3760 11
NO
IS FIELD TERMINATED BY EQUAL SIGN
NO
3705
YES
THIS IS A REPLACEMENT STMT. CLEAR "DOING SPEC STMTS" SWITCH
3730
MARK SYMBOL TABLE ENTRY FOR LEFT-SIDE NAME AS CLASS = VARIABLE
B
10

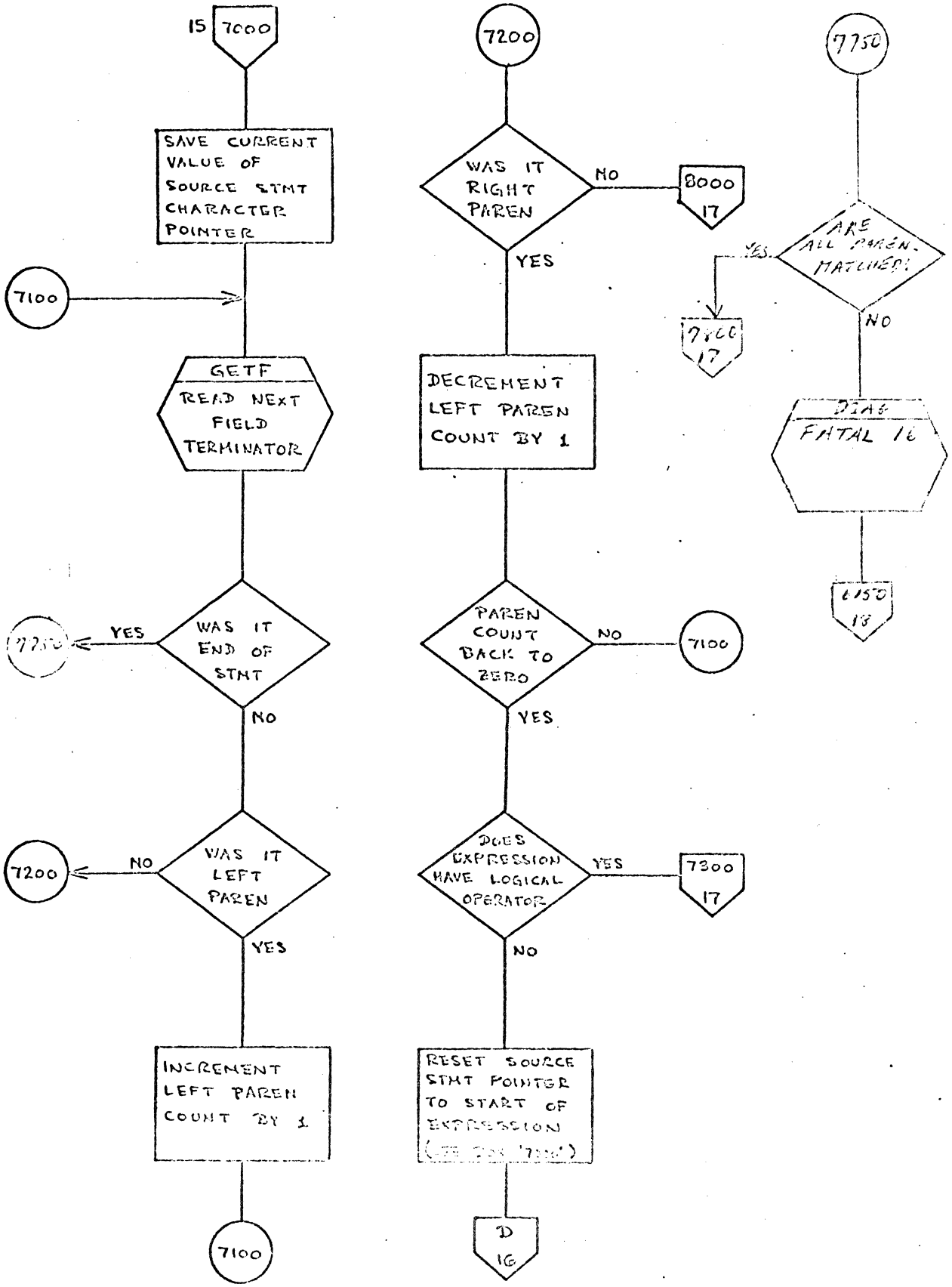

10 B
3745
ARITH
BUILD TREE IN OUTPUT, FOR LEFT SIDE VARIABLE
ARITH
BUILD TREE IN OUTPUT, FOR RIGHT SIDE EXPRESSION
3750
LEFT SIDE ENDS AT EQUAL SIGN
YES
NO
COMPILING A STATEMENT FUNCTION
NO
YES
6150 18
3735
LEFT SIDE ENDS AT RIGHT PAREN
NO
YES
DESTROY SYMBOL TABLE ENTRIES FOR DUMMY ARGUMENTS OF FUNCTION
DUMMY MAY HAVE SAME NAME AS ANOTHER SYMBOL BUT HAS SEPARATE ENTRY
3705 9
6150 18
GETF
READ NEXT FIELD OF STMT
NULL FIELD TERMINATED BY EQUAL SIGN
NO
YES
3705 9
3745

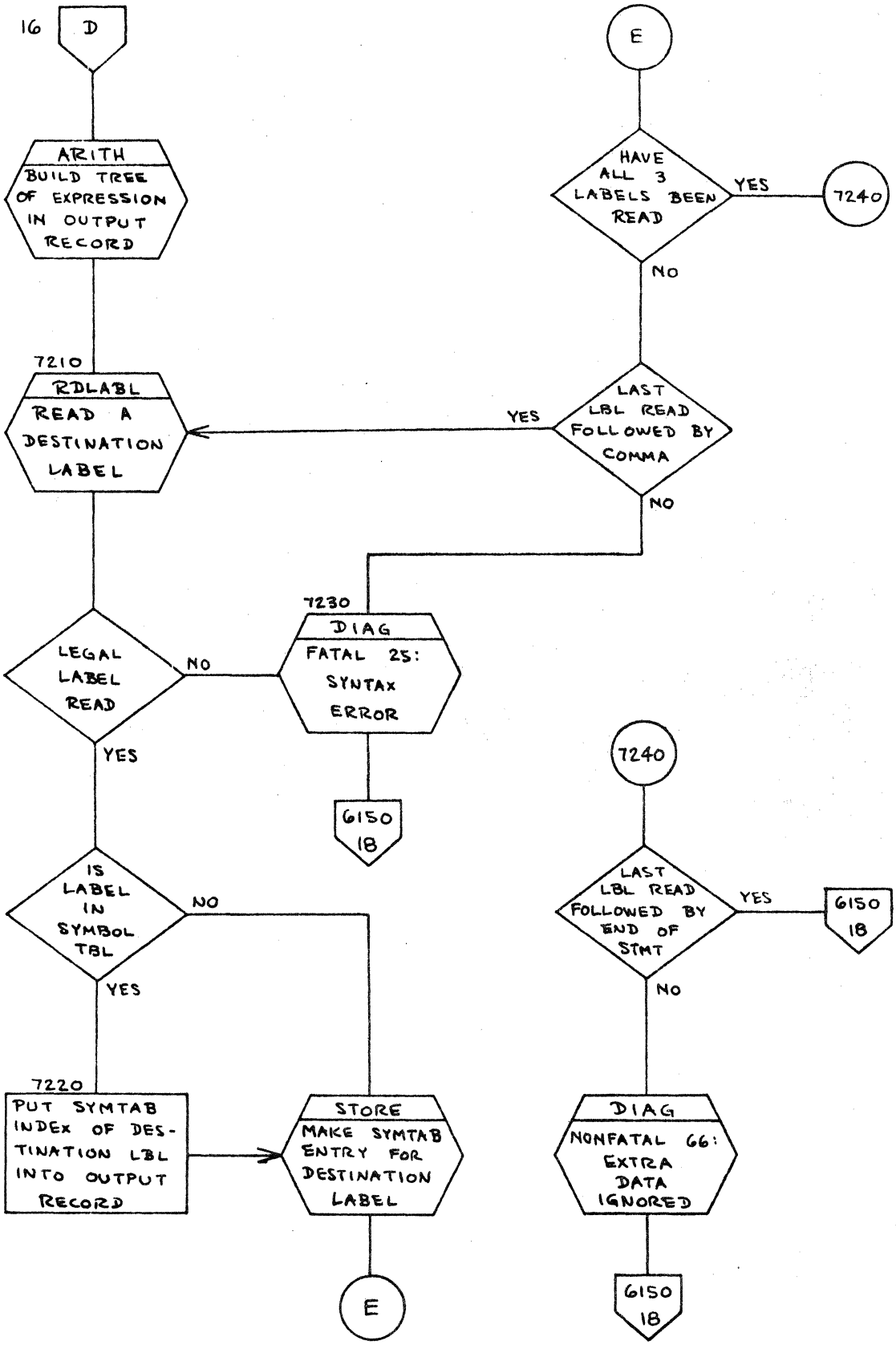

11 3760
IS CLASS OF LEFT-SIDE NAME UNKNOWN
NO
3730 9
YES
IS STATEMENT LABELLED
YES
DIAG
NONFATAL 54: LABEL IGNORED
NO
SET STMT TYPE TO "STATEMENT FUNCTION"
DELETE INDICATION OF STATEMENT LABEL
GETF
READ NEXT FIELD
DUMMY ARGUMENT NAME
NO
3705 9
YES
C
C
SYMBOL, STORE
PUT DUMMY ARGUMENT NAME IN SYMBOL TBL
PUT POINTER TO THAT TBL ENTRY INTO OUTPUT RECORD
NAME FOLLOWED BY A COMMA
NO
3735 10
YES

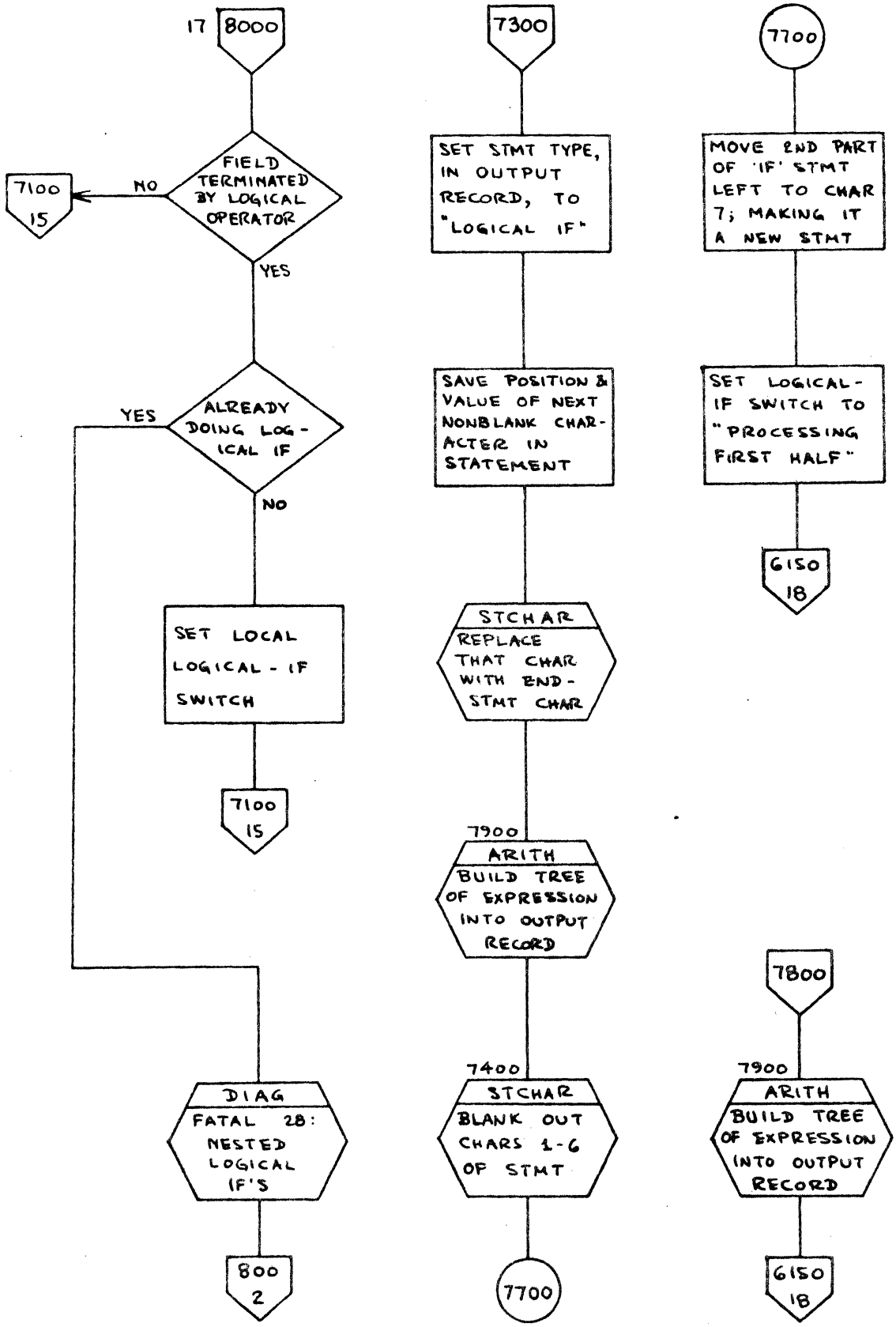

12 4400
GETF
READ FIRST FIELD AFTER "GO TO"
WAS FIELD NULL, ENDED BY LEFT PAREN
YES
NO
4435
SET STATEMENT TYPE, IN OUTPUT RECORD, TO "COMPUTED GO TO"
4415
RDLABL
READ A DESTINATION LABEL
STORE
ENTER LABEL INTO SYMTAB IF NOT ALREADY THERE
PUT SYMTAB INDEX OF DESTINATION LABEL INTO OUTPUT RECORD
STMT LABEL = DESTINATION LABEL
YES
NO
DIAG
NONFATAL 74: JUMP TO SELF
4440
4440
IS LABEL FOLLOWED BY COMMA
YES
NO
IS LABEL FOLLOWED BY RIGHT PAREN
NO
4405
YES
4445 13
4410 13
YES
WAS FIELD AN INTEGER
NO
4455 14
YES
WAS FIELD A SYMBOL
NO
4405
DIAG
FATAL 25: SYNTAX ERROR
6150 18

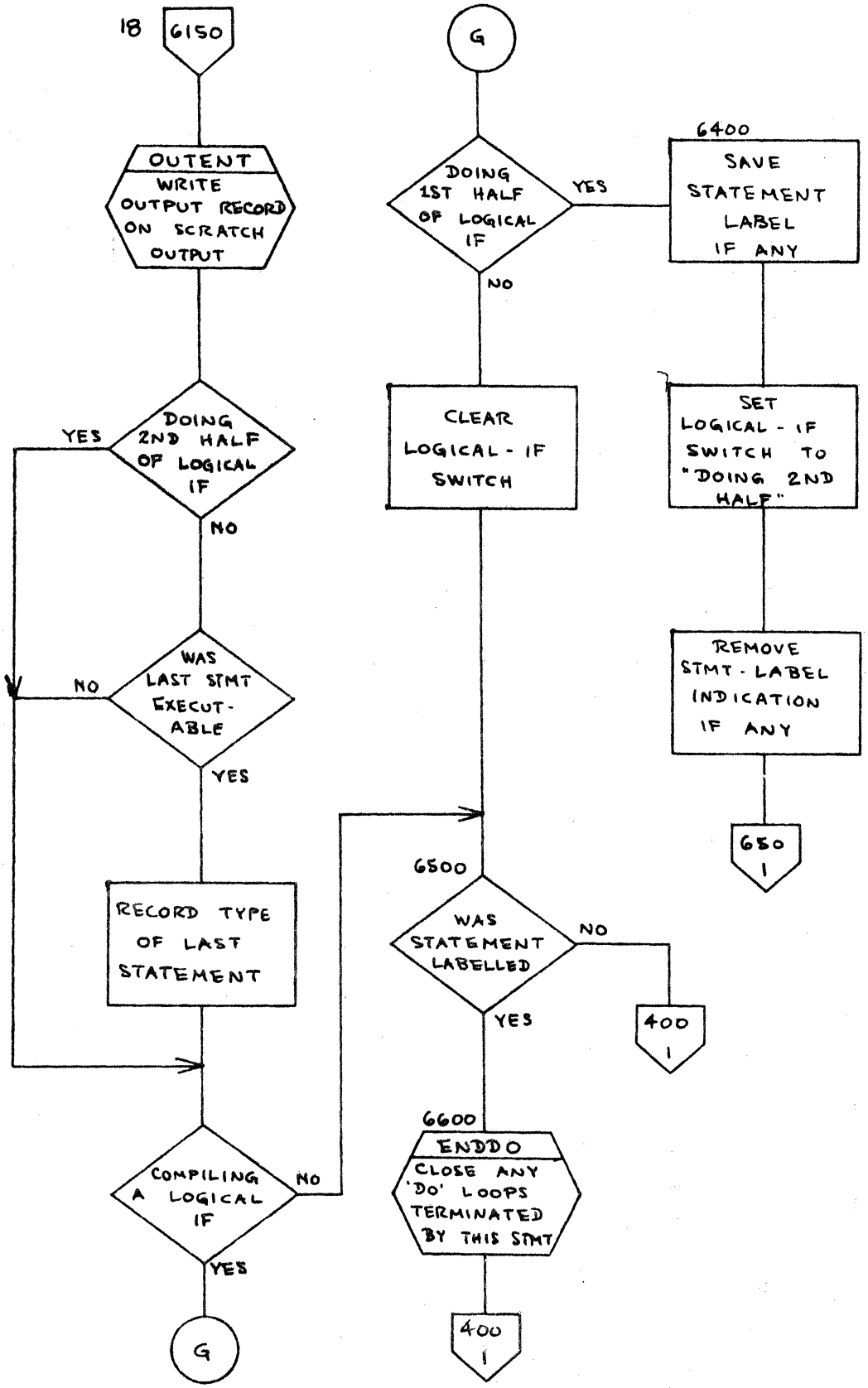

13
4445
IS PAREN FOLLOWED BY COMMA
YES
NO
DIAG
NONFATAL 52: COMMA ASSUMED
4450
ARITH
BUILD TREE IN OUTPUT RECORD FOR INDEX VARIABLE
6150
18
4410
4415
RDLABL
READ A DESTINATION LABEL
STORE
PUT LABEL IN SYMTAB IF NOT ALREADY THERE
PUT SYMTAB INDEX OF DESTINATION LABEL INTO OUTPUT RECORD
STMT LABEL = DESTINATION LABEL
NO
YES
DIAG
NONFATAL 74: JUMP TO SELF
4425
4425
DEST. LABEL FOLLOWED BY END OF STMT
YES
NO
DIAG
NONFATAL 66: EXTRA DATA IGNORED
6150
18

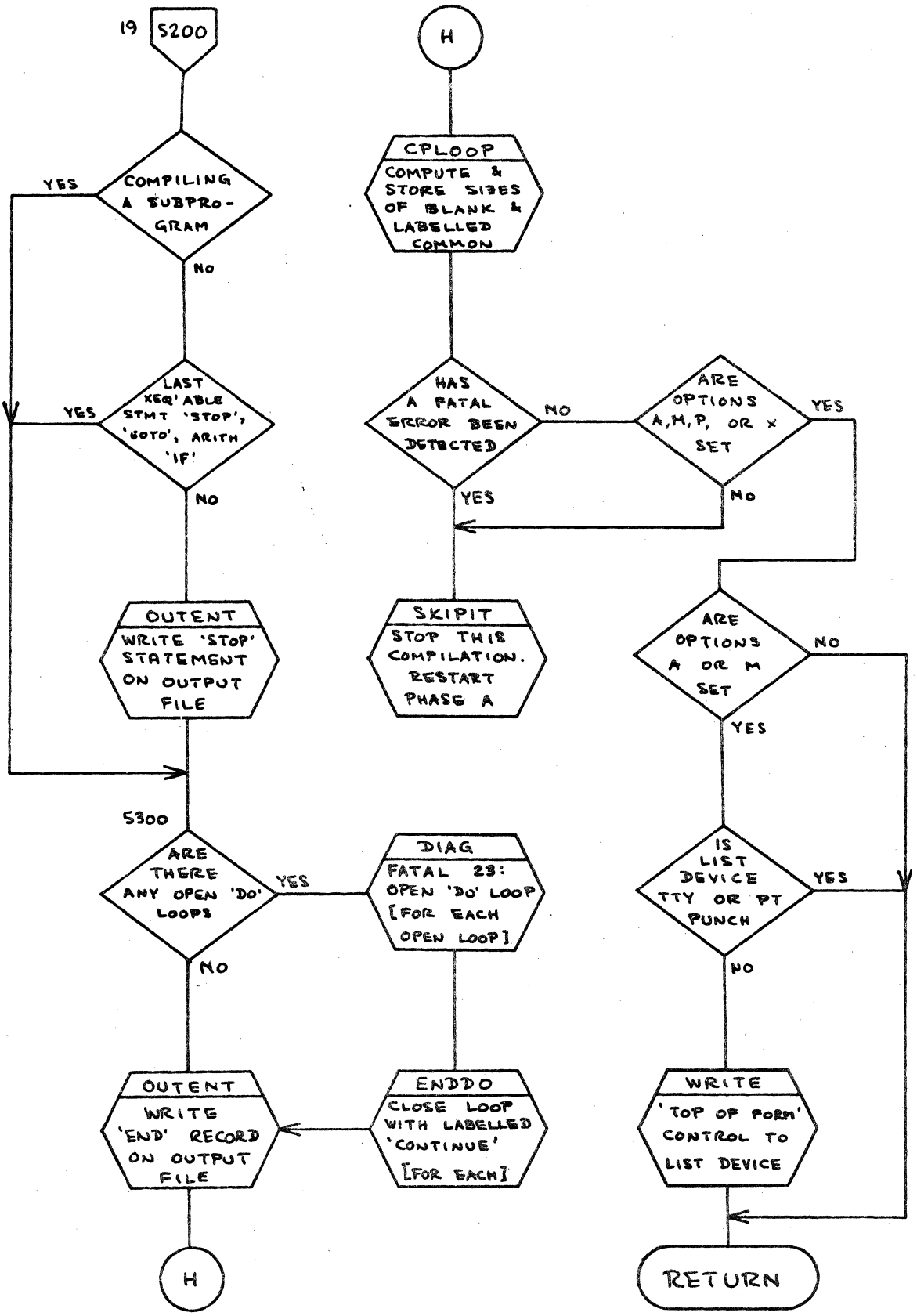

14 4455
NAME OF ASSIGN VARIABLE IN SYMTAB
YES
SIMPLE INTEGER OR UNKNOWN CLASS
NO
DIAG
FATAL 69: FIELD MUST BE INTEGER VARIABLE
G150 18
YES
4465
NO
STORE
ENTER ASSIGN VARIABLE INTO SYMTAB
ASSIGN VBL NAME IMPLICITLY INTEGER
NO
DIAG
NONFATAL 64: NOT IMPLICITLY INTEGER
4465
YES
SET CLASS OF ASSIGN VBL TO INTEGER
4470
PUT SYMTAB INDEX OF ASSIGN VBL IN OUTPUT. SET STMT TYPE TO "ASSIGNED GO TO"
G150 18
4600
MORE DATA IN STMT
NO
1700 5
YES
DIAG
FATAL 66: EXTRA DATA IGNORED
SET "FATAL ERROR" FLAG; 'OUTENT' WILL GENERATE A 'CONTINUE'
G150 18


15 7000
SAVE CURRENT VALUE OF SOURCE STMT CHARACTER POINTER
7100
GETF
READ NEXT FIELD TERMINATOR
WAS IT END OF STMT
YES
NO
7250
WAS IT LEFT PAREN
NO
7200
YES
INCREMENT LEFT PAREN COUNT BY 1
7100
7200
WAS IT RIGHT PAREN
NO
8000 17
YES
DECREMENT LEFT PAREN COUNT BY 1
PAREN COUNT BACK TO ZERO
NO
7100
YES
DOES EXPRESSION HAVE LOGICAL OPERATOR
YES
7300 17
NO
RESET SOURCE STMT POINTER TO START OF EXPRESSION
D 16
7250 17
7250
ARE ALL PAREN. MATCHED?
YES
NO
DIAG FATAL 16
7250 18


16 D
ARITH
BUILD TREE OF EXPRESSION IN OUTPUT RECORD
7210
RDLABL
READ A DESTINATION LABEL
LEGAL LABEL READ
NO
YES
7230
DIAG
FATAL 25: SYNTAX ERROR
6150 18
IS LABEL IN SYMBOL TBL
NO
YES
7220
PUT SYMTAB INDEX OF DESTINATION LBL INTO OUTPUT RECORD
STORE
MAKE SYMTAB ENTRY FOR DESTINATION LABEL
E
E
HAVE ALL 3 LABELS BEEN READ
YES
7240
NO
LAST LBL READ FOLLOWED BY COMMA
YES
NO
7240
LAST LBL READ FOLLOWED BY END OF STMT
YES
6150 18
NO
DIAG
NONFATAL 66: EXTRA DATA IGNORED
6150 18


17 8000
FIELD TERMINATED BY LOGICAL OPERATOR
NO
7100 15
YES
ALREADY DOING LOGICAL IF
YES
NO
SET LOCAL LOGICAL - IF SWITCH
7100 15
DIAG
FATAL 28: NESTED LOGICAL IF'S
800 2
7300
SET STMT TYPE, IN OUTPUT RECORD, TO "LOGICAL IF"
SAVE POSITION & VALUE OF NEXT NONBLANK CHARACTER IN STATEMENT
STCHAR
REPLACE THAT CHAR WITH END-STMT CHAR
7900
ARITH
BUILD TREE OF EXPRESSION INTO OUTPUT RECORD
7400
STCHAR
BLANK OUT CHARS 1-6 OF STMT
7700
7700
MOVE END PART OF 'IF' STMT LEFT TO CHAR 7; MAKING IT A NEW STMT
SET LOGICAL-IF SWITCH TO "PROCESSING FIRST HALF"
6150 18
7800
7900
ARITH
BUILD TREE OF EXPRESSION INTO OUTPUT RECORD
6150 18


18
6150
OUTENT
WRITE OUTPUT RECORD ON SCRATCH OUTPUT
DOING 2ND HALF OF LOGICAL IF
YES
NO
WAS LAST STMT EXECUT-ABLE
NO
YES
RECORD TYPE OF LAST STATEMENT
COMPILING A LOGICAL IF
NO
YES
G
G
DOING 1ST HALF OF LOGICAL IF
YES
NO
6400
SAVE STATEMENT LABEL IF ANY
CLEAR LOGICAL - IF SWITCH
SET LOGICAL - IF SWITCH TO "DOING 2ND HALF"
REMOVE STMT - LABEL INDICATION IF ANY
650
1
6500
WAS STATEMENT LABELLED
NO
YES
400
1
6600
ENDDO
CLOSE ANY 'DO' LOOPS TERMINATED BY THIS STMT
400
1


19 5200
COMPILING A SUBPRO-GRAM
YES
NO
LAST XEQ'ABLE STMT 'STOP', 'GOTO', ARITH 'IF'
YES
NO
OUTENT
WRITE 'STOP' STATEMENT ON OUTPUT FILE
5300
ARE THERE ANY OPEN 'DO' LOOPS
YES
NO
OUTENT
WRITE 'END' RECORD ON OUTPUT FILE
H
H
CPLOOP
COMPUTE & STORE SIZES OF BLANK & LABELLED COMMON
HAS A FATAL ERROR BEEN DETECTED
NO
YES
SKIPIT
STOP THIS COMPILATION. RESTART PHASE A
DIAG
FATAL 23: OPEN 'DO' LOOP [FOR EACH OPEN LOOP]
ENDDO
CLOSE LOOP WITH LABELLED 'CONTINUE' [FOR EACH]
ARE OPTIONS A,M,P, OR X SET
YES
NO
ARE OPTIONS A OR M SET
NO
YES
IS LIST DEVICE TTY OR PT PUNCH
YES
NO
WRITE 'TOP OF FORM' CONTROL TO LIST DEVICE
RETURN

2.2.2.1 Subroutine GNST

The GNST subroutine reads the source statements which are to be processed by the PHASEA subroutine.

With the exception of comment cards, each source card read by GNST is processed in three stages.

STAGE 1.

The GNST subroutine calls the READ entry point in IOPRBA to read an 82 character record from unit 1 in the ASCII mode. The input buffer, ISRS, occupies 41 words of Phase A blank common. The GNST routine then scans ISRS from right to left and stores in the variable LENGTH the location of the right-most word containing a non-blank character (not equal to $20020).

STAGE 2.

The contents of the ISRS buffer are printed and transferred to the source buffer ISORS, which occupies 208 words of Phase A blank common.

STAGE 3.

The characters stored in the source buffer ISORS are converted from ASCII code to the internal Fortran code (see 2.2.2.0.3).

Continuation records are processed through all three stages during one call to GNST. The initial record of a Fortran statement other than END is passed through stage 1 on one call to GNST, and through stages 2 and 3 on the following call. Because GNST reads at least one record ahead in the source language input, at least two consecutive source records will be processed during a single call to this subroutine.

2.2.2.1.1 Normal Flow of Subroutine GNST

This section describes the operations accomplished by a single call to GNST after the first non-comment record has been processed and before the END record has been read. At entry to GNST, the input buffer ISRS will contain the initial record of a Fortran statement that has previously been processed through stage 1. The first task of GNST is to process this ISRS buffer through stage 2;

that is, to print the record (if the list option switch, IL, is on) with an internal statement number, and then move the record to ISORS. A full seventy-two characters (36 words) are transfered to ISORS to allow for later continuation cards. The location of the rightmost non-blank word in the record, formerly saved in LENGTH, is transferred to LENSAV.

The subroutine GNST next reads a new input record into ISRS and processes it through state 1, any characters stored as $FF by the paper tape or magnetic tape drivers are changed to blank ($20) and the location of the rightmost non-blank word is ISRS is stored in LENGTH.

After processing the new input record through stage 1, GNST checks for three special cases - a blank record, a comment record, or a continuation record (not zero or blank in character #6). These three types of records are immediately processed as follows:

a) Blank record - ignored. A new input record is read and processed through stage 1.

b) Comment record - printed and ignored. If the IL switch is on, the comment record is printed without a statement number and then a new record is read and processed through stage 1.

c) Continuation record - completely processed in one call. The contents of ISRS are printed without a statement number (if IL = 1). Characters 7 through 72 of the continuation record are added to the end of the statement in the ISORS buffer. ISORSX (set to 1 on entry) is set to the beginning of the added record. The variable LENSAV is reset equal to LENGTH so that LENSAV contains the location of the rightmost non-blank character in the last record added to the ISORS buffer.

If the record in ISRS is the initial line of a Fortran statement other than END, further processing is deferred until the next call to GNST. At this point in the flow, a full Fortran statement (with all continuations) is contained in ISORS. The final step of the GNST subroutine is to convert this statement from ASCII code to internal Fortran code. GNST exits with a full converted Fortran statement in ISORS and the initial line of a Fortran statement in ASCII in ISRS.

2.2.2.1.2. First and last source records.

Until the first non comment record is read, stages two and three are skipped in the GNST flow. The variable LENSAV is set to zero by a data statement, and therefore remains zero until a non-comment statement is read. When LENSAV is equal to zero, the printing and transfer of ISRS at the beginning of GNST and the code conversion at the end are not executed. Initial comment cards are printed as usual.

At the end of stage two, after ISRS has been transferred to ISORS, ISRS is tested for the presence of an END record. If the card in ISRS is an END card no more input records are read and the END card is immediately converted to internal code.

2.2.2.1.3. Errors

When processing a non-comment statement, GNST tests for the following illegalities.

1. That the first non-comment record is an END statement.
2. That the first non-comment record is a continuation card.
3. That there are more than five continuation cards for a source statement.
4. That characters illegal in Fortran occur in the source statement.

2.2.2.1.3.1. End statement as first non-comment record.

The variable LENSAV is set to zero until the first non-comment card is read and processed. If the END card is read while LENSAV =0, an error diagnostic is given, and further processing of this program is terminated by a call to subroutine SKIPIT.

2.2.2.1.3.2. Continuation card as first non-comment card of deck.

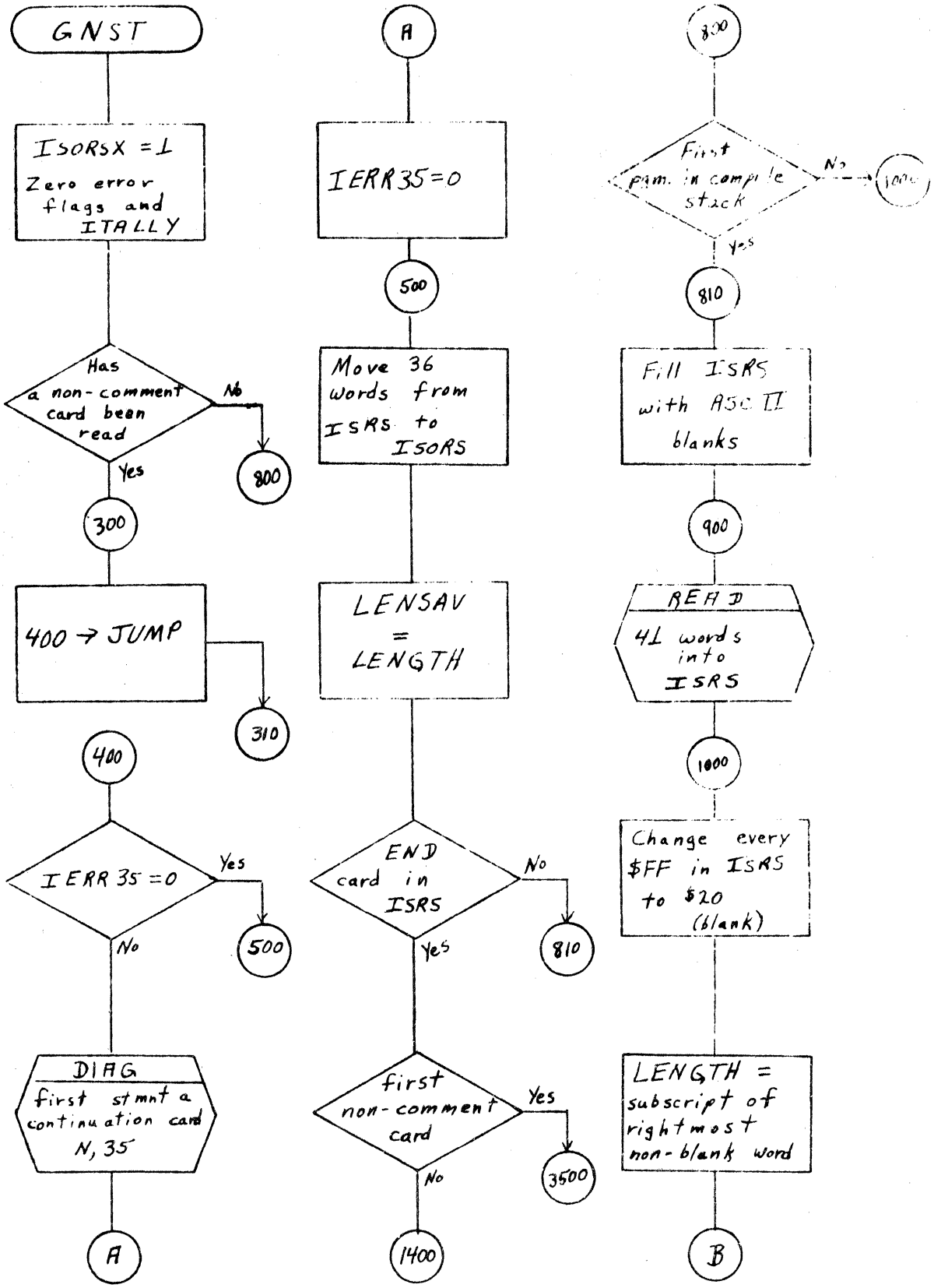
Error flag IERR35 is set, character number 6 is set to blank, and the record is treated as the initial record of a Fortran statement. After the record is fully processed the appropriate error diagnostic is printed.

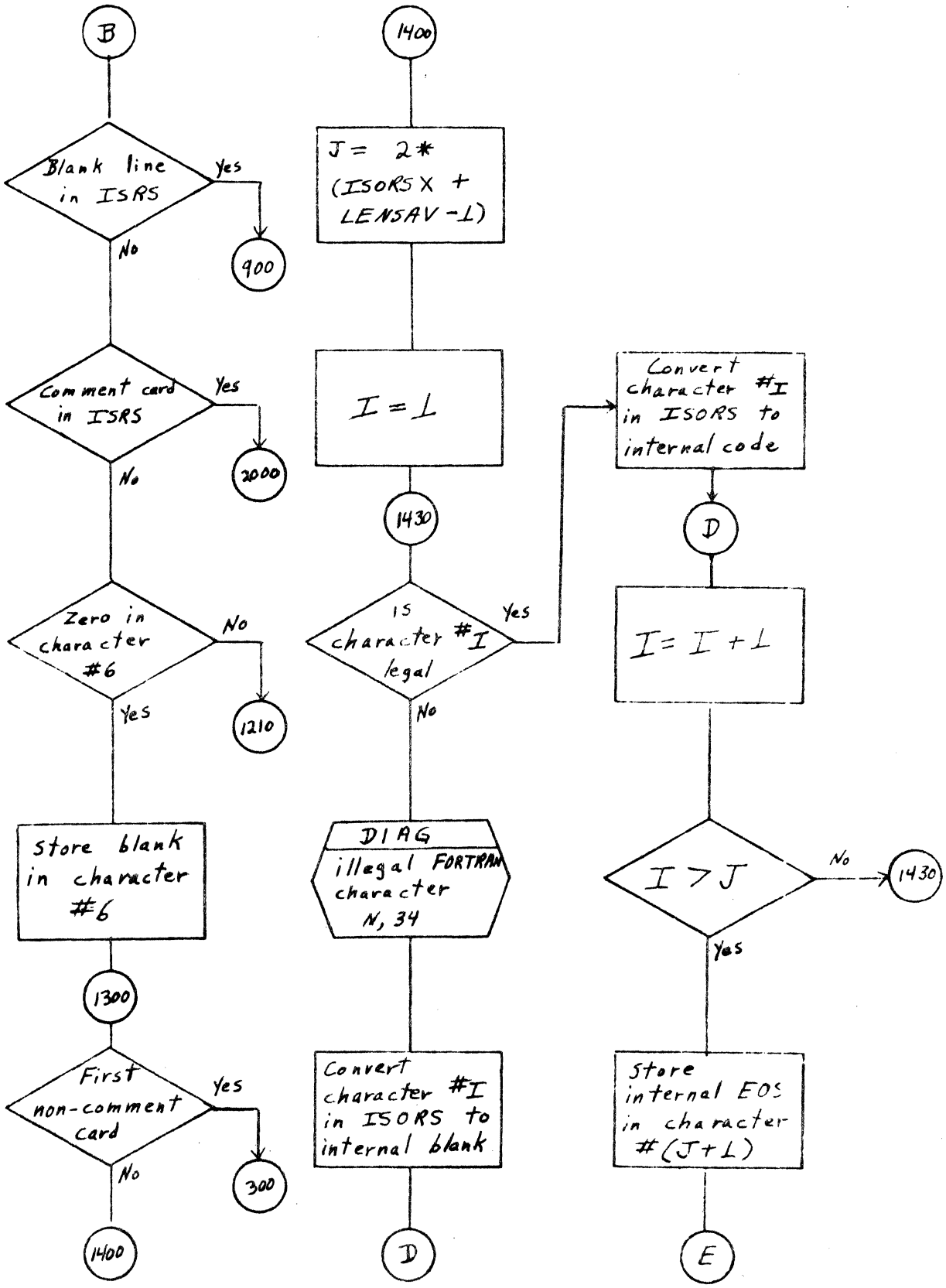
2.2.2.1.3.3. More than 5 continuation cards.

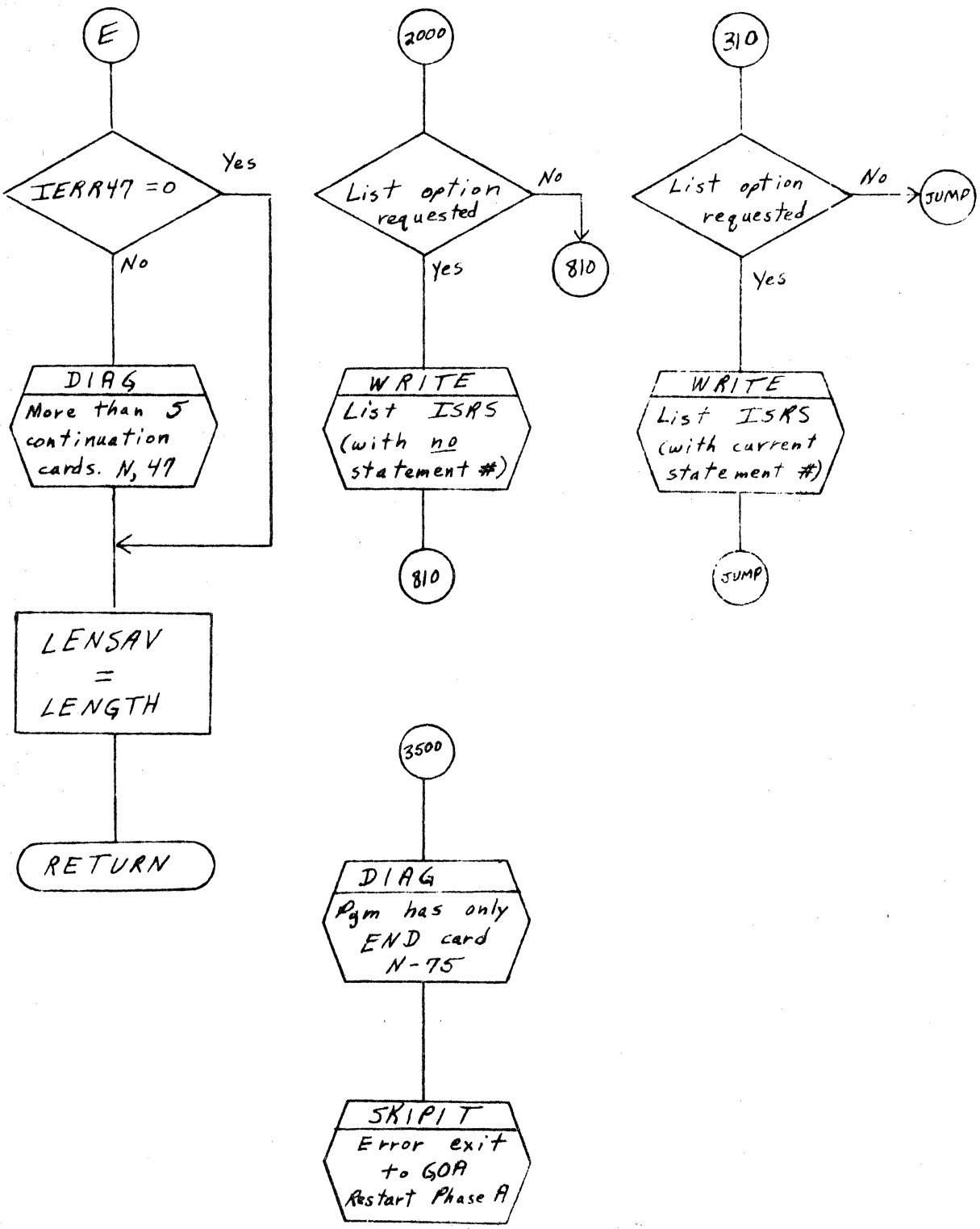
ITALLY is a tally register for the number of continuation cards per statement. ITALLY is set to zero on entry to GNST and increased by 1 for each continuation card read. When ITALLY exceeds 5 in value the error flag IERR47 is set. Continuation cards in excess of 5 are printed but are not added to ISORS. When the next initial statement card is read, the diagnostic is printed.

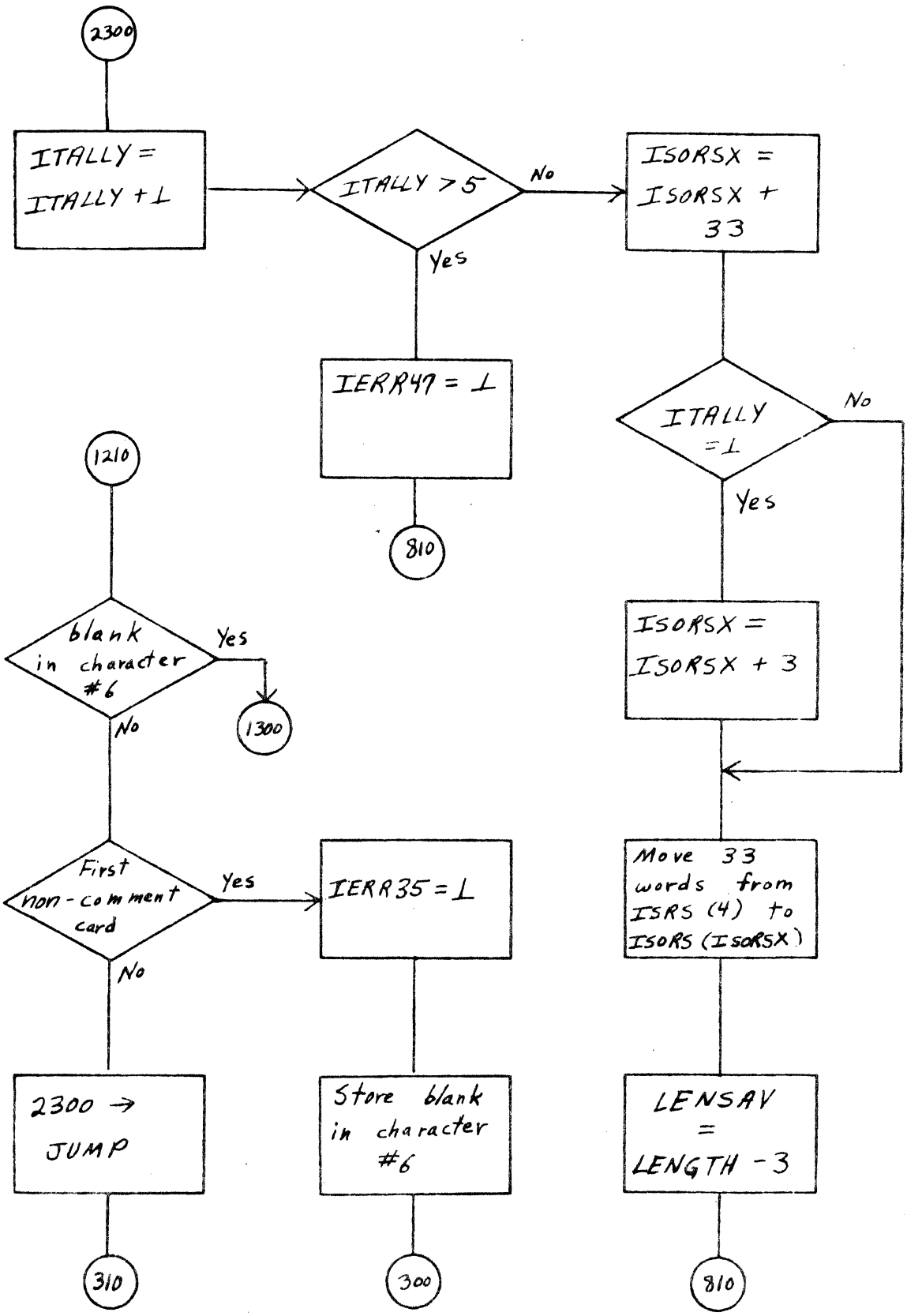
2.2.2.1.3.4. Illegal characters

Characters not contained in the Fortran character set are converted to blanks in stage 3, after printing the record. A non-fatal diagnostic is printed for each illegal character.


GNST
ISORSX = 1
Zero error flags and ITALLY
Has a non-comment card been read
No
Yes
800
300
400 → JUMP
310
400
IERR 35 = 0
Yes
No
500
DIAG
first stmnt a continuation card
N, 35
A
A
IERR35=0
500
Move 36 words from ISRS to ISORS
LENSAV = LENGTH
END card in ISRS
No
Yes
810
first non-comment card
Yes
No
3500
1400
800
First parm. in compile stack
No
Yes
810
Fill ISRS with ASC II blanks
900
READ
41 words into ISRS
1000
Change every $FF in ISRS to $20 (blank)
LENGTH = subscript of rightmost non-blank word
B


B
Blank line in ISRS
Yes
No
900
Comment card in ISRS
Yes
No
2000
Zero in character #6
No
Yes
1210
Store blank in character #6
1300
First non-comment card
Yes
No
300
1400
1400
J = 2* (ISORS X + LENSAV -1)
I = 1
1430
IS character #I legal
Yes
No
DIAG illegal FORTRAN character N, 34
Convert character #I in ISORS to internal blank
D
Convert character #I in ISORS to internal code
D
I = I + 1
I > J
No
1430
Yes
Store internal EOS in character #(J+1)
E


E
IERR47 = 0
Yes
No
DIAG
More than 5 continuation cards. N, 47
LENSAV = LENGTH
RETURN
2000
List option requested
No
810
Yes
WRITE
List ISRS (with no statement #)
810
310
List option requested
No
JUMP
Yes
WRITE
List ISRS (with current statement #)
JUMP
3500
DIAG
Pgm has only END card N-75
SKIPIT
Error exit to GOA Restart Phase A


2300
ITALLY = ITALLY + 1
ITALLY > 5
No
Yes
ISORSX = ISORSX + 33
IERR47 = 1
810
ITALLY = 1
No
Yes
1210
ISORSX = ISORSX + 3
blank in character #6
Yes
No
1300
First non-comment card
Yes
No
IERR35 = 1
Move 33 words from ISRS (4) to ISORS (ISORSX)
2300 → JUMP
Store blank in character #6
LENSAV = LENGTH - 3
310
300
810

2.2.2.2 TYPE

The TYPE subroutine is used by PHASE A to determine the type number for the statement. If the TYPE subroutine is able to determine the type number for the statement, it will place the number in the location IBUF2{3} and exit. If TYPE is unable to identify the statement, it sets an error indicator and returns to the calling routine.

2.2.2.2.1 Entrance to TYPE

Entrance to the TYPE subroutine is made by the following statement:

```
CALL TYPE {error flag}
```

The subroutine statement which leads the TYPE subroutine is as follows:

```
SUBROUTINE TYPE {IX1}
```

where IX1 is the name of the formal parameter corresponding to °error flag°.

2.2.2.2.2 Initialization for TYPE

Immediately after entrance is made to the TYPE subroutine, the following steps of initialization occur:

1. The error flag is cleared.
   0 → IX1
2. The parentheses level tally register is cleared.
   0 → ITS1
3. The column counter is set to the character position in the source buffer at which scanning of the source statement begins.
   7 → ISORSX

2.2.2.2.3 Identification of Source Statement

If the TYPE subroutine is able to identify the source statement as a replacement statement, it will place the appropriate type number in the holder IBUF2{3} and exit. {See item 2.2.2.2.3.1} If the TYPE subroutine is unable to identify the source statement as a replacement it will search the ISTN table in order to obtain a tape number. {See item 2.2.2.2.3.2}

2.2.2.2.3.1 Replacement Statement

The TYPE subroutine begins a scan of the source statement beginning with the 7th character from the beginning. The necessary and sufficient conditions for identifying the statement as a replacement is as follows:

> The statement must contain an "=" sign not enclosed in parenthesis which is not followed by a comma not enclosed in parenthesis.

For purposes of scanning, the GETC subroutine is used to extract a single character from the source statement and place it in JCHAR. If the switch JBLANK is set to zero when a call is made to GETC, the space character, whose internal FORTRAN code = 46 is not treated as a significant character. Therefore given a source statement -

```
D 0 1 0 I = 1 , 5
```

the TYPE subroutine sees only the significant character and not the intervening spaces which causes it to appear as -

```
DO10I=1,5
```

This ordinarily would be regarded as a statement in which -

```
name = expression,
```

except according to the previously stated rule for a replacement statement, the comma which is not enclosed by parentheses excludes the possibility of this being a replacement statement.

Another special case occurs for the following statement:

```
FORMAT  10HX=A(1,5)
```

In this example TYPE encounters an "=" sign which is followed by a comma which is enclosed in parentheses. Ordinarily it would appear to the TYPE subroutine as a

```
name = expression
```

type of statement wherein -

1. The name FORMAT10HX is illegal since it is 10 characters long, and -
2. the expression consists of an element of an array, A(1,5).

However, as a result of scanning, the TYPE subroutine encounters a numeric field terminated with an H. If the value of the numeric field is less than some arbitrarily large number (in TYPE it is 400) an assumption is made by TYPE that it has encountered a FORMAT statement with a Hollerith Field Specification. Therefore, TYPE counts characters according to the number preceding the "H" and beginning with the character following the "H".

In order to count characters, the TYPE subroutine sets the switch JBLANK to 1. It then calls the GETC routine the number of times specified by the numeric field preceding the "H". Each time GETC is called, the column counter is increased by 1. Since (JBLANK) = 1, spaces in the field, following the "H" are regarded as significant characters. When the end of the Hollerith string is encountered, JBLANK is reset to 0 and the column counter is positioned for the character following the Hollerith string.

When slewing the Hollerith string, the TYPE subroutine does not recognize the equal sign it encounters. Therefore, it will not erroneously regard a FORMAT statement as a replacement.

2.2.2.2.2.3.2 Other Statements

The TYPE subroutine attempts to identify statements other than replacements by searching the ISTN table. An entry in the ISTN table contains the name of a statement stored 1 character to a word, the characters represented by the Internal FORTRAN code. The name of the statement is followed by the type number for the statement, and preceded by a pointer to the start of the next entry in the ISTN table. If the space occupied by the ISTN table is regarded as a one dimensional array, the pointer to the start of the next table entry may be regarded as a subscript with which to reference the element (or word) which is the start position of the next ISTN entry. If the name of a statement has K characters, the ISTN table entry has K+2 elements (or words). The 1st ISTN table entry is as follows:

(ISTN(1)) = 12 pointer to next table entry; next entry begins at ISTN(12).

DIMENSION	name
0	type number; type number is stored at {ISTN{1}}-1.

The 1st ISTN table entry occupies storage reserved for elements ISTN{1} through ISTN{11}. The second ISTN entry begins at ISTN{12}.

The final ISTN table entry is for the end statement. It appears as follows:

```
341
E
N
D
32          = type number
```

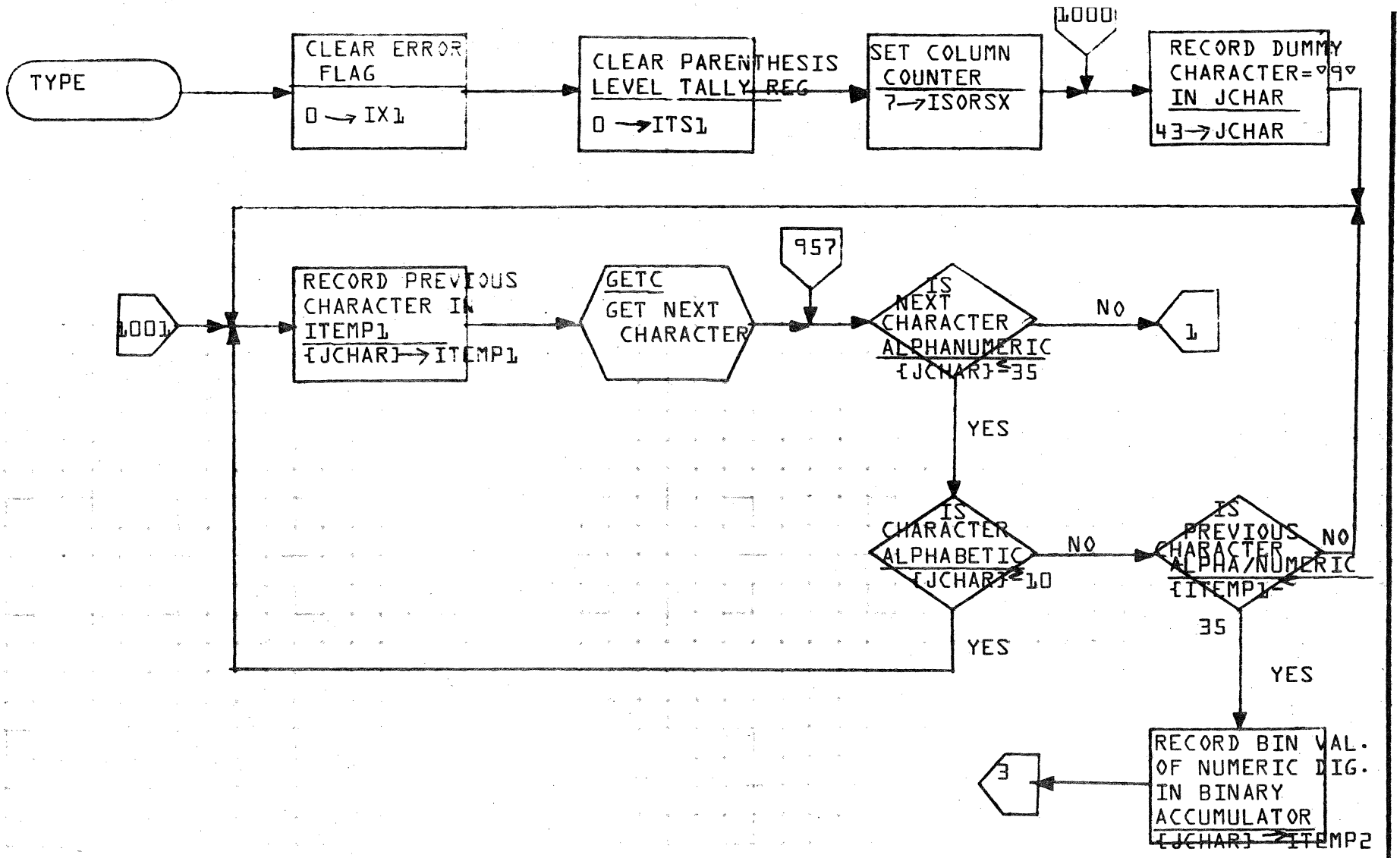
If the flag {ISPEC} = 0 enabling PHASEA to process specification statements, the entire ISTN table is searched in order to identify the source statement. If {ISPEC} = 1 indicating that no subsequent specification statements will be processed, the search of the ISTN table begins with the 1st entry containing a non-specification statement name.

If the TYPE subroutine is able to identify a source statement as a result of an ISTN table search, the type number is read from the last word of the matching entry, placed in IBUF2{3}. An exit is made from TYPE.

If the entry containing the END statement is encountered and no match is made, an error flag is set by -

1 → IX1,

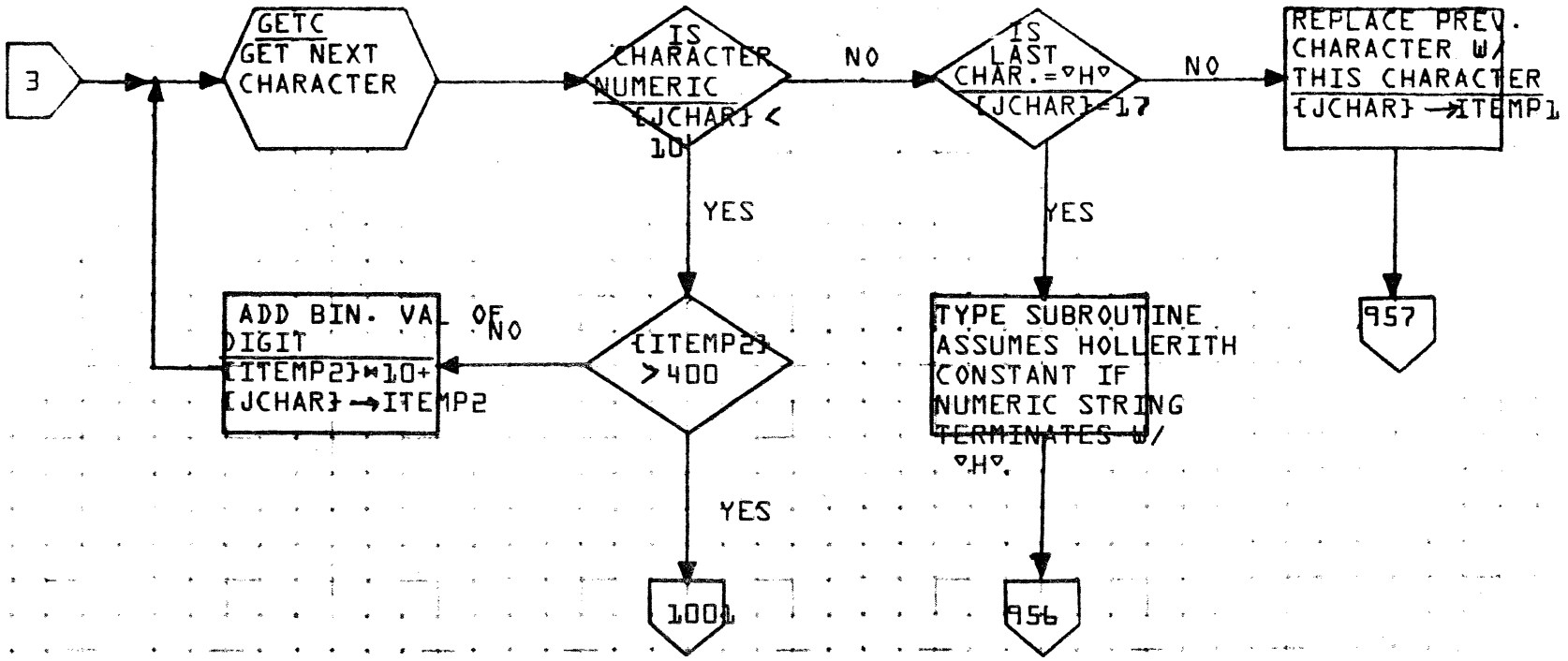
and an exit is made from the TYPE subroutine.



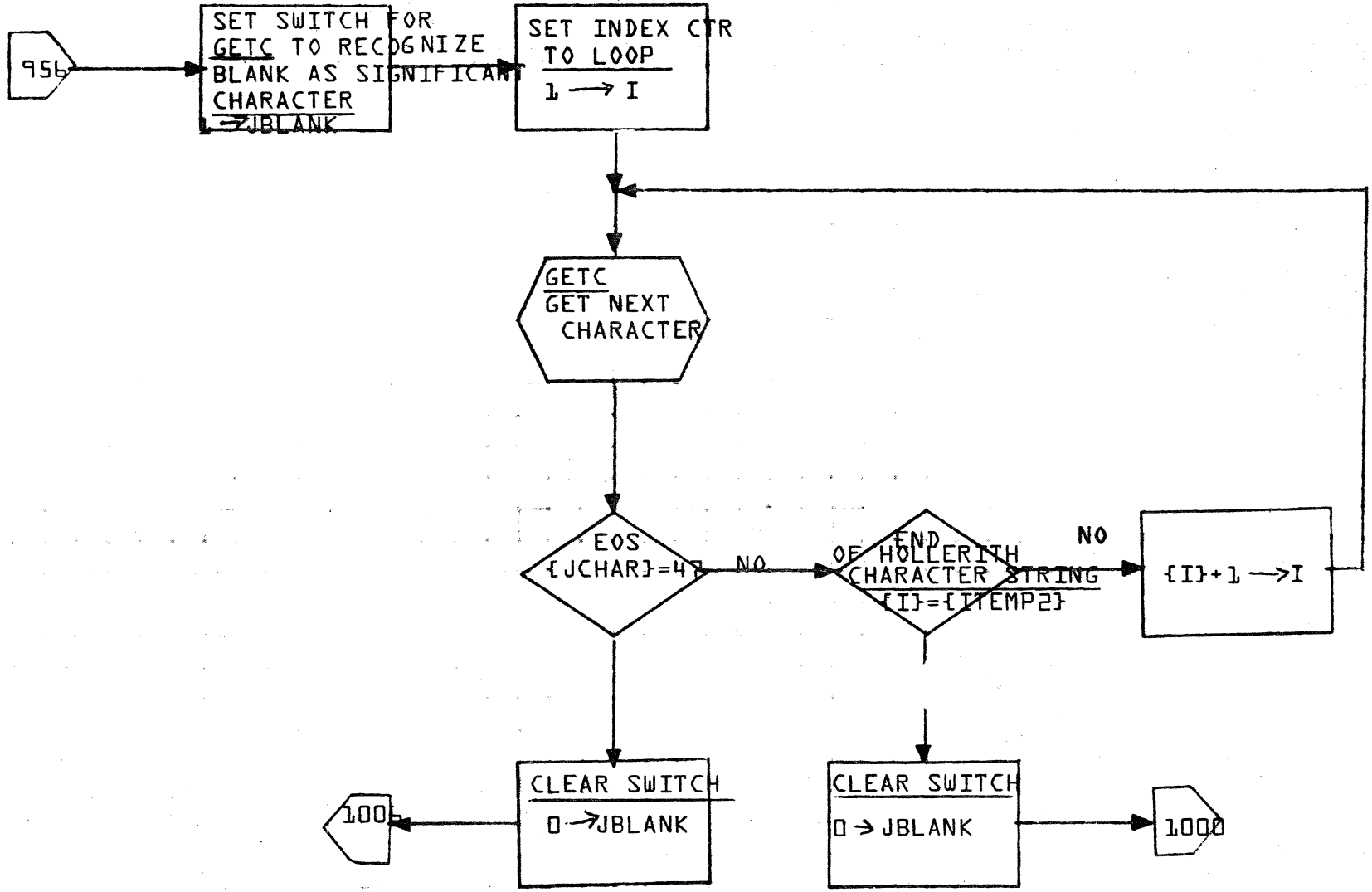
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TYPE		PROJECT MGR.			
		PAGE 1 OF 8	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1365 (FORMERLY CA127-1)

PRINTED IN U.S.A.



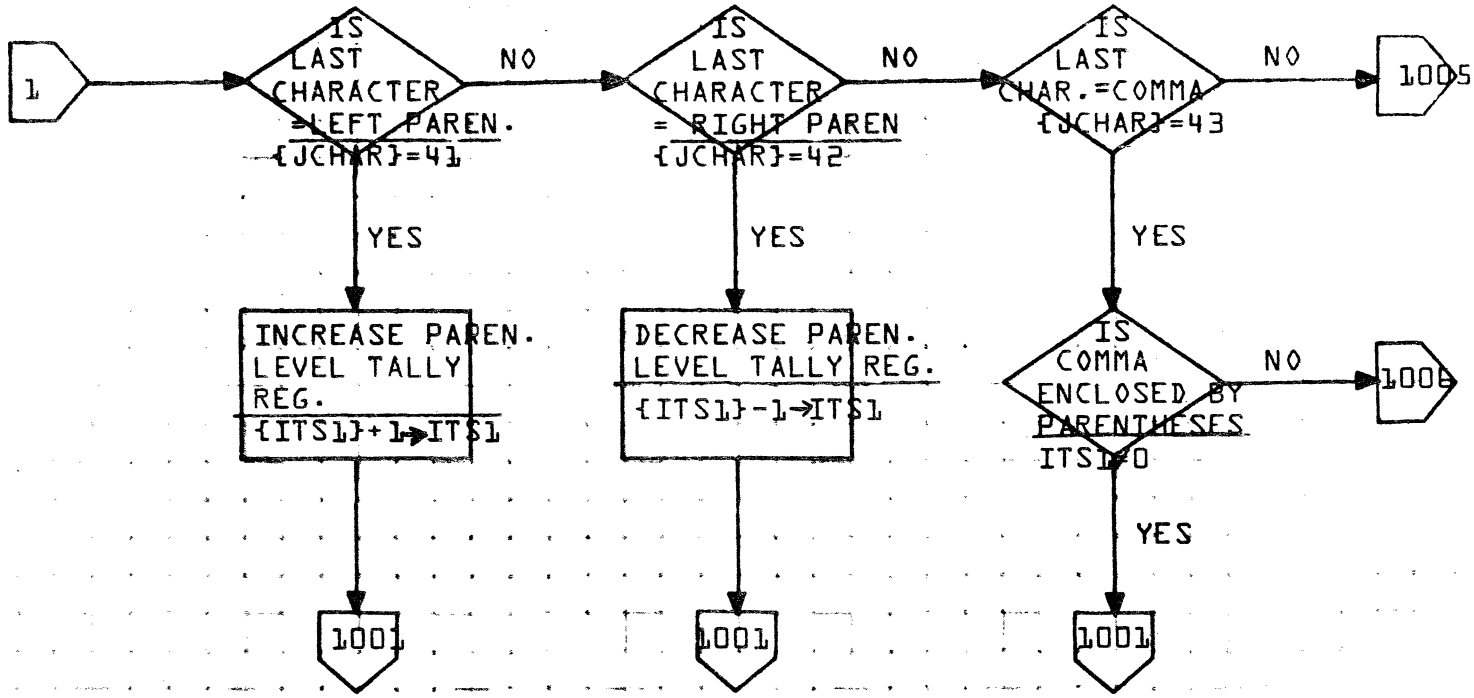
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TYPE		PROJECT MGR.		
	PAGE 2 OF 8		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



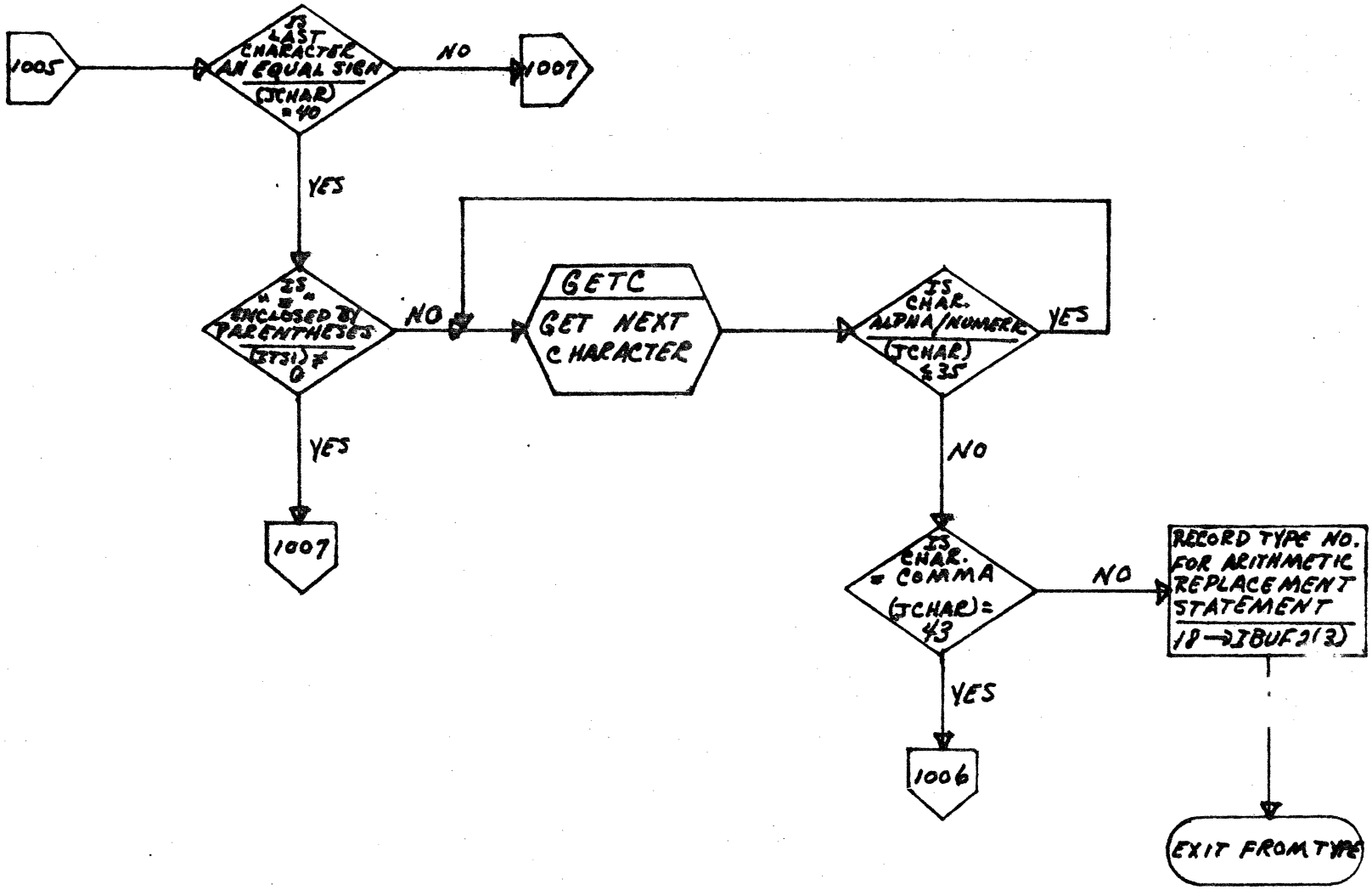
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE TYPE		PROJECT MGR.			
FLOWCHART ☒		PAGE 3 OF 8	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1388 (FORMERLY CA127-1)

PRINTED IN U.S.A.

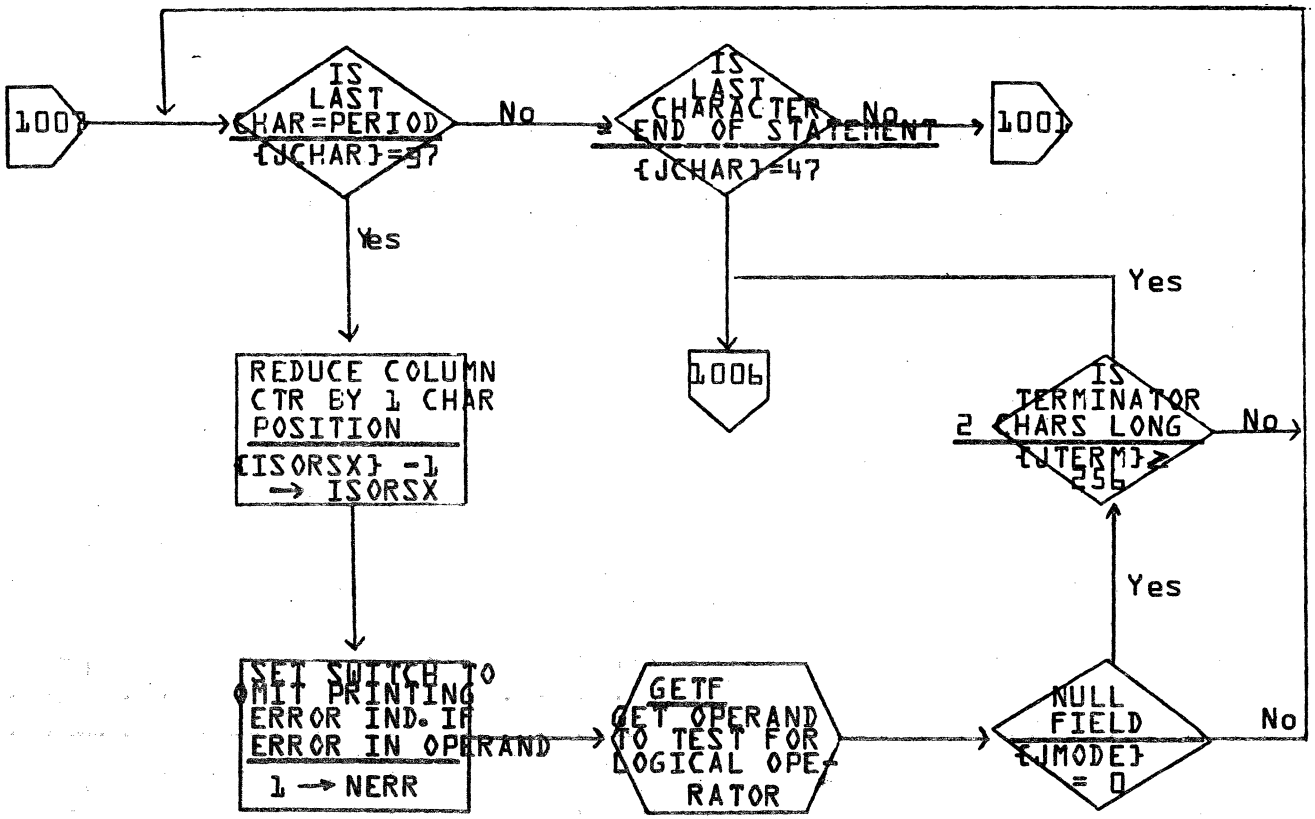


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TYPE		PROJECT MGR.		
		PAGE 4 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

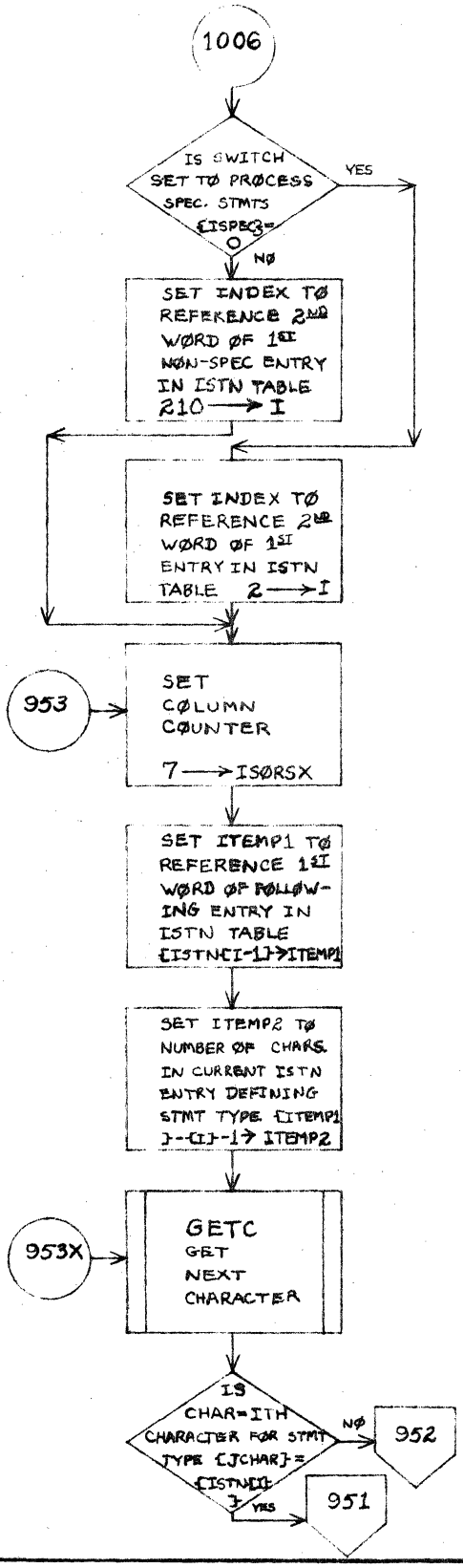


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TYPE		PROJECT MGR.			
		PAGE 5 OF 8	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

CA 127-1

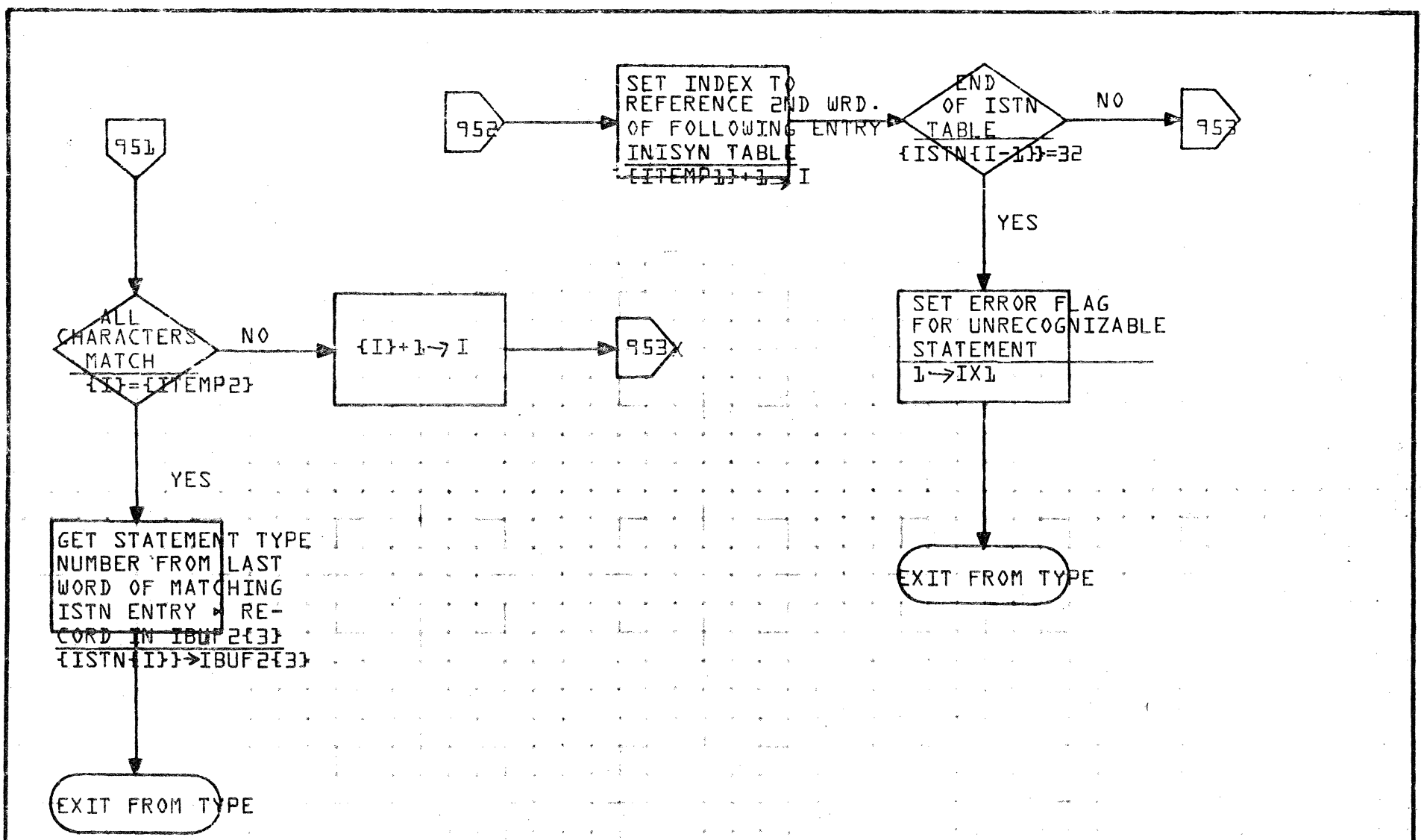


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TYPE		PROJECT MGR.		
		PAGE 6 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



TITLE	TYPE		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET OF

CSD 203



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE TYPE		PROJECT MGR.		
		PAGE 8 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		

2.2.2.3 PEQVS - The Equivalence Table Builder

This routine is called by Phase A upon encountering an executable statement or the END line. EQUIVALENCE statements, which have been saved, in a different form, in the buffer KEQV, are merged into equivalence classes and stored, in a shorthand form, in the equivalence table KEQV. Each entry in KEQV is four words. A symbol table pointer, followed by three words containing subscript values, if any.

This routine has three logical parts or phases. In the first part it checks the subscripts for validity, and calculates a single value for the subscript, if any.

In the second part, all equivalenced variable or array parameters (called a chain) plus the single calculated value are moved into an intermediate table MEQV by chain. Each entry in MEQV has two words, a pointer to the name in the symbol table, and the single calculated value.

In the third part, each chain is sorted by increasing increment size, and the ordered chains are then moved to IEQV. Each entry in IEQV is two words and has the same format as MEQV.

KEQV

1	SYMTAB Pointer	
2		Subscript Values if any
3		
4		

MEQV

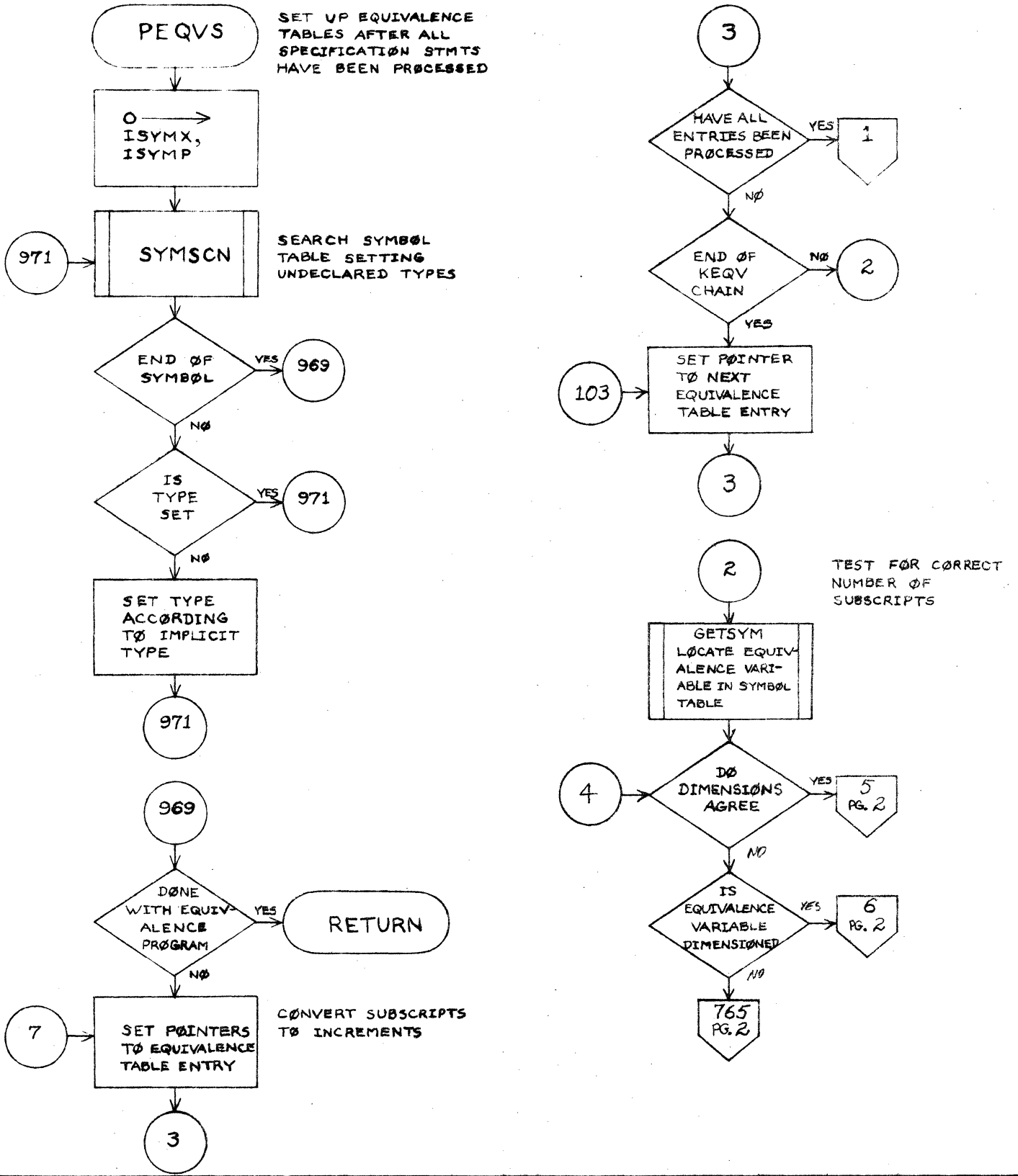
1	SYMTAB Pointer	
2		Calculated Value

IEQV (Sorted)

1	SYMTAB Pointer	
2		

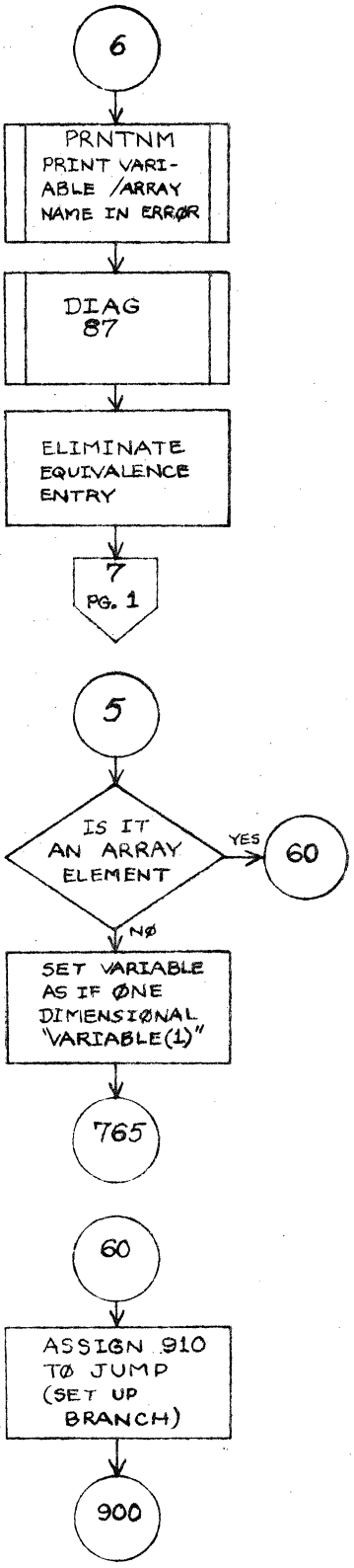
CONTROL DATA

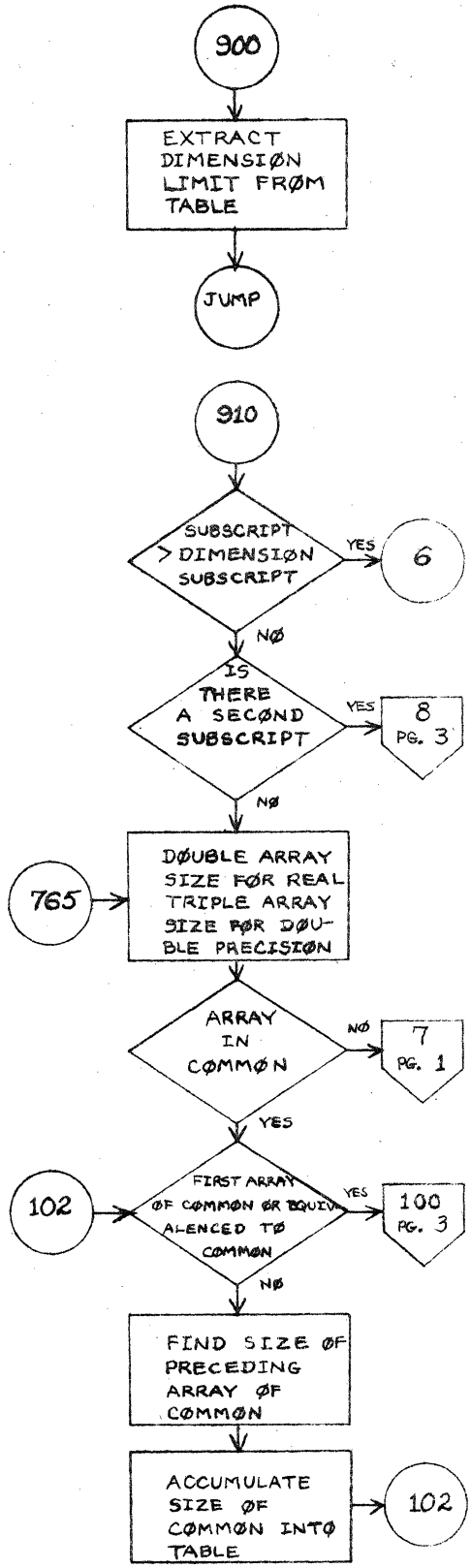
LA JOLLA FACILITY



TITLE	PEQVS	DRG. NO.
		REVISION
DRAWN BY	PROJ.	DATE	SHEET 1 OF

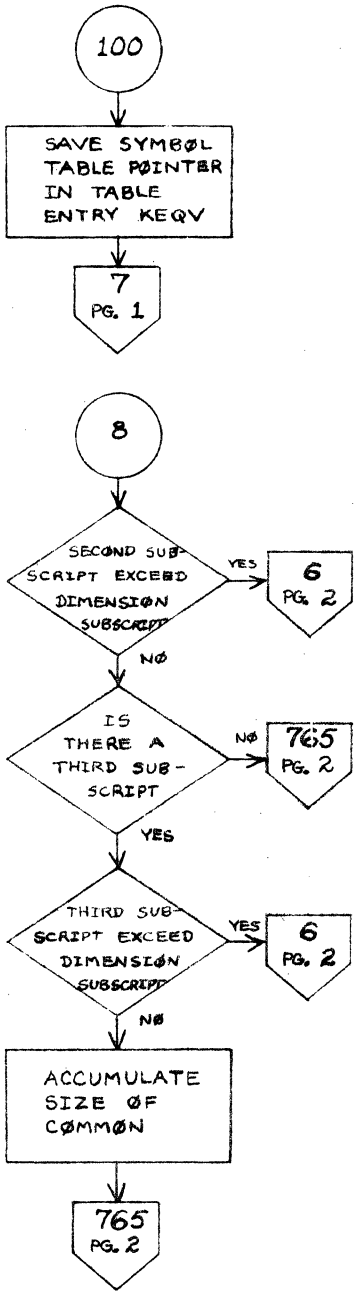
CSD 203





TITLE PEQVS			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET OF

CSD 203



TITLE PEQVS			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET OF

CSD 203

{THIS PAGE INTENTIONALLY LEFT BLANK}

18. Merge sets of equivalence chains into equivalence classes. Start by finding unused chain for new equivalence class.

19. Jump if end of entries to YES leg.

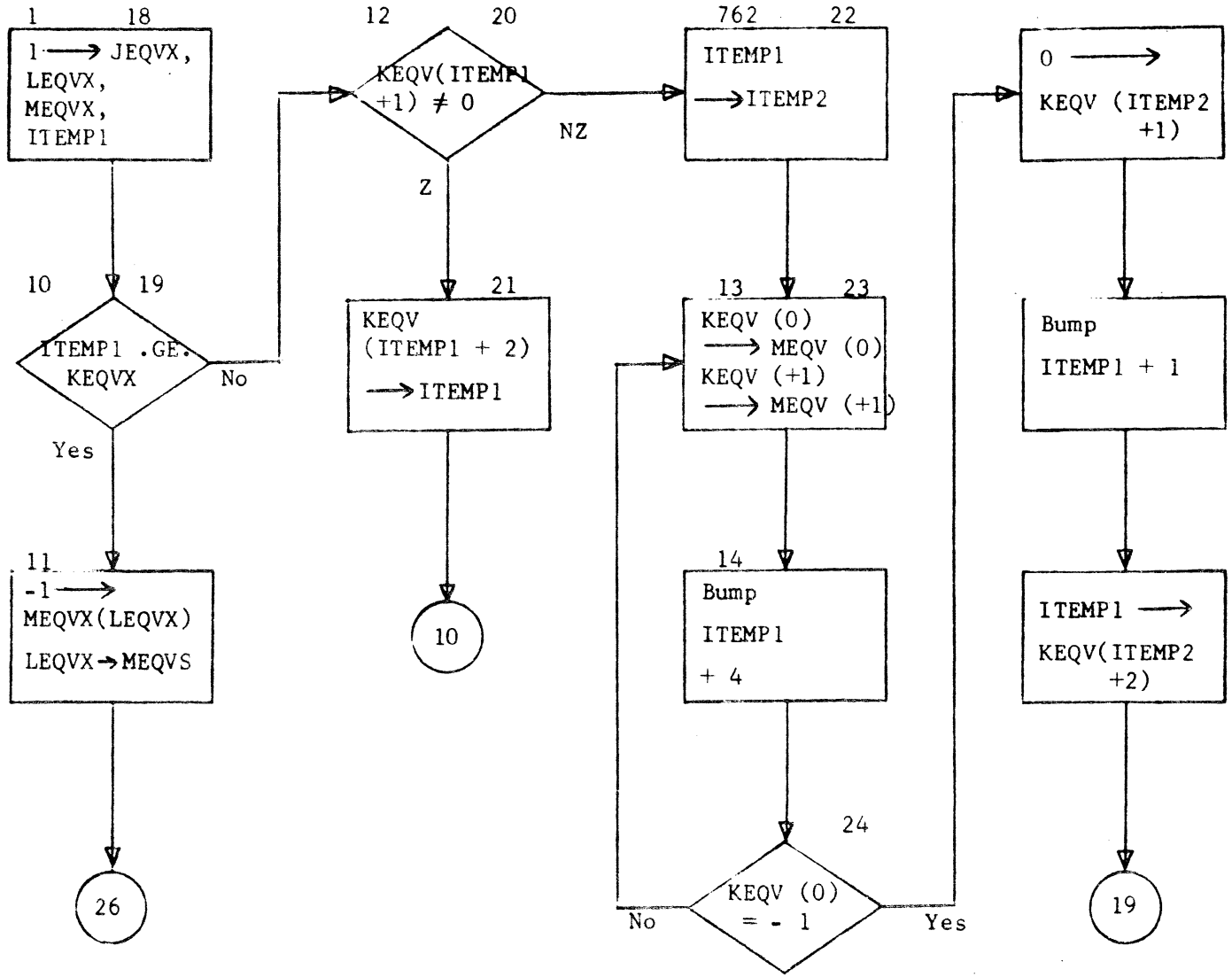
20. Jump if this chain has not previously been removed.

21. Move to next chain.

22. Move chain to MEQV, deleting illegal entries.

23. ITEMP1 is index to KEQV, LEQVX is index to MEQV.

24. Back to 13 if not end of chain.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PEQVS		PROJECT MGR.		
		PAGE 5 OF 16	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

25. Jump if there are MEQV class elements left.

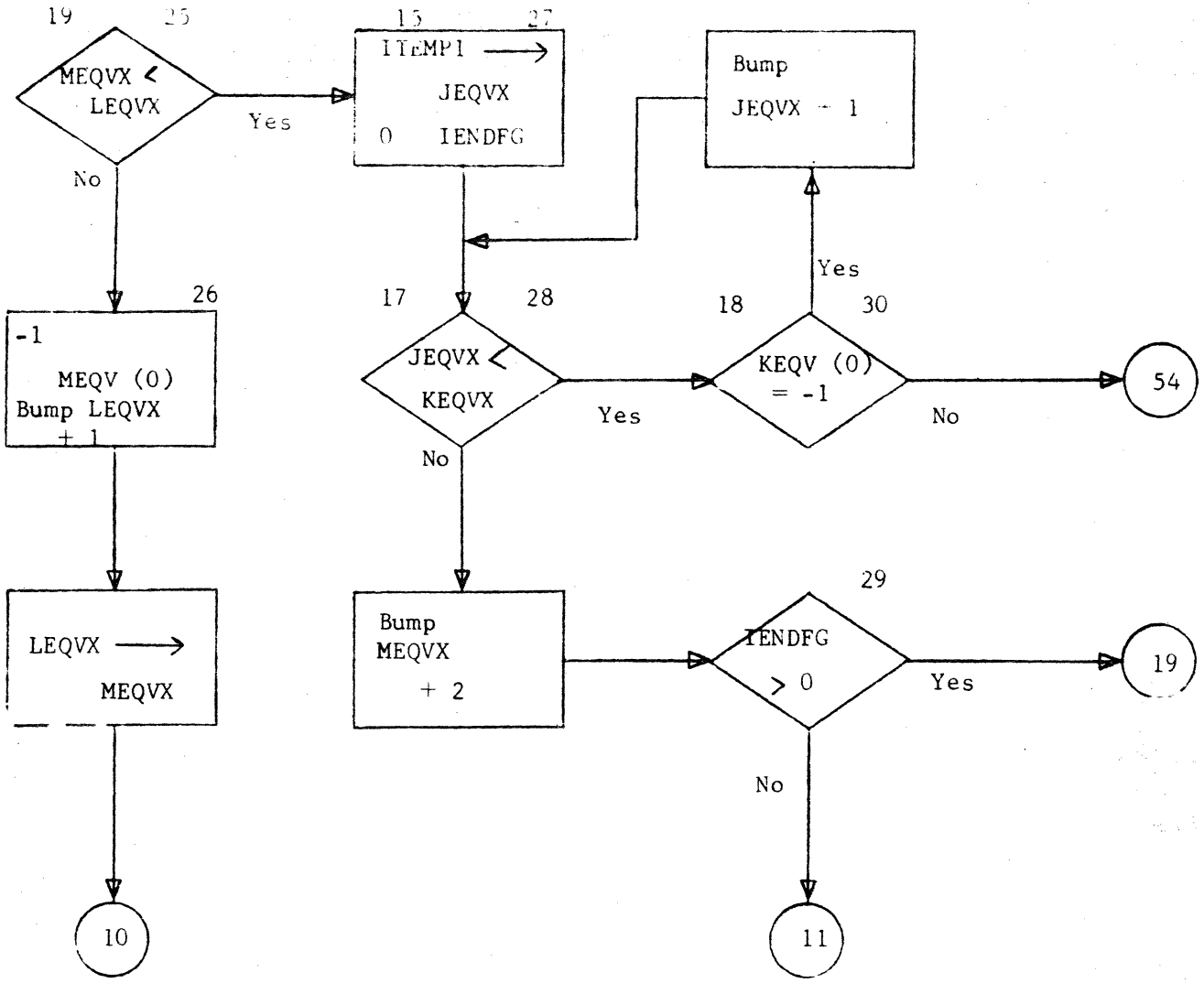
26. Set end of class mark.

27. Try to find equivalent chain, equivalence established by MEQV.

28. Go to 18 if not end of entries.

29. Jump to 19 if something left in KEQV.

30. Jump to 54 if not end of chain.



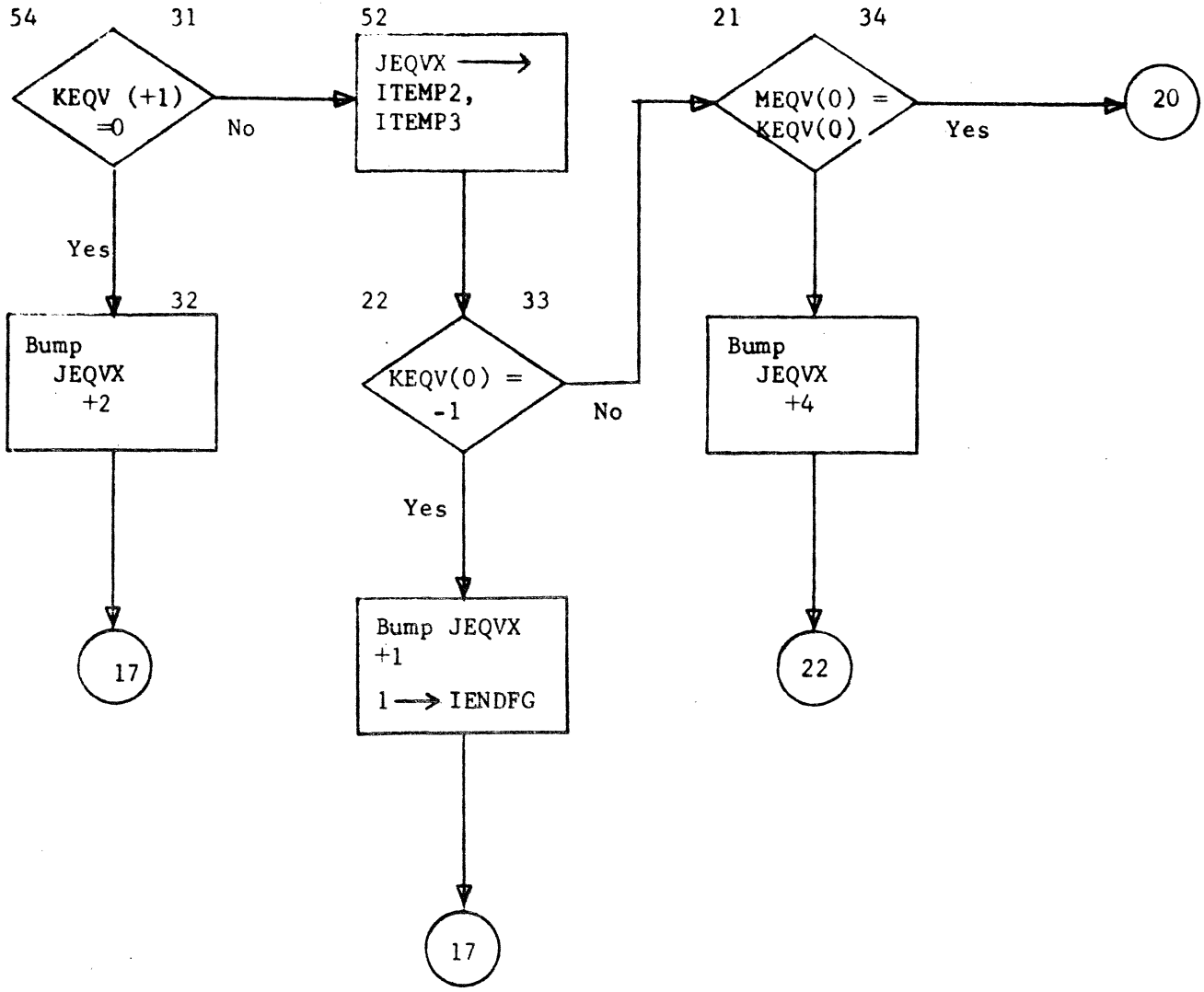
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MEQVS	PROJECT MGR.			
	PAGE 6 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

31. Jump to 52 if chain has not been assigned to class.

32. Jump to next chain.

33. Jump to 21 if not end of chain.

34. Is MEQV (MEQVX) an element in this chain. Jump to 20 if so.

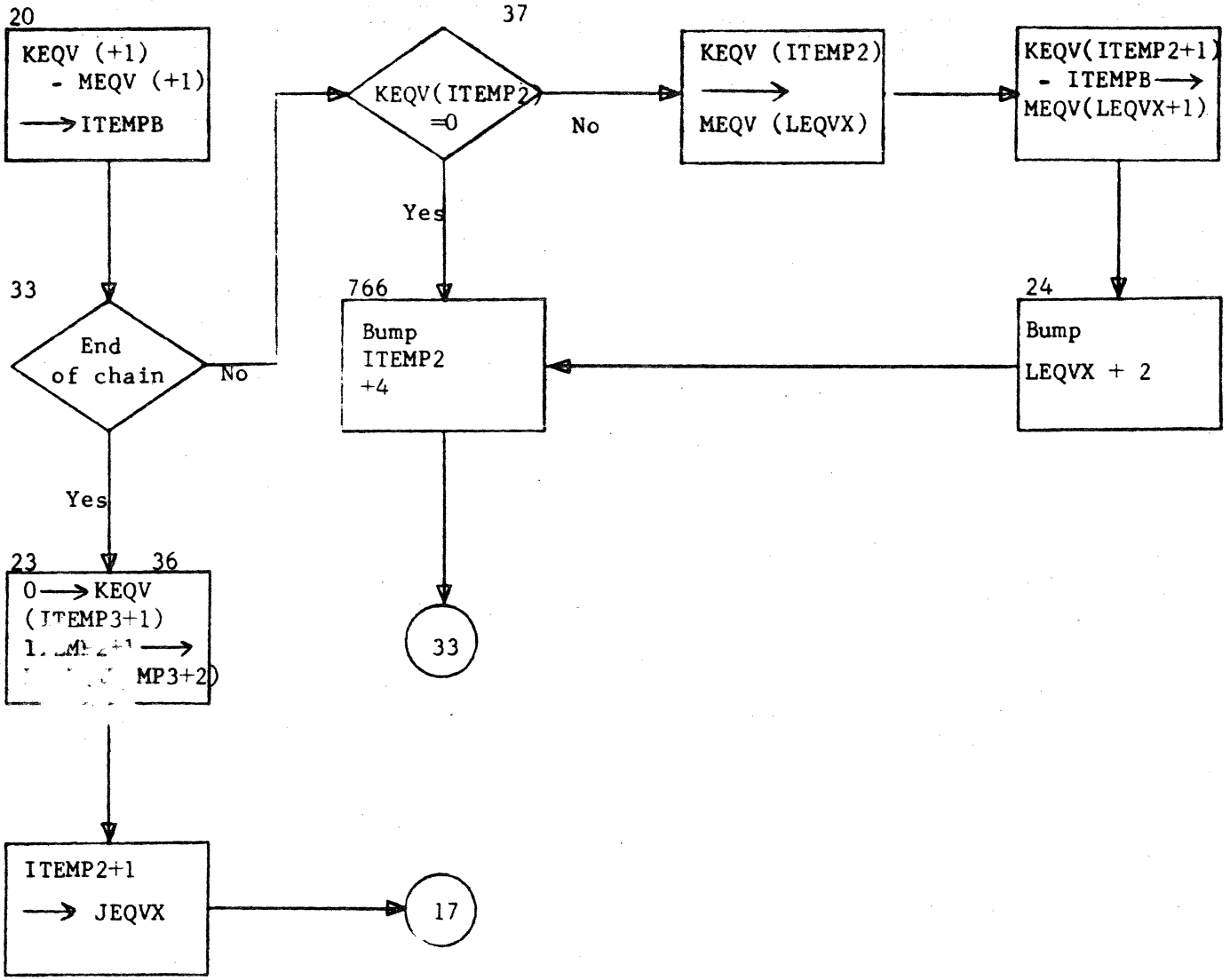


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED	DATE
	DOCUMENT TITLE PEQVS		PROJECT MGR.			
		PAGE 7 OF 16	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

35. Integrate chain into partial class. Must adjust common increment. At this point KEQV indexed by JEQVX, MEQV indexed by MEQVX.

36. Remove this chain from KEQV.

37. Jump to 766 if illegal entry.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACHINE TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PEQVS	PROJECT MGR.			
	PAGE 8 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

CA 127-1

2-80

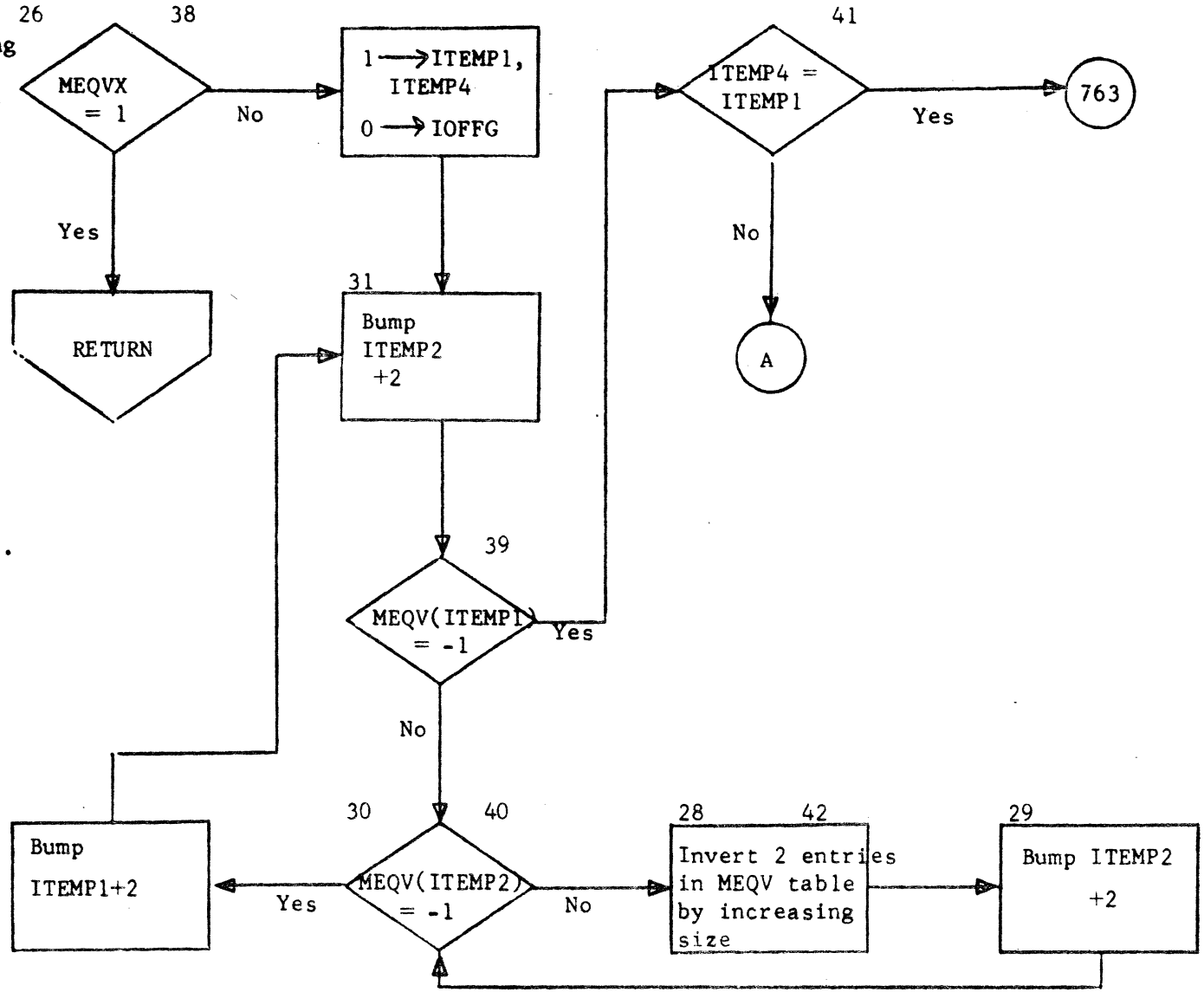
38. Set up IEQV by ordering increments correctly and deleting illegalities. Sort class by increasing increment size.

39. Not end chain for primary scan, go to 30.

40. Not end of chain for secondary scan, go to 30.

41. Go to 763 if class empty.

42. ITEMP1 and ITEMP2 are pointers to MEQV entries to be inverted.

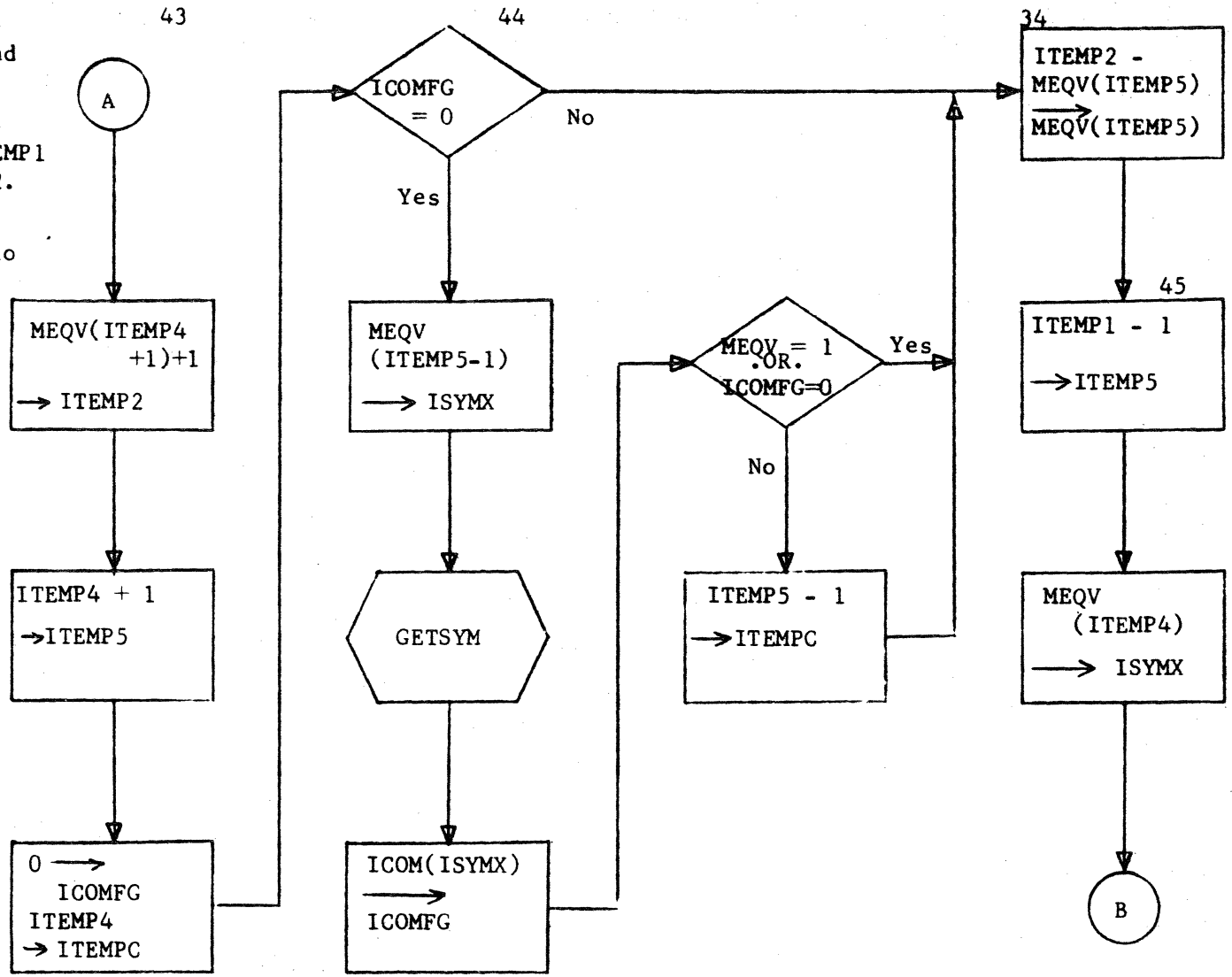


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MEQVS		PROJECT MGR.			
		PAGE 9 OF 16	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

43. Set up class as increment over base and find first common block.

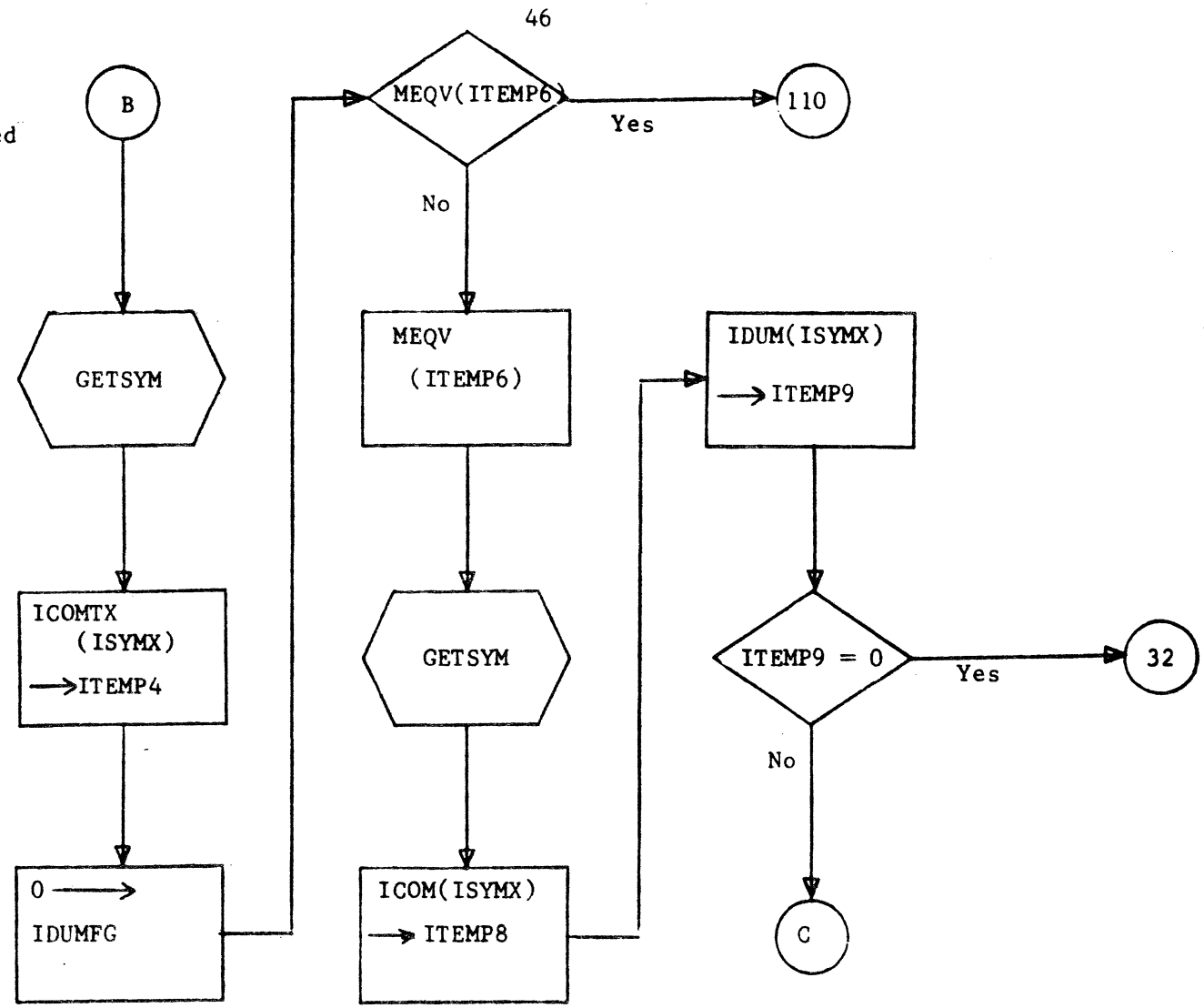
44. DO loop from to statement 34, ITEMP5 to ITEMP1 times incremented by 2.

45. Compare all pairs for legality and enter into IEQV.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PEQVS	PROJECT MGR.			
	PAGE 10 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 7/6/	TASK NAME			

46. DO loop to 110. ITEMP6 = ITEMPC, ITEMP5,2. Jump if pair has been deleted to DO loop counter.

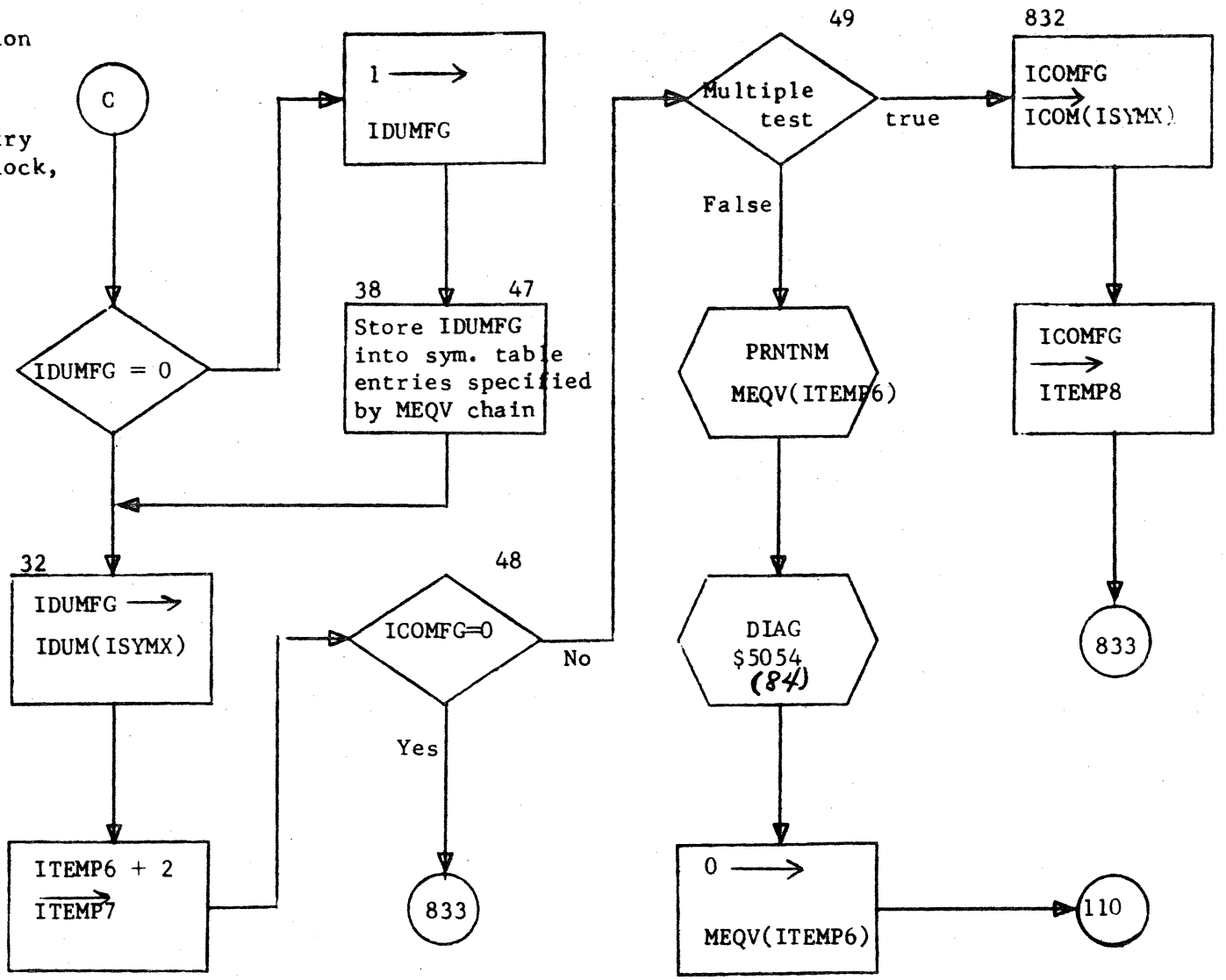


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE AEQIS		PROJECT MGR		
		PAGE 11 OF 16	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

47. DO loop with induction variables ITEMPD = ITEMPC, ITEMP6,2

48. Set symbol table entry to correct common block, if any.

49. Multiple test is:
ITEMP8 = 0
.OR.
ITEMP8 = ICOMFG
.AND.
ITEMP9 = 0



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PEQVS		PROJECT MGR.		
		PAGE 12 OF 16	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

50. DO loop to 35 with induction variables:

    ITEMP7=ITEMP7,. ITEMP5, 2.

    Jump if item been deleted to 35.

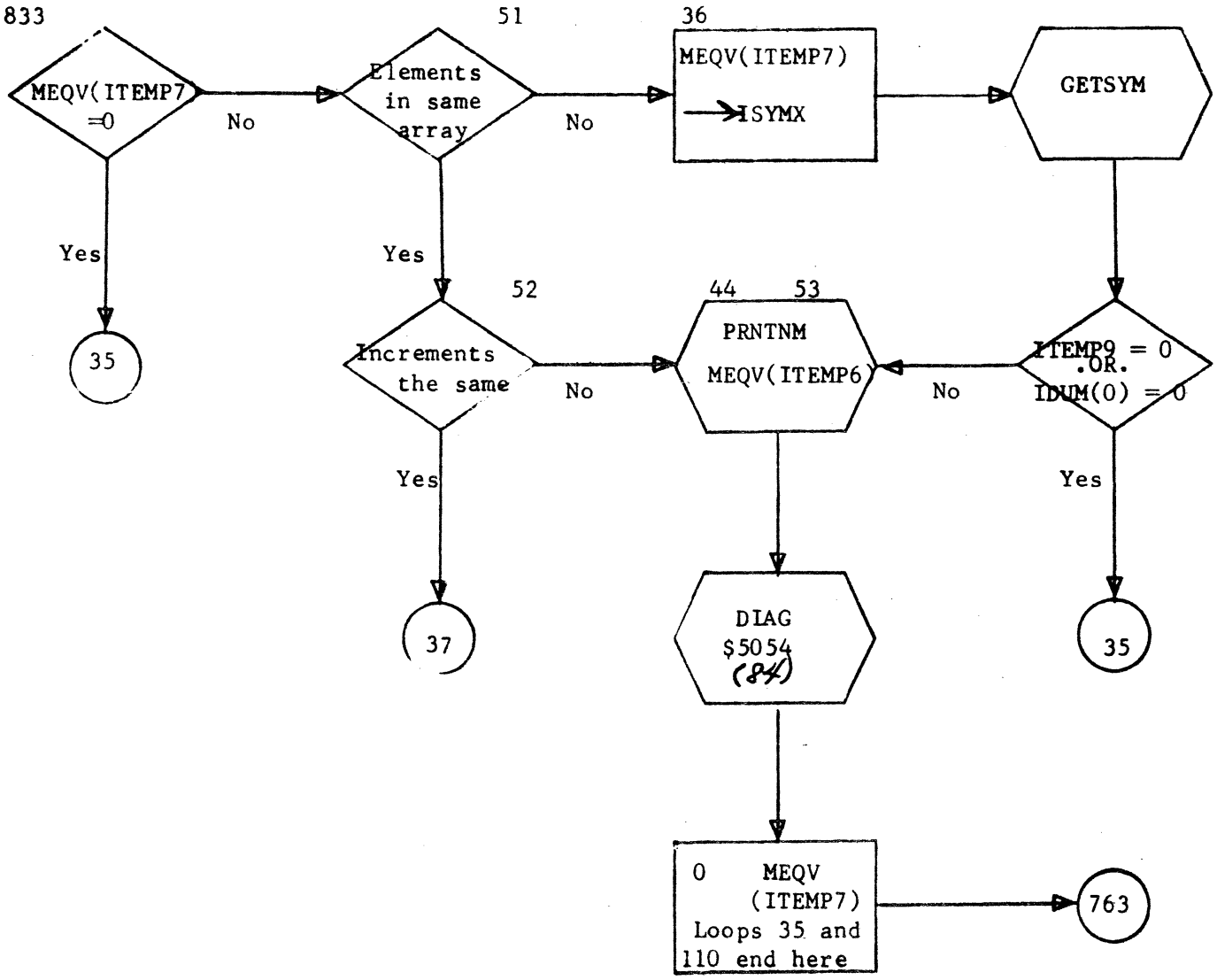
51. Test is:

    MEQV(ITEMP6) ≠ MEQV(ITEMP7) true means no.

52. Test is:

    MEQV(ITEMP6+1) = MEQV(ITEMP7+1) true means yes.

53. Print out names for: MEQV(ITEMP6) and MEQV(ITEMP7).

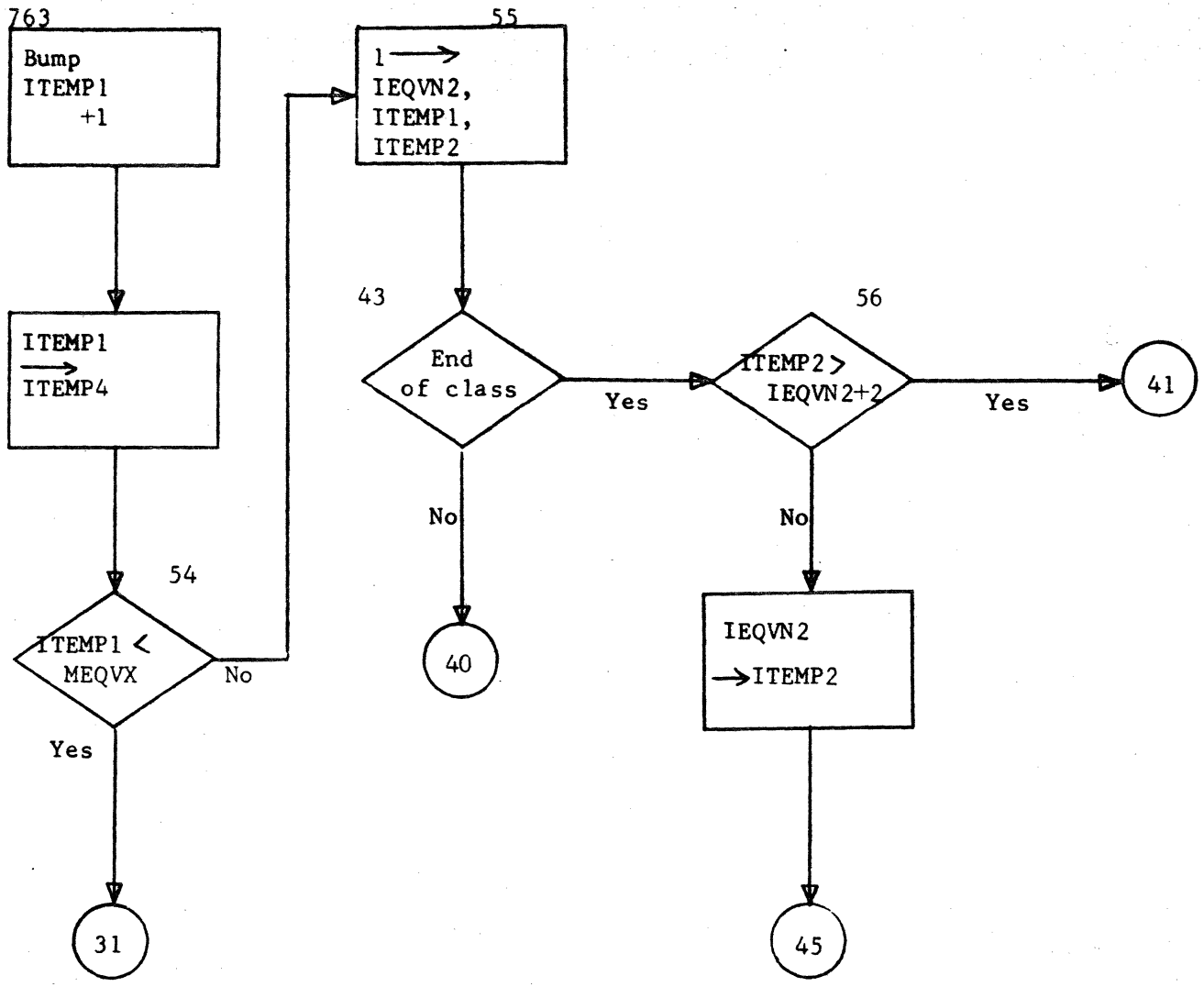


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PEQ16		PROJECT MGR.		
		PAGE 8 OF 16	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3 67	TASK NAME		

54. Jump to 31 to process more classes.

55. Prepare to set up IEQV tables.

56. Jump to 41 if class contains two or more names.

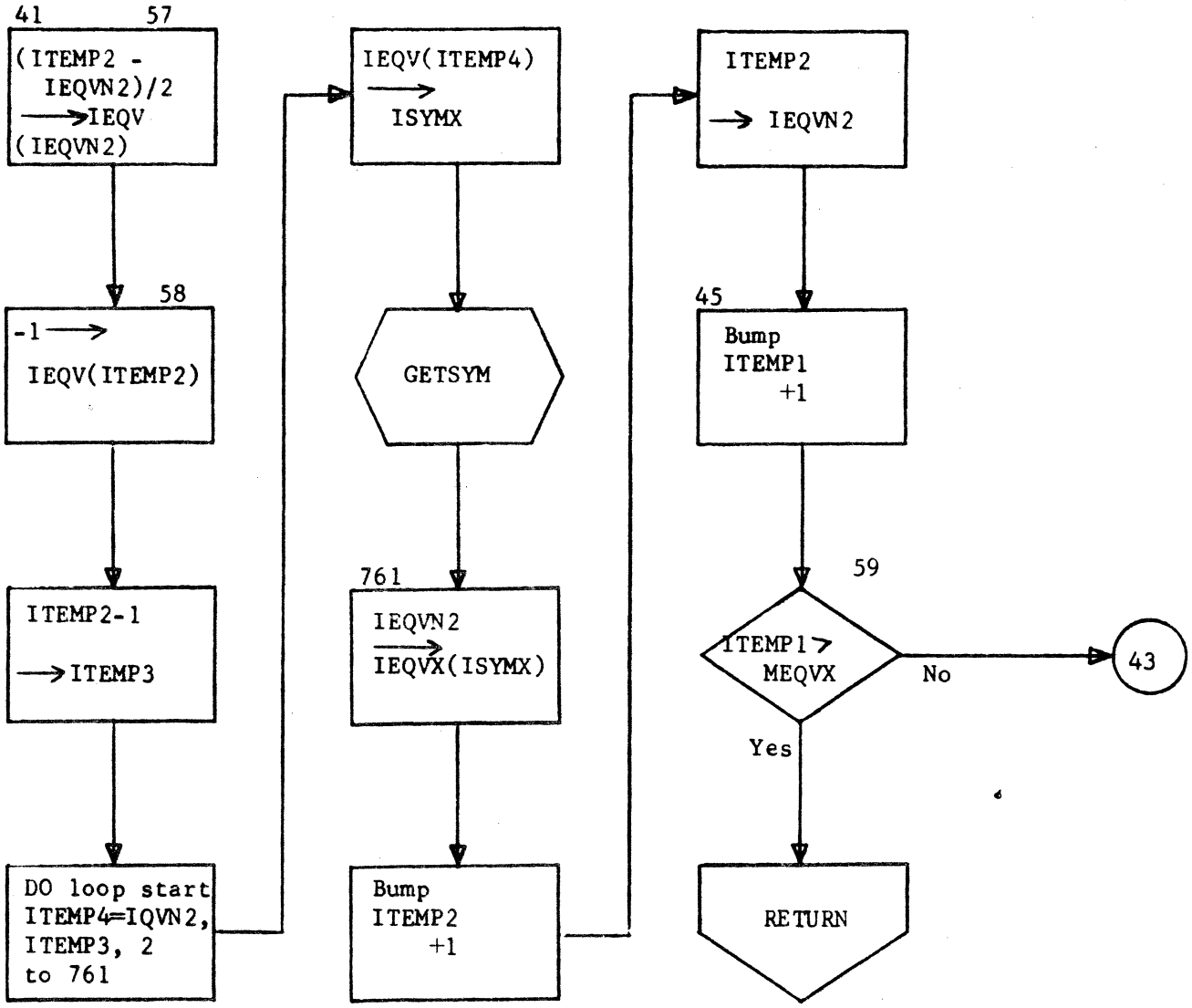


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PEQVS	PROJECT MGR.			
	PAGE 14 OF 67	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 2-16-71	TASK NAME			

57. Set up number of elements in this class.

58. Set end of class flag.

59. Return if all equivalence entries have been processed.

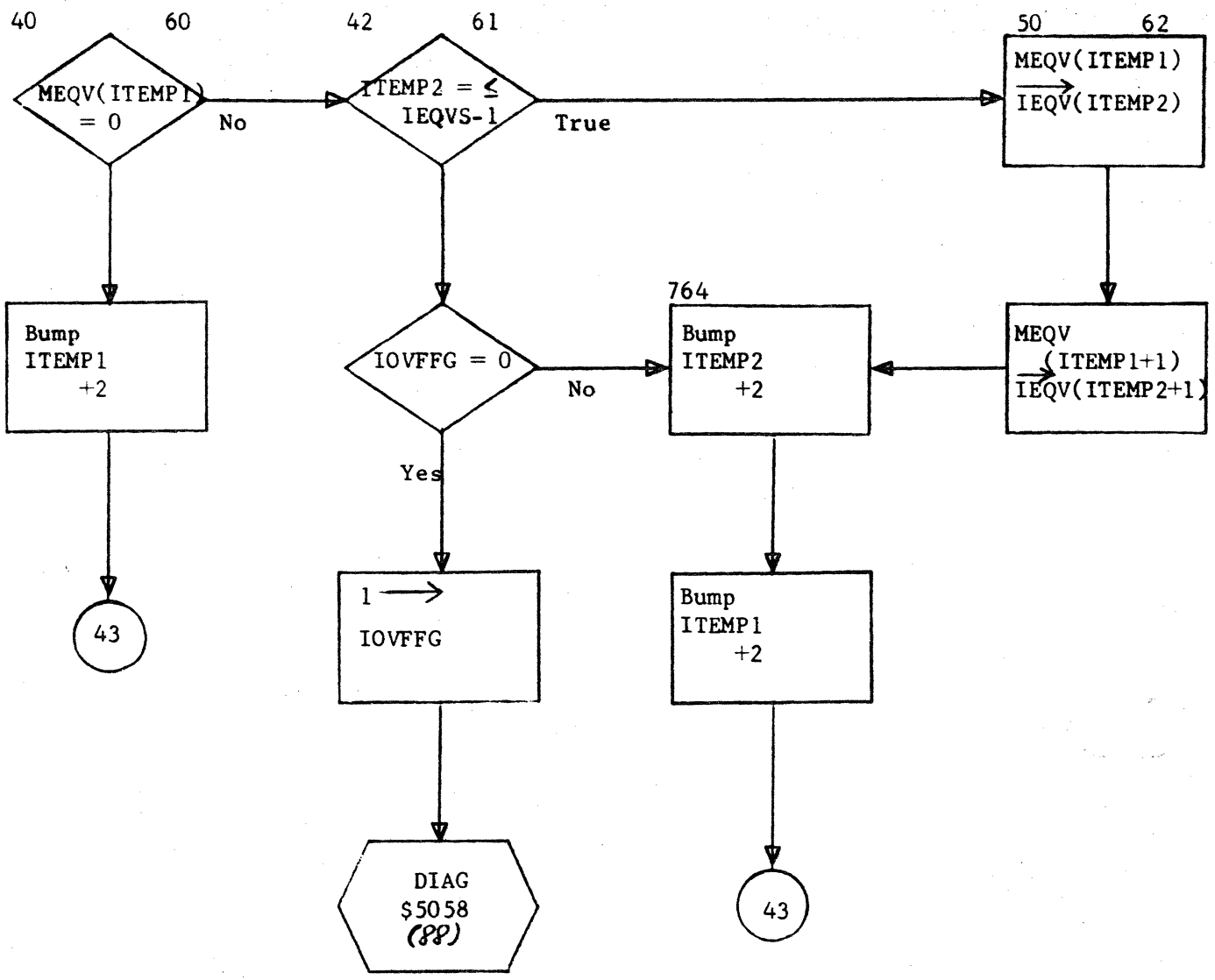


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE AEQVS	PROJECT MGR.			
	PAGE 15 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-1	TASK NAME			

60. Jump if entry is not to be deleted.

61. Check for table overflow. Jump if none.

62. Store name into equivalence table.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PEQV5	PROJECT MGR.			
	PAGE 16 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

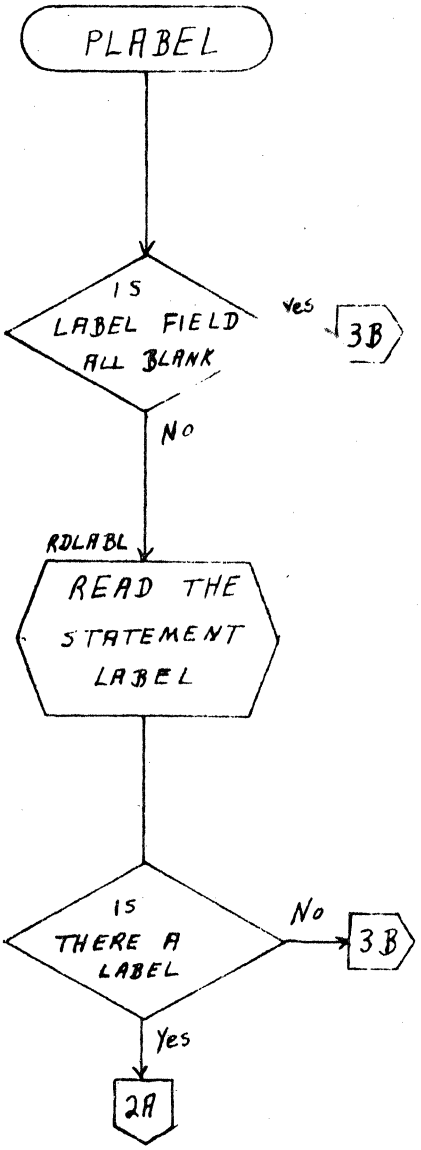
2.2.2.4 Subroutine PLABEL

PLABEL is called by PHASEA to process the label field of an input statement. If the label field is blank, the routine returns immediately with ISORSX = 7.

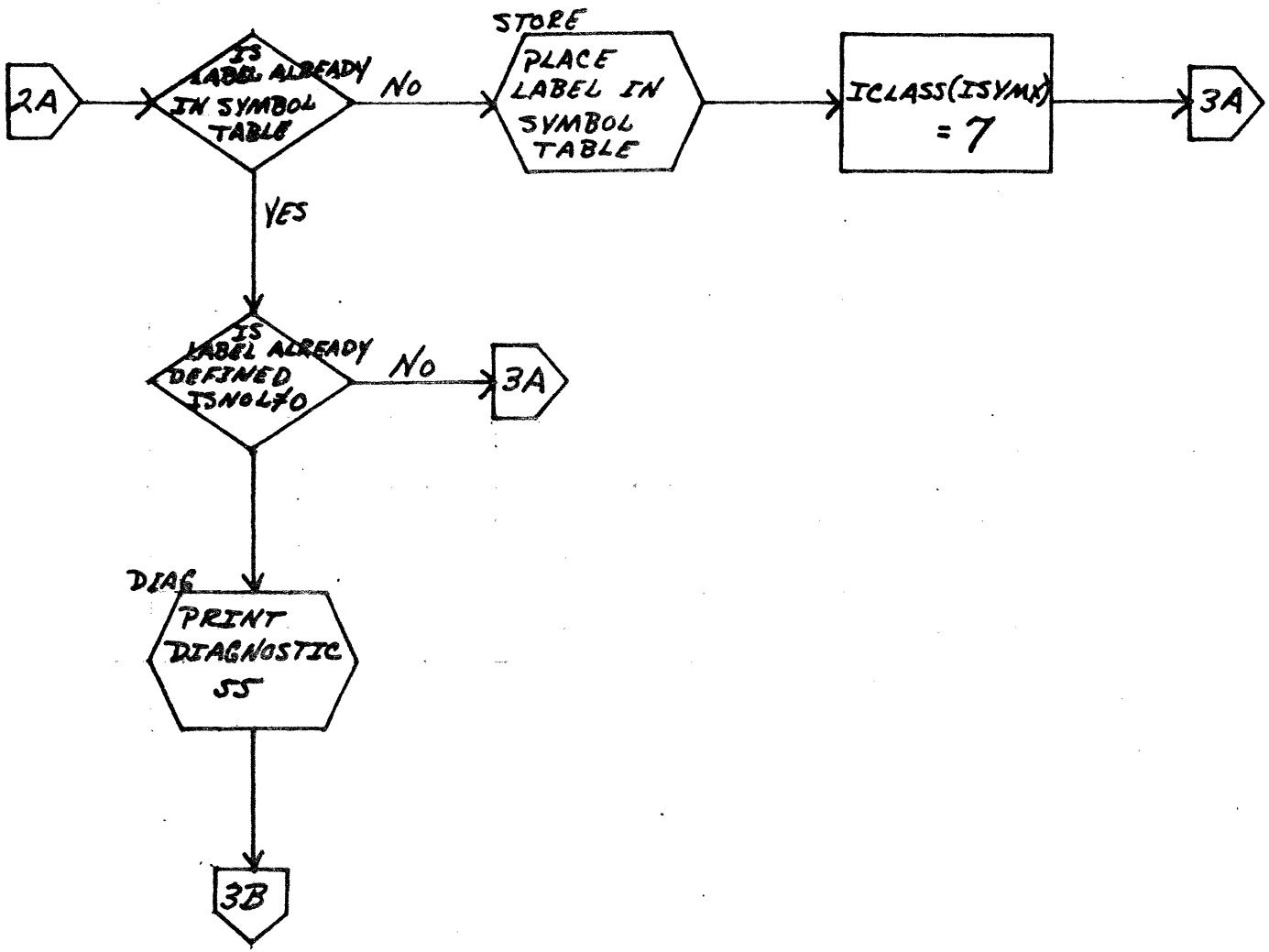
Otherwise, it uses the routine RDLABL to read the field. If the field contains a label, a check is made to see if this label is currently in the symbol table (SYMTAB). If it is not, it is entered there by a call to STORE. If it is, a check is made to see if the label has been previously defined. If it has, an error is output and the routine exits with ILLABL = 0 and ISORSX = 7.

Otherwise, ISNOL of the SYMTAB entry of this label is set to the current statement number, thus setting the label as defined.

The routine then exits with ILLABL equal to the SYMTAB index of the label entry, ISORSX = 7 (the next column to process), IBUF2(5) = the SYMTAB index for this label, and IBUF2(2) set to indicate a labelled statement (negative statement number).

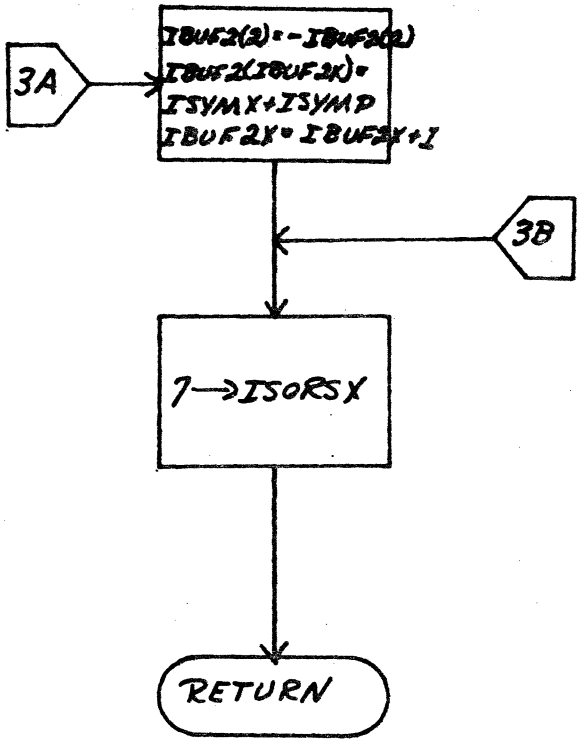

PLABEL
IS LABEL FIELD ALL BLANK
Yes
3B
No
RDLABL
READ THE STATEMENT LABEL
IS THERE A LABEL
No
3B
Yes
2A

PLABEL



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PLABEL	PROJECT MGR.			
	PAGE 2 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

PLABEL

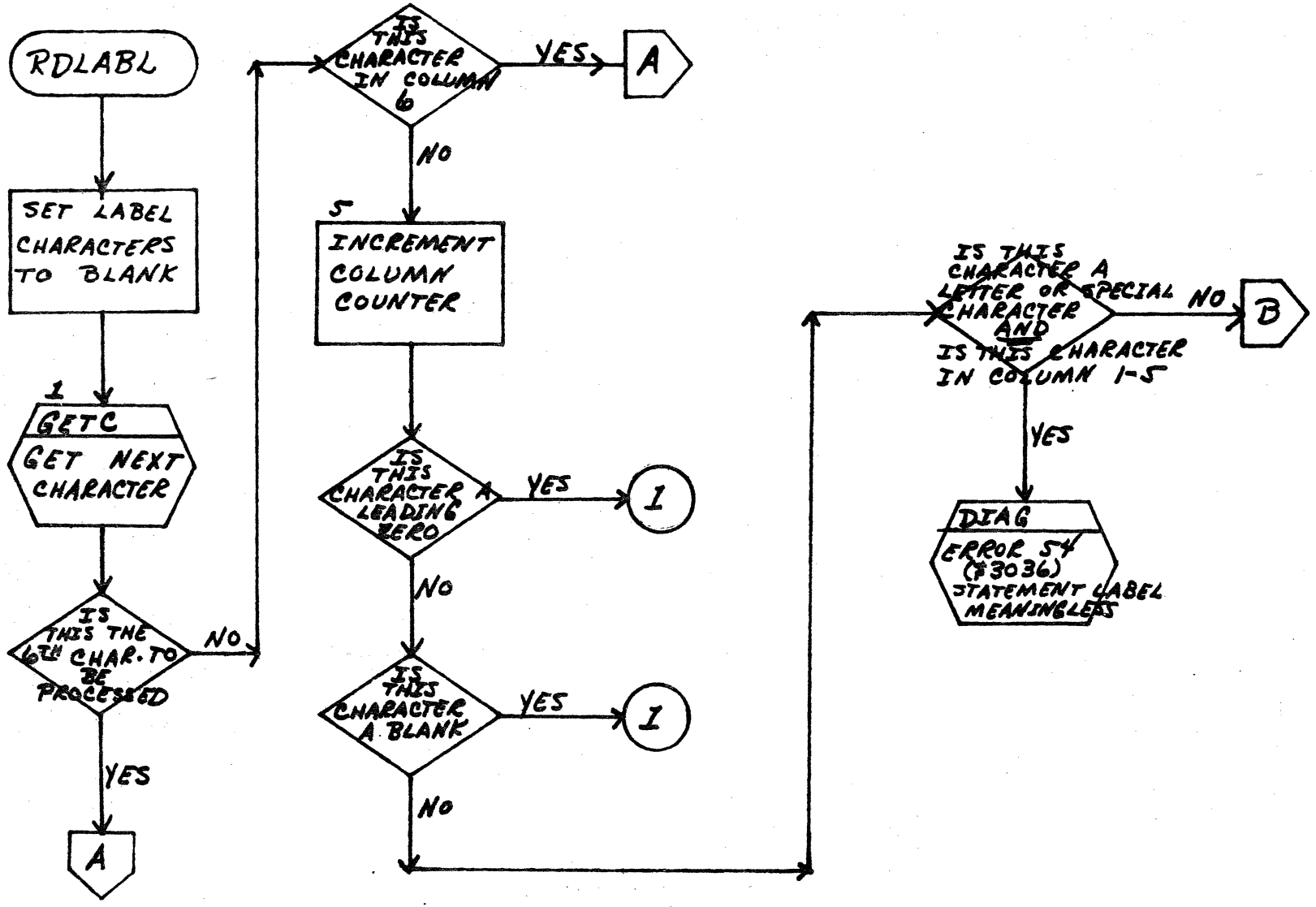


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PLABEL	PROJECT MGR.			
	PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

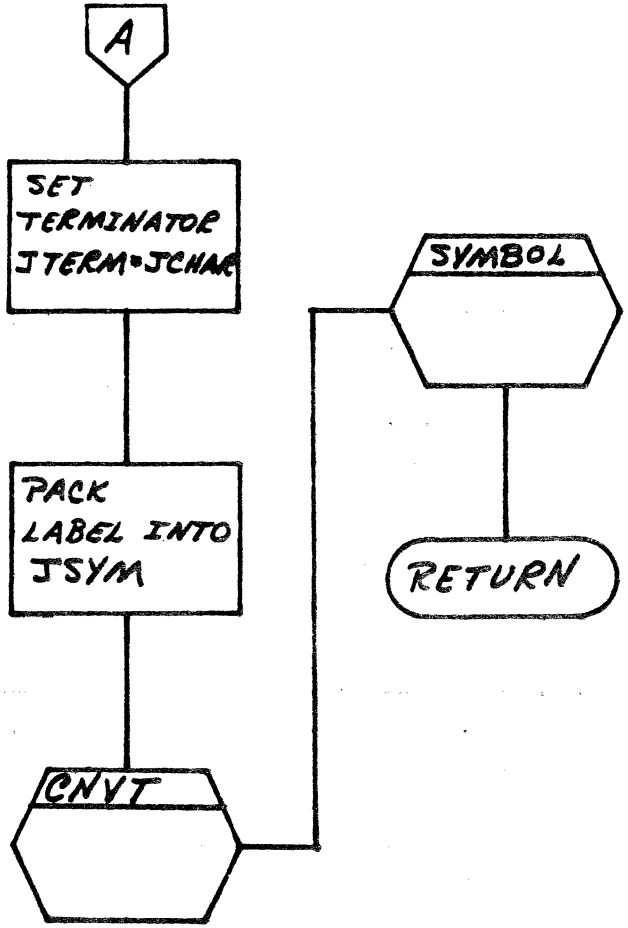
CA 127-1

2.2.2.4.1 Subroutine RDLABL

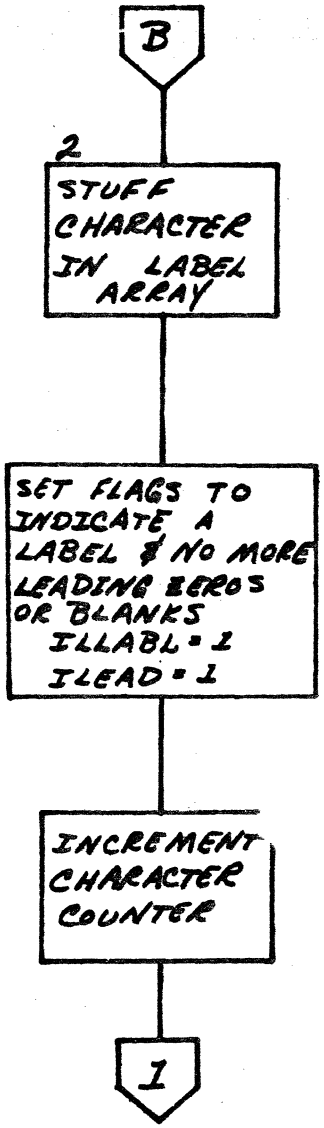
RDLABL is called to read a label. It reads the label and checks to see if it is a number. If it is not, it prints out diagnostic 54. If the label is legal, RDLABL packs the label into JSYM and returns.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE RDLABL		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE RDLABL		PROJECT MGR.			
		PAGE 2 OF 3	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE RDLABL	PROJECT MGR.			
	PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

2.2.2.5 Option Selection - Subroutine OPTIONS

OPTIONS is the routine, written in 1700 assembly language, which processes option selection in 1700 Mass Storage Fortran. It is entered from G0A. Upon entrance, OPTIONS clears the option selection holders in the DATA block (IR, IK, IP, IA, IL, IM, IXLG0), backgrounds its input buffer and solicits input from the standard input device. If there was no ° OPT° card, the standard options LPX are assumed and the option selection holders in the DATA block IP, IL, and IXLG0 are set ≠0. If no options are listed on the ° OPT° card, OPTIONS solicits input from the input comment medium by outputting OPTIONS and waiting for a carriage return.

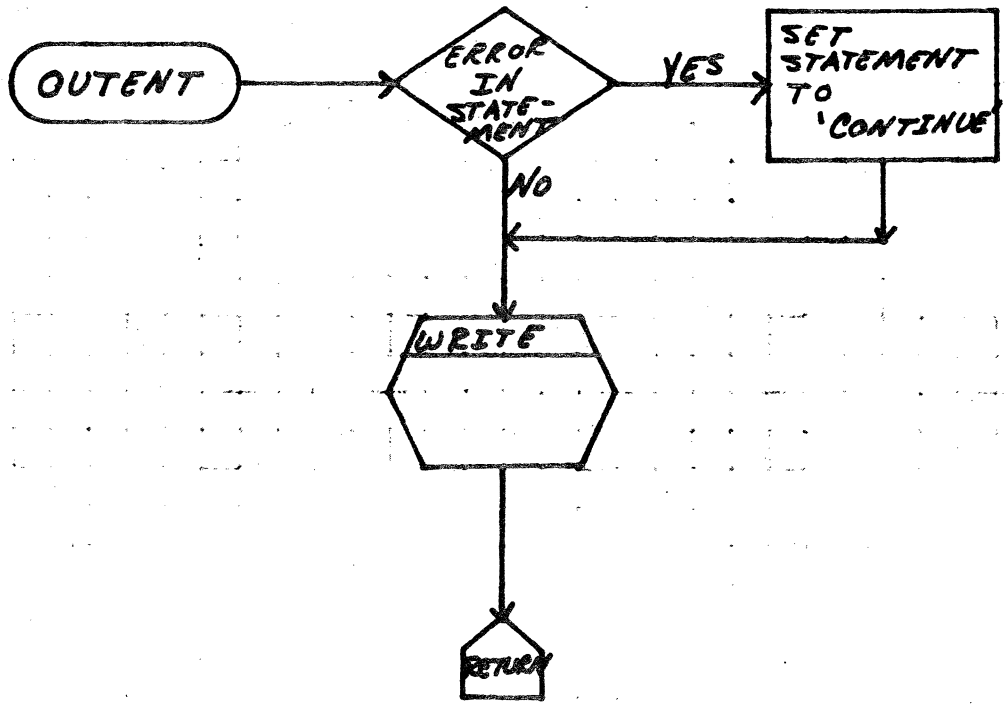
Upon receipt of a carriage return or finding options on the ° OPT° card, OPTIONS interrogates each character input. If it is recognizable (R,K,P,A,L,M,X) the corresponding holder is marked selected (≠0). Unrecognizable characters are ignored.

OPTIONS then returns to G0A.

2.2.2.6 Subroutine OUTENT

OUTENT is the subroutine used to output a record to the scratch output device. If an error has occurred in the statement, it is changed to a CONTINUE and the error-in-statement switch (IERR) is cleared.

OUTENT calls subroutine IOPR (entry point WRITE) to perform the output.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE OUTENT	PROJECT MGR.			
PAGE 1 OF 1	PROJECT NAME			
NUMBER ISSUE DATE	TASK NO.			
DRAWN BY DATE	TASK NAME			

2.2.3.2 Subroutine ENDDO - The DO Loop Termination Routine

ENDDO is called by PHASEA upon completion of processing of a labeled statement and by IOSPR to terminate DO loops and complete the LOOPT table.

If there are no DO loops open, ENDDO exits. If the label of the current statement terminates the innermost open DO, an ENDDO entry is output and the LOOPT for this DO loop is completed. This DO entry is then erased from the LOOPT table (DO nesting) and the check of label vs. innermost DO is repeated.

When ENDDO encounters an open innermost DO which does not terminate at this statement label, the routine searches the rest of the LOOPT table to ascertain that no outer DO loops terminate here. If any do, an error is output and the erroneous entry is erased from LOOPT.

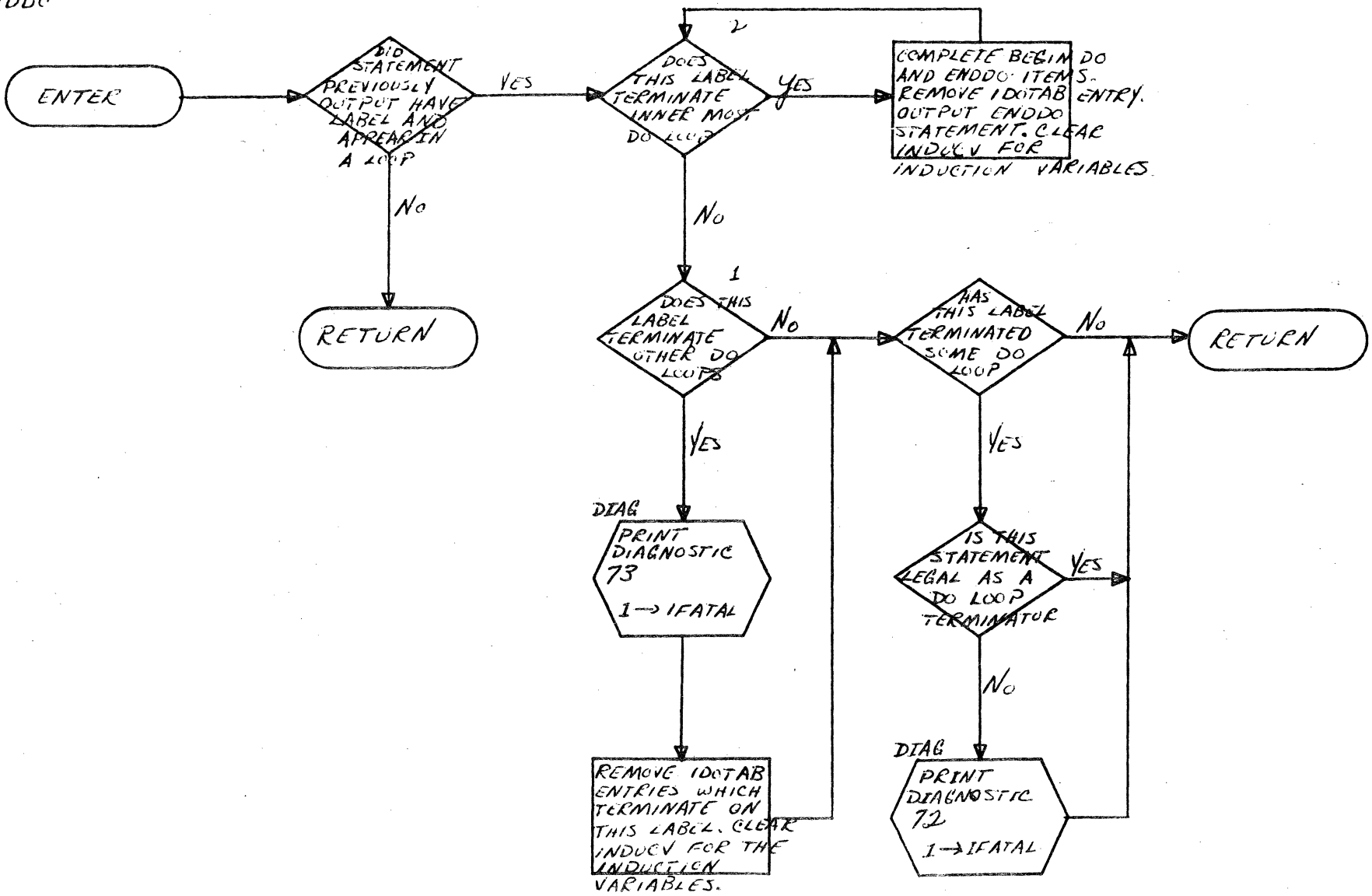
If any DO loops have terminated at the current statement label, ENDDO checks the statement associated with this label. If the statement is of a type illegal for DO termination an error is output and the statement changed to a CONTINUE.

2.2.3.3 Subroutine SAVEID

SAVEID was written to allow the use of up to 47 characters of comments (copyright statement) on the source routine identification card which will be output in a format similar to Macro Assembler produced binary NAM blocks.

The format of the binary NAM card includes up to 47 characters of comments preceded by a slash from the source routine identification card (PROGRAM, SUBROUTINE, FUNCTION, or BLOCK DATA) on the binary NAM card. If no slash and no comments appear on the source card, the binary NAM card will be punched entirely with blanks. If there is no slash, but comments do appear on the card, diagnostic #66 will be issued, and the binary NAM card will contain blanks. A continuation card may also be used.

ENDDO



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ENDDO		PROJECT MGR.		
FLOWCHART ☒		PAGE / OF /	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐					

2.3 General Pass 1 Routines

2.2.1 Symbol Table Manipulating Routines

2.3.1.1 SYMBOL

To locate a name in the symbol table, store the name in the three cell array, JSYM. If the name is not a constant, pack it into JSYM(1) and JSYM(2) by calling CNVT. The name may be a statement label. If the name is an integer constant, the constant's value should be in JSYM(1) and the constant 2424_{16} should be in JSYM(3). If the name is a real constant, the constant's value should be in JSYM(1) and JSYM(2) and the constant 2525_{16} should be in JSYM(3). If the name is a double precision constant, the constant's value should be in JSYM(1), JSYM(2), and JSYM(3). The common variable JMODE should equal 6 for double precision constant. Issue the statement

```
CALL SYMBOL
```

Upon exit, if the common variable ISYMD is zero, the symbol is not in the table. If ISYMD is non-zero, the COMMON variable ISYMX indexes the symbol entry, that is, each item in that symbol's entry may be obtained as

```
item name (ISYMX)
```

If ISYMD is zero, ISYMX points to the next available symbol table entry. There are two versions of SYMBOL, as follows:

Phase A: If SYMTAB paging is necessary, only the current page is written on disk before being overlaid; later pages are simply overlaid. SYMBOL accomplishes this by setting ISYMX negative before calling GETSYM after the current page has been written.

Phase B: If it is known that the name is not in the symbol table, SYMBOL may be called with ISYMX negative. In this case SYMBOL does not search the symbol table but returns with ISYMX pointing to the next available entry.

2.3.1.2 Subroutine STORE

Assume that JSYM(1), JSYM(2), and JSYM(3) are filled, probably by GETF, but possible by the programmer himself. Also assume that ISYMX is pointing to the next available

entry in the symbol table. SYMBOL will set ISYMX in this manner if an entry is not in the table. Then the symbol in JSYM will be inserted into the symbol table. ISYMN (current size of symbol table) will be incremented. If JSYM(3) is 2424_{16} or 2525_{16}, or if JMODE is 6 for double precision constants, ICLASS and ITYPE will be set. Issue the statement

CALL STORE

Thus, the normal sequence of events leading to an insertion in the symbol table is

CALL GETF

which then does

CALL SYMBOL

to see if the field is in the symbol table. Upon return from GETF, the calling program does

CALL STORE

to insert the field in the symbol table if it is not in it already.

2.3.1.3 Subroutine GETSYM

GETSYM insures that the symbol table page requested by ISYMX is in core, updates the current symbol table page holder (ISYMPC) if necessary, and adjusts ISYMX to the in-core subscript of the desired symbol table entry (ISYMX = ISYMX - ISYMP).

There are five versions of GETSYM, distinguished as follows:

Phases A and B:

If ISYMX is positive on entry and paging is necessary, the current symbol table page is written on disk before the desired page is read. If ISYMX is negative, it is set positive and the current symbol table page is not saved.

The symbol table page size is 960 words. The current ISTABX page (96 words) is read and written along with the SYMTAB page, therefore eleven sectors are read and written for each page.

Phase C and Phase D/E 2.0B version.

The symbol table page size is 2400 words. When reading and writing the large page every eleventh sector (formerly an ISTABX page) is skipped. The current symbol

table page is always saved on disk before a new page is read.

Phase D local 1, 2.0A version:

This version reads and writes the compressed symbol table with a page size of 1000 words (see Assembler Output Phase, section 5.2). The current symbol table page is always saved.

Phase D local 2, 2.0A version:

This version reads the normal symbol table with a page size of 480 words. Every eleventh disk sector is skipped. The current symbol table page is never saved.

Phase E, 2.0A version:

The symbol table page size is 480 words. Every eleventh disk sector is skipped. The current symbol table page is always saved.

2.3.1.4 Subroutine CNVT

CNVT packs a six-character variable name into two words for storage in the symbol table. Only alphanumeric characters and blanks can be packed. CNVT packs the leftmost three characters into the first word of JSYM and the rightmost three characters into the second word as follows:

Let x, y, z, a, b, and c be the internal Fortran codes for X, Y, Z, A, B, and C. Then the name XYZABC is stored as:

$$JSYM(1) = 1521x + 39y + z$$
$$JSYM(2) = 1521a + 39b + c$$

Blanks, which are 46 in internal Fortran code, are charged to 38 before packing.

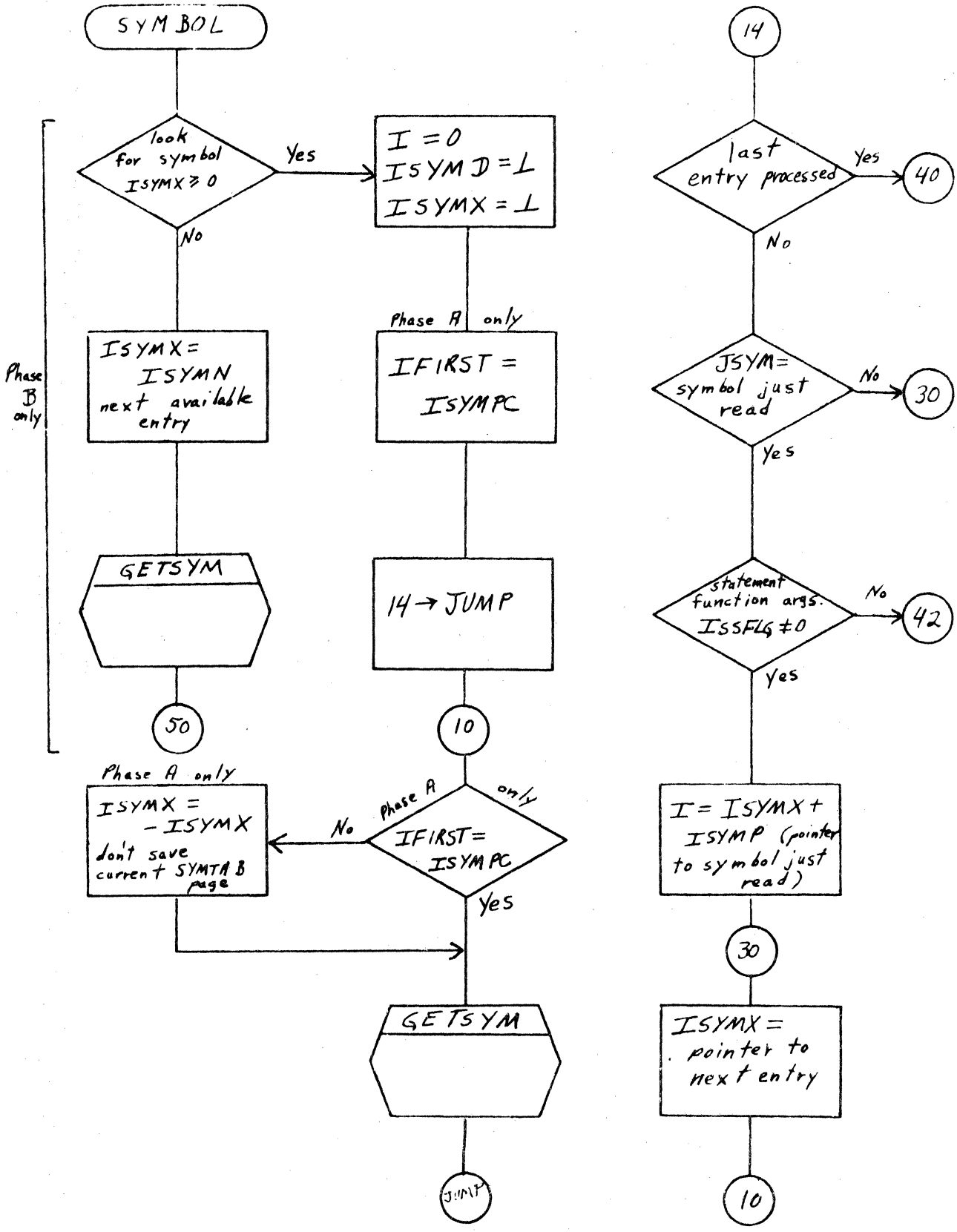
2.3.1.5 Subroutine SYMSCN

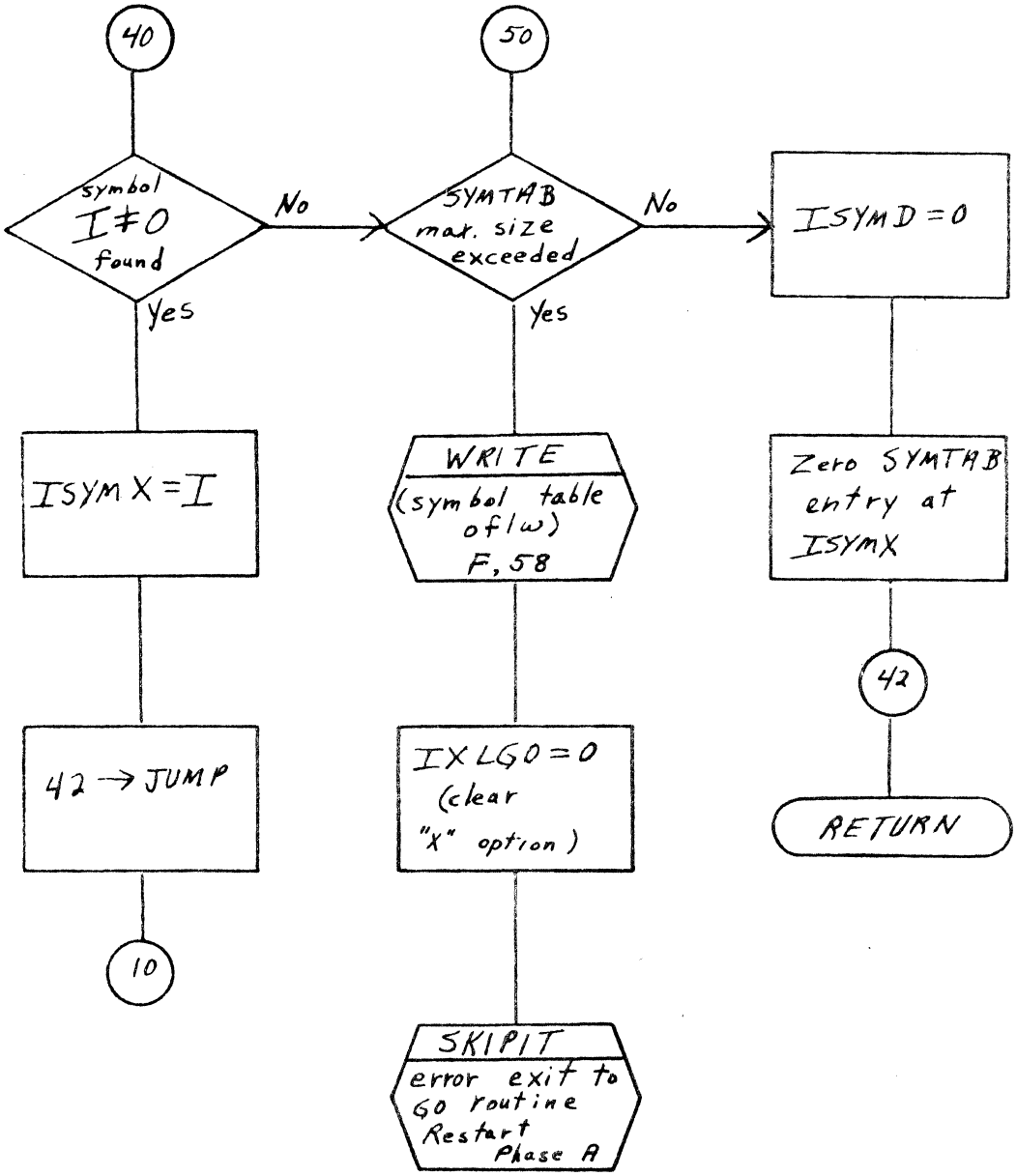
SYMSCN gets the entry <u>following</u> the current entry from the symbol table. Upon entrance, ISYMX = 0 requests the first symbol table entry, and ISYMX = (an in-core symbol table subscript) requests the entry (ISYMX + ISYMP + ISYMFL). Upon return, ISYMP = 1 says there are no more entries. GETSYM is called just before return.

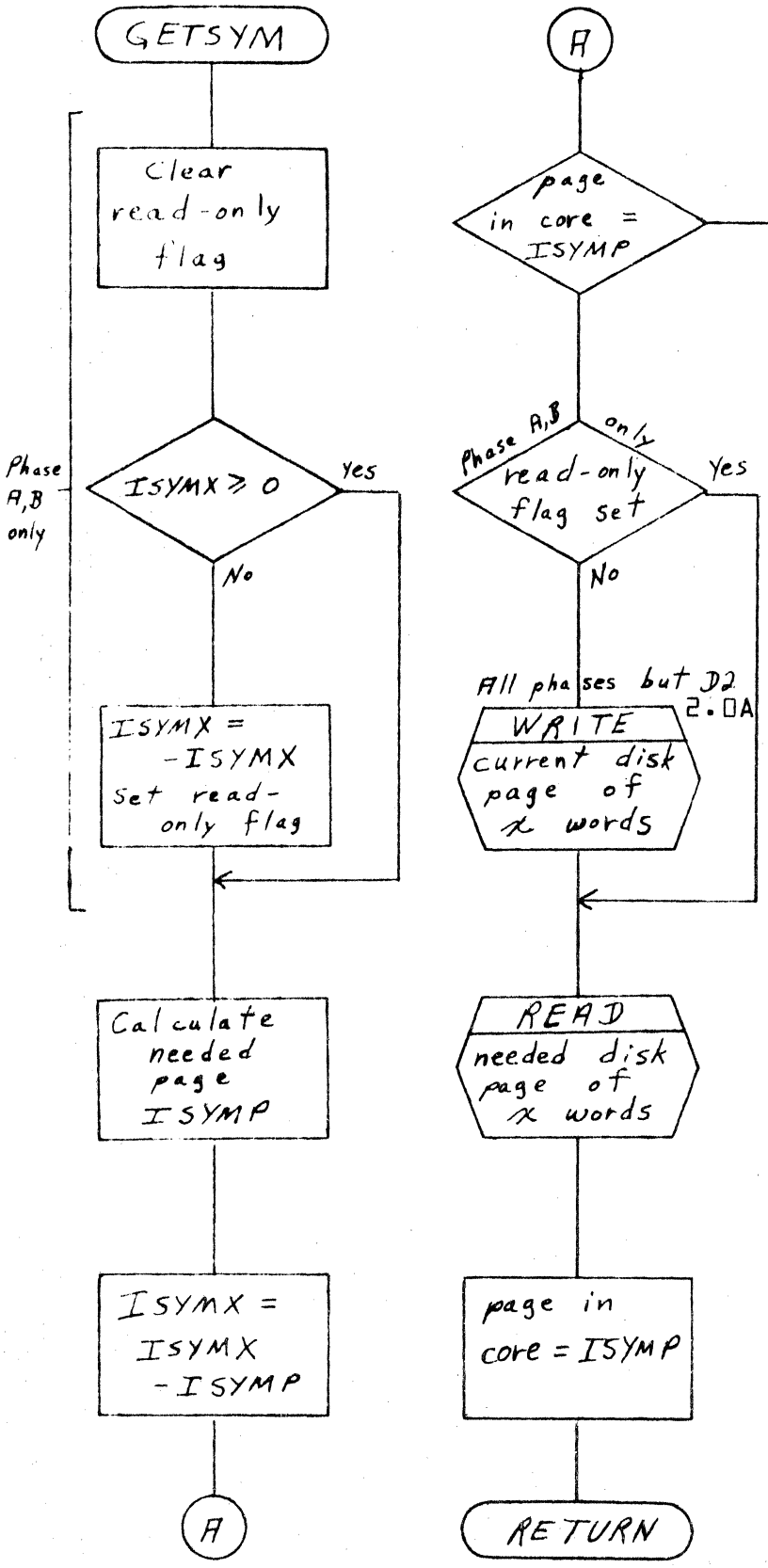
Input parameters: ISYMX (in-core subscript), ISYMP set to current entry.

Output parameters: ISYMX (in-core subscript), ISYMP updated to next entry.

There are four versions of SYMSCN. The phase D local 1 2.0A version references the compressed symbol table (see Assembly Output Phase, 5.2). The other three versions differ only in the dimension of the array SYMTAB in common.


SYMBOL
look for symbol ISYMX ≥ 0
Yes
No
I = 0
ISYMD = 1
ISYMX = 1
Phase A only
IFIRST = ISYMPC
14 → JUMP
10
Phase A only
IFIRST = ISYMPC
No
Yes
Phase A only
ISYMX = -ISYMX
don't save current SYMTAB page
GETSYM
JUMP
Phase B only
ISYMX = ISYMN next available entry
GETSYM
50
14
last entry processed
Yes
40
No
JSYM = symbol just read
No
30
Yes
statement function args. ISSFLG ≠ 0
No
42
Yes
I = ISYMX + ISYMP (pointer to symbol just read)
30
ISYMX = pointer to next entry
10


40
50
symbol I≠0 found
No
SYMTAB max. size exceeded
No
ISYMD=0
Yes
Yes
ISYMX=I
WRITE
(symbol table oflw)
F, 58
Zero SYMTAB entry at ISYMX
42
42 → JUMP
IXLGO=0
(clear "X" option)
RETURN
10
SKIPIT
error exit to GO routine
Restart Phase A

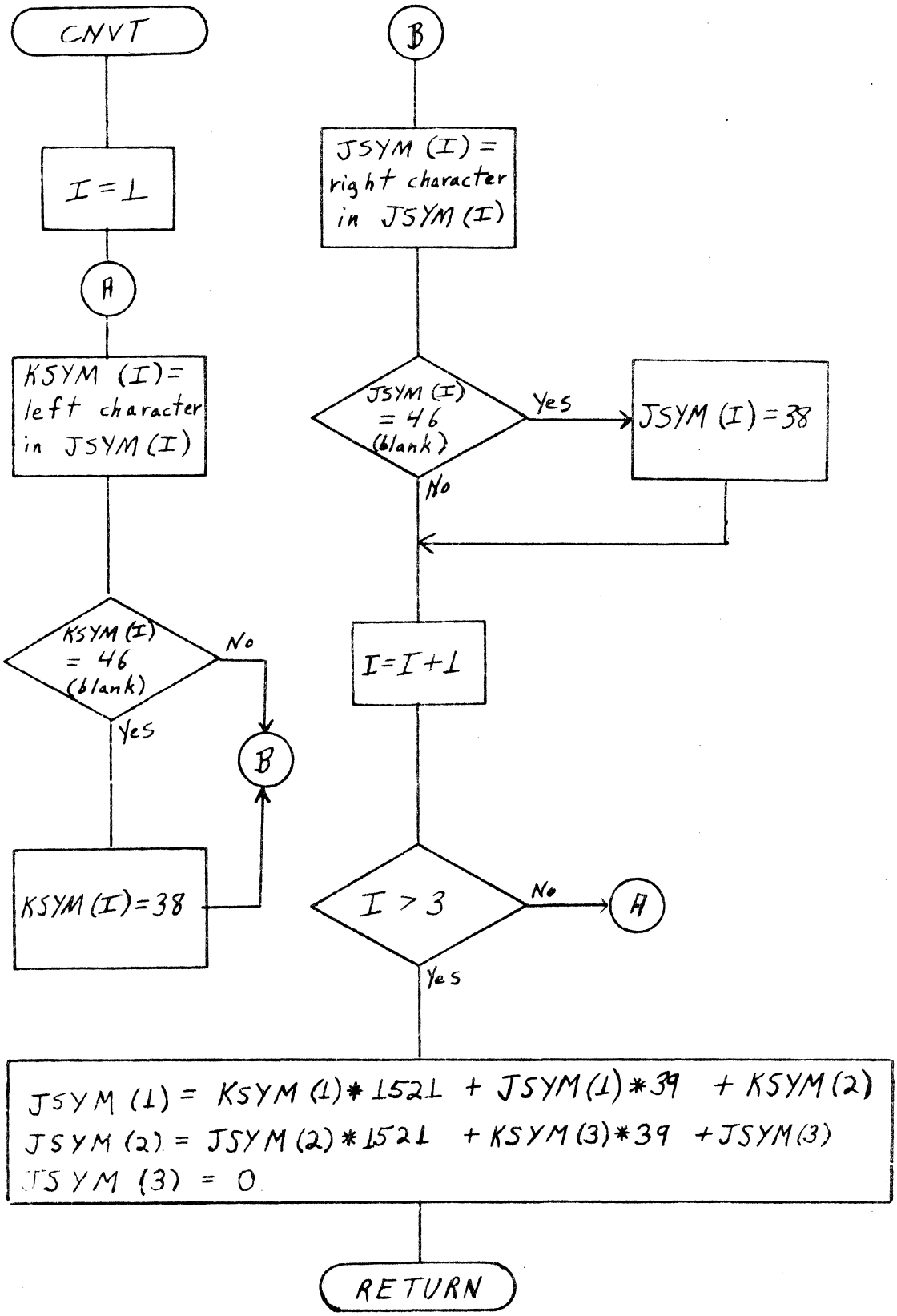


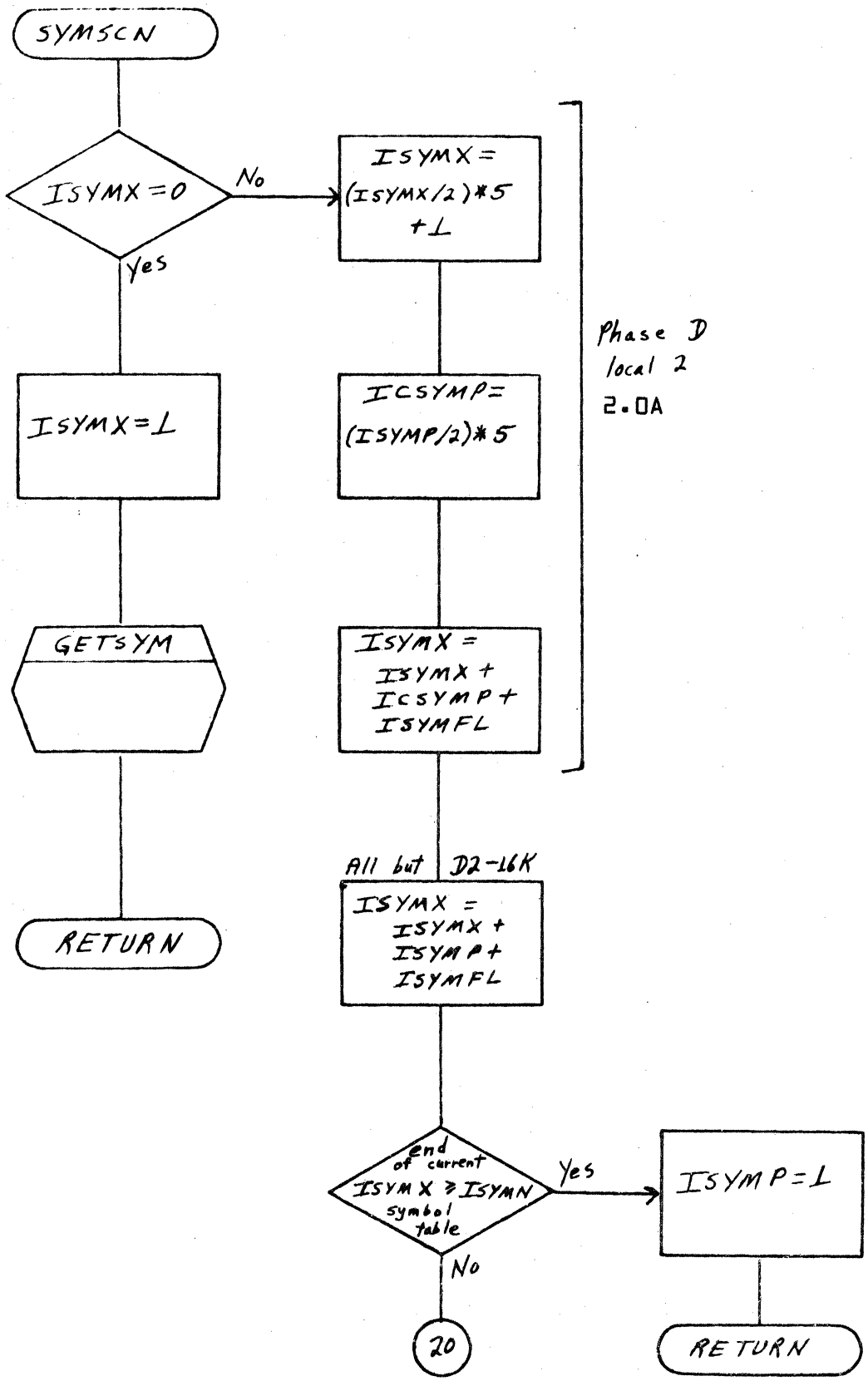
Phases A and B
$x = 960 + 96$
(SYMTAB and ISTABX)

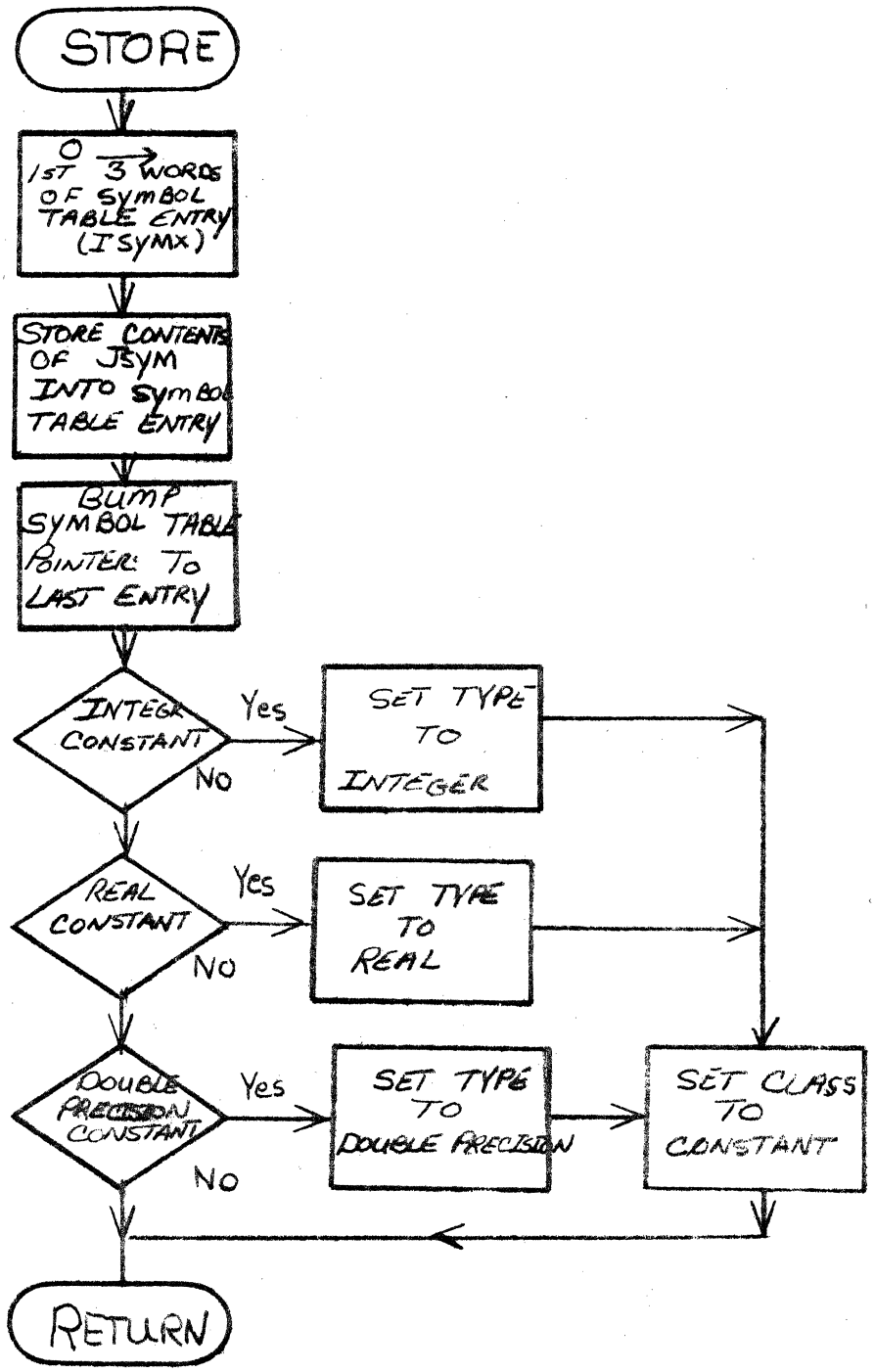
Phases C and D/E 2.0B
$x = 2400$
(skip every 11th sector)

Phases D2, E 2.0A
$x = 480$
(skip every 11th sector)

Phase D local 1 2.0A
$x = 1000$
Read and write CSYMTAB
(compressed symbol table)


CNVT
I = 1
A
KSYM (I) = left character in JSYM (I)
KSYM (I) = 46 (blank)
No
Yes
KSYM (I) = 38
B
B
JSYM (I) = right character in JSYM (I)
JSYM (I) = 46 (blank)
Yes
JSYM (I) = 38
No
I = I + 1
I > 3
No
A
Yes
JSYM (1) = KSYM (1) * 1521 + JSYM (1) * 39 + KSYM (2)
JSYM (2) = JSYM (2) * 1521 + KSYM (3) * 39 + JSYM (3)
JSYM (3) = 0
RETURN


SYMSCN
ISYMX=0
No
Yes
ISYMX=(ISYMX/2)*5+1
ISYMX=1
ICSYMP=(ISYMP/2)*5
GETSYM
ISYMX=ISYMX+ICSYMP+ISYMFL
Phase D
local 2
2.0A
All but D2-16K
ISYMX=ISYMX+ISYMP+ISYMFL
end of current symbol table
ISYMX ≥ ISYMN
Yes
No
ISYMP=1
RETURN
20
RETURN



TITLE	IMS STORE		DRG. NO.
			REVISION
DRAWN BY J. HARRINGTON	PROJ.	DATE 1-16-73	SHEET 1 OF 1

CSD 203

2.3.2 Field Processing Routines

2.3.2.1 Subroutine GETF

GETF is the subroutine which extracts a field from the source input in ISORS*. The column at which to start is given in ISORSX*. If the field is alphabetic (first character is A-Z), the field is placed left-justified in JSYM and JMODE set to 2. JESWT is set to indicate the implicit type of the name (1 = integer, 2 = real). If the field is a fixed, floating point, or double precision floating point constant, it is converted and placed in JSYM. JMODE is set to indicate the mode of the constant. Integer constants are placed in JSYM(1) converted as if they were decimal unless OCT ≠ 0 upon entrance to GETF, or if the constant is preceded by $. If OCT ≠ 0, an integer constant is considered octal and non-integer fields are considered errors. OCT is set to zero upon exit from GETF. If the constant is preceded by $, the constant is considered hexadecimal and the only permissible digits are 0-9, A-F. For integer constants: JSYM(2) = 0, JSYM(3) = 2424_{16}. Real constants are placed in JSYM(1) and JSYM(2) and JSYM(3) is set to 2525_{16}. Double precision floating point constants are placed in JSYM(1), JSYM(2), and JSYM(3) and common variables JMODE and JFLOT are set to 6 and 3 respectively.

If the field is null, JMODE is set = 0.

SYMBOL is called for alphabetic or numeric fields if NERR = 0 upon entrance to GETF.

The field terminator is placed in JTERM.

JTERM settings for 3 character logical operators are:

.AND. = AD
.NOT. = NT

Error messages will be printed as errors are encountered if NERR = 0 upon entrance to GETF.

NERR is set to zero upon exit from GETF.

JERR = 0 - no error encountered in field switch
JERR ≠ 0 - error encountered

*If AND (IFLAGS,$10) is not zero, input is in IBUF2 (IBUF2X). In 2.0A, there are two versions of GETF.

GETF switches

JESWT	constant analysis indicator 0 digits only encountered 1 decimal point encountered 2 E or D encountered 3 exponent is + 4 exponent is -
JFLOT	mode of constant 0 - integer 2 - real 3 - double precision
JOCHAR	character count
JOL	overflow indicator for constant conversion
JCHAR	character being looked at
JE	fraction count
JEL	exponent count
JCT	JSYM character indicator
JCTL	JSYM word count

If an error is encountered and NERR = 0, an error message is output and depending on the severity of error, JERR may be set ≠ 0. If JERR ≠ 0 upon exit from GETF, the contents of JSYM and JMODE are meaningless. NERR is set = 0 upon exit from GETF.

INPUT Parameters

OCT	convert integer constant as octal if OCT≠0. If field is not an octal integer, an error is assumed.
ISORSX	column of input at which to start
ISORS	source input table
NERR	≠ 0 says don't print errors, don't call SYMBOL.

OUTPUT Parameters

JSYM	3 word holder for field
JTERM	1 word holder for field terminator
JMODE	mode of JSYM 0 = null 2 = alphabetic 3 = integer - only JSYM(1) is significant 5 = real - JSYM(1) and (JSYM(2) contain value 6 = double precision - JSYM(1), JSYM(2), JSYM(3) contain value

ISORSX	updated to reflect start of next field or end of input
JERR	error switch
JESWT	for JMODE = 2, implicit type of JSYM 1 = integer 2 = real

1700 INTERNAL CODE

CHARACTER		CODE (Decimal)
digits	0-9	0-9
letters	A-Z	10-35
dollar sign	$	36
period	.	37
plus sign	+	38
minus sign	-	39
equal sign	=	40
left parenthesis	(	41
right parenthesis	)	42
comma	,	43
slash	/	44
asterisk	*	45
blank	⊿	46
End of statement	⊿	47
single quote	'	48

2.3.2.2 Subroutine GETC

GETC gets one character from ISORS. The column to get is given in ISORSX. If JBLANK is zero upon entrance to GETC, blank characters are skipped. If AND (IFLAGS,$10) is not zero, the character is from IBUF2 (IBUF2X). In 2.0A there are two versions. JBLANK is not initialized upon exit.

INPUT Parameters

ISORSX	card column at which to start
ISORS	source input table
JBLANK	present blanks switch. 0 = skip blanks; ≠ 0 - present blanks

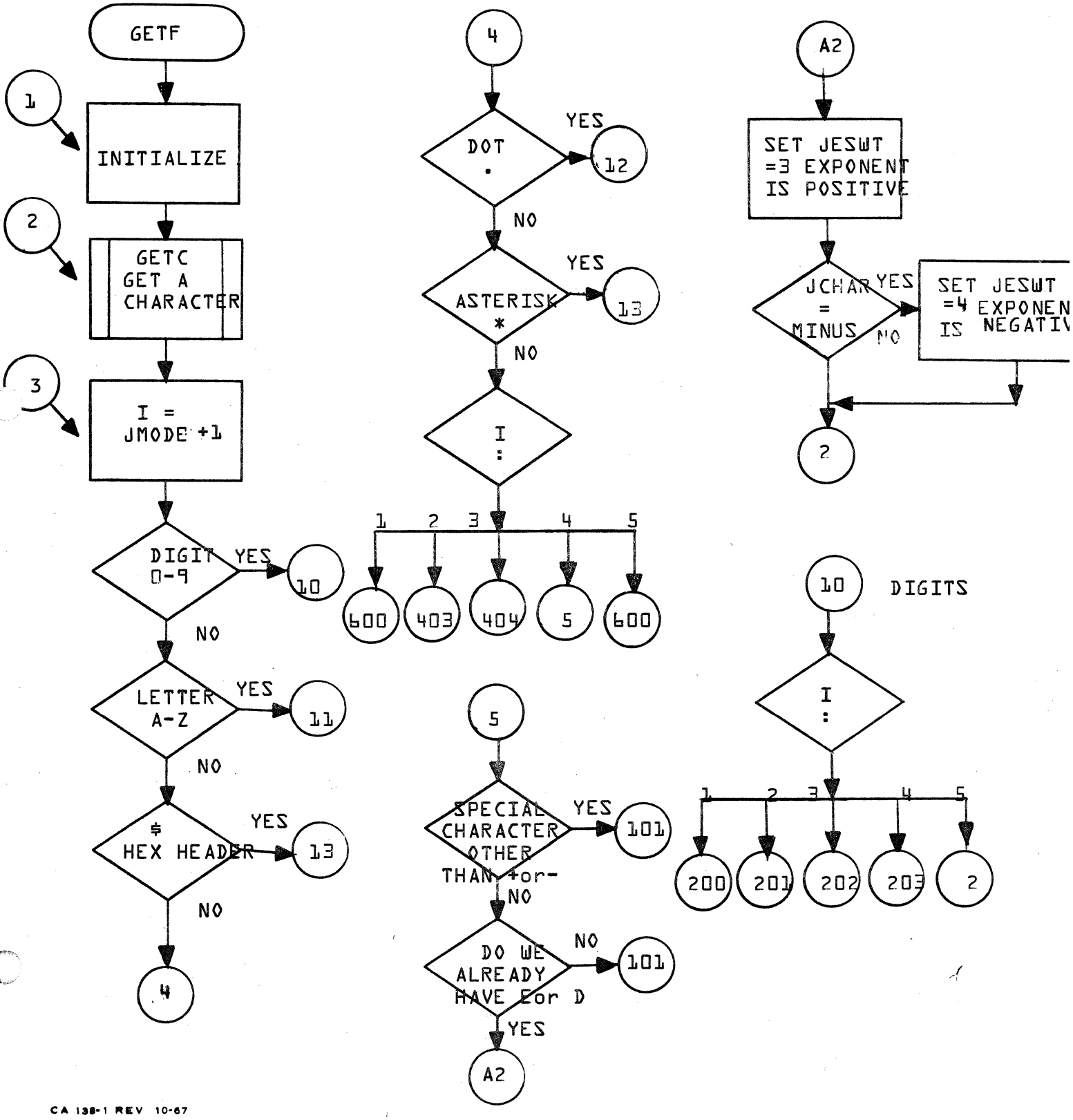
OUTPUT Parameters

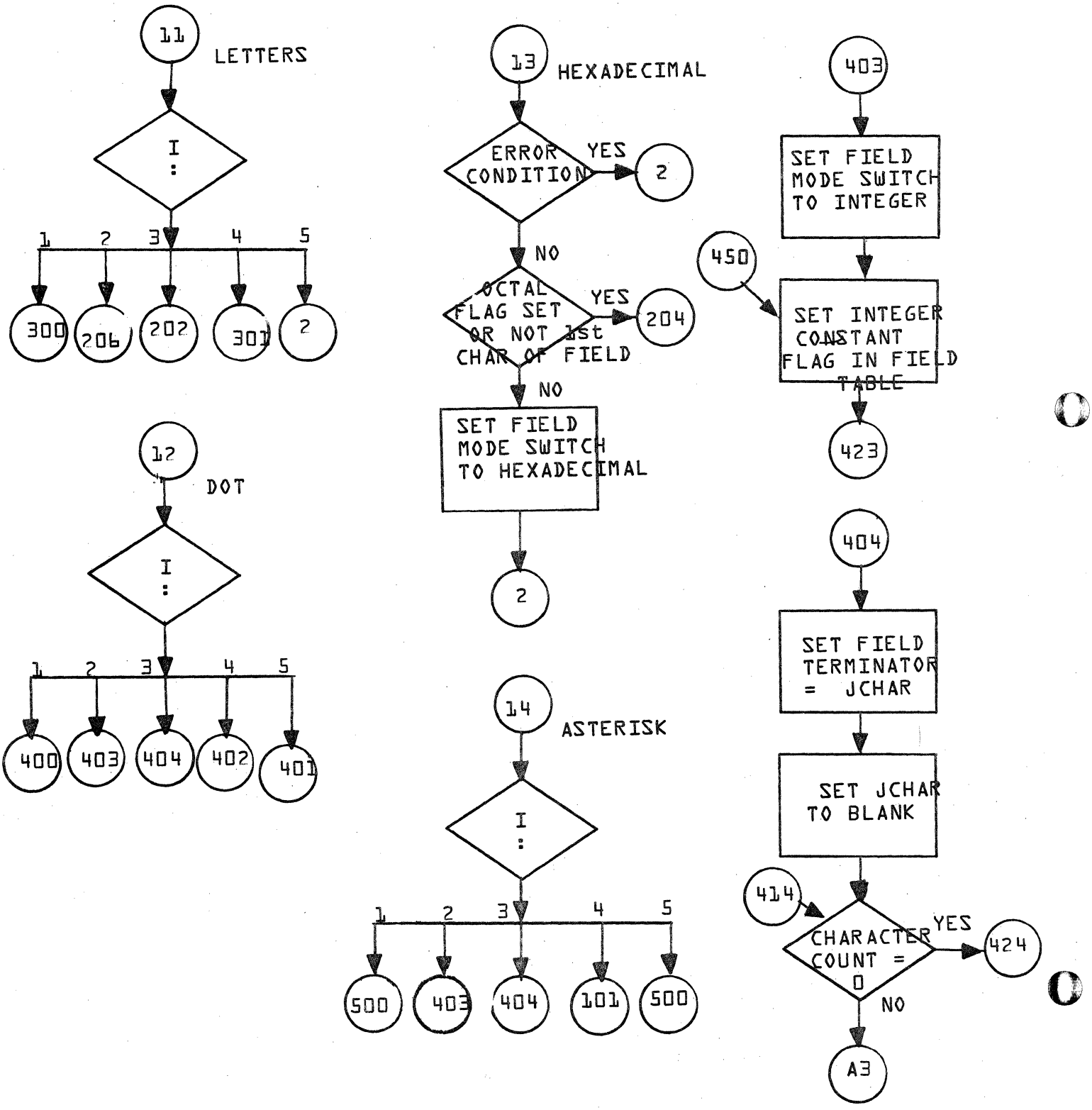
JCHAR	next significant character {right justified}
ISORSX	updated

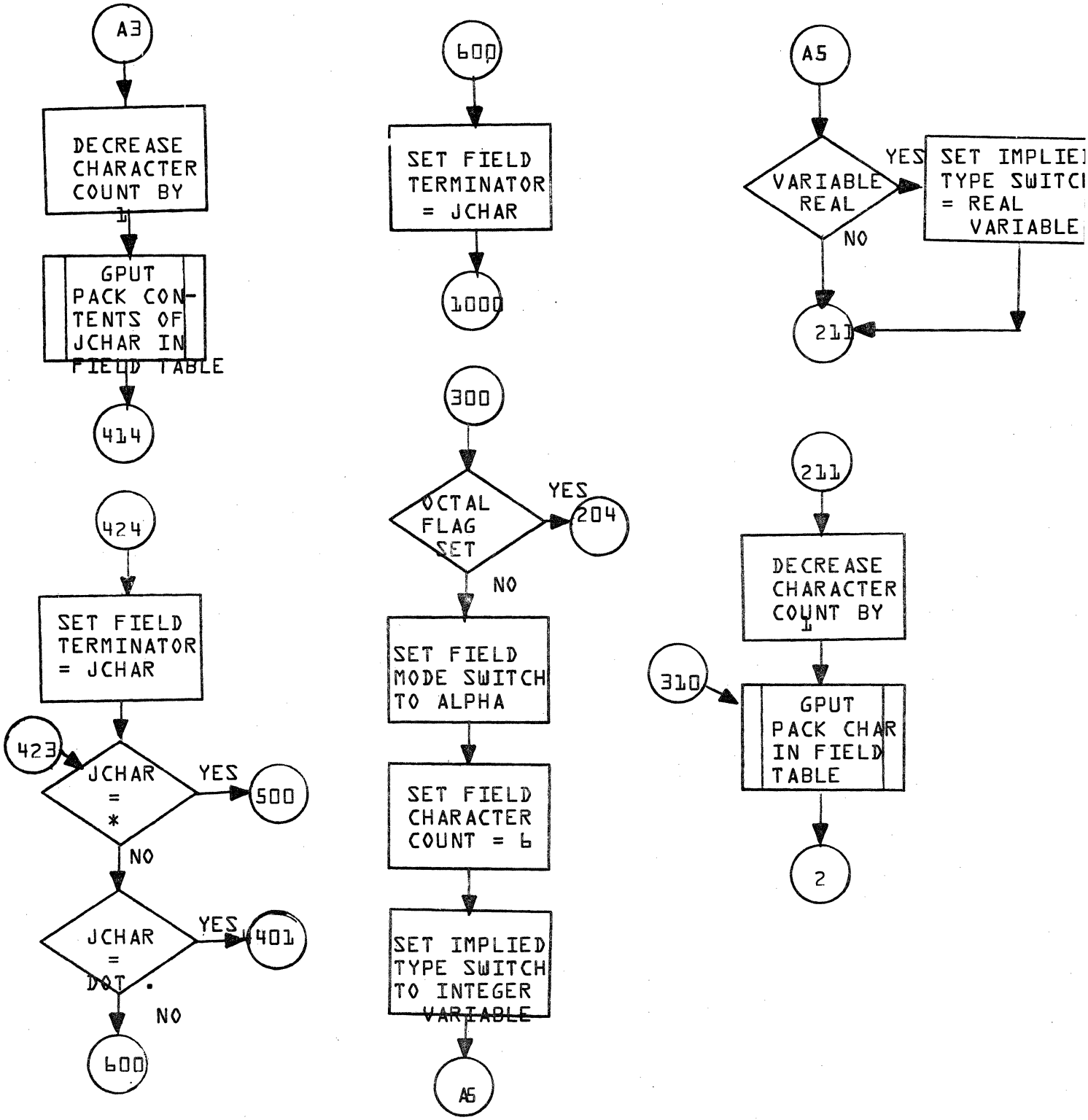
2.3.2.3 GPUT

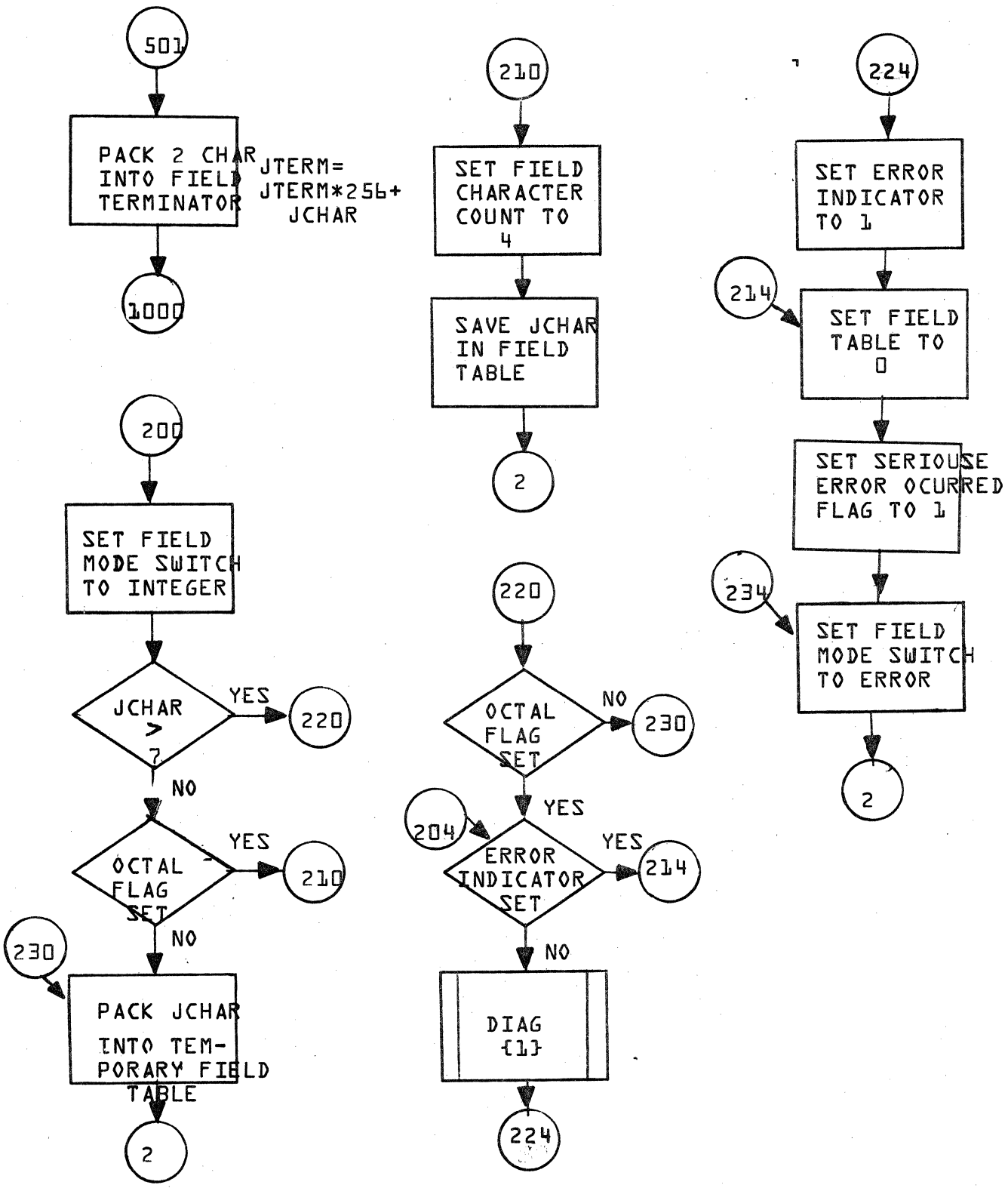
GPUT packs the character in JCHAR into the three word buffer JSYM into the 8 bit byte signaled by the current setting of JCT1 {character number}.

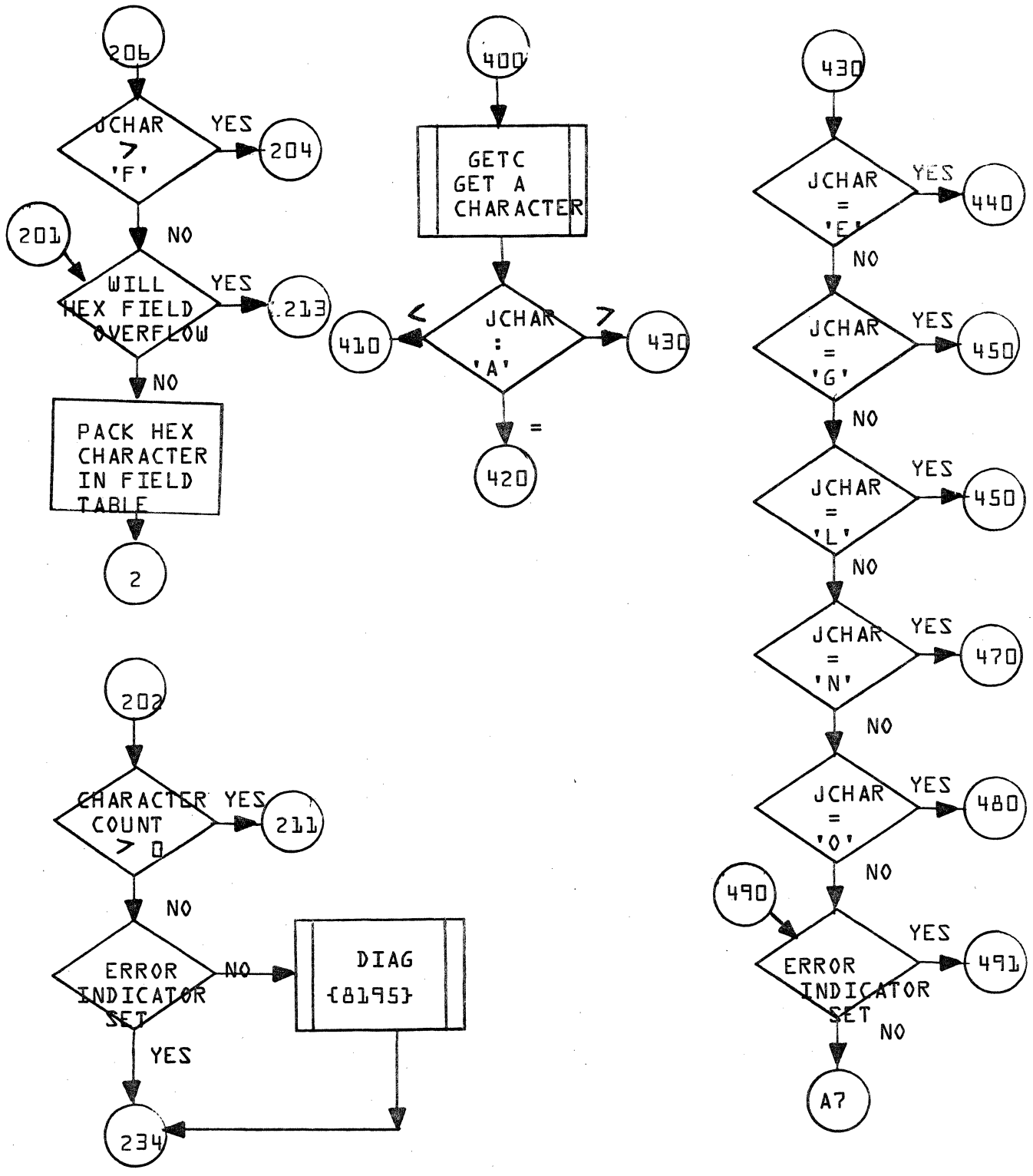
The characters are packed two per word.

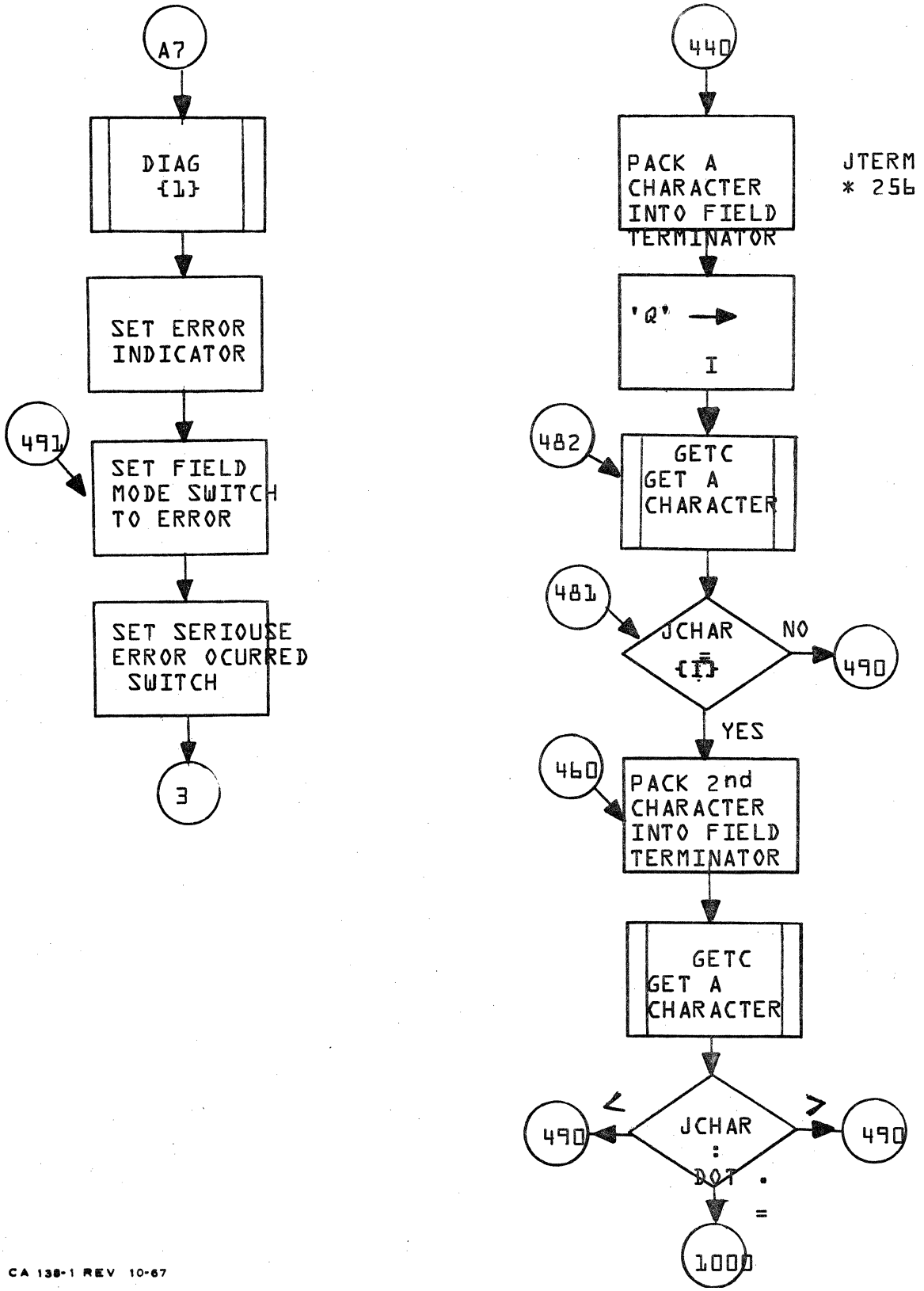

GETF
1
INITIALIZE
2
GETC
GET A
CHARACTER
3
I =
JMODE +1
DIGIT
0-9
YES
10
NO
LETTER
A-Z
YES
11
NO
$
HEX HEADER
YES
13
NO
4
4
DOT
.
YES
12
NO
ASTERISK
*
YES
13
NO
I
:
1 2 3 4 5
600 403 404 5 600
5
SPECIAL
CHARACTER
OTHER
THAN +or-
YES
101
NO
DO WE
ALREADY
HAVE E or D
NO
101
YES
A2
A2
SET JESWT
=3 EXPONENT
IS POSITIVE
JCHAR
=
MINUS
YES
SET JESWT
=4 EXPONEN
IS NEGATIV
NO
2
10
DIGITS
I
:
1 2 3 4 5
200 201 202 203 2

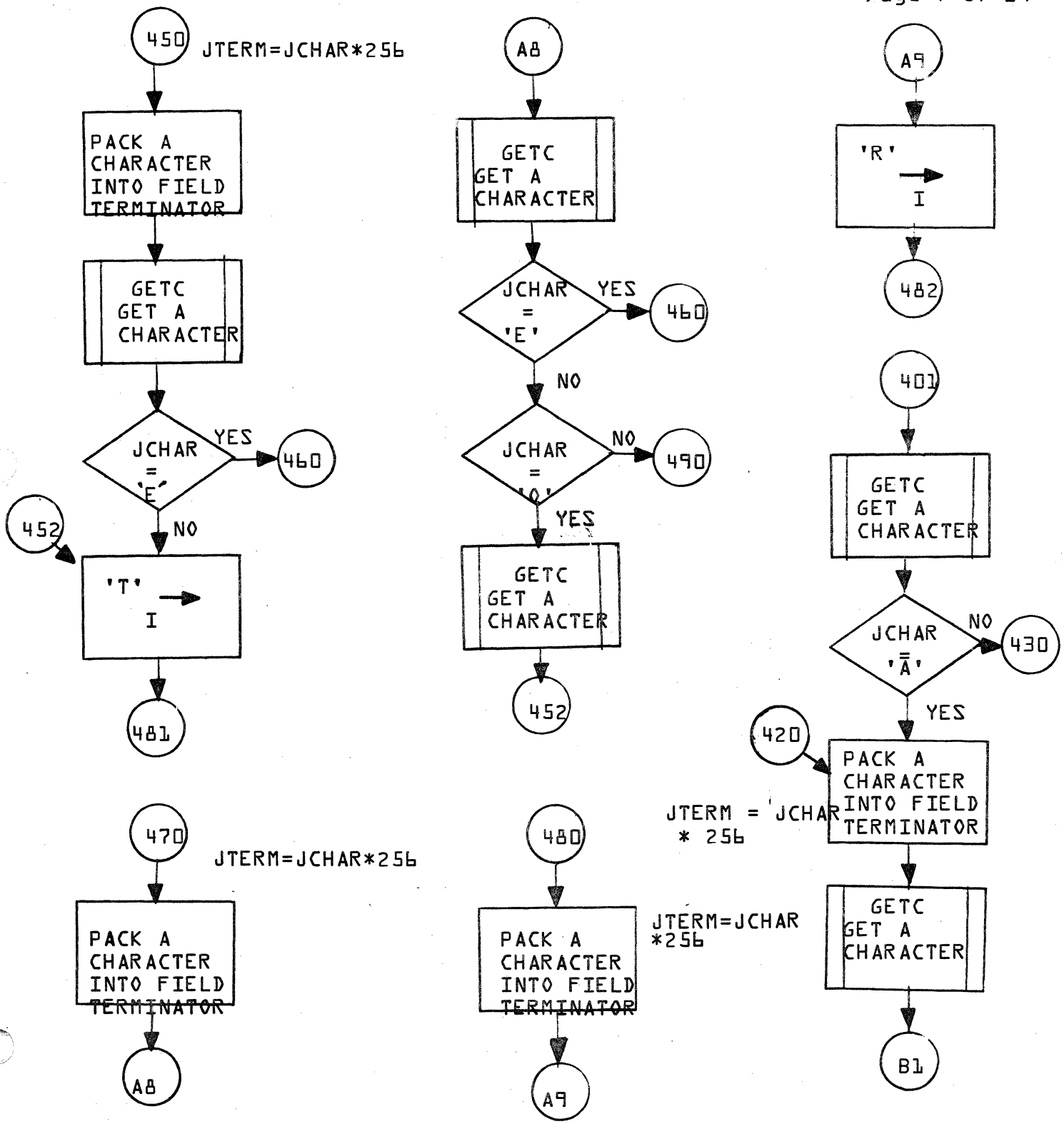

11
LETTERS
I
:
1 2 3 4 5
300 206 202 301 2
12
DOT
I
:
1 2 3 4 5
400 403 404 402 401
13
HEXADECIMAL
ERROR CONDITION
YES
2
NO
OCTAL FLAG SET OR NOT 1st CHAR OF FIELD
YES
204
NO
SET FIELD MODE SWITCH TO HEXADECIMAL
2
14
ASTERISK
I
:
1 2 3 4 5
500 403 404 101 500
403
SET FIELD MODE SWITCH TO INTEGER
450
SET INTEGER CONSTANT FLAG IN FIELD TABLE
423
404
SET FIELD TERMINATOR = JCHAR
SET JCHAR TO BLANK
414
CHARACTER COUNT = 0
YES
424
NO
A3

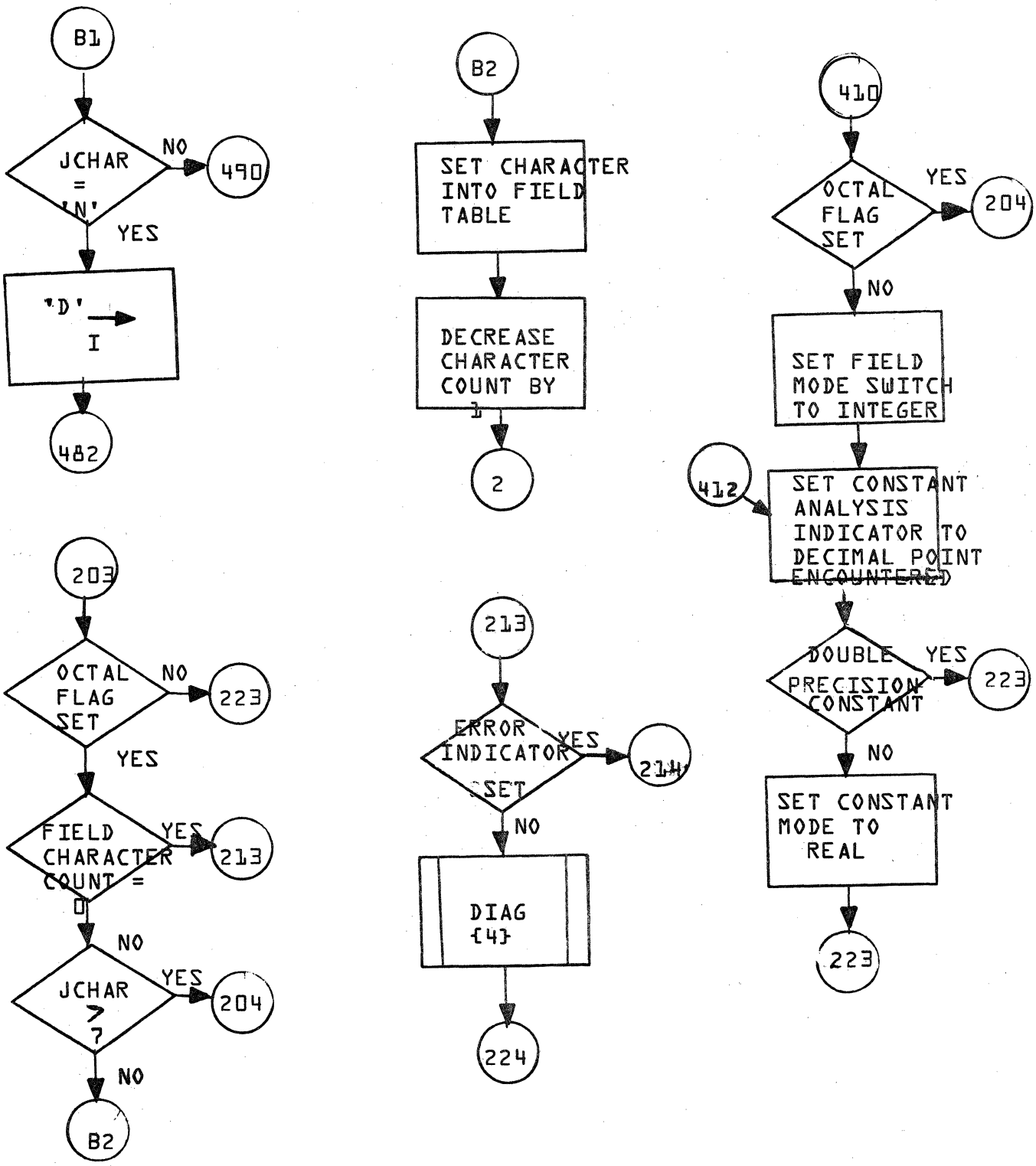

A3
DECREASE CHARACTER COUNT BY 1
GPUT PACK CONTENTS OF JCHAR IN FIELD TABLE
414
424
SET FIELD TERMINATOR = JCHAR
423
JCHAR = *
YES
500
NO
JCHAR = DOT .
YES
401
NO
600
600
SET FIELD TERMINATOR = JCHAR
1000
300
OCTAL FLAG SET
YES
204
NO
SET FIELD MODE SWITCH TO ALPHA
SET FIELD CHARACTER COUNT = 6
SET IMPLIED TYPE SWITCH TO INTEGER VARIABLE
A5
A5
VARIABLE REAL
YES
SET IMPLIE TYPE SWITCH = REAL VARIABLE
NO
211
211
DECREASE CHARACTER COUNT BY 1
310
GPUT PACK CHAR IN FIELD TABLE
2

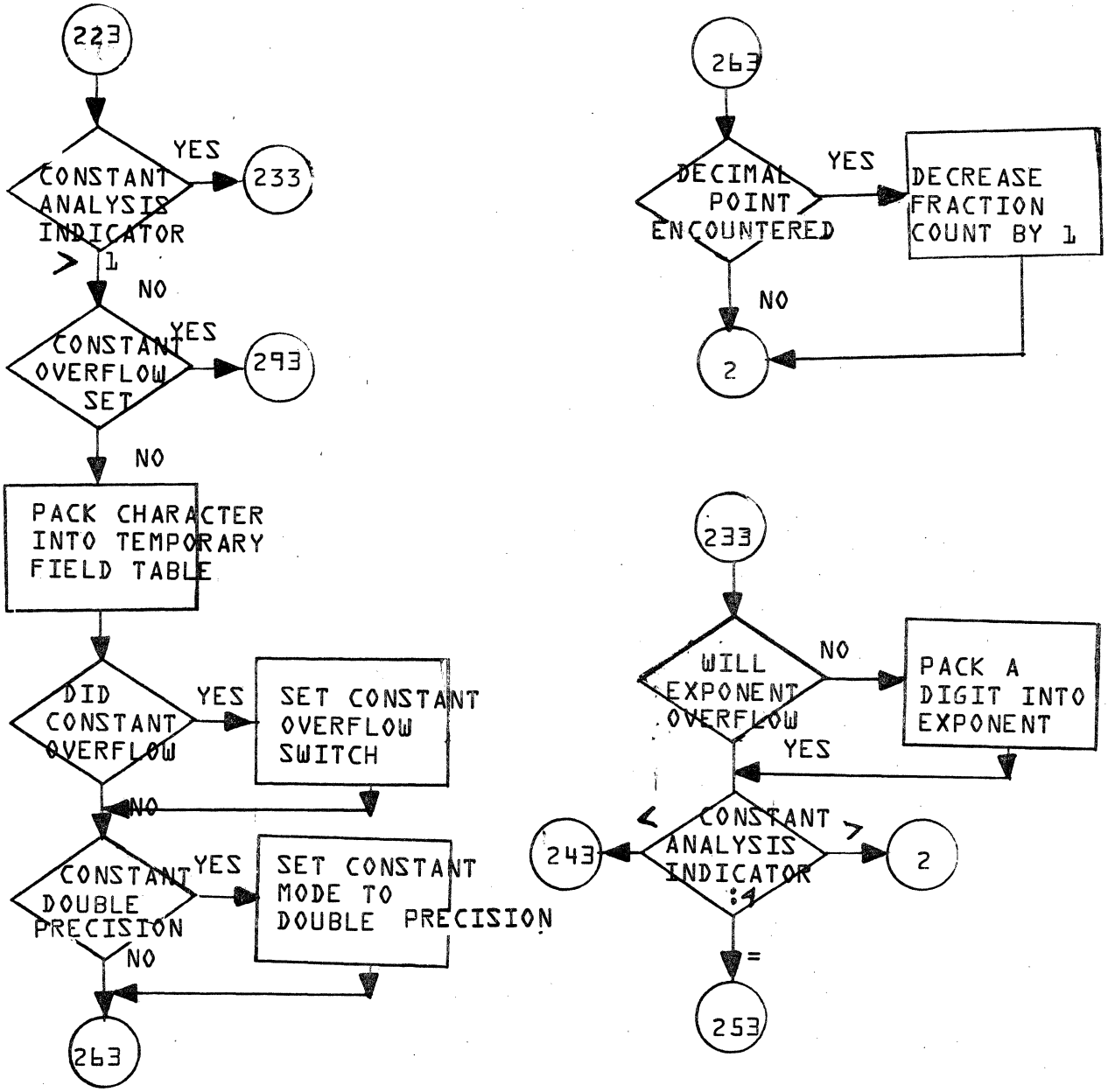

501
PACK 2 CHAR INTO FIELD TERMINATOR
JTERM= JTERM*256+ JCHAR
1000
200
SET FIELD MODE SWITCH TO INTEGER
JCHAR > 7
YES
220
NO
OCTAL FLAG SET
YES
210
NO
230
PACK JCHAR INTO TEMPORARY FIELD TABLE
2
210
SET FIELD CHARACTER COUNT TO 4
SAVE JCHAR IN FIELD TABLE
2
220
OCTAL FLAG SET
NO
230
YES
204
ERROR INDICATOR SET
YES
214
NO
DIAG (1)
224
224
SET ERROR INDICATOR TO 1
214
SET FIELD TABLE TO 0
SET SERIOUSE ERROR OCURRED FLAG TO 1
234
SET FIELD MODE SWITCH TO ERROR
2

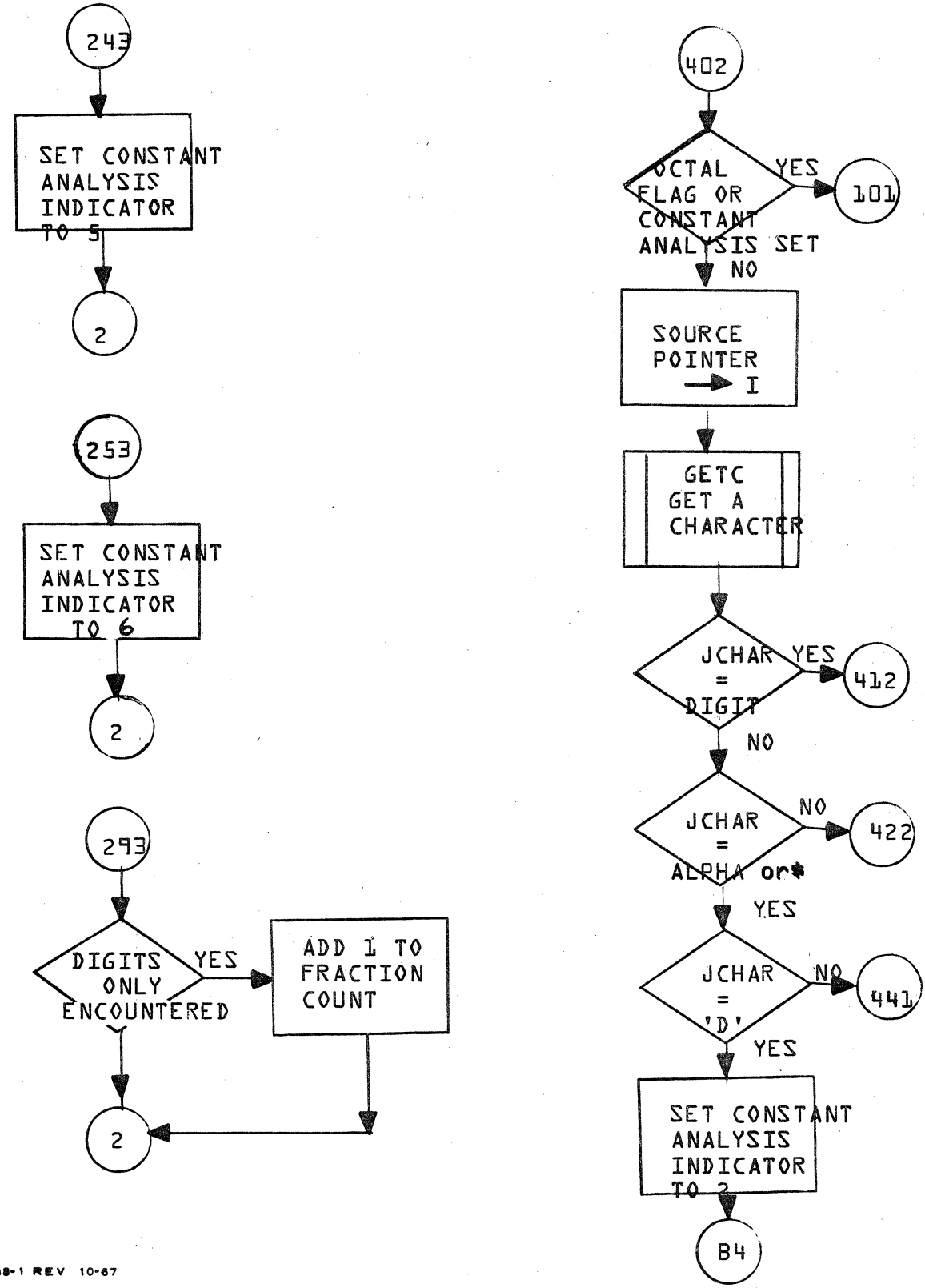

206
JCHAR > 'F'
YES
204
NO
201
WILL HEX FIELD OVERFLOW
YES
213
NO
PACK HEX CHARACTER IN FIELD TABLE
2
400
GETC GET A CHARACTER
JCHAR : 'A'
<
410
>
430
=
420
430
JCHAR = 'E'
YES
440
NO
JCHAR = 'G'
YES
450
NO
JCHAR = 'L'
YES
450
NO
JCHAR = 'N'
YES
470
NO
JCHAR = 'O'
YES
480
NO
490
ERROR INDICATOR SET
YES
491
NO
A7
202
CHARACTER COUNT > 0
YES
211
NO
ERROR INDICATOR SET
NO
DIAG (8195)
YES
234

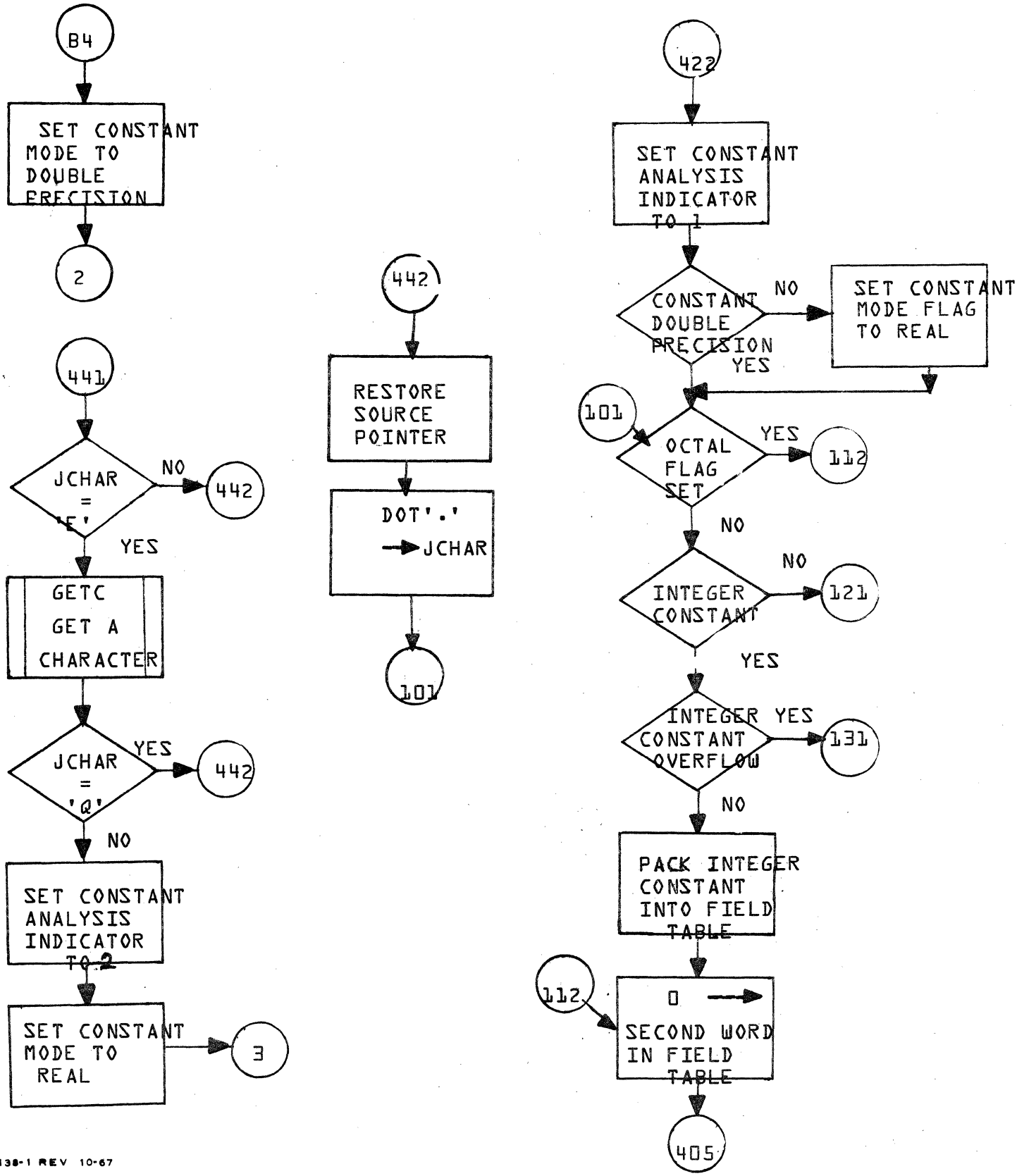

A7
DIAG {1}
SET ERROR INDICATOR
491
SET FIELD MODE SWITCH TO ERROR
SET SERIOUSE ERROR OCURRED SWITCH
3
440
PACK A CHARACTER INTO FIELD TERMINATOR
JTERM = JCHAR * 256
'Q' → I
482
GETC GET A CHARACTER
481
JCHAR {I}
NO
490
YES
460
PACK 2nd CHARACTER INTO FIELD TERMINATOR
GETC GET A CHARACTER
JCHAR
<
>
:
.
=
490
490
1000

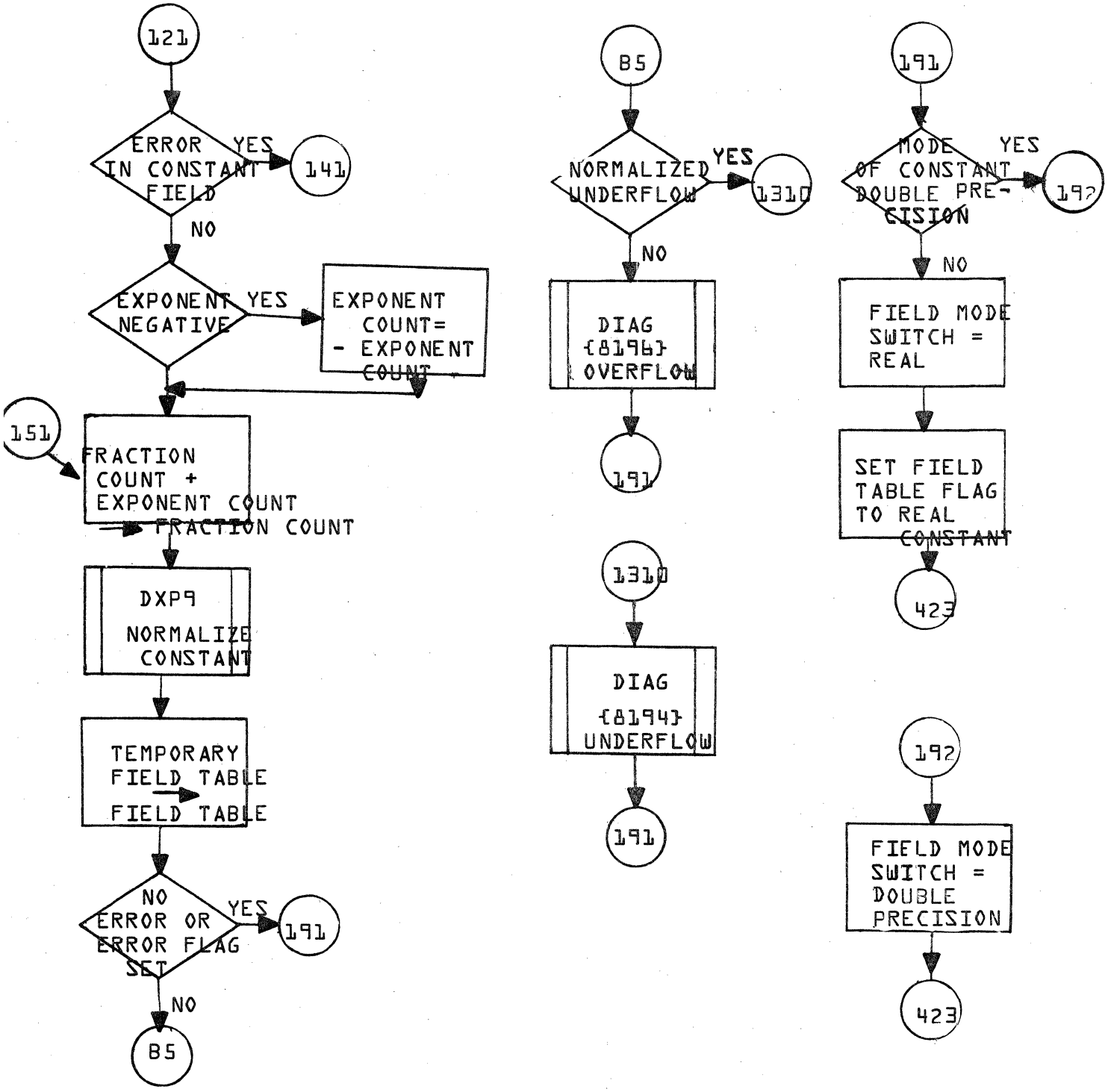

450
JTERM=JCHAR*256
PACK A CHARACTER INTO FIELD TERMINATOR
GETC GET A CHARACTER
JCHAR = 'E'
YES
460
NO
452
'T' I
481
A8
GETC GET A CHARACTER
JCHAR = 'E'
YES
460
NO
JCHAR = 'O'
NO
490
YES
GETC GET A CHARACTER
452
A9
'R' I
482
401
GETC GET A CHARACTER
JCHAR = 'A'
NO
430
YES
420
PACK A CHARACTER INTO FIELD TERMINATOR
JTERM = JCHAR * 256
GETC GET A CHARACTER
B1
470
JTERM=JCHAR*256
PACK A CHARACTER INTO FIELD TERMINATOR
A8
480
PACK A CHARACTER INTO FIELD TERMINATOR
JTERM=JCHAR *256
A9

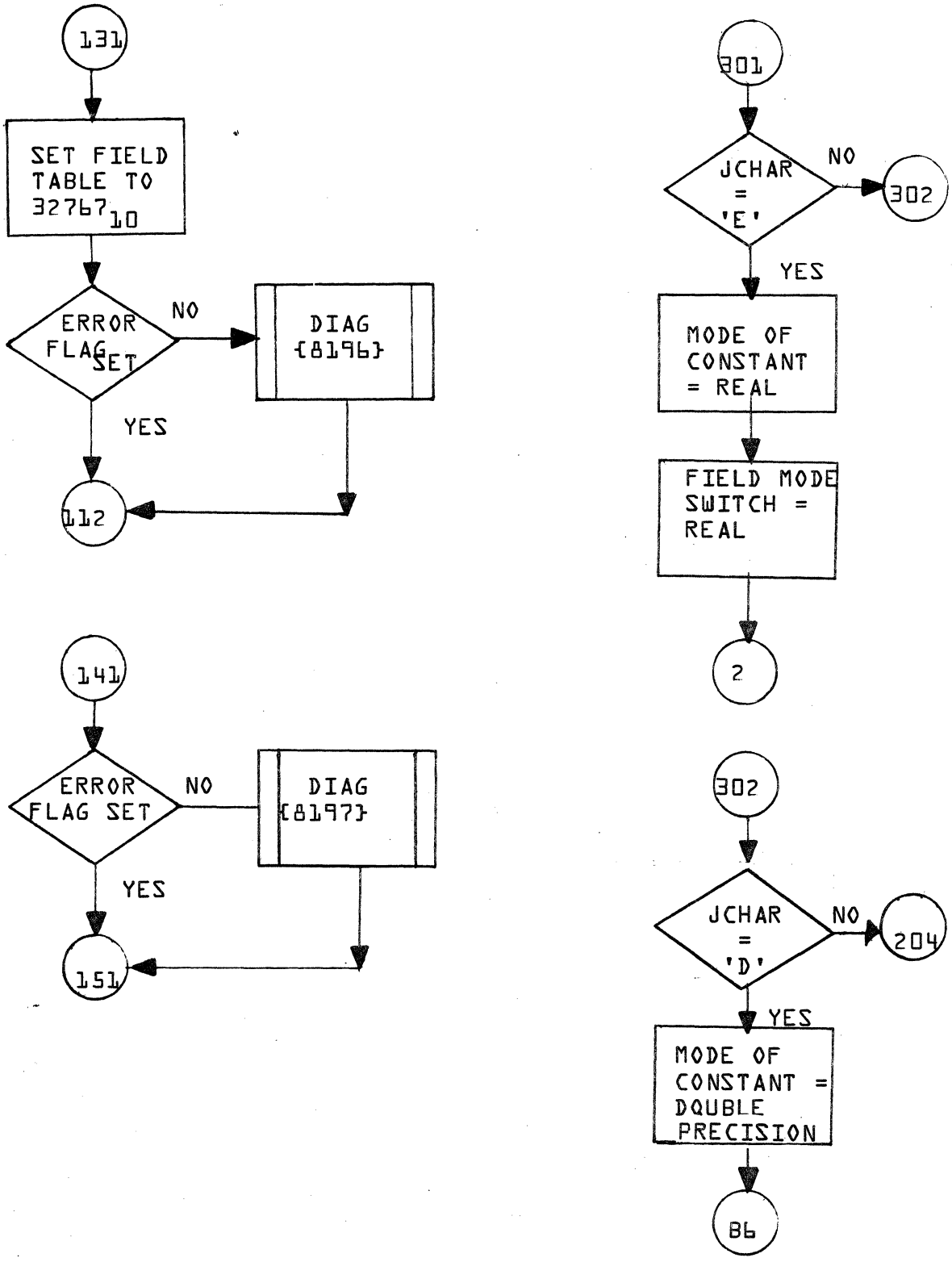

B1
JCHAR = 'N'
NO
490
YES
'D' → I
482
B2
SET CHARACTER INTO FIELD TABLE
DECREASE CHARACTER COUNT BY 1
2
410
OCTAL FLAG SET
YES
204
NO
SET FIELD MODE SWITCH TO INTEGER
412
SET CONSTANT ANALYSIS INDICATOR TO DECIMAL POINT ENCOUNTERED
DOUBLE PRECISION CONSTANT
YES
223
NO
SET CONSTANT MODE TO REAL
223
203
OCTAL FLAG SET
NO
223
YES
FIELD CHARACTER COUNT = 0
YES
213
NO
JCHAR > ?
YES
204
NO
B2
213
ERROR INDICATOR SET
YES
214
NO
DIAG {4}
224

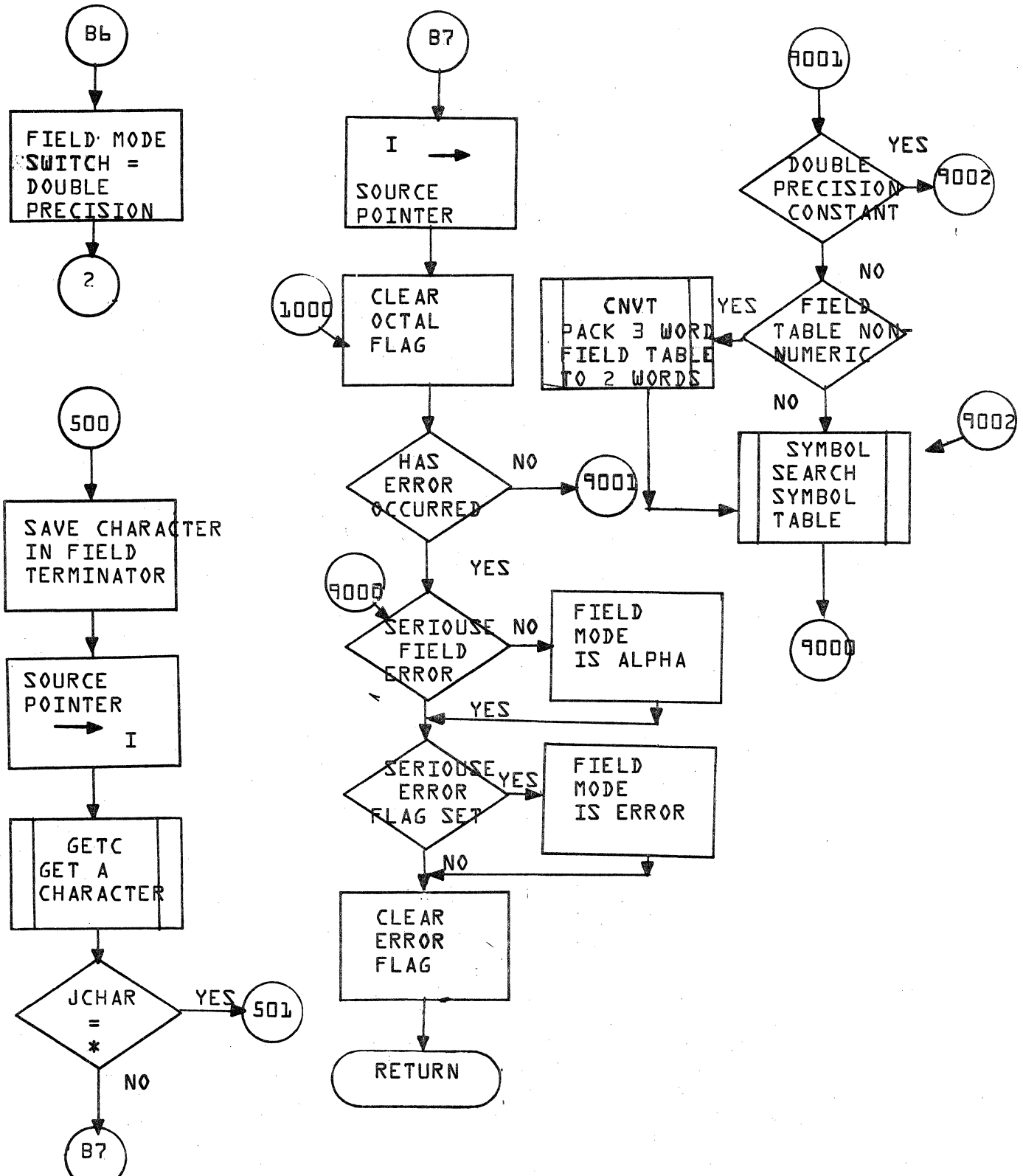

223
CONSTANT ANALYSIS INDICATOR
YES
233
> 1
NO
CONSTANT OVERFLOW SET
YES
293
NO
PACK CHARACTER INTO TEMPORARY FIELD TABLE
DID CONSTANT OVERFLOW
YES
SET CONSTANT OVERFLOW SWITCH
NO
CONSTANT DOUBLE PRECISION
YES
SET CONSTANT MODE TO DOUBLE PRECISION
NO
263
263
DECIMAL POINT ENCOUNTERED
YES
DECREASE FRACTION COUNT BY 1
NO
2
233
WILL EXPONENT OVERFLOW
NO
PACK A DIGIT INTO EXPONENT
YES
< CONSTANT ANALYSIS INDICATOR >
243
2
= 4
=
253


243
SET CONSTANT ANALYSIS INDICATOR TO 5
2
253
SET CONSTANT ANALYSIS INDICATOR TO 6
2
293
DIGITS ONLY ENCOUNTERED
YES
ADD 1 TO FRACTION COUNT
2
402
OCTAL FLAG OR CONSTANT ANALYSIS SET
YES
101
NO
SOURCE POINTER → I
GETC GET A CHARACTER
JCHAR = DIGIT
YES
412
NO
JCHAR = ALPHA or *
NO
422
YES
JCHAR = 'D'
NO
441
YES
SET CONSTANT ANALYSIS INDICATOR TO 2
B4


B4
SET CONSTANT MODE TO DOUBLE PRECISION
2
441
JCHAR = 'E'
NO
442
YES
GETC GET A CHARACTER
JCHAR = 'Q'
YES
442
NO
SET CONSTANT ANALYSIS INDICATOR TO 2
SET CONSTANT MODE TO REAL
3
442
RESTORE SOURCE POINTER
DOT '.' → JCHAR
101
422
SET CONSTANT ANALYSIS INDICATOR TO 1
CONSTANT DOUBLE PRECISION
NO
SET CONSTANT MODE FLAG TO REAL
YES
101
OCTAL FLAG SET
YES
112
NO
INTEGER CONSTANT
NO
121
YES
INTEGER CONSTANT OVERFLOW
YES
131
NO
PACK INTEGER CONSTANT INTO FIELD TABLE
112
0 → SECOND WORD IN FIELD TABLE
405


121
ERROR IN CONSTANT FIELD
YES
141
NO
EXPONENT NEGATIVE
YES
EXPONENT COUNT= - EXPONENT COUNT
151
FRACTION COUNT + EXPONENT COUNT → FRACTION COUNT
DXP9 NORMALIZE CONSTANT
TEMPORARY FIELD TABLE → FIELD TABLE
NO ERROR OR ERROR FLAG SET
YES
191
NO
B5
B5
NORMALIZED UNDERFLOW
YES
1310
NO
DIAG (8196) OVERFLOW
191
1310
DIAG (8194) UNDERFLOW
191
191
MODE OF CONSTANT DOUBLE PRECISION
YES
192
NO
FIELD MODE SWITCH = REAL
SET FIELD TABLE FLAG TO REAL CONSTANT
423
192
FIELD MODE SWITCH = DOUBLE PRECISION
423


131
SET FIELD TABLE TO 32767 10
ERROR FLAG SET
NO
DIAG (8196)
YES
112
141
ERROR FLAG SET
NO
DIAG (8197)
YES
151
301
JCHAR = 'E'
NO
302
YES
MODE OF CONSTANT = REAL
FIELD MODE SWITCH = REAL
2
302
JCHAR = 'D'
NO
204
YES
MODE OF CONSTANT = DOUBLE PRECISION
86

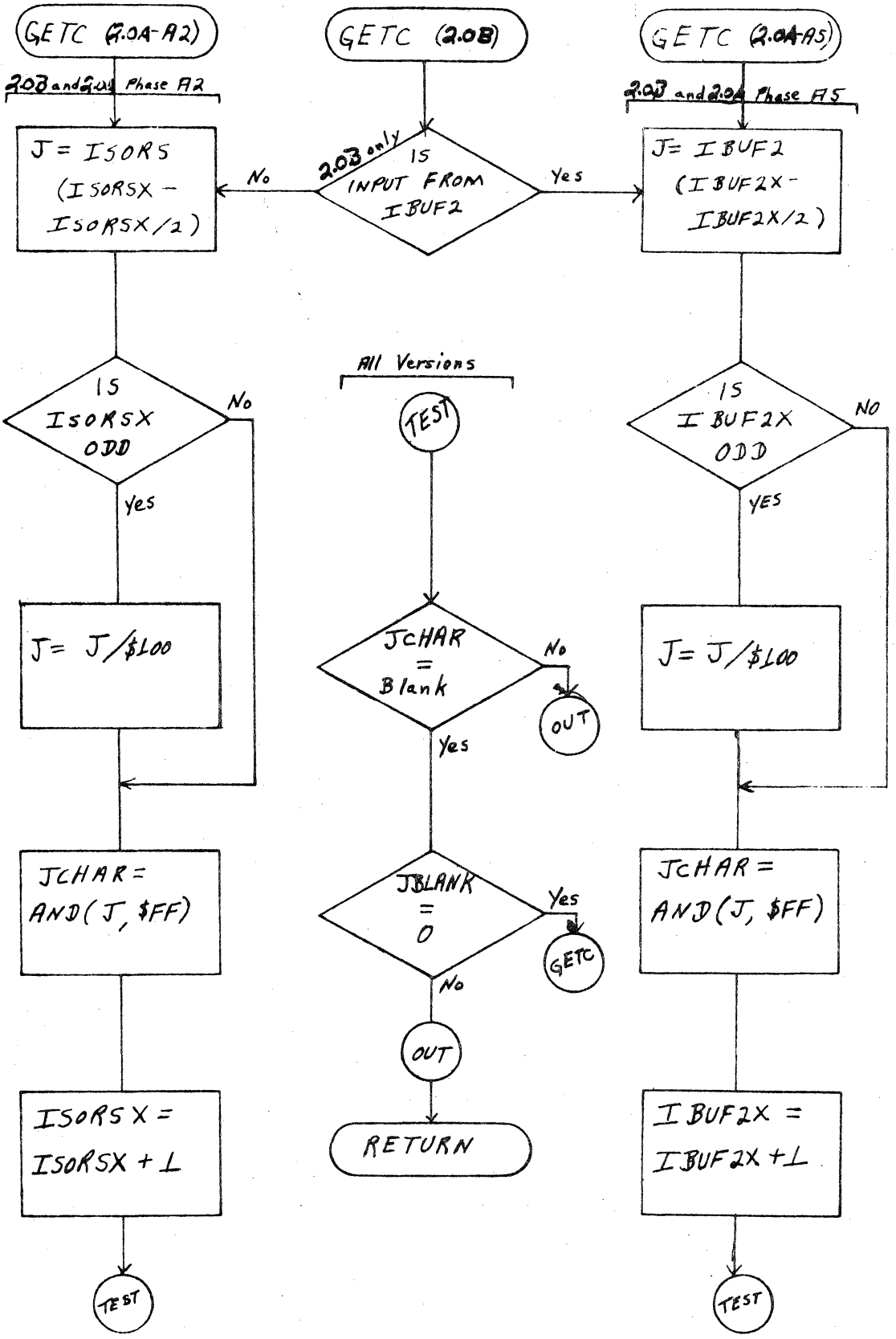

B6
FIELD MODE SWITCH = DOUBLE PRECISION
2
500
SAVE CHARACTER IN FIELD TERMINATOR
SOURCE POINTER I
GETC GET A CHARACTER
JCHAR = *
YES
501
NO
B7
B7
I SOURCE POINTER
1000
CLEAR OCTAL FLAG
HAS ERROR OCCURRED
NO
9001
YES
9000
SERIOUSE FIELD ERROR
NO
FIELD MODE IS ALPHA
YES
SERIOUSE ERROR FLAG SET
YES
FIELD MODE IS ERROR
NO
CLEAR ERROR FLAG
RETURN
9001
DOUBLE PRECISION CONSTANT
YES
9002
NO
FIELD TABLE NON-NUMERIC
YES
CNVT PACK 3 WORD FIELD TABLE TO 2 WORDS
NO
9002
SYMBOL SEARCH SYMBOL TABLE
9000

CONTROL DATA CORPORATION
LA JOLLA RESOURCE CENTER DIVISION

DOCUMENT CLASS IMS PAGE NO. 2-129 thru 2-135
PRODUCT NAME 1700 MASS STORAGE FORTRAN
PRODUCT MODEL NO. C005*3.1 A/B MACHINE SERIES 1700

PAGES 2-129 THRU 2-135

ARE INTENTIONALLY LEFT BLANK


GETC (2.0A-A2)
2.03 and 2.04 Phase A2
J = ISORS (ISORSX - ISORSX/2)
IS ISORSX ODD
No
Yes
J = J/$100
JCHAR = AND(J, $FF)
ISORSX = ISORSX + L
TEST
GETC (2.0B)
2.03 only
IS INPUT FROM IBUF2
No
Yes
All Versions
TEST
JCHAR = Blank
No
OUT
Yes
JBLANK = 0
Yes
GETC
No
OUT
RETURN
GETC (2.0A-A5)
2.03 and 2.04 Phase A5
J = IBUF2 (IBUF2X - IBUF2X/2)
IS IBUF2X ODD
NO
YES
J = J/$100
JCHAR = AND(J, $FF)
IBUF2X = IBUF2X + L
TEST

2.3.2.4 Function IGETCF

IGETCF is used by subroutine GNST to get a specific character from an input string.

Input parameters:

IMAGE - location of start of string

ICOLMN - character within string

2.3.2.5 Subroutine STCHAR

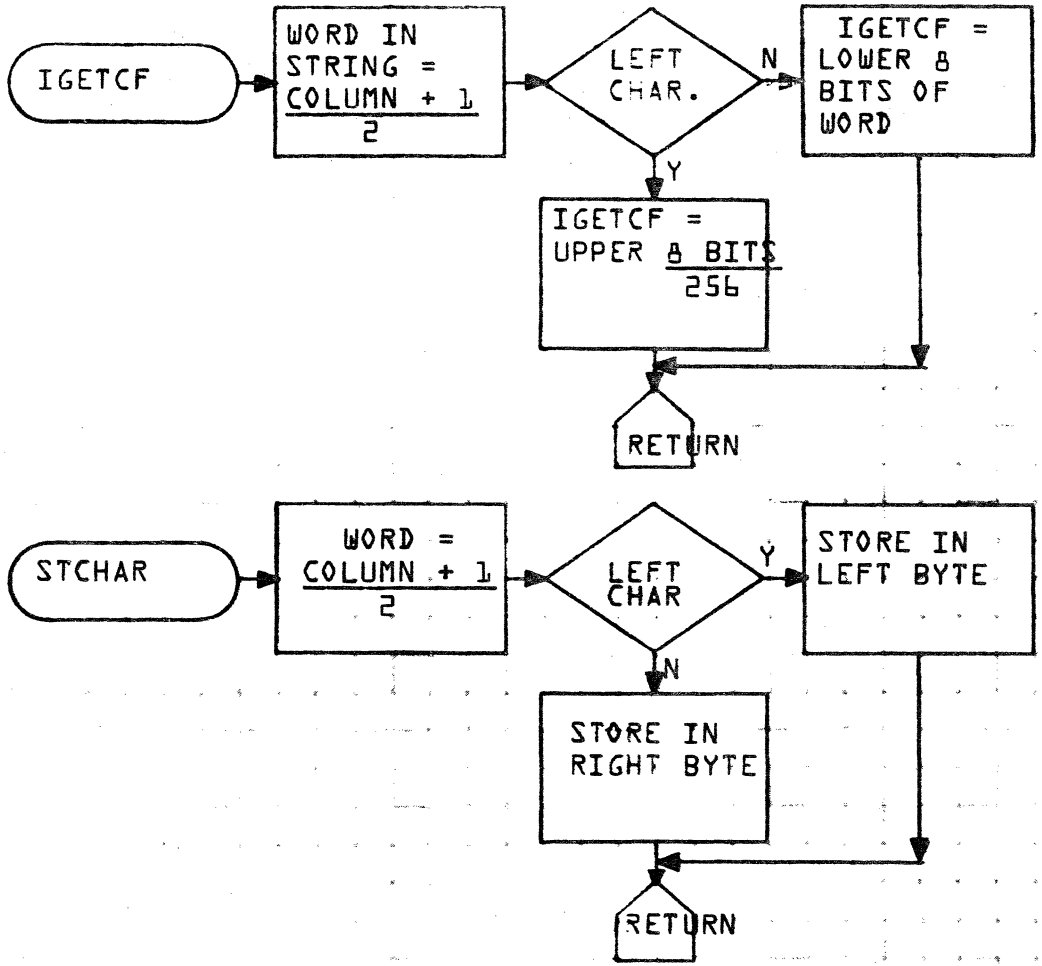
STCHAR is used by subroutine GNST to store an eight bit character into a character string.

Input parameters:

IMAGE - location of start of string

ICOLMN - character within string

ICHARC - character to store



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE IGETCF, STCHAR		PROJECT MGR.			
FLOWCHART ☒	PAGE 1 OF 1		PROJECT NAME FORTRAN			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1388 (FORMERLY CA127-1)

PRINTED IN U.S.A.

2.3.3 Special Field Handling Routines

2.3.3.1 CONSUB

Given an element of an array, the CONSUB subroutine is used to compute the increment from the subscripts of the element in order to reference the location of the element in the array. The CONSUB subroutine requires that subscripts be integer constants. It computes the nth element of an array.

Given the array defined by

```
DIMENSION A (I,J,K),
```

and the element

```
A (I1, J1, K1)
```

the increment is computed and recorded in IVCFLG as follows:

```
IVCFLG = (K1-1)*I*J+(J1-1)*I+I1
```

The resulting value in IVCFLG is such that

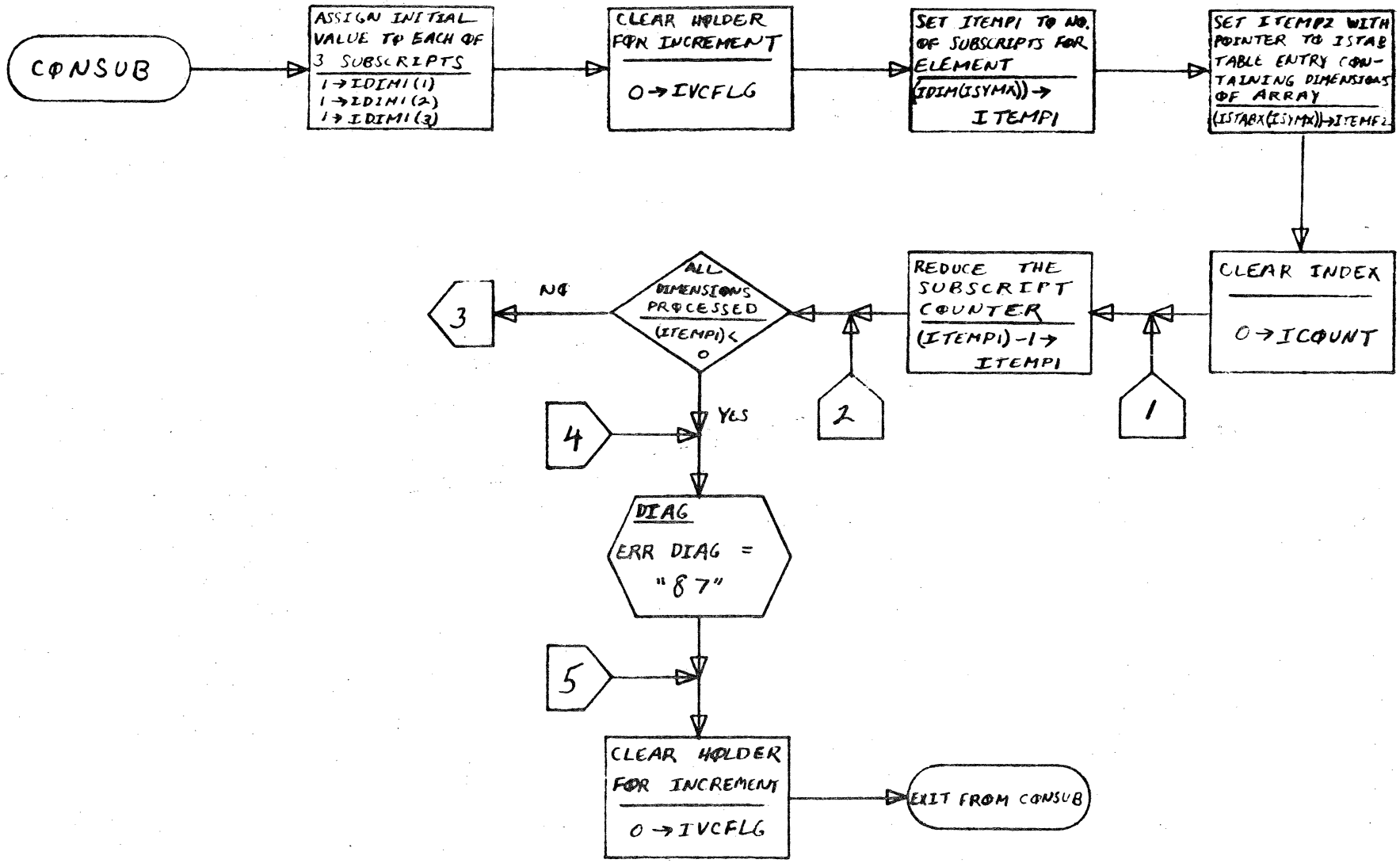
A(I1, J1, K1) = the IVCFLGth element in the array A(I, J, K). The original dimensions of the array A (I, J, K) are obtained from an entry in the ISTAB table.

2.3.3.1.1 Errors

The CONSUB subroutine gives diagnostics for the following errors:

1. An element with no subscripts,
2. an element with more subscripts than the array of which it is a member has dimensions,
3. an element with a subscript which exceeds the value of the corresponding dimension for the array of which it is a member,
4. an element with a subscript which is something other than an integer, and
5. a format error in which the subscripts are not separated by commas and/or not enclosed in parentheses.

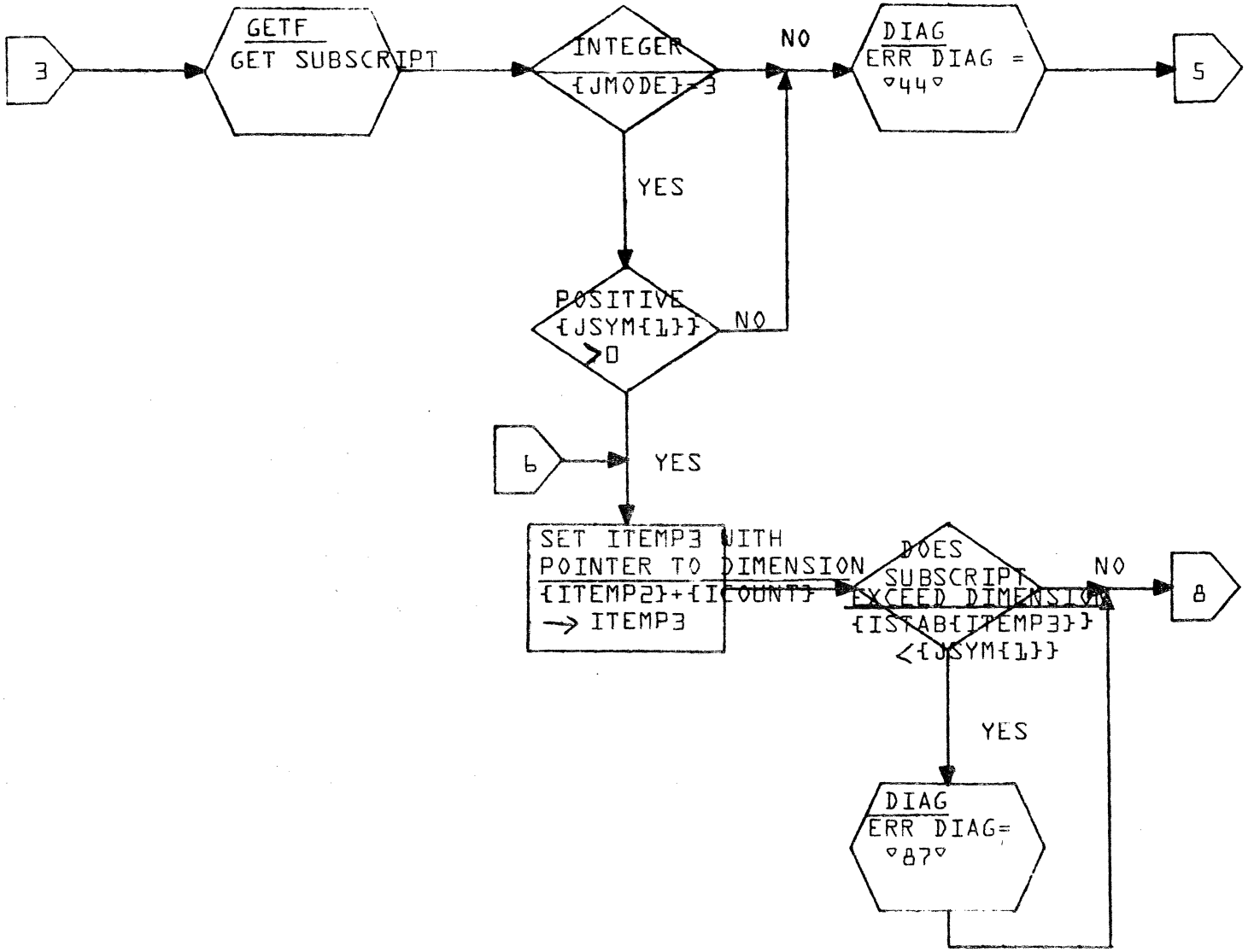
Errors 1, 2, 4 & 5 will cause CONSUB to terminate operation. Following the error indication, the holder IVCFLG is set to zero, and an exit is made from CONSUB. Error 3 will not cause the CONSUB operation to terminate following the error indication.



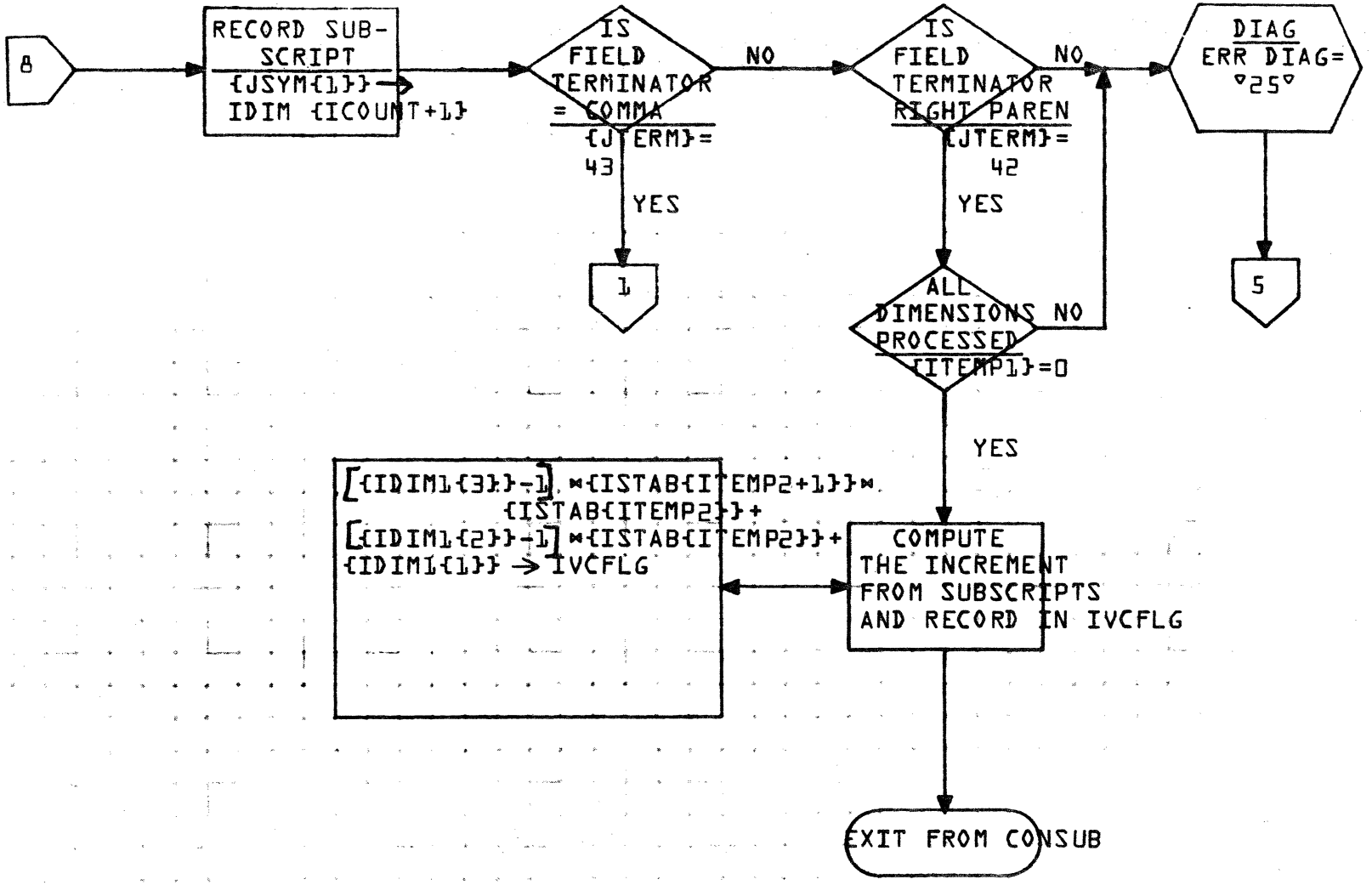
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE CONSUB		PROJECT MGR.		
		PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

2-140

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE CONSUB		PROJECT MGR.			
FLOWCHART ☒		PAGE 2 OF 5	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE CONSUB		PROJECT MGR.			
FLOWCHART ☒		PAGE 3 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

2.3.3.2 CFIVOC - Checks for Integer Variable or Integer Constant

CFIVOC reads the next field in ISORS. If the field is an integer variable or constant, the field is stored in the symbol table (if necessary), the type set to integer and the class set appropriately. IVCFLG is set to signal the type of field

0 = not integer variable or constant
1 = constant
2 = variable

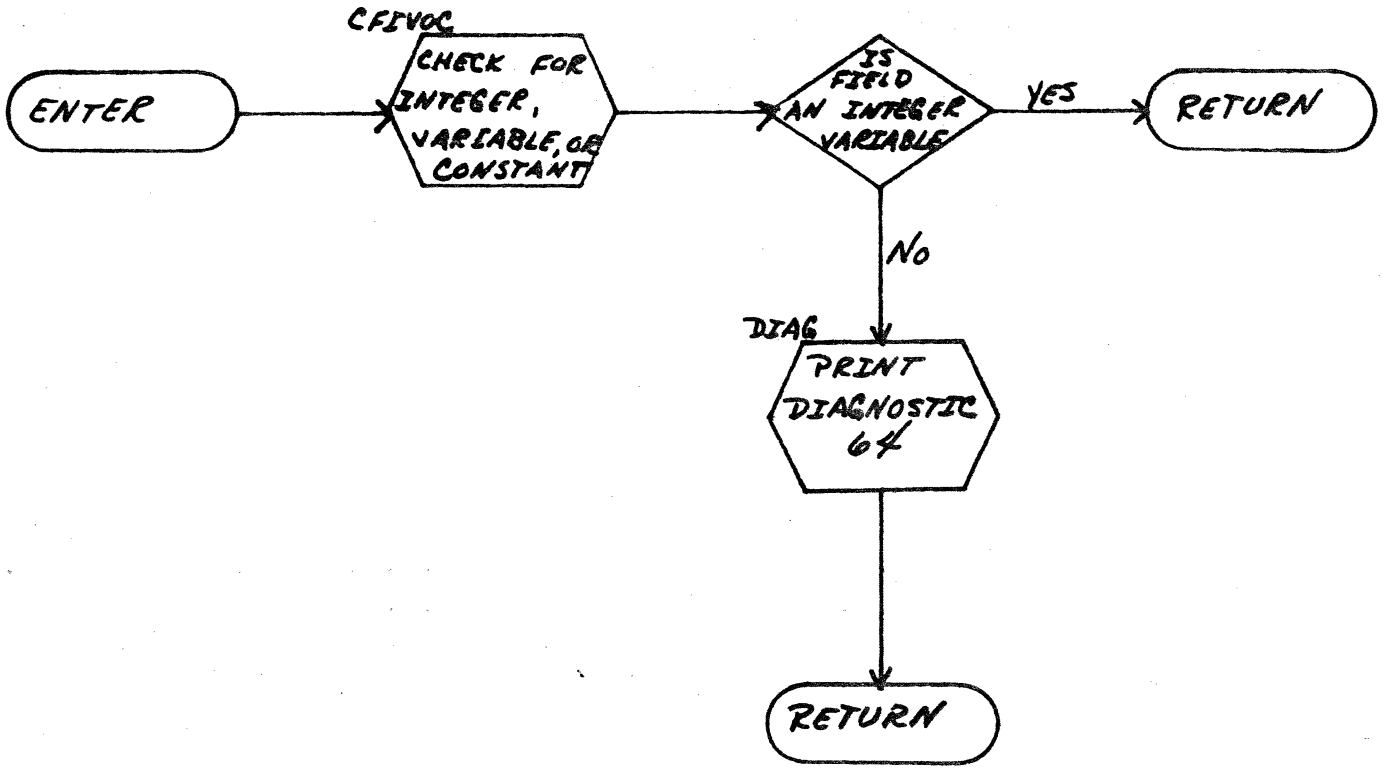
2.3.3.3 CKIVC -Checks for Integer Variable or Constants and Provides Diagnostics

CKIVC calls CFIVOC to read next ISORS field. If the field is neither an integer variable or an integer constant (IVCFLG = 0), diagnostic 63 is output.

2.3.3.4 CKNAME - Checks for Integer Name

CKNAME calls CFIVOC to read the next ISORS field. If the field is not an integer name (IVCFLG = 2), diagnostic 64 is output.

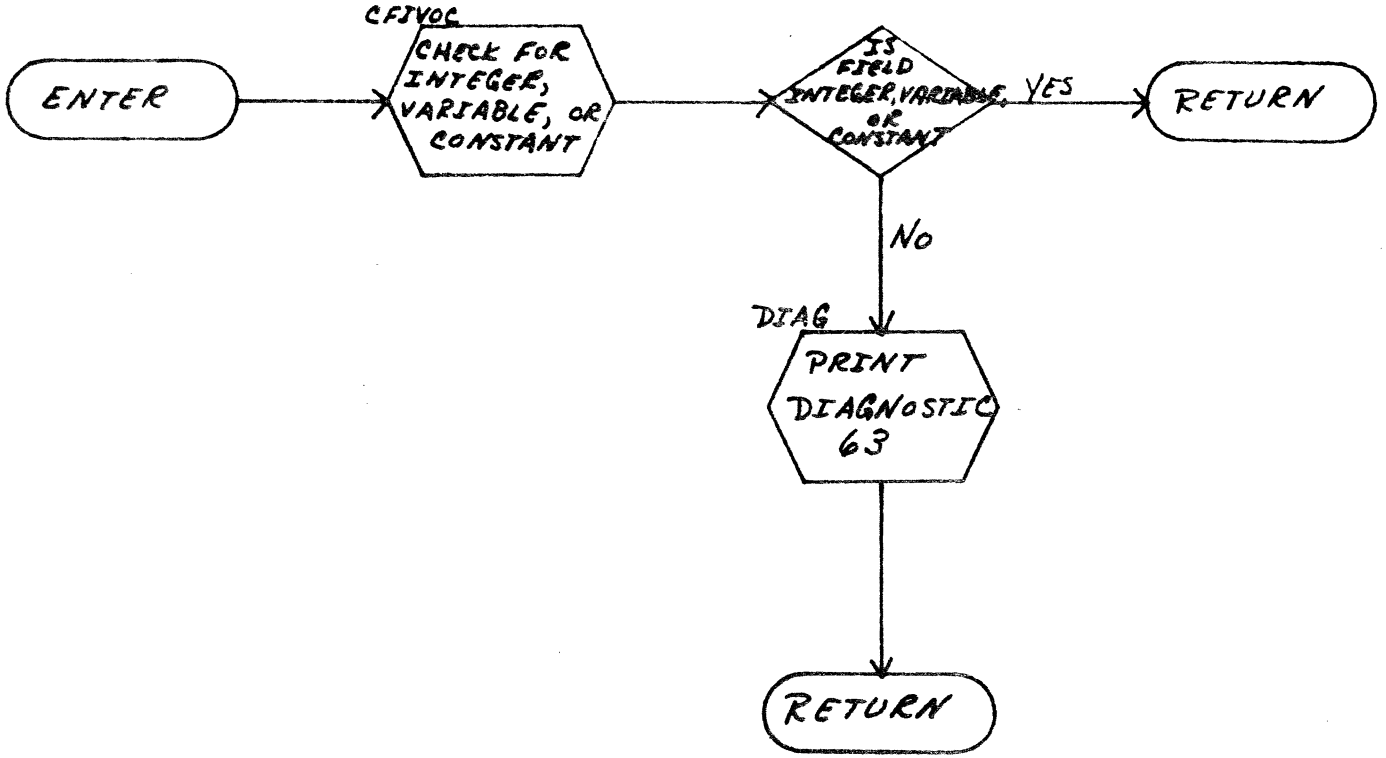
CKNAME



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE CKNAME		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

CA 127-1

CKIVC



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE CKIVC		PROJECT MGR		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

2.3.4 Diagnostic Handling Routine

2.3.4.1 Subroutine DIAG

DIAG is a multiphase subroutine which outputs error diagnostics. The error message output is of the form:

```
*C E,S      xxxxxx
```

Where C is F or N for fatal or non-fatal
E is the error number
S is the offending statement number
xxxxxx are six characters from the source input buffer surrounding the current setting of ISORSX

The input to DIAG is one packed parameter giving the error number in bits 11-0 and flags in bits 14, 13, 12.

The flags are:

Bit 14 0 = print statement number
1 = don't print statement number

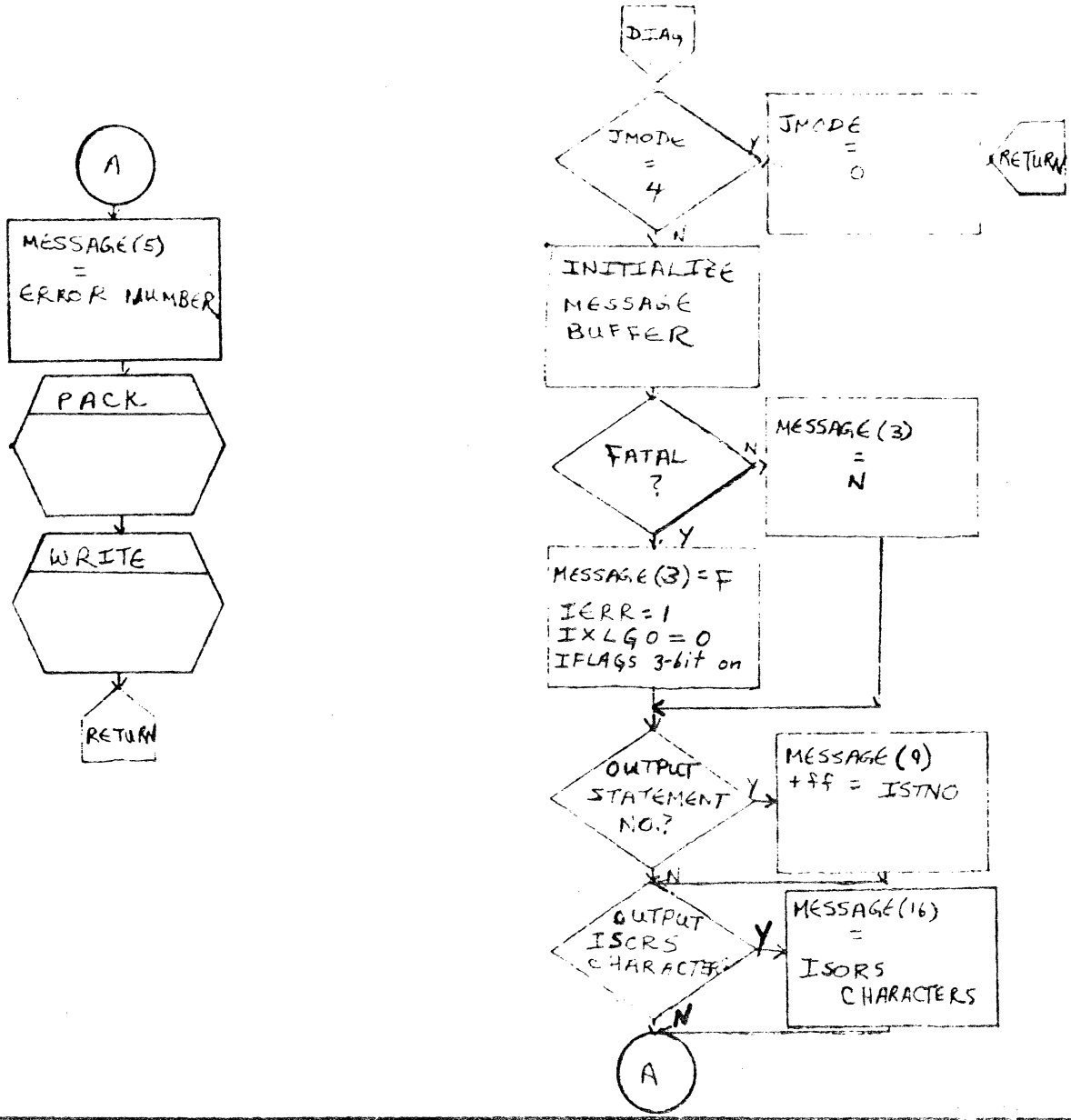
Bit 13 0 = fatal error
1 = non-fatal error

Bit 12 0 = Print ISORS characters
1 = Don't print ISORS characters

If this is a fatal error, IERR is set $\neq 0$.

If DIAG is entered with JMODE = 4 (errpr mode in GETF), no error is output and JMODE is set = 0. This is to minimize the output of redundant errors.

If this is a fatal error, the "X" option flag IXLGO is cleared and bit 3 is set in IFLAGS.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART X DECISION TABLE OTHER	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE DIAG PAGE 1 OF 1 NUMBER ISSUE DATE AUTHOR DATE	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

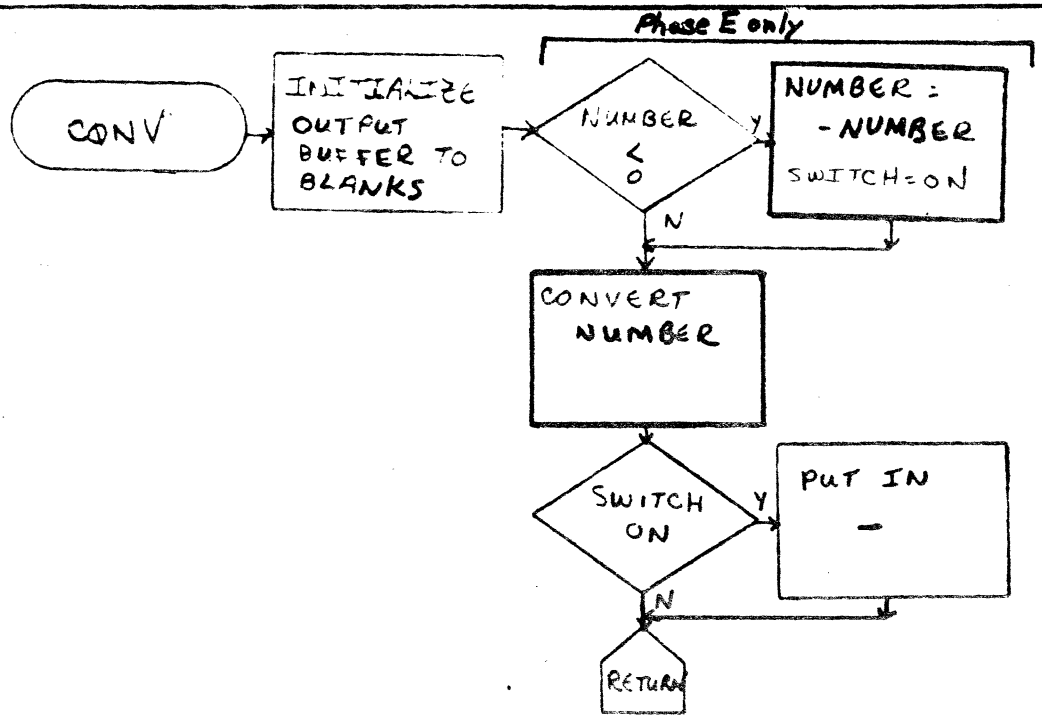
2.3.4.2 Subroutine CONV

CONV converts a binary integer to external code (ASCII) and puts the result in an output buffer, one character per word. Its input parameters are:

IX1 number to convert

IX2 output buffer - six words long

CONV is coded in assembler. The version of CONV used in PHASE A cannot convert negative numbers, while the version used in PHASE E can convert any binary integer.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE CONV		PROJECT MGR		
	PAGE 1 OF 1		PROJECT NAME FORTRAN		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE 3-67	TASK NAME		

2.3.4.3 Subroutine PACK

PACK packs a buffer which comes to it one character per word into the form of two characters per word.

Input parameters:

IX1 Location of buffer

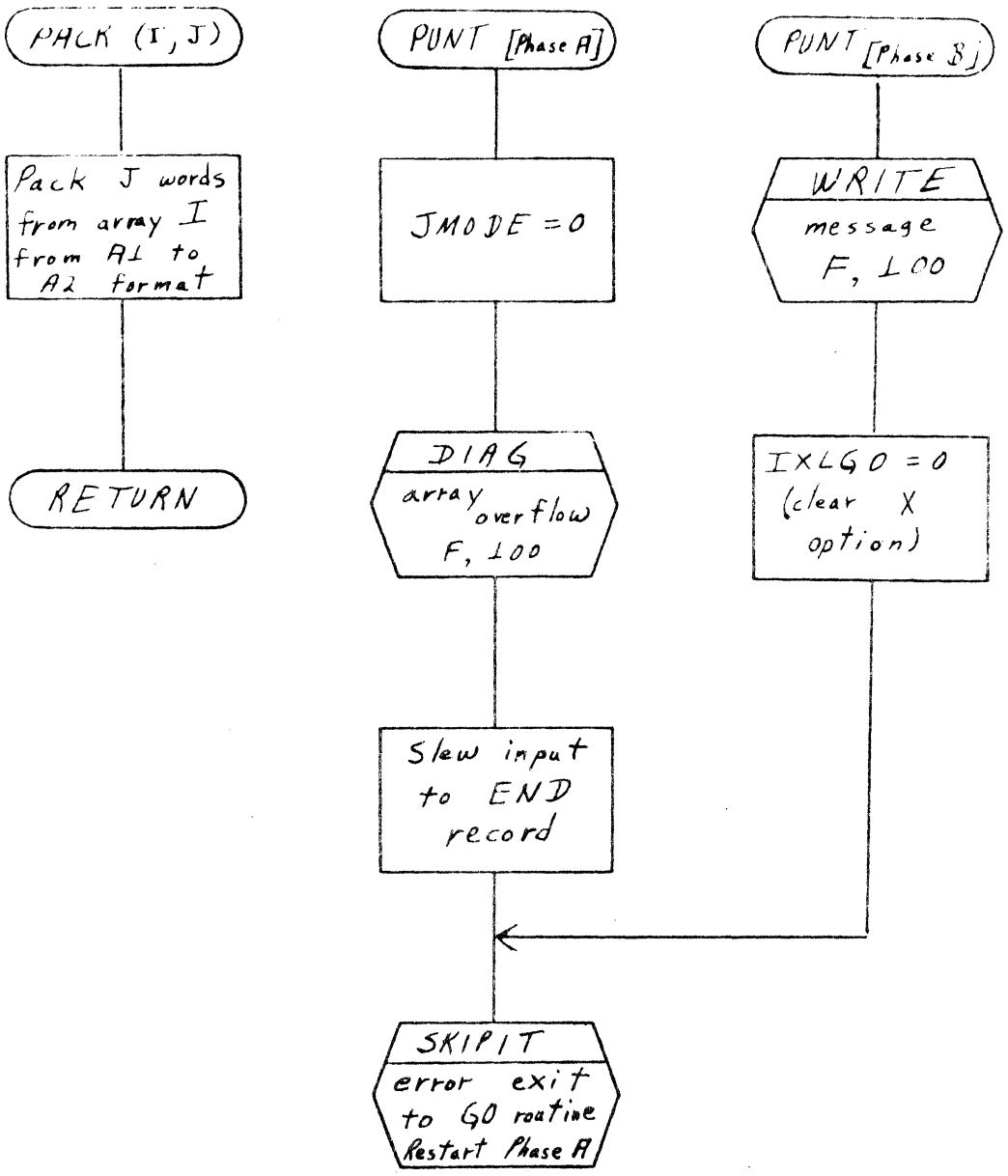
IX2 number of words to pack

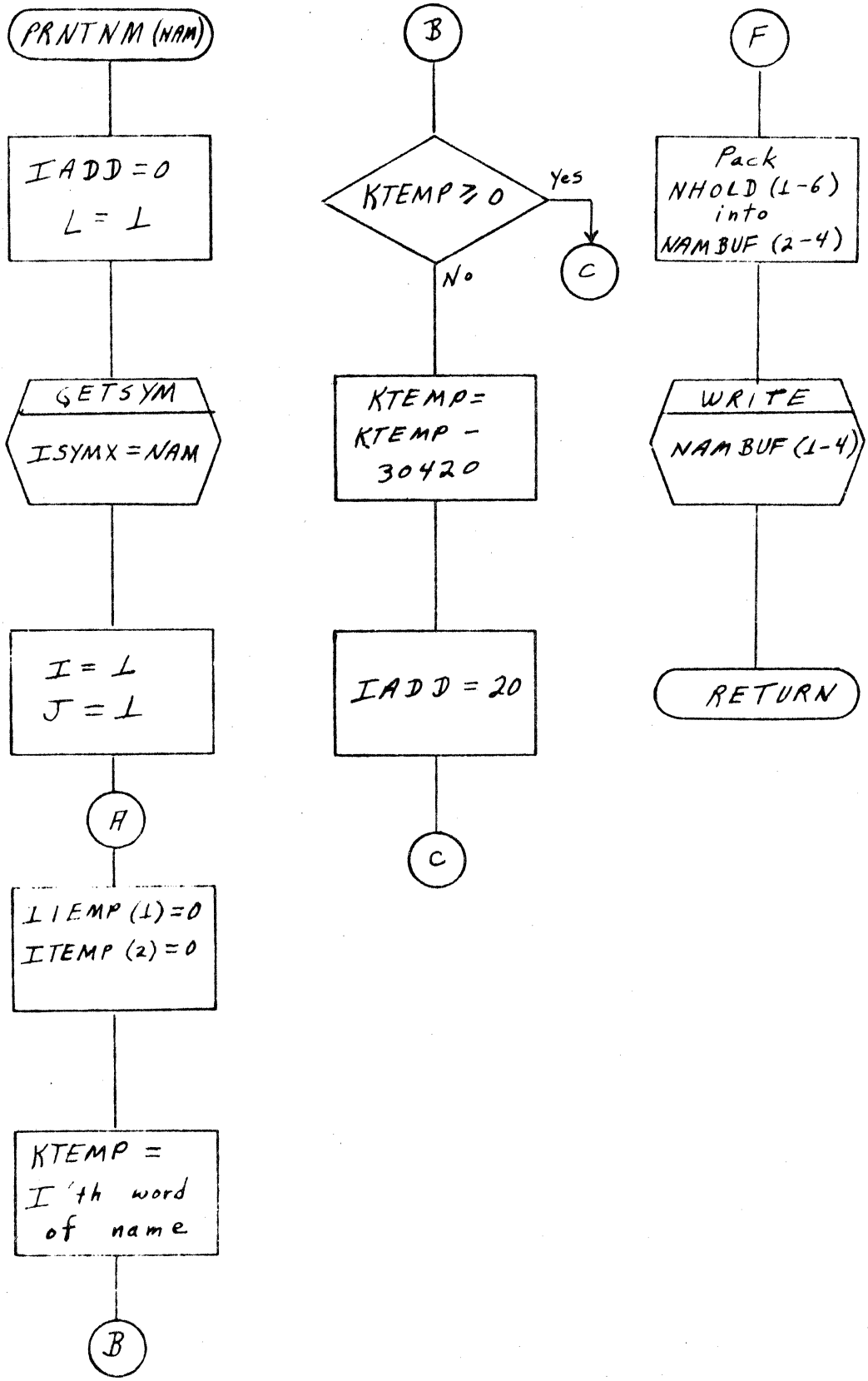
2.3.4.4 Subroutine PUNT

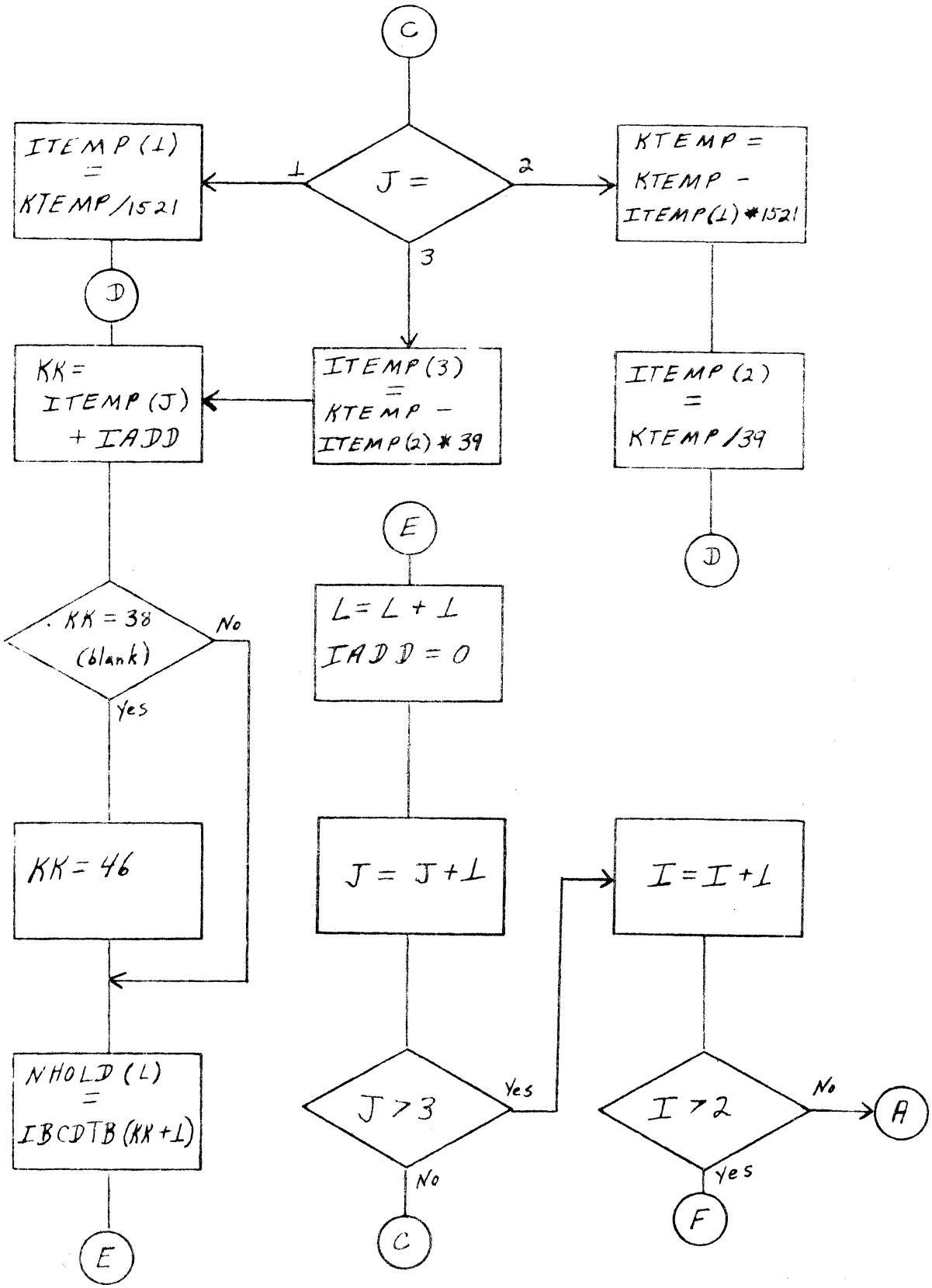
PUNT is called when a catastrophic table overflow occurs in the compiler. It outputs error diagnostic 100 and calls SKIPIT. The Phase A version slews the input to an END card before calling SKIPIT.

2.3.4.5 Subroutine PRNTNM

PRNTNM is called by subroutine PEQVS when an illegal equivalence chain is encountered. PRNTNM converts the name of the illegal variable to external code (ASCII) and outputs it through the use of subroutine IOPR (entry point WRITE).


PACK (I, J)
Pack J words from array I from A1 to A2 format
RETURN
PUNT [Phase A]
JMODE = 0
DIAG
array overflow
F, 100
Slew input to END record
PUNT [Phase B]
WRITE
message
F, 100
IXLGO = 0
(clear X option)
SKIPIT
error exit to GO routine
Restart Phase A


PRNTNM (NAM)
IADD = 0
L = 1
GETSYM
ISYMX = NAM
I = 1
J = 1
A
ITEMP (1) = 0
ITEMP (2) = 0
KTEMP = I 'th word of name
B
B
KTEMP ≠ 0
Yes
No
C
KTEMP = KTEMP - 30420
IADD = 20
C
F
Pack NHOLD (1-6) into NAMBUF (2-4)
WRITE
NAMBUF (1-4)
RETURN


C
J =
1
2
3
ITEMP(1) = KTEMP/1521
KTEMP = KTEMP - ITEMP(1)*1521
D
KK = ITEMP(J) + IADD
ITEMP(3) = KTEMP - ITEMP(2)*39
ITEMP(2) = KTEMP/39
D
E
KK = 38 (blank)
No
Yes
L = L + 1
IADD = 0
KK = 46
J = J + 1
I = I + 1
NHOLD(L) = IBCDTB(KK+1)
J > 3
Yes
No
I > 2
No
Yes
A
E
C
F

2.3.4.6 Subroutine IOPR

This assembly-language routine performs all I/O for the compiler. In order to optimize core and disk use, there is a separate version of IOPR for each phase of each {2.0A and 2.0B} compiler; but all versions share the following functional description.

Entry points

IOPR is the entry called by GO at the start of each phase. It allocates the disc scratch area, initializes pointers, and begins input.

READ and WRITE perform both unit record and mass storage I/O. Unit record I/O in phase A is buffered; records on the scratch files are packed by WRITE and unpacked by READ.

RESET is available in phase D/E; it performs a logical "rewind" of a scratch file.

EXIT issues an EXIT request to MSOS.

Disc allocation

The speed of the compiler is highly dependent on the placement of its files within the MSOS-allotted disk scratch area. This area is allocated starting at the first sector of the first cylinder beyond the current top of the load-and-go file; seek operations are minimized by never allowing a file to "break" between two cylinders.

The first portion of this area is allocated for local files. (Even though the present phase may not have locals, this area must be stepped over in order to find the symbol table and scratch files.) The partitioning of this area is "customized" for each phase, generally leaving each file several sectors more than it requires to allow for later expansion.

Following the local files, an area is assigned to the symbol table. Since the table is organized in eleven-sector "pages", this area starts 11n sectors from the bottom of a cylinder and overlaps onto the next cylinder. The disk allocation is such that only programs with a very large number of symbols will use any of the SYMTAB pages on the next cylinder, and so that the most-used local files will share the "prime" SYMTAB cylinder.

After the symbol table, the remainder of the scratch area is available for the two scratch files. These files, like the symbol table, are organized in "pages" of several sectors; in order to minimize length of seeks, pages of file 1 are interleaved with pages of file 2. No effort is made to keep scratch pages from crossing cylinder boundaries, since the "broken" page would be referred to only once per phase.

Scratch file packing

To minimize disk usage, the two scratch files are packed into streams of logical records without regard for sector boundaries. (The first word of each logical record contains the total record length.) The files are then broken up into "pages" of several sectors; a "page" being as much of the file as will fit into core. (In several cases, a phase keeps only part of a page in core at a time; the page size is chosen for maximum efficiency over all four phases of a compiler version.) A call to READ causes copying of a logical record from the unpacking area to the user-designated address; if the end of the unpacking area is reached before the entire record is moved, then the next page of the input file is read into the unpacking area and the copying proceeds. A call to WRITE similarly loads the output packing area, which is similarly emptied when full. (A special call is provided to force the output onto disk at the end of each phase.)

It is important to note that on scratch-file I/O, the logical record length is controlled only by word 1 of the record; The record-length parameter of the call to READ or WRITE is ignored.

Unit record buffering

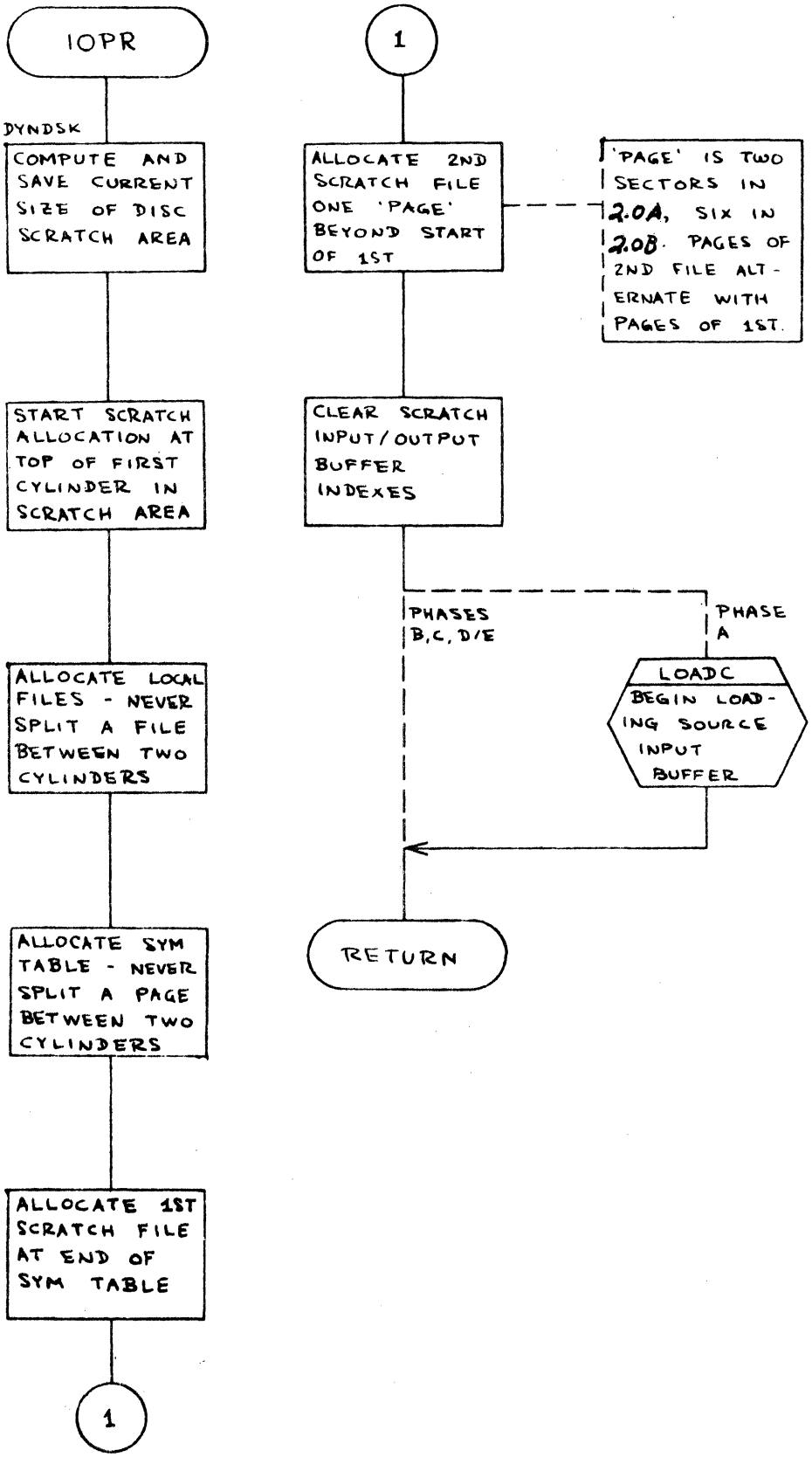
Phase A unit record I/O...source statement input and listing... is buffered, so that phase A execution time is overlapped as much as possible. Space is provided in the phase A IOPR to hold one line of input and one or two lines of output; each line is associated with an FREAD or FWRITE request. When the IOPR entry is called, at the start of a phase, it activates the FREAD request that is to fill the input buffer. Thereafter, a call to READ will: (1) wait until the request is unthreaded; (2) copy the record just read, from the buffer

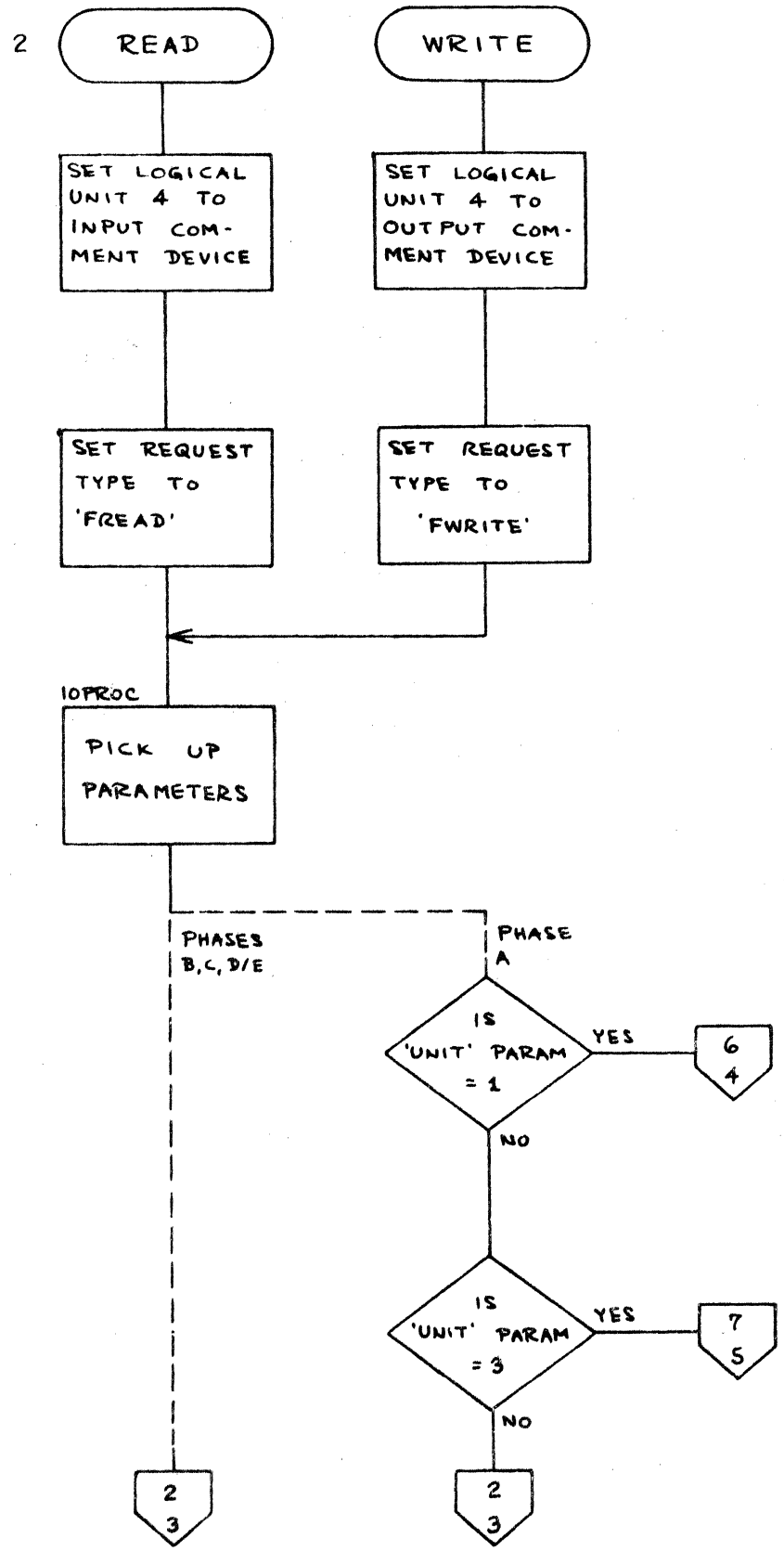
to the user's area; (3) reissue the FREAD request. Similarly, a call to WRITE waits for a buffer to be available, fills it, issues an FWRITE request for it, and returns.

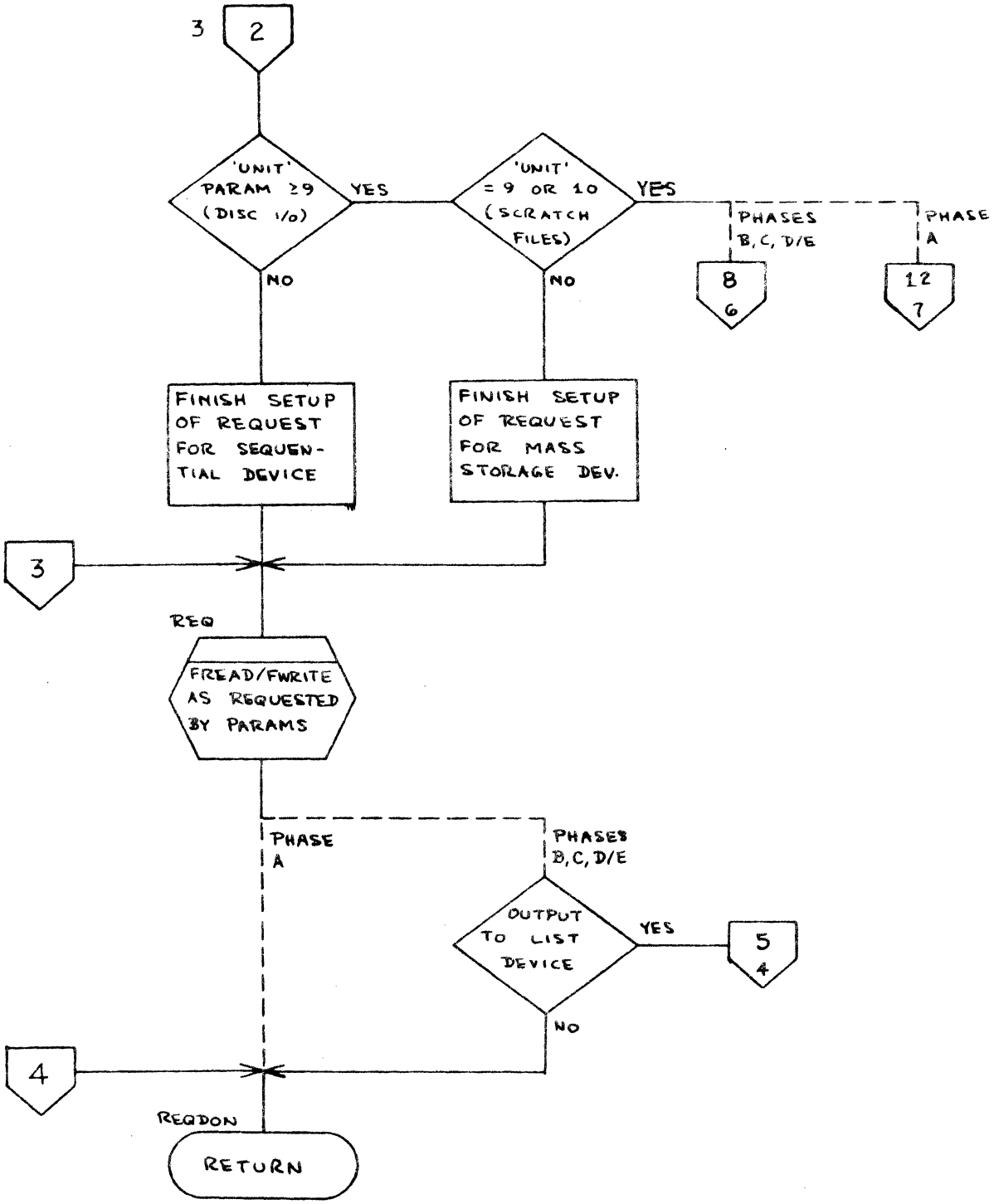
The READ routine does not reissue the FREAD request if the buffer contains an END or MON card. The buffer is blanked before the request is issued. The requested record length is 41 words so that, regardless of actual record length, the operating system will store the actual last-word-address-plus-one outside the area to be processed by Fortran.

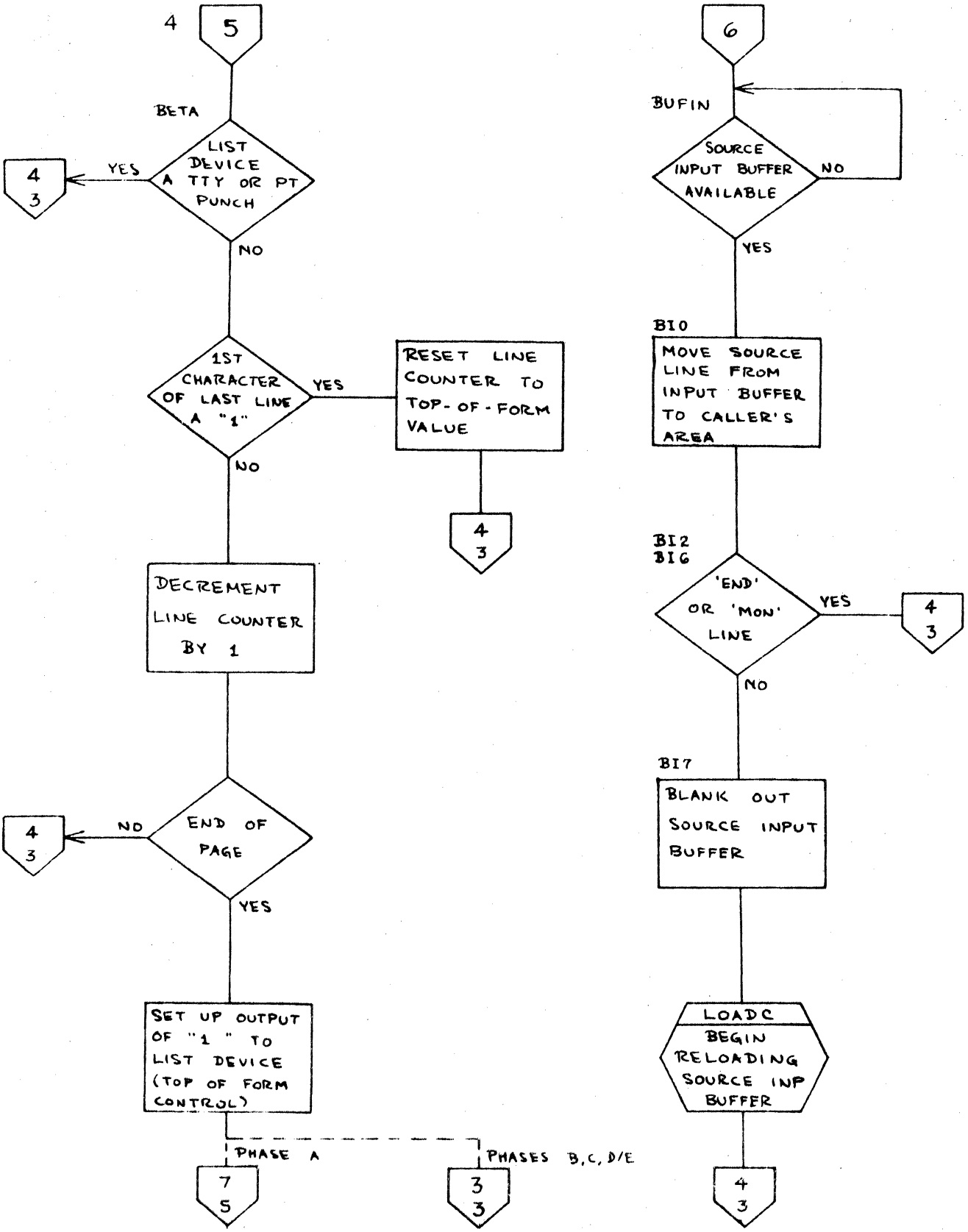
Page ejection

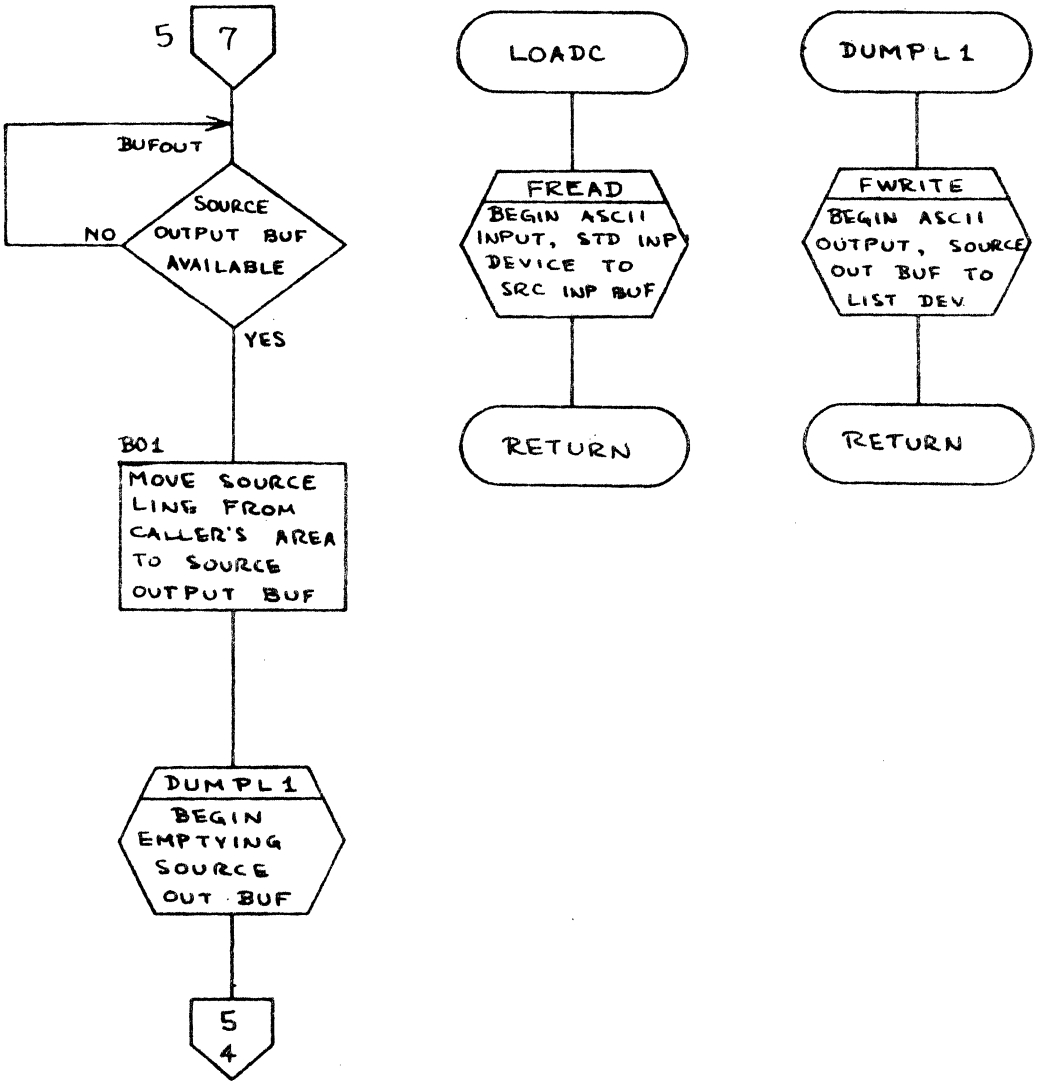
If the list output device is not a teletype or paper tape punch, IOPR issues a page-eject control ($3120) after each 60 lines of output. The maximum and current number of lines per page are held in labelled common and are set by the phase A block data subprogram and subroutine PHASEA, respectively. PHASEA is responsible for issuing page ejects before the source and assembly listings.

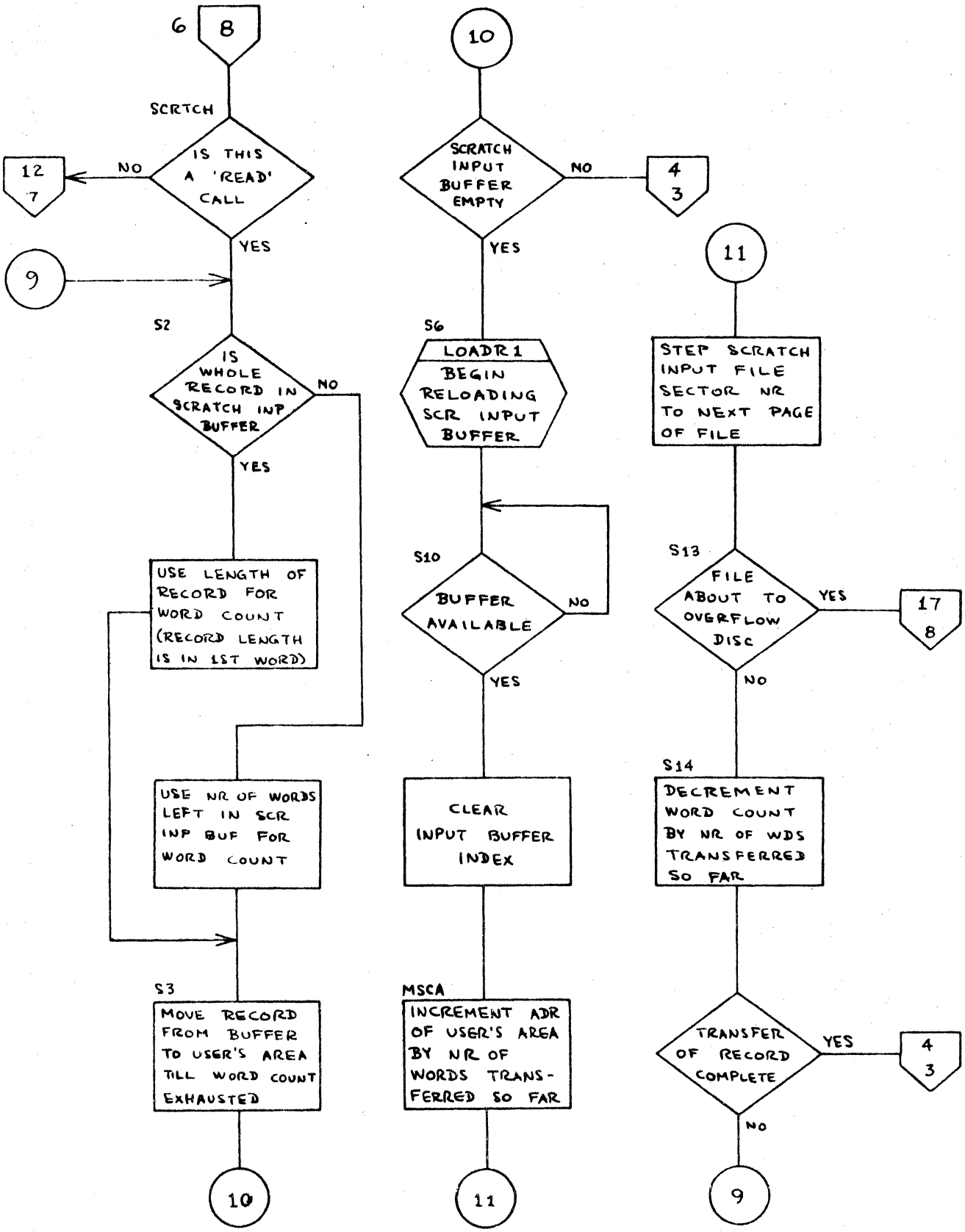

IOPR
DYNDSK
COMPUTE AND SAVE CURRENT SIZE OF DISC SCRATCH AREA
START SCRATCH ALLOCATION AT TOP OF FIRST CYLINDER IN SCRATCH AREA
ALLOCATE LOCAL FILES - NEVER SPLIT A FILE BETWEEN TWO CYLINDERS
ALLOCATE SYM TABLE - NEVER SPLIT A PAGE BETWEEN TWO CYLINDERS
ALLOCATE 1ST SCRATCH FILE AT END OF SYM TABLE
1
1
ALLOCATE 2ND SCRATCH FILE ONE 'PAGE' BEYOND START OF 1ST
'PAGE' IS TWO SECTORS IN 2.0A, SIX IN 2.0B. PAGES OF 2ND FILE ALTERNATE WITH PAGES OF 1ST.
CLEAR SCRATCH INPUT/OUTPUT BUFFER INDEXES
PHASES B,C,D/E
PHASE A
LOADC
BEGIN LOADING SOURCE INPUT BUFFER
RETURN

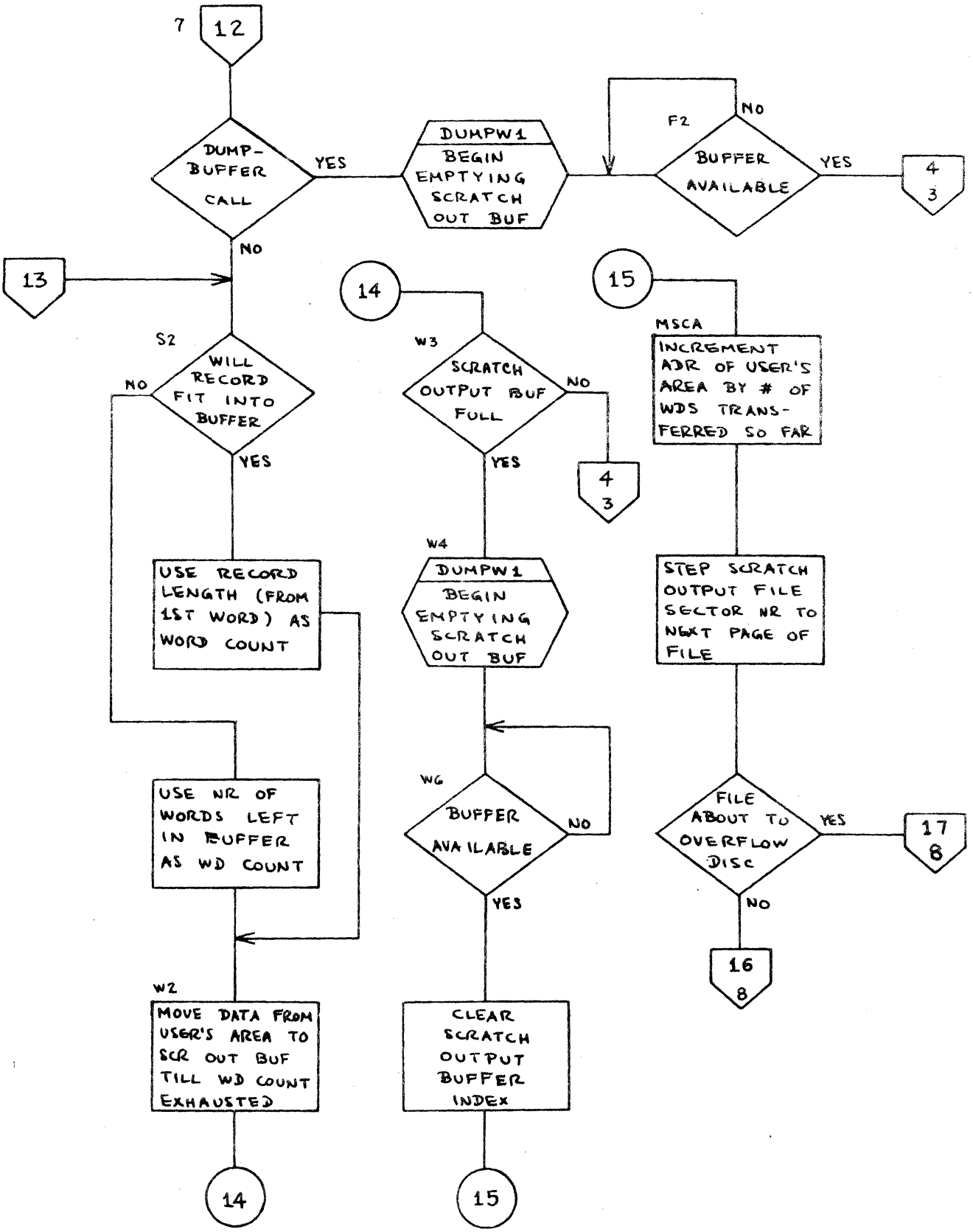

2
READ
WRITE
SET LOGICAL UNIT 4 TO INPUT COM-MENT DEVICE
SET LOGICAL UNIT 4 TO OUTPUT COM-MENT DEVICE
SET REQUEST TYPE TO 'FREAD'
SET REQUEST TYPE TO 'FWRITE'
IOPROC
PICK UP PARAMETERS
PHASES B,C,D/E
PHASE A
IS 'UNIT' PARAM = 1
YES
6 4
NO
IS 'UNIT' PARAM = 3
YES
7 5
NO
2 3
2 3

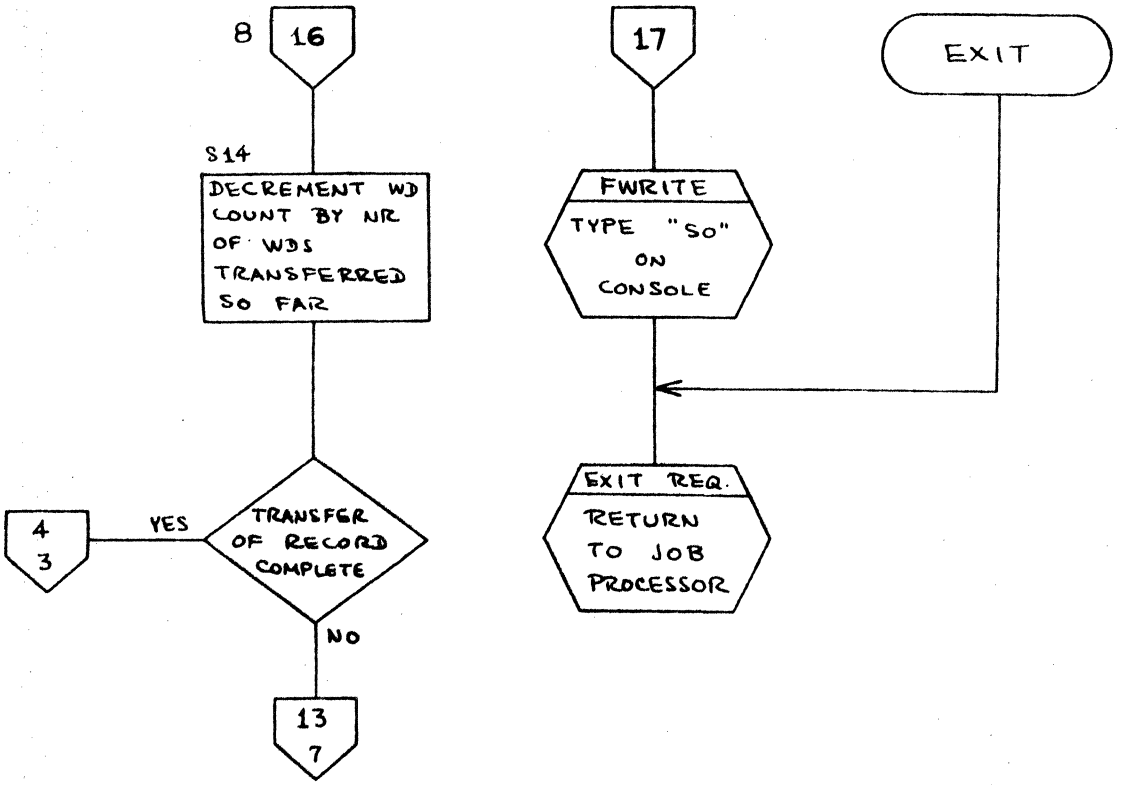

3
2
'UNIT' PARAM ≥9 (DISC I/O)
YES
'UNIT' = 9 OR 10 (SCRATCH FILES)
YES
PHASES B, C, D/E
8
6
PHASE A
12
7
NO
NO
FINISH SETUP OF REQUEST FOR SEQUENTIAL DEVICE
FINISH SETUP OF REQUEST FOR MASS STORAGE DEV.
3
REQ
FREAD/FWRITE AS REQUESTED BY PARAMS
PHASE A
PHASES B, C, D/E
OUTPUT TO LIST DEVICE
YES
5
4
NO
4
REQDON
RETURN

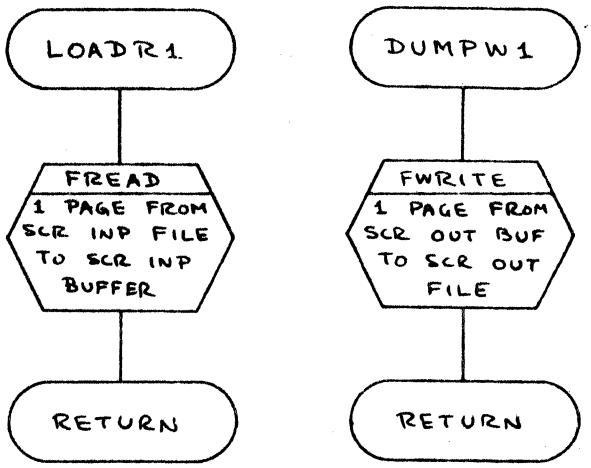

4
5
BETA
LIST DEVICE A TTY OR PT PUNCH
YES
4 3
NO
1ST CHARACTER OF LAST LINE A "1"
YES
RESET LINE COUNTER TO TOP-OF-FORM VALUE
4 3
NO
DECREMENT LINE COUNTER BY 1
END OF PAGE
NO
4 3
YES
SET UP OUTPUT OF "1" TO LIST DEVICE (TOP OF FORM CONTROL)
PHASE A
PHASES B,C,D/E
7 5
3 3
6
BUFIN
SOURCE INPUT BUFFER AVAILABLE
NO
YES
BI0
MOVE SOURCE LINE FROM INPUT BUFFER TO CALLER'S AREA
BI2
BI6
'END' OR 'MON' LINE
YES
4 3
NO
BI7
BLANK OUT SOURCE INPUT BUFFER
LOADC
BEGIN RELOADING SOURCE INP BUFFER
4 3

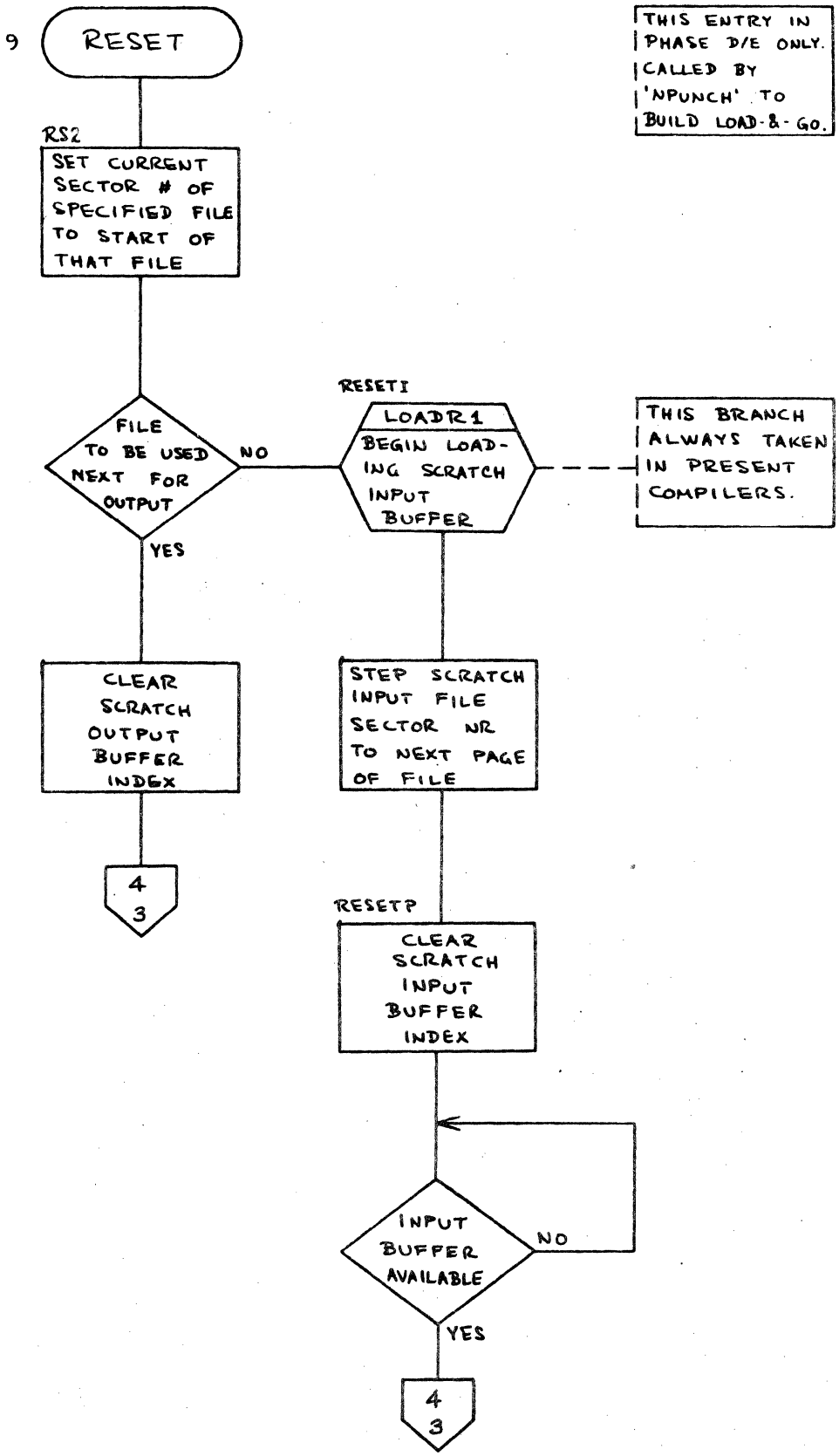

5 7
BUFOUT
SOURCE OUTPUT BUF AVAILABLE
NO
YES
B01
MOVE SOURCE LING FROM CALLER'S AREA TO SOURCE OUTPUT BUF
DUMPL1
BEGIN EMPTYING SOURCE OUT BUF
5 4
LOADC
FREAD
BEGIN ASCII INPUT, STD INP DEVICE TO SRC INP BUF
RETURN
DUMPL1
FWRITE
BEGIN ASCII OUTPUT, SOURCE OUT BUF TO LIST DEV
RETURN


6
8
SCRTCH
IS THIS A 'READ' CALL
NO
12
7
YES
9
S2
IS WHOLE RECORD IN SCRATCH INP BUFFER
NO
YES
USE LENGTH OF RECORD FOR WORD COUNT (RECORD LENGTH IS IN 1ST WORD)
USE NR OF WORDS LEFT IN SCR INP BUF FOR WORD COUNT
S3
MOVE RECORD FROM BUFFER TO USER'S AREA TILL WORD COUNT EXHAUSTED
10
10
SCRATCH INPUT BUFFER EMPTY
NO
4
3
YES
S6
LOADR1
BEGIN RELOADING SCR INPUT BUFFER
S10
BUFFER AVAILABLE
NO
YES
CLEAR INPUT BUFFER INDEX
MSCA
INCREMENT ADR OF USER'S AREA BY NR OF WORDS TRANSFERRED SO FAR
11
11
STEP SCRATCH INPUT FILE SECTOR NR TO NEXT PAGE OF FILE
S13
FILE ABOUT TO OVERFLOW DISC
YES
17
8
NO
S14
DECREMENT WORD COUNT BY NR OF WDS TRANSFERRED SO FAR
TRANSFER OF RECORD COMPLETE
YES
4
3
NO
9


7
12
DUMP-BUFFER CALL
YES
DUMPW1
BEGIN EMPTYING SCRATCH OUT BUF
F2
NO
BUFFER AVAILABLE
YES
4
3
NO
13
14
15
S2
WILL RECORD FIT INTO BUFFER
NO
YES
W3
SCRATCH OUTPUT BUF FULL
NO
YES
4
3
MSCA
INCREMENT ADR OF USER'S AREA BY # OF WDS TRANSFERRED SO FAR
USE RECORD LENGTH (FROM 1ST WORD) AS WORD COUNT
W4
DUMPW1
BEGIN EMPTYING SCRATCH OUT BUF
STEP SCRATCH OUTPUT FILE SECTOR NR TO NEXT PAGE OF FILE
USE NR OF WORDS LEFT IN BUFFER AS WD COUNT
W6
BUFFER AVAILABLE
NO
YES
FILE ABOUT TO OVERFLOW DISC
YES
17
8
NO
16
8
W2
MOVE DATA FROM USER'S AREA TO SCR OUT BUF TILL WD COUNT EXHAUSTED
CLEAR SCRATCH OUTPUT BUFFER INDEX
14
15


8
16
S14
DECREMENT WD COUNT BY NR OF WDS TRANSFERRED SO FAR
TRANSFER OF RECORD COMPLETE
YES
4
3
NO
13
7
17
FWRITE
TYPE "SO" ON CONSOLE
EXIT
EXIT REQ.
RETURN TO JOB PROCESSOR


LOADR1.
FREAD
1 PAGE FROM SCR INP FILE TO SCR INP BUFFER
RETURN
DUMPW1
FWRITE
1 PAGE FROM SCR OUT BUF TO SCR OUT FILE
RETURN


9
RESET
THIS ENTRY IN PHASE D/E ONLY. CALLED BY 'NPUNCH' TO BUILD LOAD-&-GO.
RS2
SET CURRENT SECTOR # OF SPECIFIED FILE TO START OF THAT FILE
FILE TO BE USED NEXT FOR OUTPUT
NO
YES
RESETI
LOADR1
BEGIN LOADING SCRATCH INPUT BUFFER
THIS BRANCH ALWAYS TAKEN IN PRESENT COMPILERS.
CLEAR SCRATCH OUTPUT BUFFER INDEX
STEP SCRATCH INPUT FILE SECTOR NR TO NEXT PAGE OF FILE
4
3
RESETP
CLEAR SCRATCH INPUT BUFFER INDEX
INPUT BUFFER AVAILABLE
NO
YES
4
3

2.4 STATEMENT SPECIFIC PHASEA ROUTINES

2.4.1 DIMENSION Statements - Subroutine DIMPR

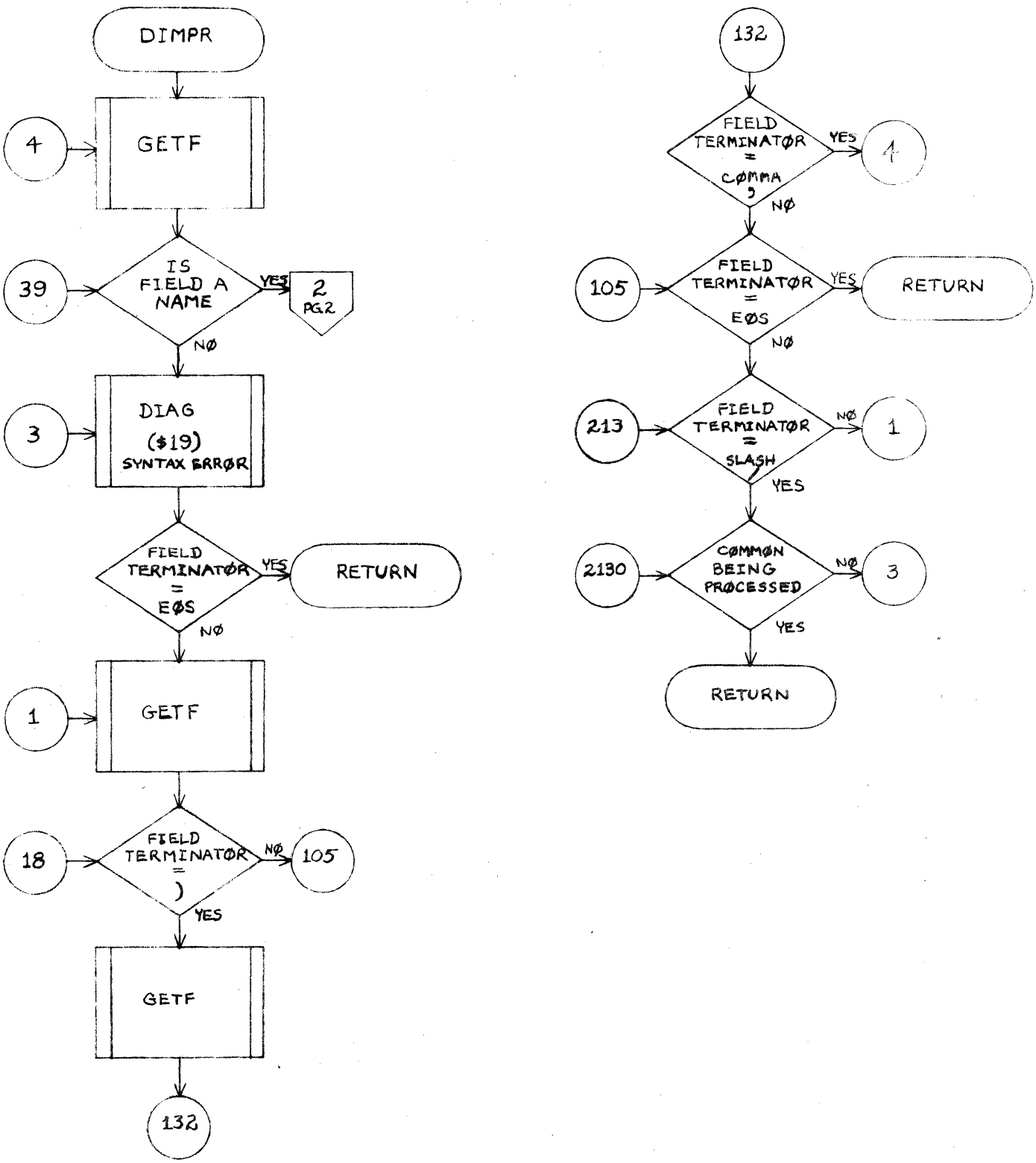
Subroutine DIMPR processes dimension information. It will be called directly by PHASE A to process DIMENSION statements or as an auxiliary routine when processing COMMON or Type statements.

Input to DIMPR is the ISORS image of the statement being processed and the current setting of ISORSX within this buffer.

DIMPR builds entries in the symbol table (SYMTAB), defining the arrays it processes. Entries are set to dimensioned arrays (ICLASS = 1, IDIM = no. dimensions, ISTABX = index into ISTAB to get dimensions). The dimensions are entered in the ISTAB table. If a COMMON statement is being processed, ICOM (common block indicator) and ICOMTX (thread of variables and arrays in common block) are set in the symbol table and ICOMBX in the common table (ICOMT) is updated to reflect the current thread of the chain of variables and arrays in a specific common block.

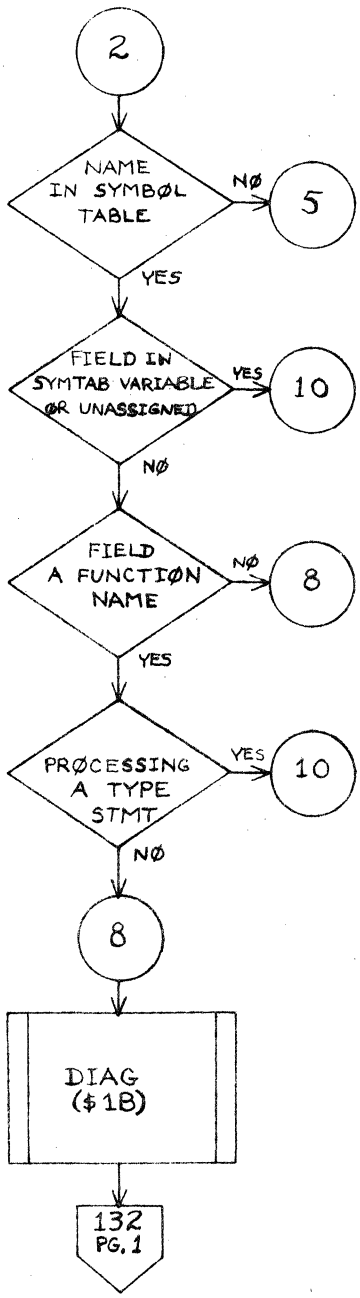
If a Type statement is being processed, the symbol table entry is set to the appropriate type (ITYPE = 1 for INTEGER or SINGLE, 2 for REAL, 3 for DOUBLE, ISNGL = 1 for SINGLE).

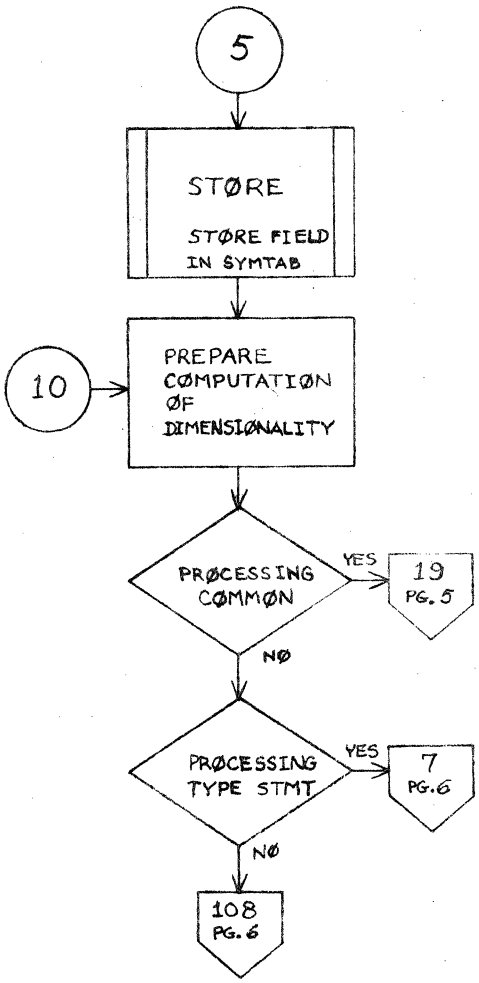
Throughout its processing, DIMPR checks the syntax of the statement it is analyzing and outputs diagnostics if errors are encountered. It also checks for conflicting information, and outputs diagnostics if such information is encountered.



TITLE	DIMPR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 1 OF 7

CSD 203



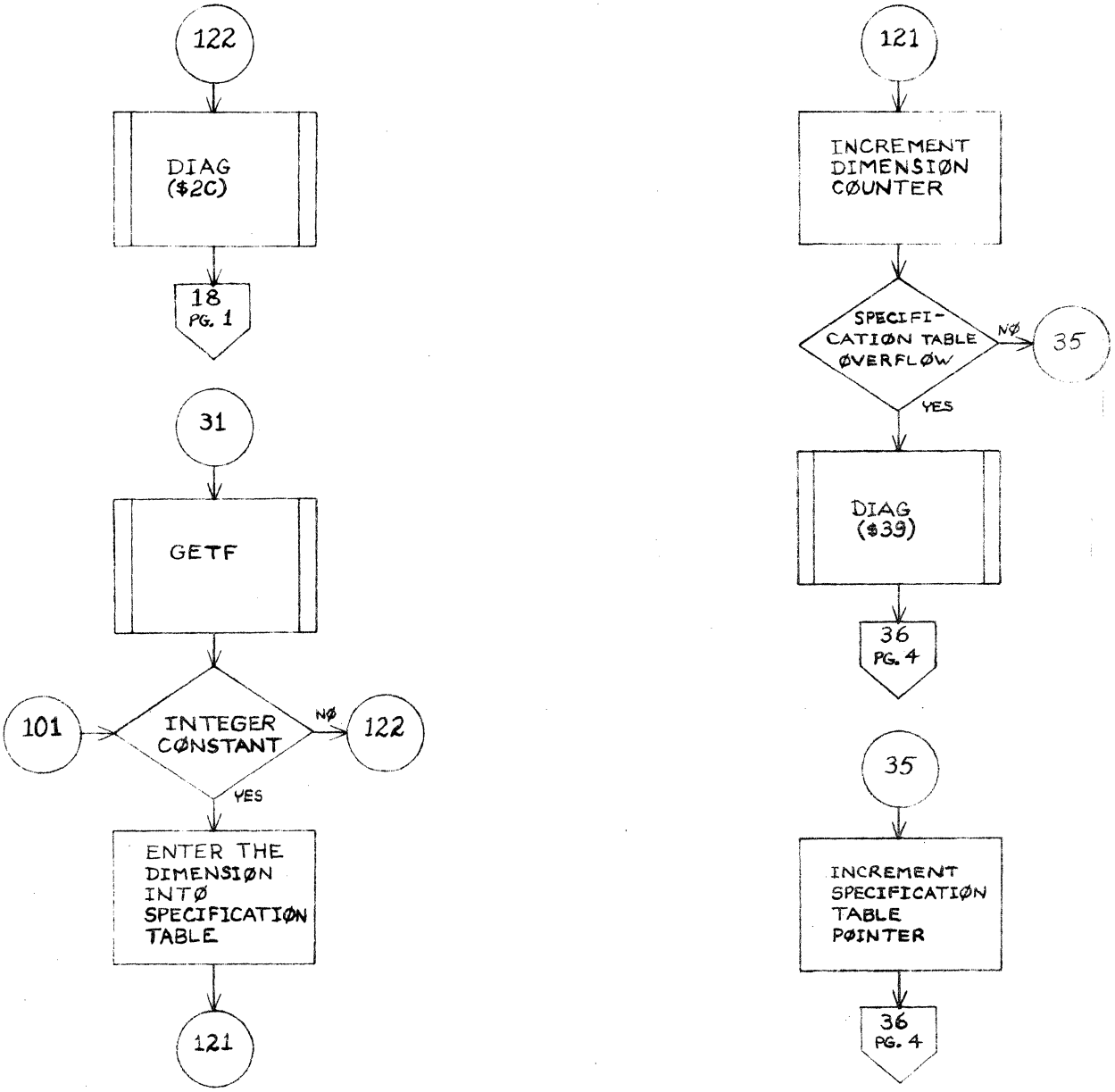


TITLE DIMPR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2 OF 7

CSD 203

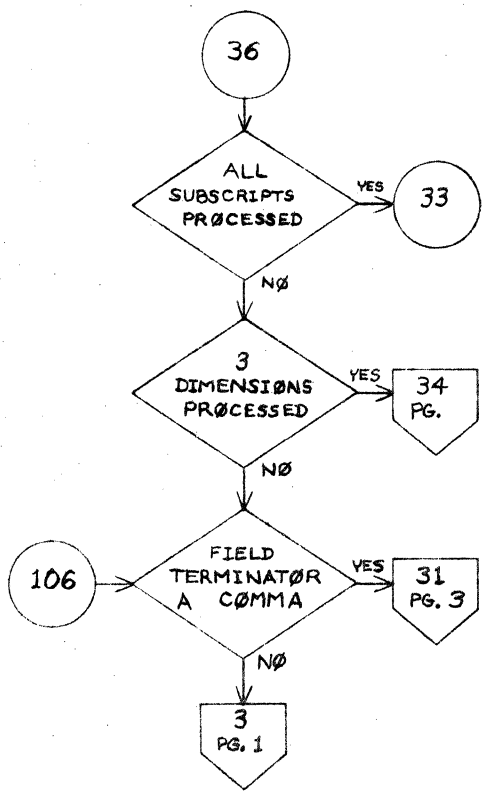
CONTROL DATA CORPORATION

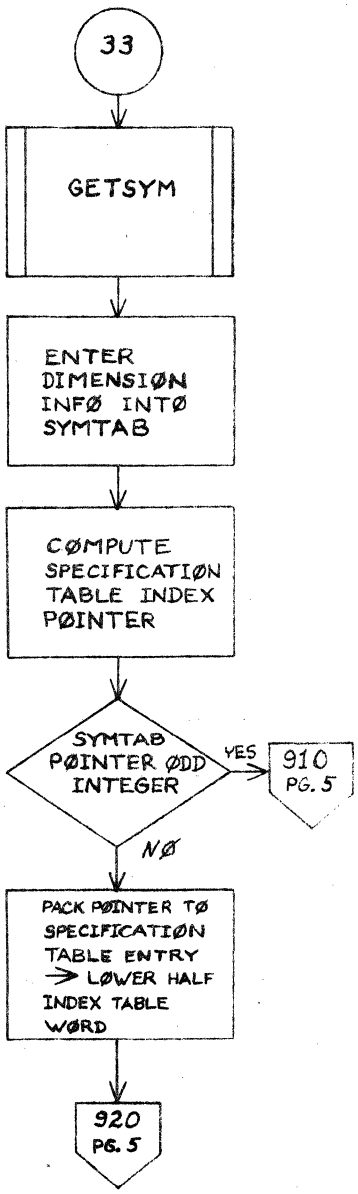
LA JOLLA FACILITY



TITLE DIMPR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF 7

CSD 203



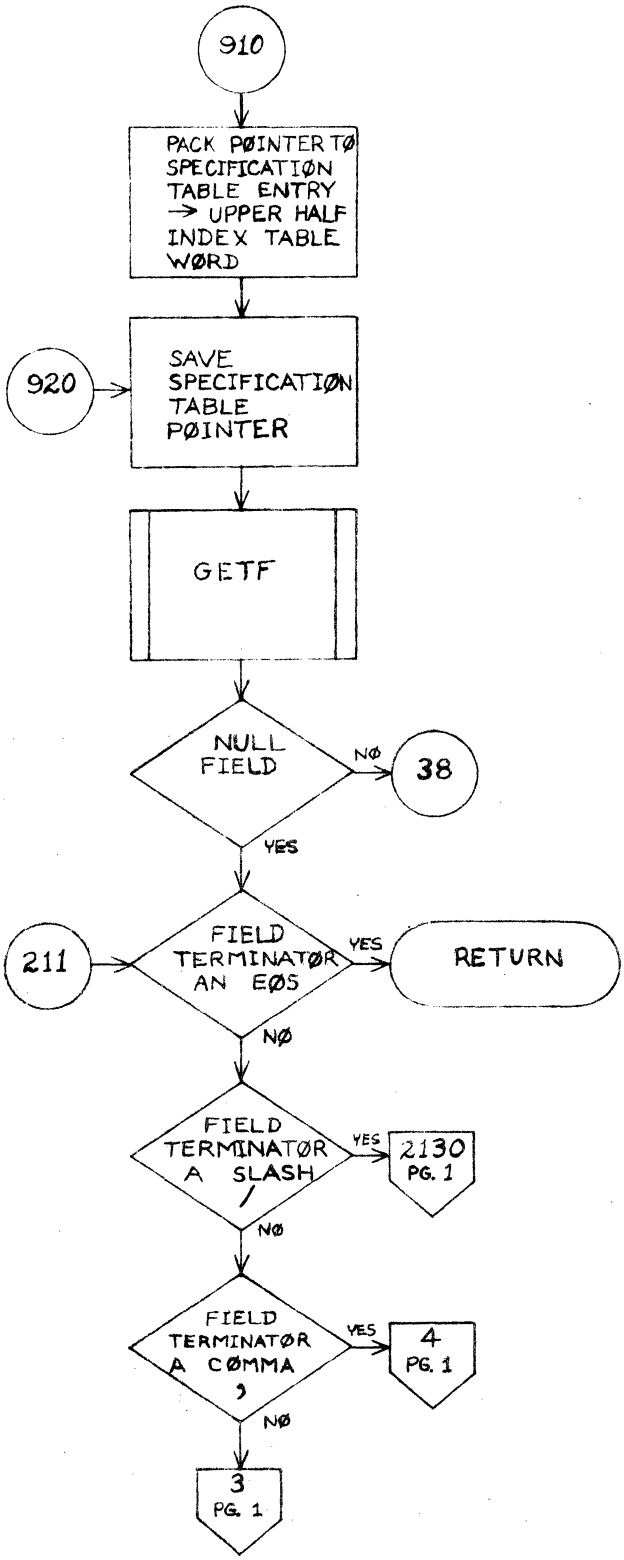


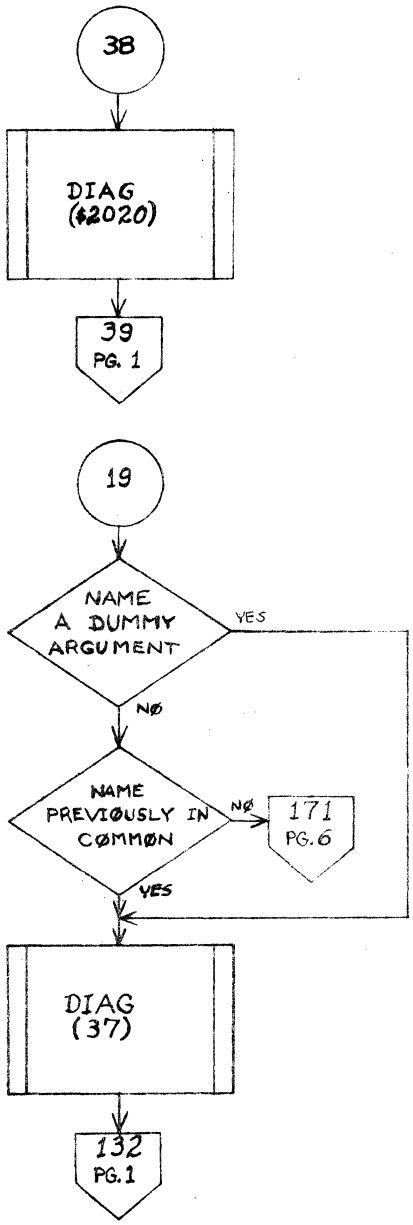
TITLE DIMPR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 4 OF 7

CSD 203

CONTROL DATA

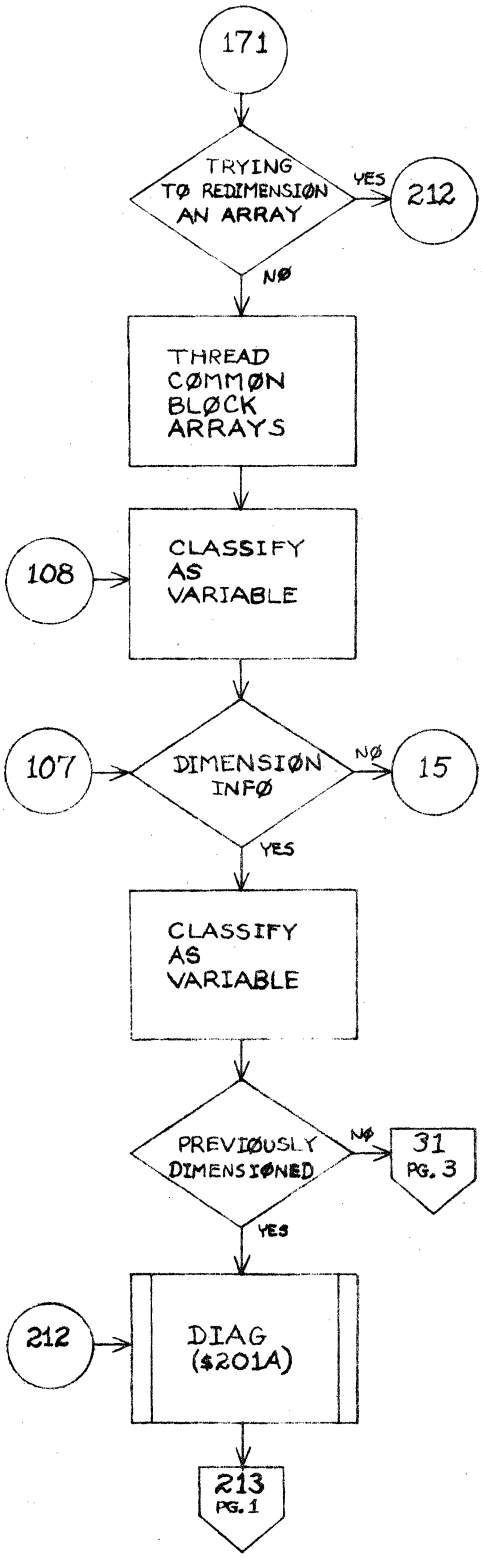
LA JOLLA FACILITY

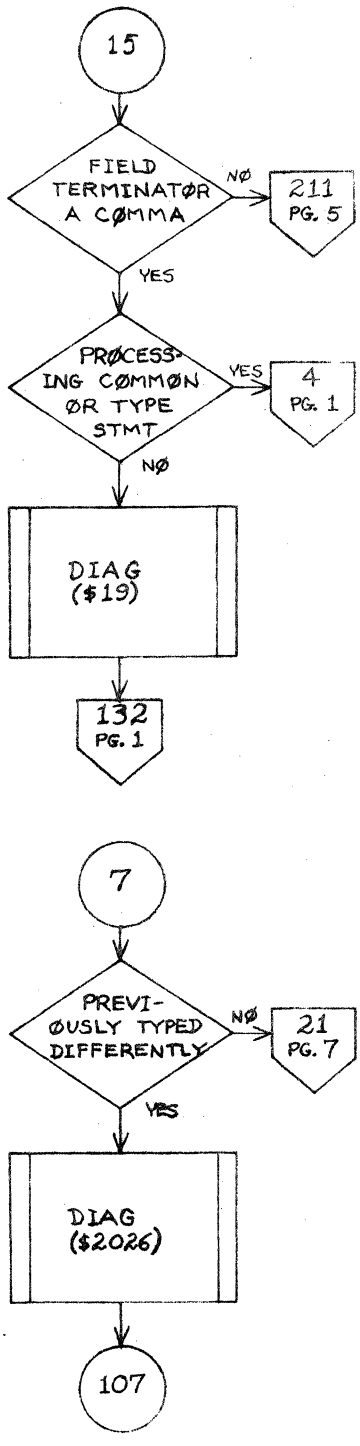




TITLE	DIMPR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 5 OF 7

CSD 203

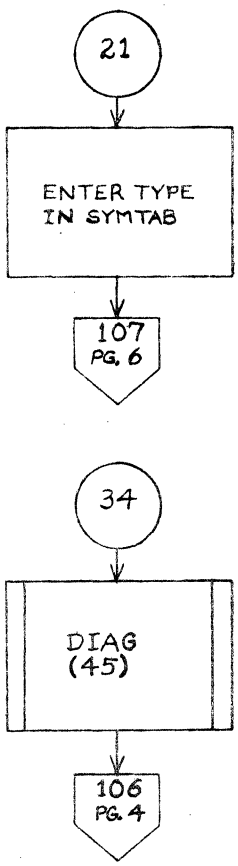




TITLE	DIMPR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 6 OF 7

CSD 203

LA JOLLA FACILITY



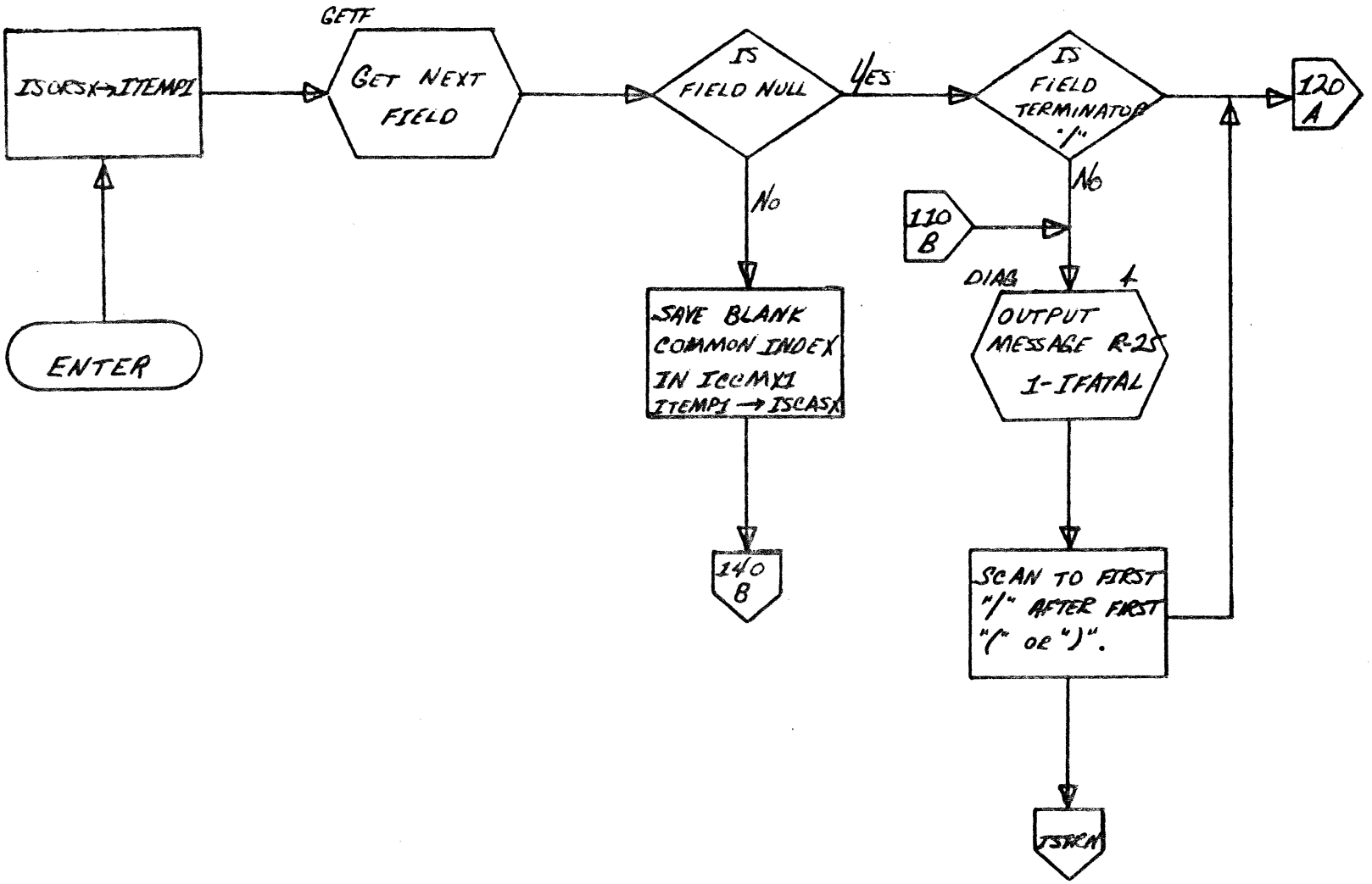
TITLE DIMPR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 7 OF 7

CSD 203

2.4.2 COMMON Statements - Subroutine COMNPR

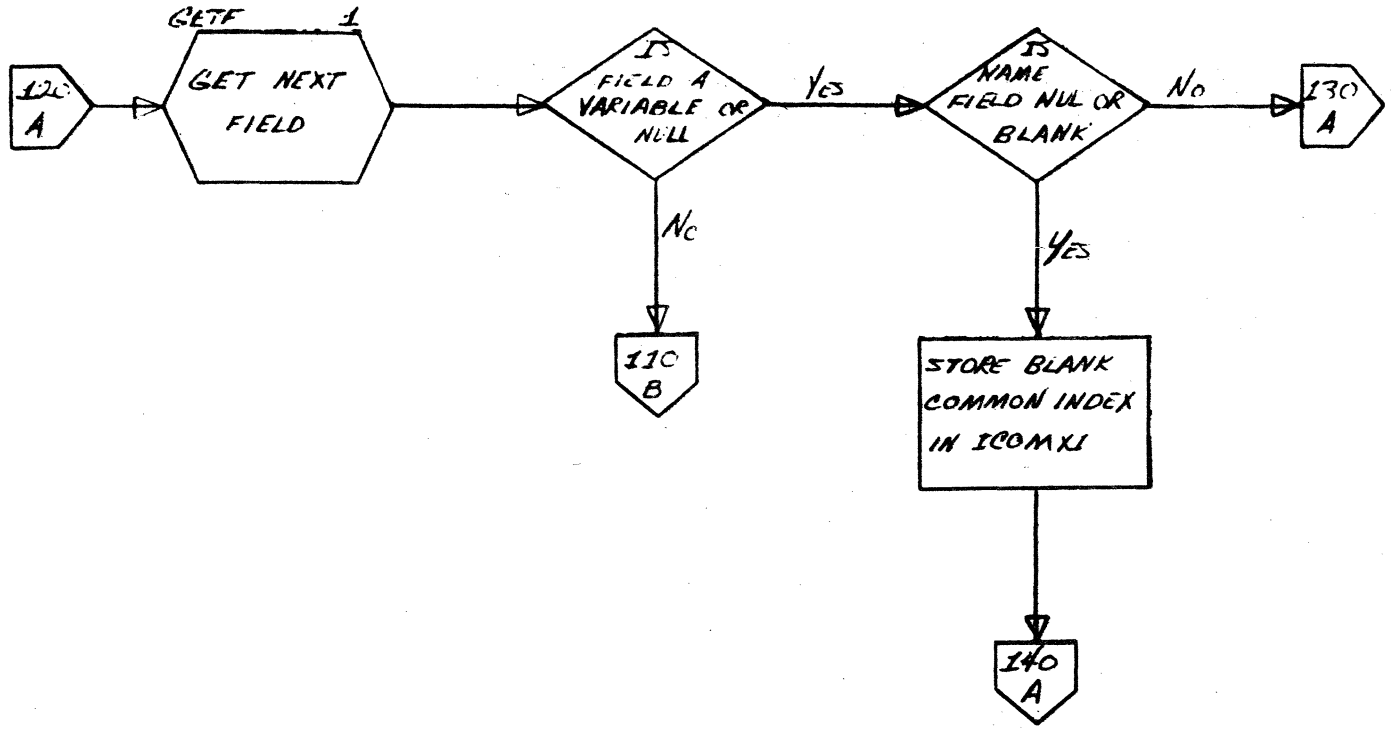
COMMON Statements are processed by the routine COMNPR. The common block is entered in ICOMT, if it is not already there. DIMPR is called. DIMPR returns to COMNPR to search for another common block.

COMMON PROCESSOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 170a	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE COMMPR	PROJECT MGR.			
	PAGE 1 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DATE	TASK NAME			

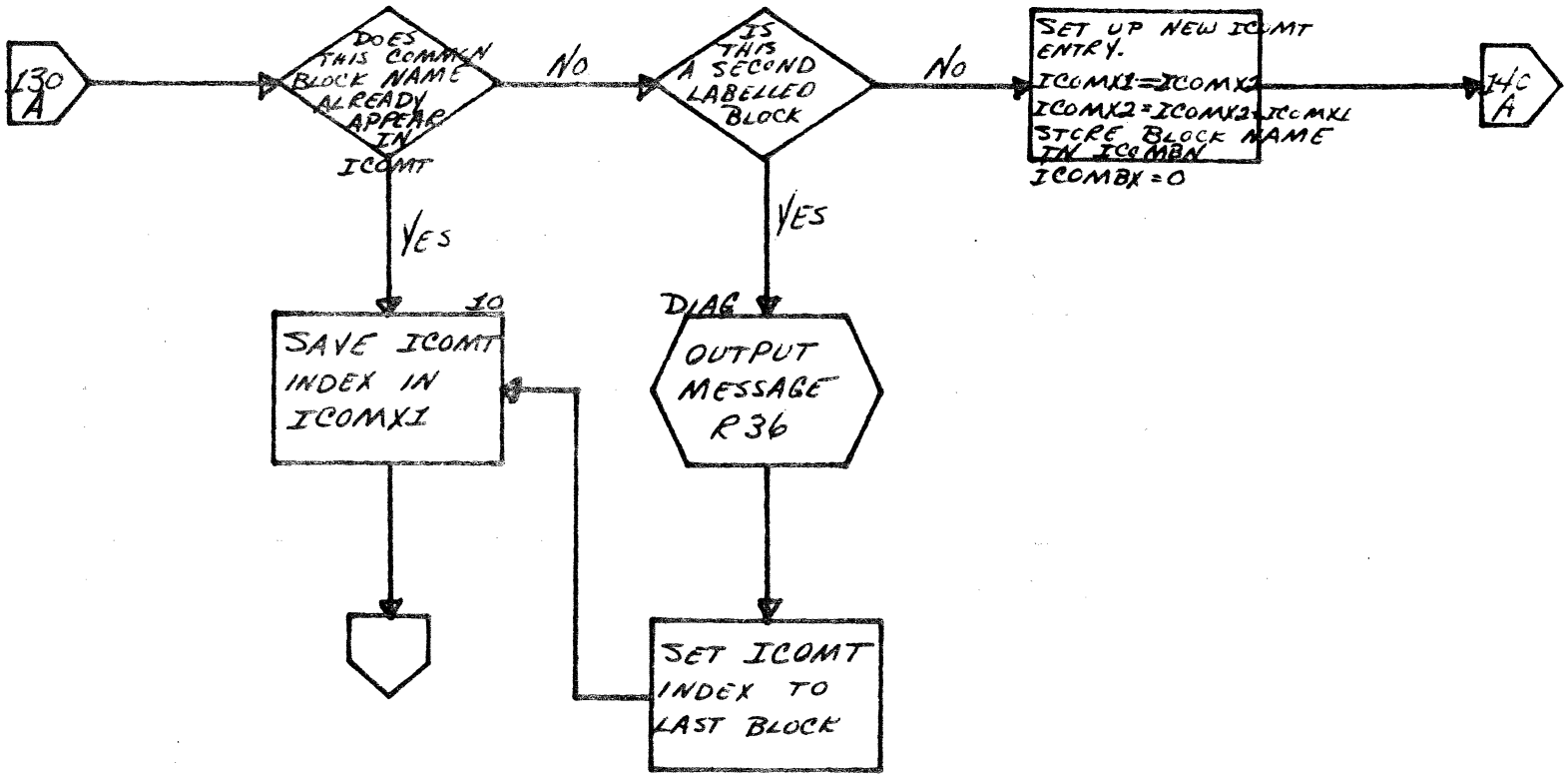
COMMON PROCESSOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE COMNPR	PROJECT MGR.			
	PAGE 2 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

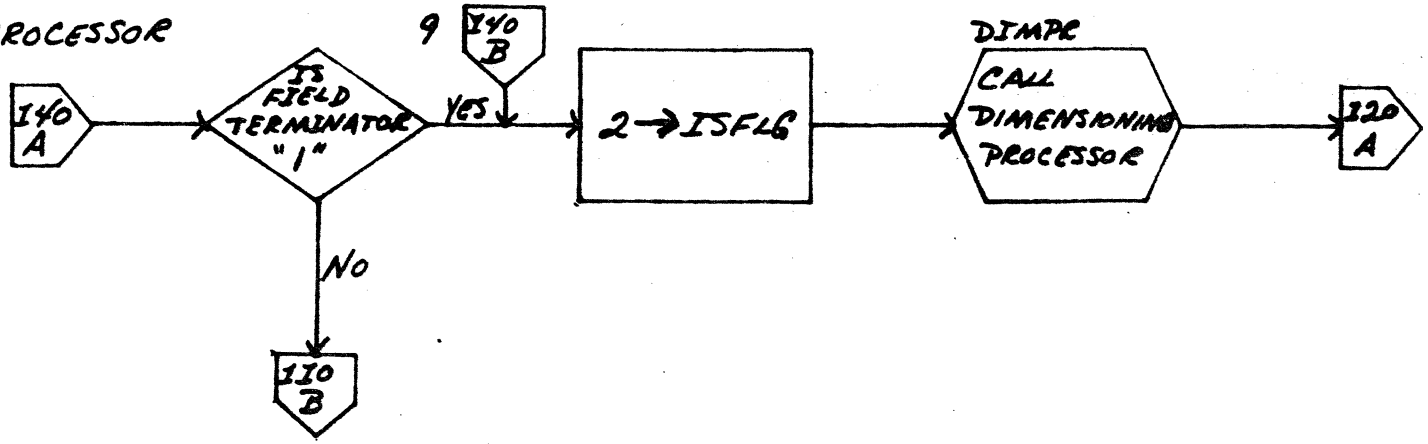
AA1385 (FORMERLY CA127-1)

COMMON PROCESSOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE COMNPR	PROJECT MGR.			
	PAGE 3 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

COMMON PROCESSOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE COMNPR	PROJECT MGR.			
	PAGE 4 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

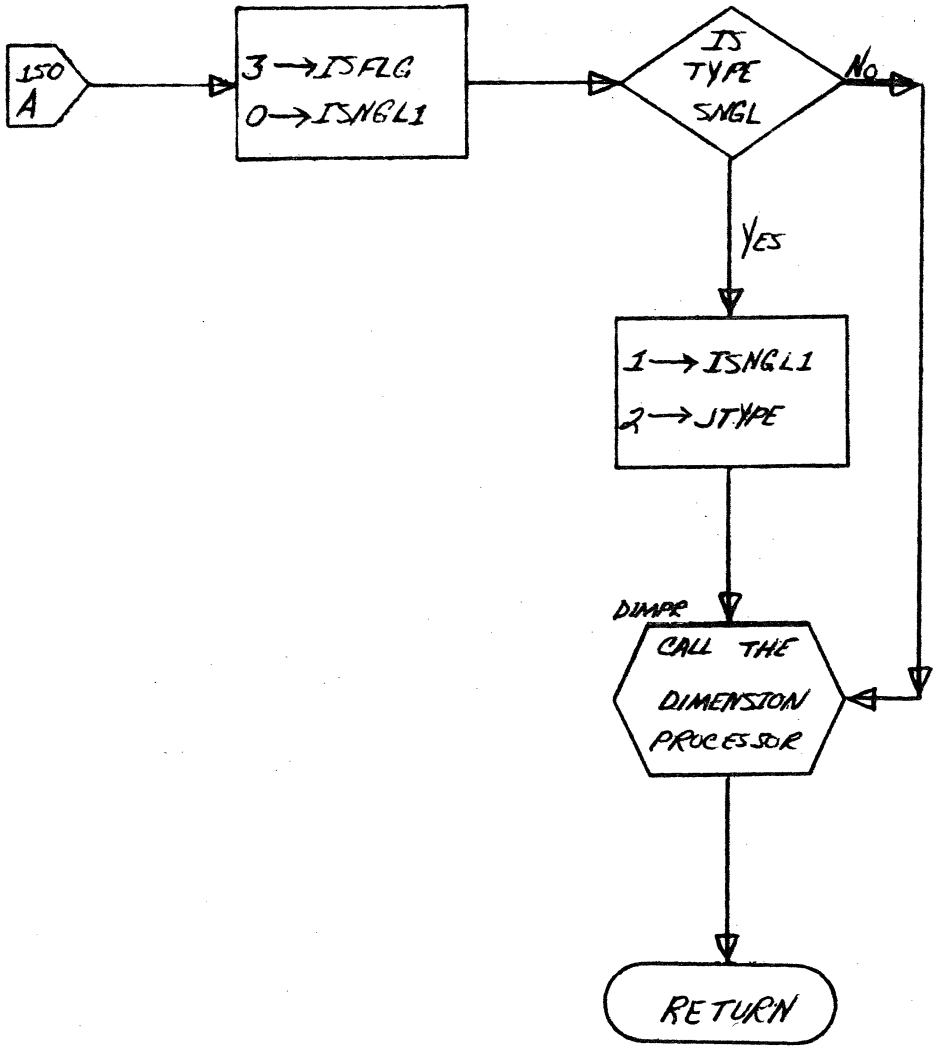
CA 127-1

2-178

2.4.3 TYPE Statements - Subroutine TYPEPR

The routine TYPEPR is entered. This routine, in turn, calls DIMPR. DIMPR enters the symbols, with their appropriate type, into the symbol table.

TYPE PROCESSOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: INS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: TYPEPR		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 5-1-7	TASK NAME		

AA1385 (FORMERLY CA127-1)

2.4.4 BYTE, EQUIVALENCE Statements - Subroutine BYEQPR

The BYEQPR subroutine is used to process BYTE, SIGNED BYTE and EQUIVALENCE statements. Upon entry to BYEQPR, the type number for the statement being processed is contained in IBUF2(3) as follows:

(IBUF2(3)) = 8 for BYTE statement.
(IBUF2(3)) = 9 for SIGNED BYTE statement.
(IBUF2(3)) = 10 for EQUIVALENCE statement.

2.4.4.1 KEQV Table

BYEQPR generates entries for the KEQV table as a result of processing either of the three statements previously mentioned. Prior to entering BYEQPR, KEQVX contains the word length for the information in KEQV, and KEQVS contains the storage capacity for KEQV. The KEQV table along with KEQVX are initially cleared to zero as part of the initialization procedure of the PHASE 1 subroutine.

LEQVX is set to the value in KEQVX immediately upon entry to BYEQPR. Entries are made to KEQV using the storage index LEQVX. There are four words per entry to KEQV. For each entry made to KEQV, LEQVX is increased by the number of words per entry --

(LEQVX) + 4 ⟶ LEQVX.

Entries made to the KEQV table have the following format:

WORD1: pointer to symbol table entry containing name of variable
WORD2: 1st subscript or 0 if subscript not specified
WORD3: 2nd subscript or 0 if subscript not specified
WORD4: 3rd subscript or 0 if subscript not specified

Consecutive entries made to KEQV correspond to related elements in the source statement. (Refer to item 2.4.4.2 for an EQUIVALENCE statement and to item 2.4.4.3 for a BYTE statement.) The last of these entries is followed by an end-of-chain mark which is a word containing a value of -1. When an entry is made to KEQV to indicate the end of chain KEQVX is increased by the number of words added to the KEQV table by -

(LEQVX) + 1 ⟶ KEQVX.

When processing begins and the next chain of related elements in the source statement, LEQVX will be set to the current word length of KEQV information by -

(KEQVX) ⟶ LEQVX.

Upon exit from BYEQPR, KEQVX will contain the word length of information stored in the KEQV table.

2.4.4.2 Processing the EQUIVALENCE Statement

Equivalence statements of the form:

EQUIVALENCE $(a_1,a_2,\ldots,a_i)$, $(b_1,b_2,\ldots,b_i)$.,.,. $(c_1,c_2,\ldots,c_i)$

are processed as follows:

A chain of related elements areenclosed in parentheses. The chains separated by commas. The last chain is followed by an EOS. Within the chain, the elements are separated by commas. A four word entry is made to KEQV in the manner described by 2.4.4.1 for each element in a chain. The last entry is followed by an end-of-chain mark. The 1st entry made for the next chain follows the end-of-chain mark for the preceding entry, etc. A chain must contain at least 2 elements.

2.4.4.2.1 Processing Errors for EQUIVALENCE Statements

Diagnostics will be given for the following errors while processing EQUIVALENCE statements:

1. A chain is not enclosed in parenthesis.
2. An element is not alphanumeric.
3. An element within a chain is already in the symbol table, and either -
   a. classed as something other than a variable name, or -
   b. the name appears either on a RELATIVE or an EXTERNAL statement.
4. An element has more than three subscripts.
5. An element has a subscript ≤ 0.
6. An element has a non-integer subscript.
7. The subscripts of an element are not separated by commas.
8. The subscripts are not enclosed in parentheses.
9. A chain contains fewer than two elements.

After the error diagnostic is printed, the entries made to the KEQV table are erased in the following manner:

Using the value in KEQVX to locate the starting point, each KEQV word is set to zero by -

$0 \longrightarrow$ KEQV(KEQVX),

$0 \longrightarrow$ KEQV(KEQVX+1),

⋮

and $0 \longrightarrow$ KEQV(LEQVX-1).

Once these entries are erased, BYEQPR will slew the remaining portion of the chain. While slewing, an exit will be made from BYEQPR when it encounters in the source buffer, either an EOS character or a right parenthesis followed by an EOS. If, while slewing, BYEQPR encounters a right parenthesis followed by a comma, it will proceed to process the next field.

2.4.4.3 Processing BYTE Statements

BYTE or SIGNED BYTE statements of the form:

$$\left\{\begin{matrix}\text{BYTE}\\ \text{SIGNED BYTE}\end{matrix}\right\} (a_1,b_1,(c_1=d_1)),\ldots, (a_i,b_i,(c_i=d_i))$$

are processed as follows:

For each byte name processed, two entries are made into the KEQV table, each in the manner described by item 2.4.4.1. The 1st entry is made for the byte name a_i followed by the entry for the second variable b_i. This entry is followed by an end-of-chain mark of "-1". The b_i variable may or may not be subscripted. However, the a_i variable must not be subscripted. Therefore, the 2nd, 3rd and 4th words of the KEQV entry for a_i must contaii zeros.

If $(c_i,b_i(c_i=d_i))$

may be regarded as a chain, the chain of the BYTE or SIGNED BYTE statement must be separated by commas and the last followed by an EOS.

The byte limits c_i and d_i must both be integer constants and maintain the relationship -

$$15 \geq c_i \geq d_i \geq 0.$$

An indicator is set in the symbol table entry containing the name a_i to show that it is a byte name by either -

a. 1 ⟶ IPART(ISYMX) if a_i is not a signed byte, or -
b. 2 ⟶ IPART(ISYMX) if a_i is a signed byte.

The byte limits are recorded in the symbol table entry as follows:

1. c_i ⟶ IPARTL(ISYMX), and -
2. d_i ⟶ IPARTR(ISYMX).

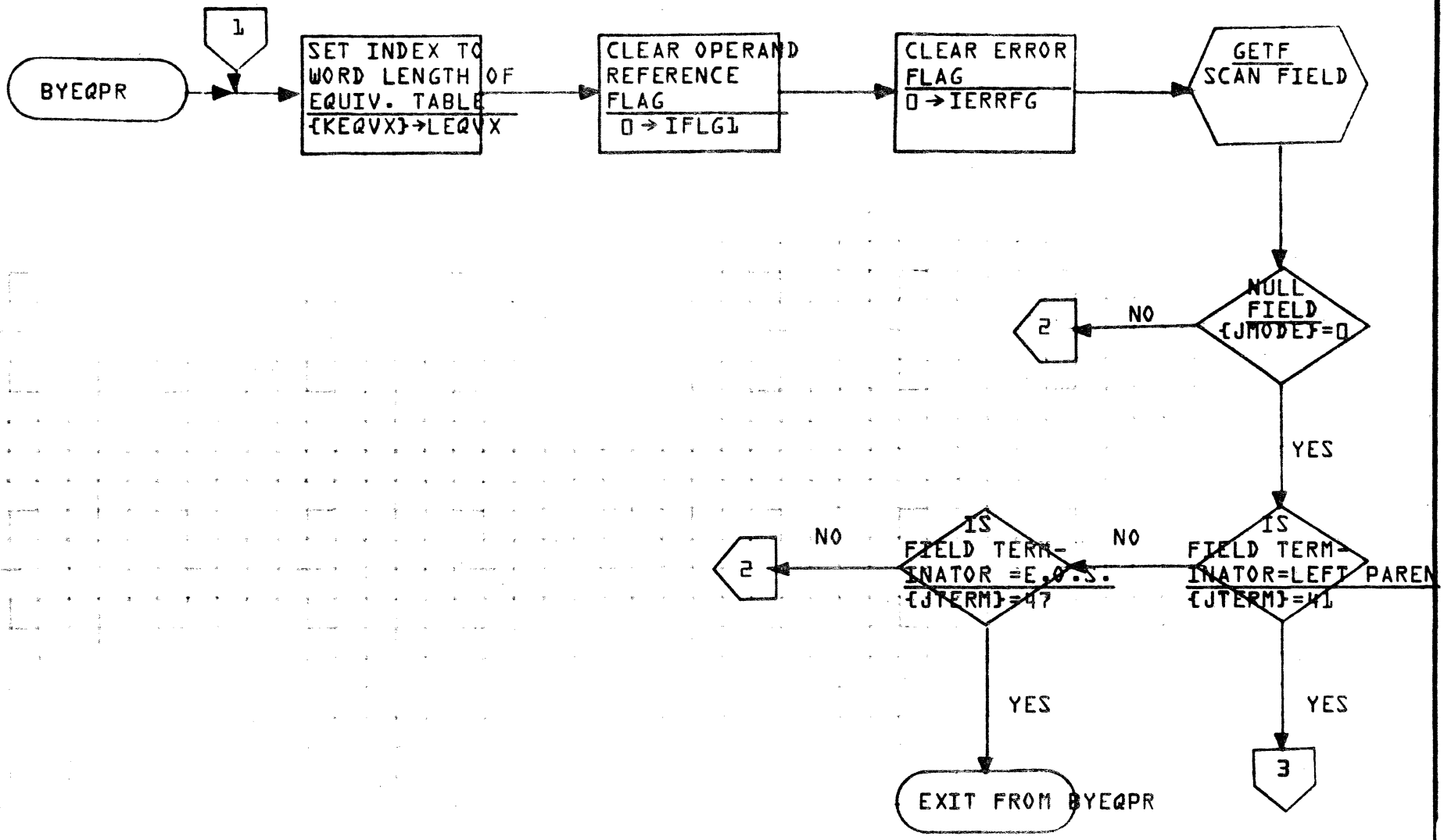
2.4.4.3.1 Processing Errors for BYTE and SIGNED BYTE Statements

Diagnostics will be given for the following errors while processing BYTE and SIGNED BYTE statements:

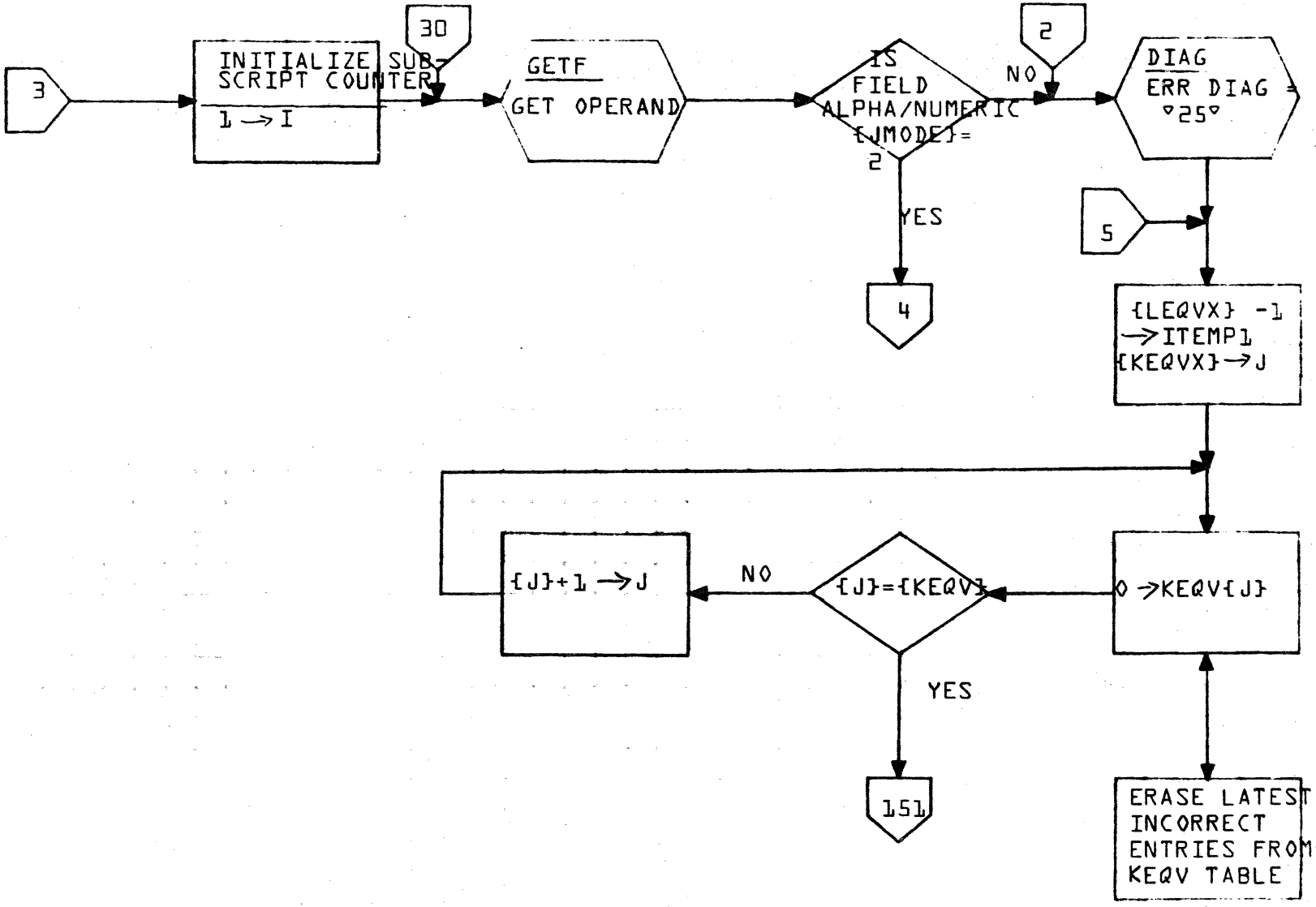
1. A chain (as defined in item 2.4.4.3) is not enclosed in parenthesis.
2. An element, either a_i or b_i is not alphanumeric.
3. Some byte name a_i appears in the symbol table either -
   a. typed as something other than an integer variable,
   b. indicated to be a dummy variable, or -
   c. previously specified as a byte.
4. The byte name a_i is subscripted.
5. The second variable b_i is not alphanumeric.
6. The second variable b_i has more than three subscripts.
7. The second variable b_i has a non-integer subscript.
8. The second variable b_i has a subscript < 0.
9. The subscripts for b_i are not enclosed within parentheses.
10. Either of the byte limits c_i or d_i is not an integer constant.
11. The byte limits c_i and d_i are not enclosed in parentheses and/or the limits c_i and d_i are separated by something other than equal sign.
12. The byte limits c_i and d_i do not satisfy the condition -

$$15 \geq c_i \geq d_i \geq 0.$$

Following the diagnostic output, the procedure followed for errors 1-11 is the same as the procedure described in item 2.4.4.2.1 concerning errors in EQUIVALENCE statements. After printing the diagnostic, the procedure for error 12 is to substitute a 15 for c_i and a 0 for d_i and proceed as if no error had occurred.



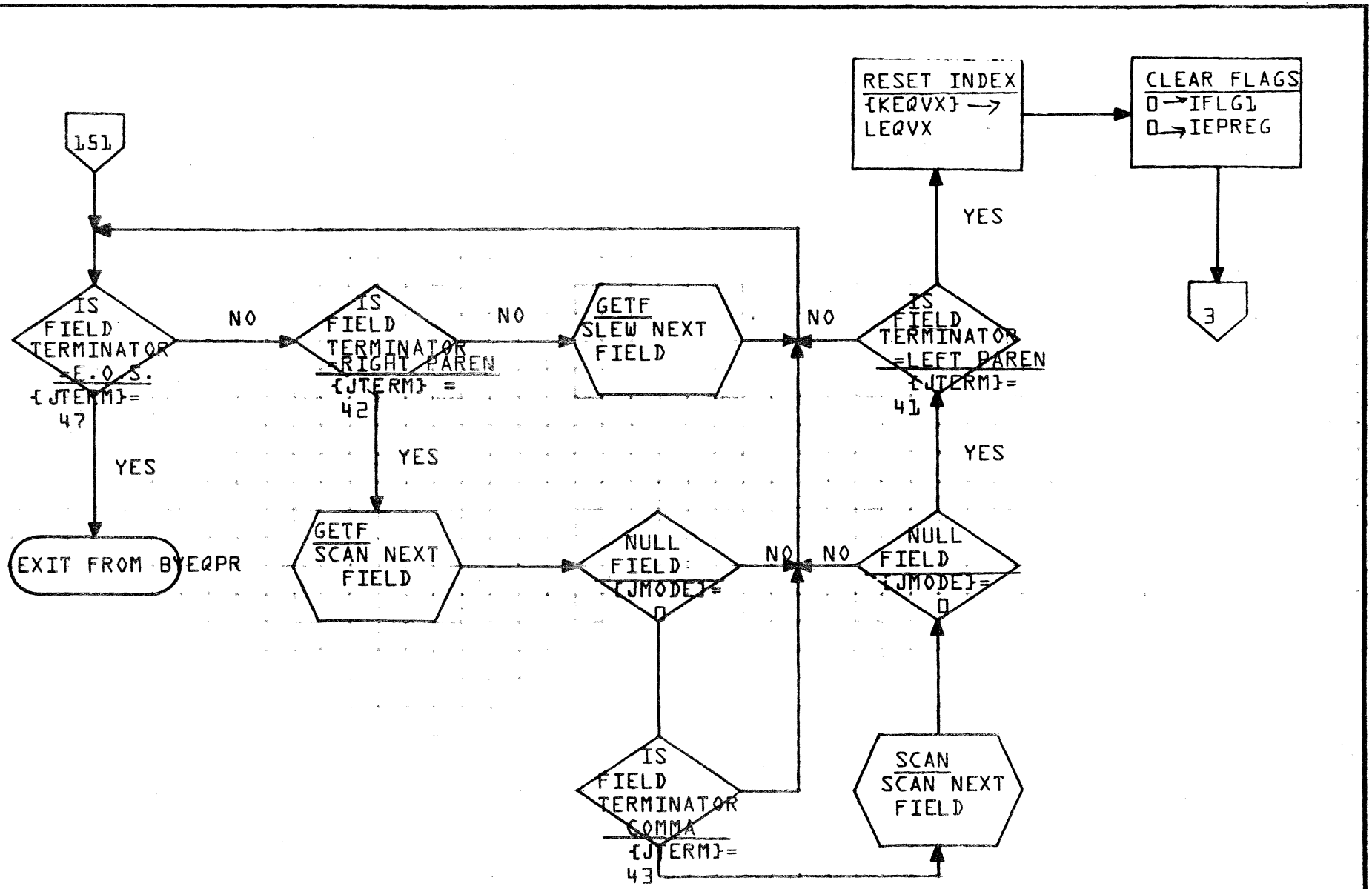
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: BYEQPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 15	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



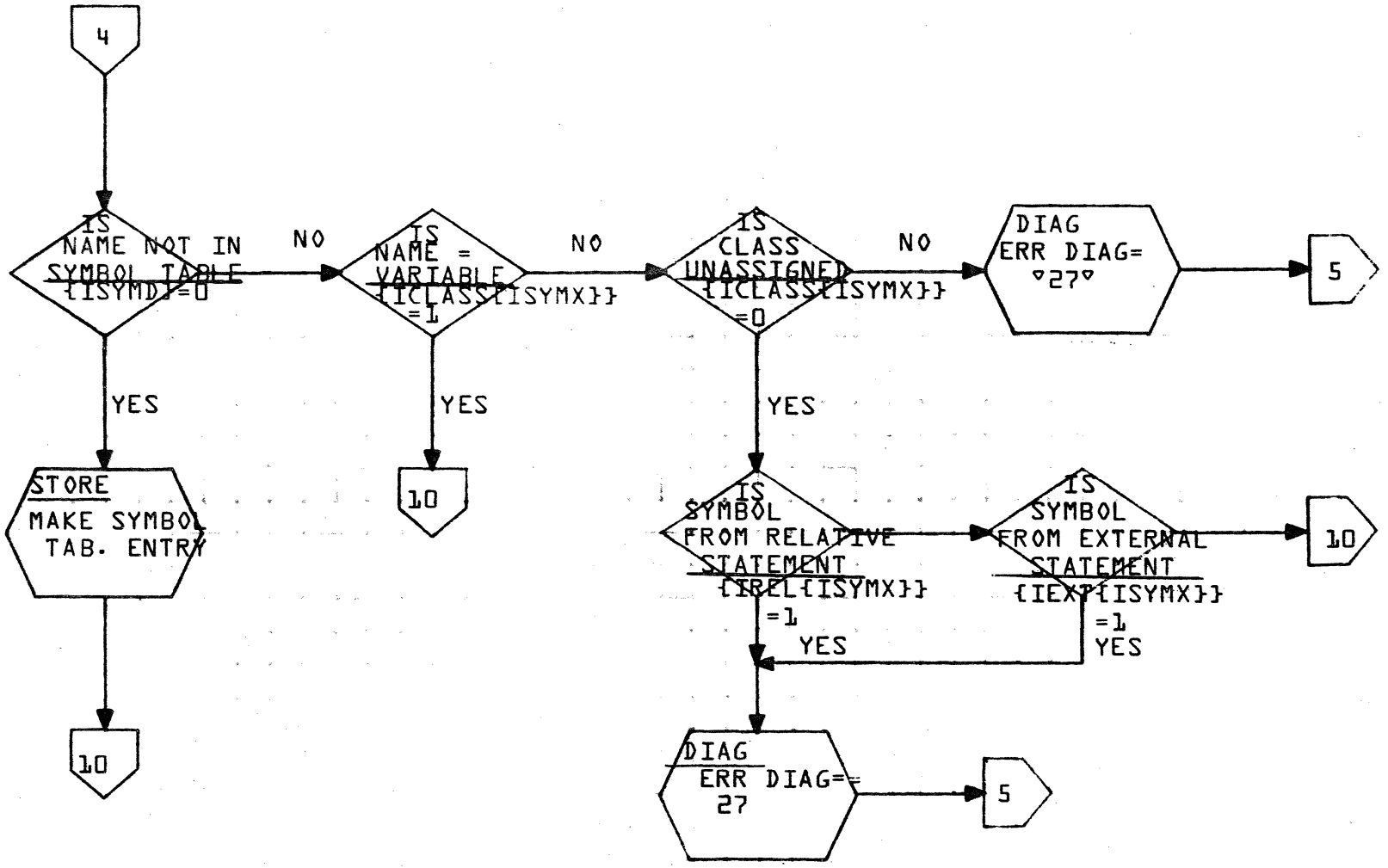
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BYEQPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 2 OF 15	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

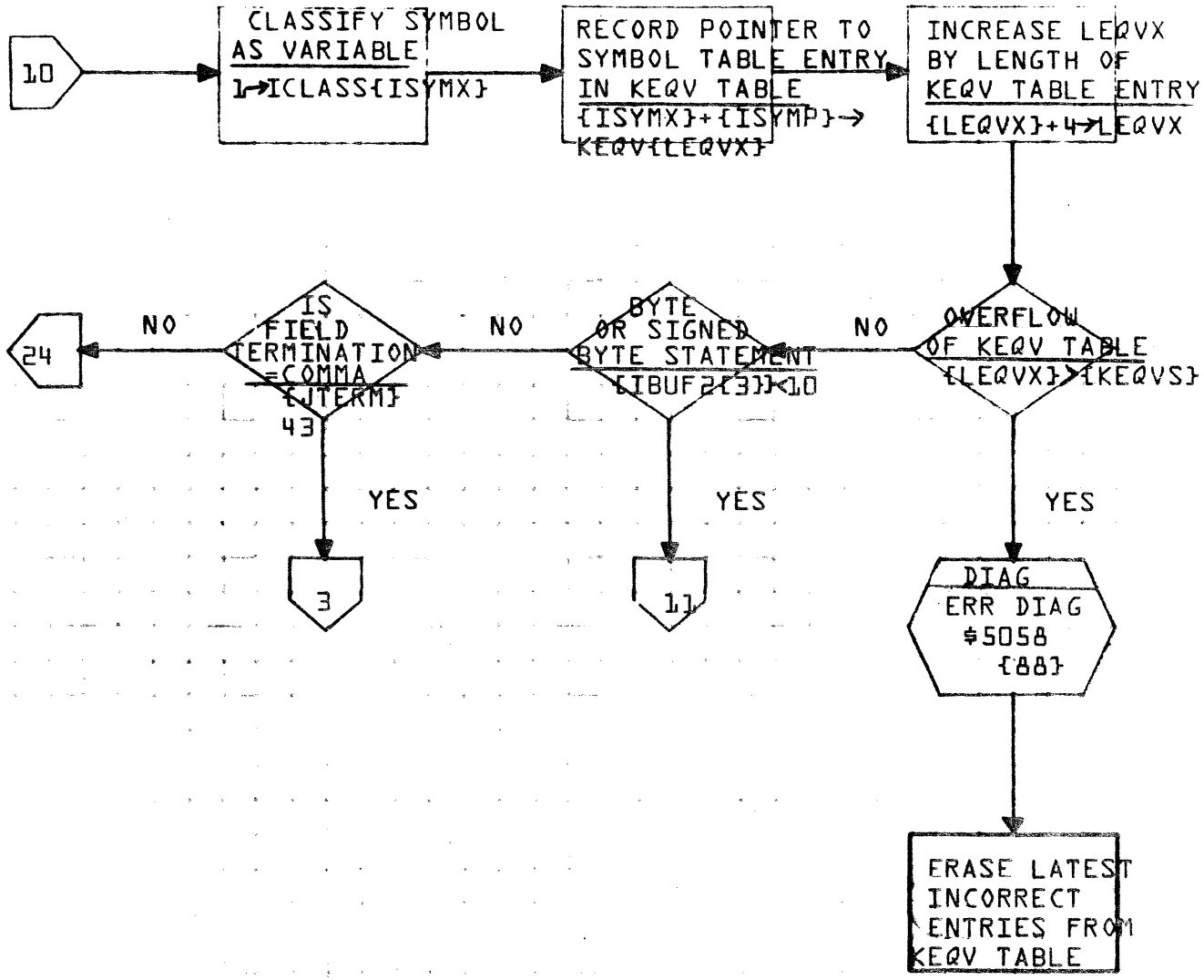


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BYEQPR		PROJECT MGR.			
		PAGE 3 OF 15	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

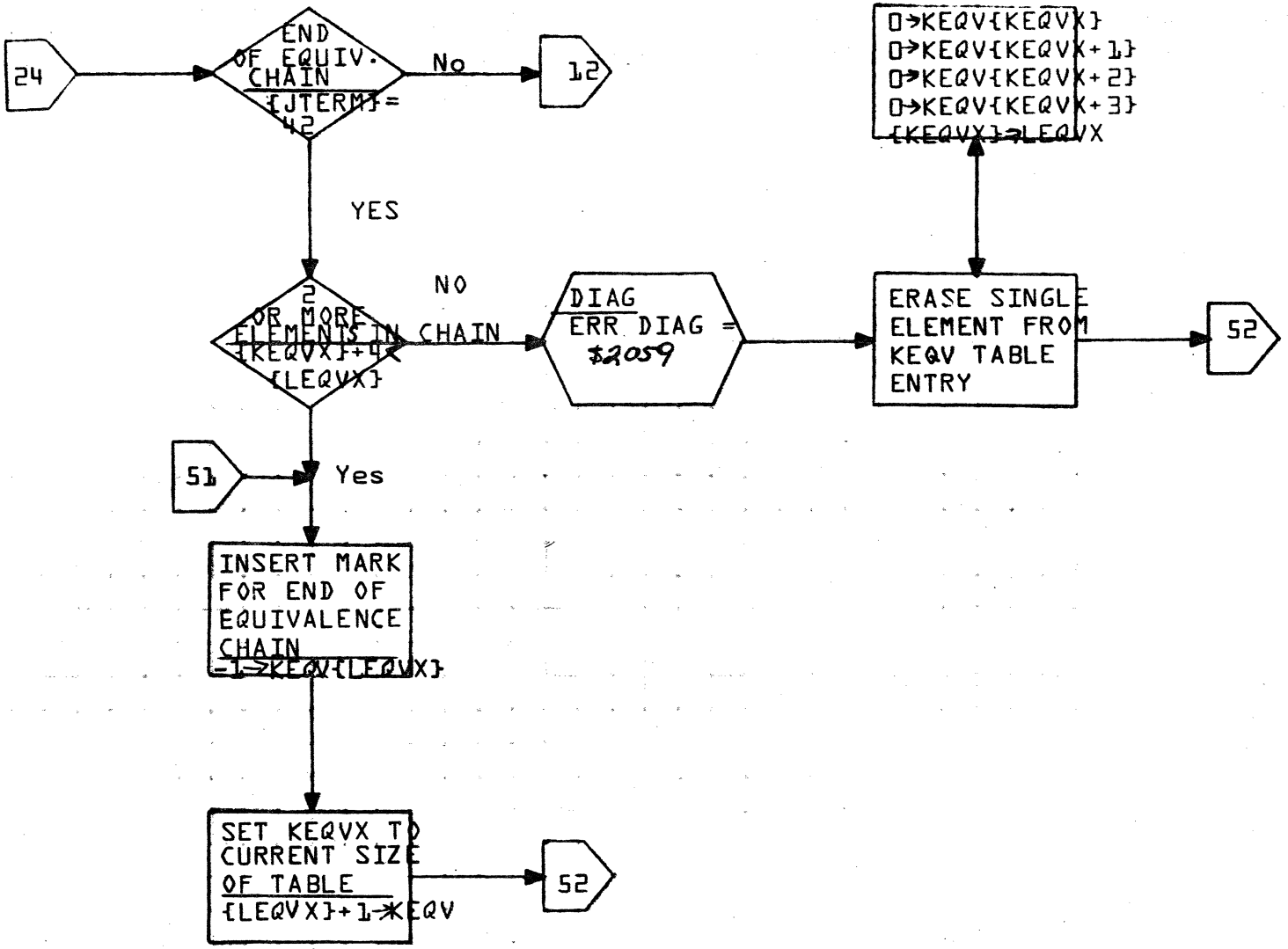


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BYEQPR		PROJECT MGR.		
		PAGE 4 OF 15	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

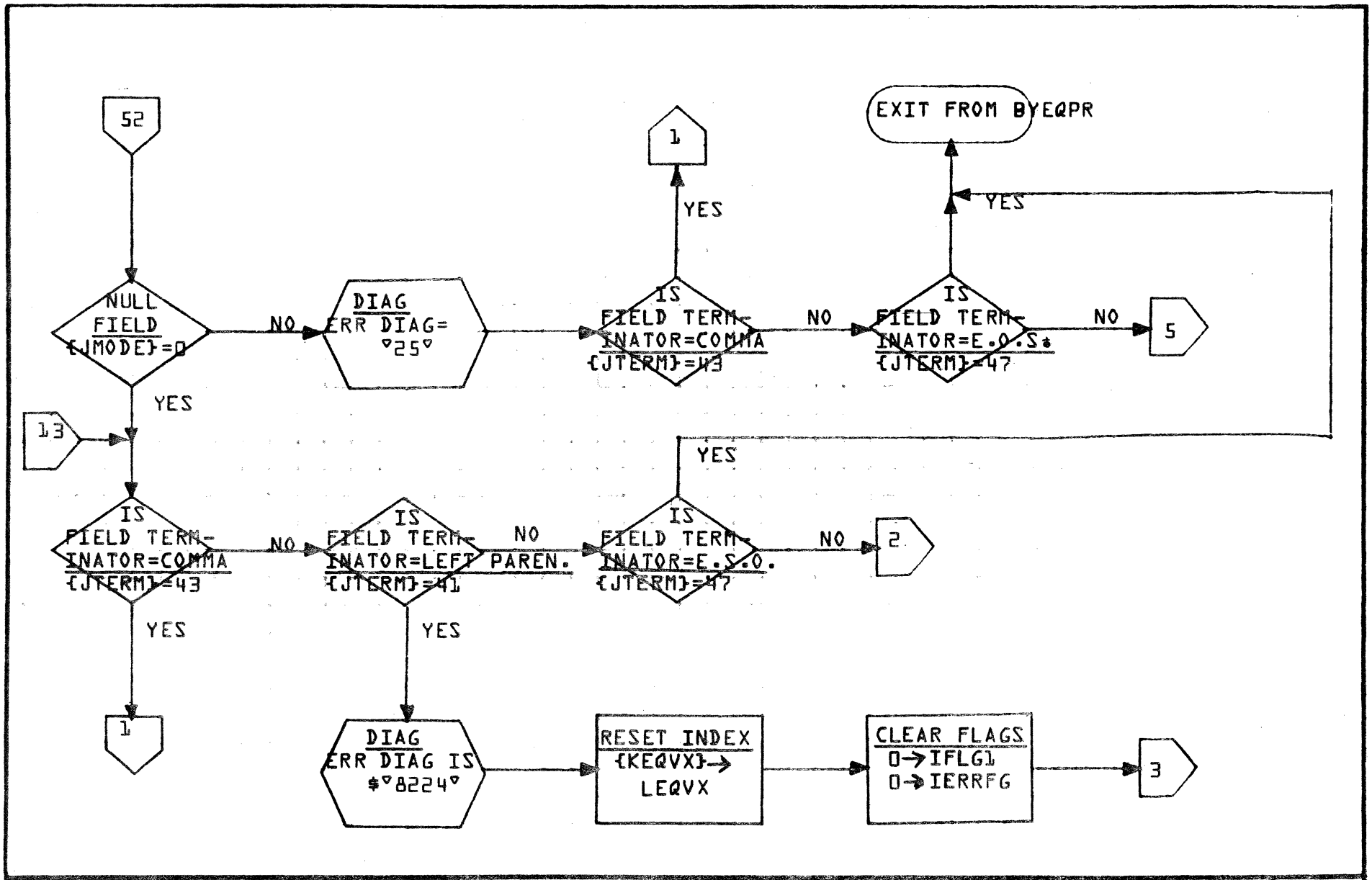
2-188



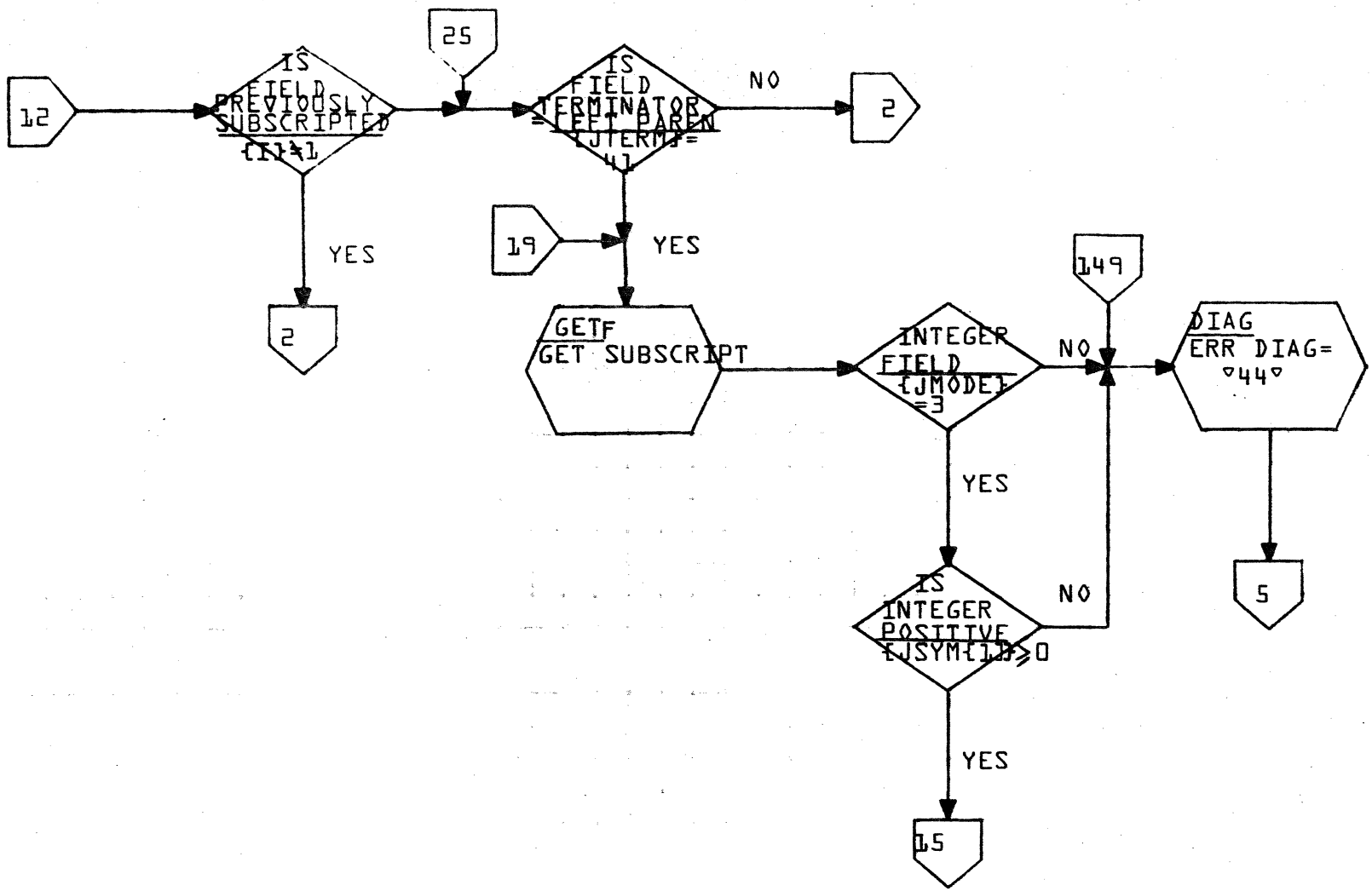
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BYEQPR	PROJECT MGR.			
	PAGE 5 OF 15	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



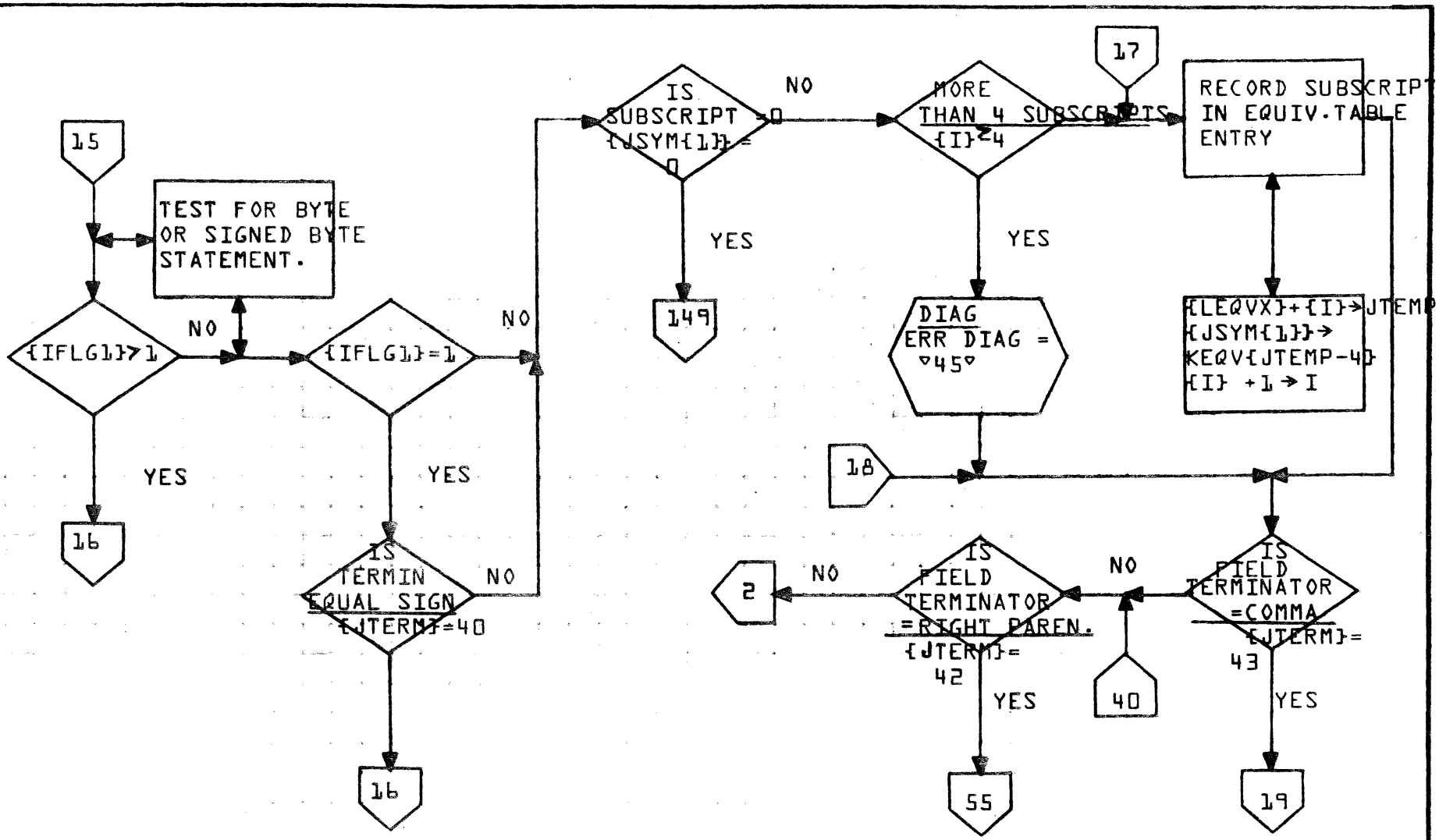
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BYEQPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 6 OF 15	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



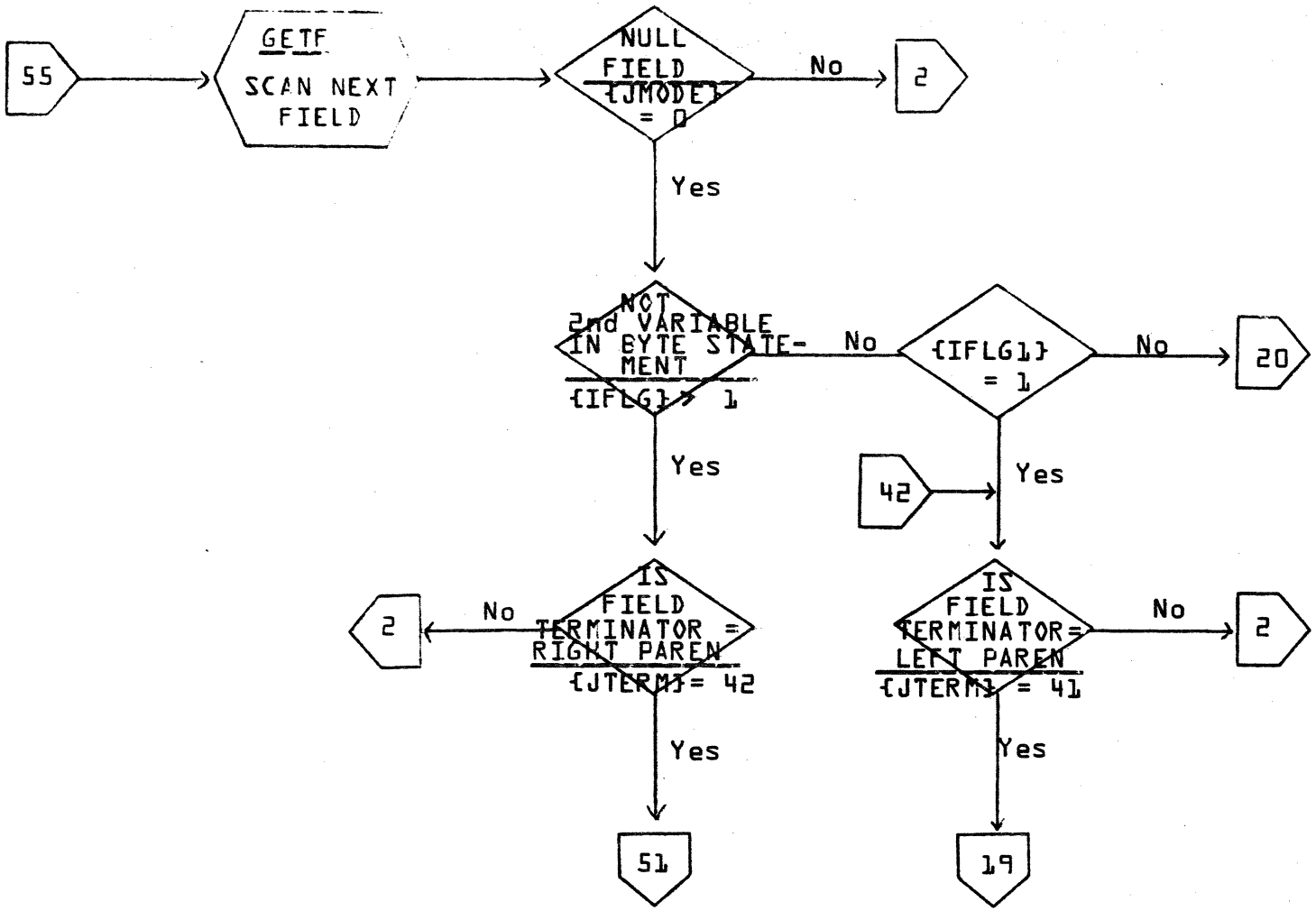
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: BYEQPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 7 OF 15	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE BYEQPR		PROJECT MGR.		
		PAGE 8 OF 15	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

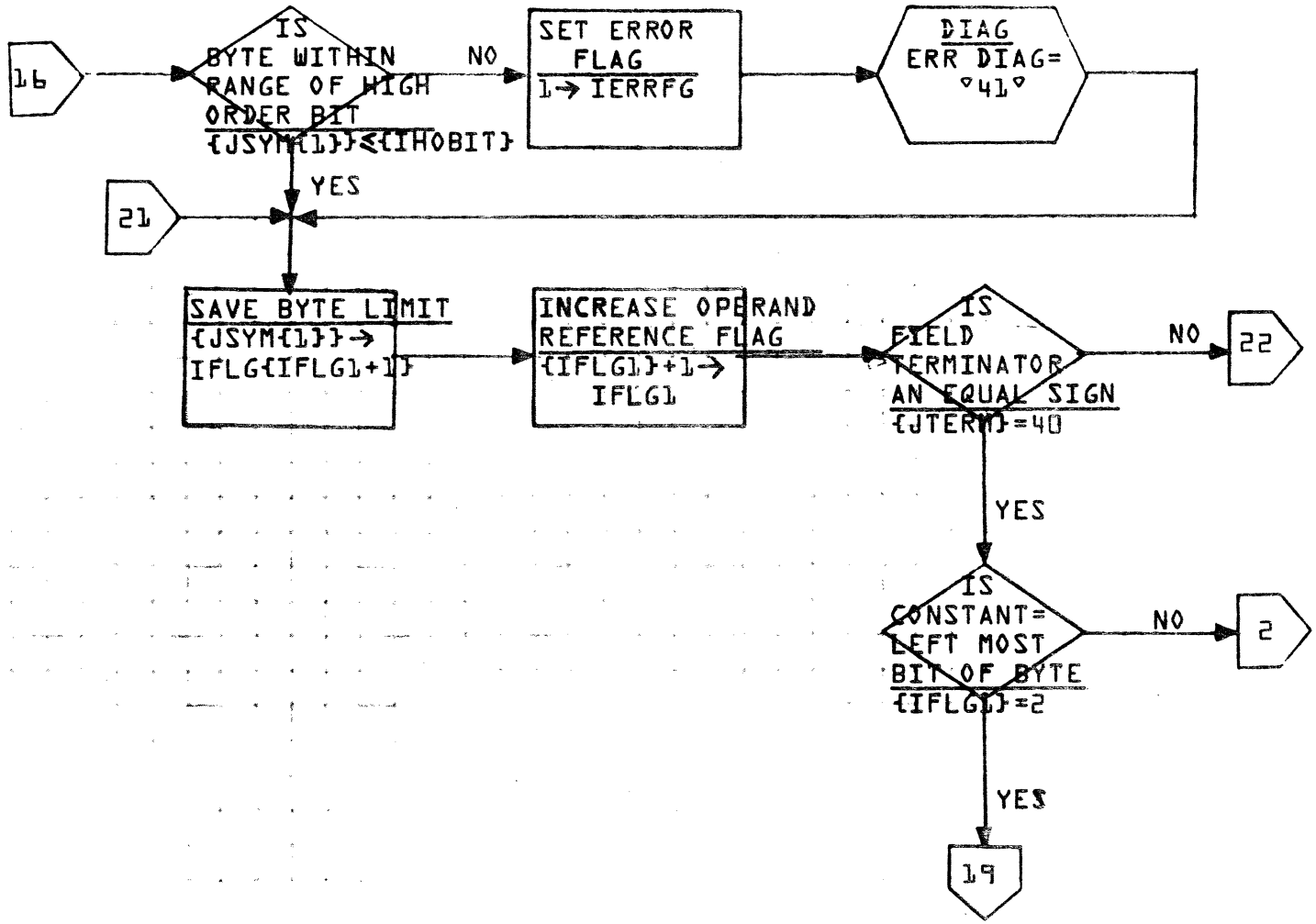


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BYEQPR	PROJECT MGR.			
	PAGE 9 OF 15	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

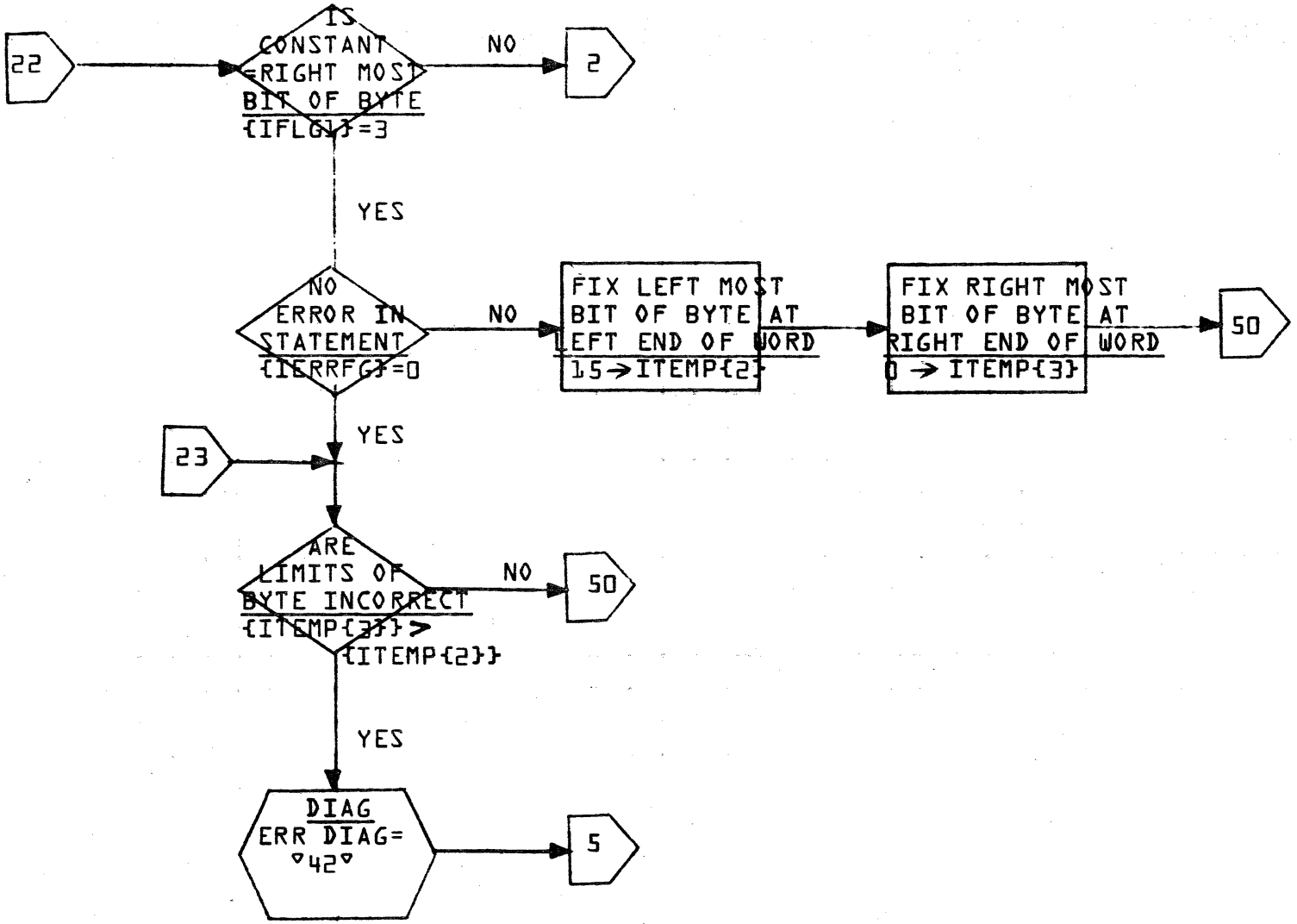


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BYEQPR		PROJECT MGR.			
FLOWCHART ☐		PAGE 10 OF 15	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)



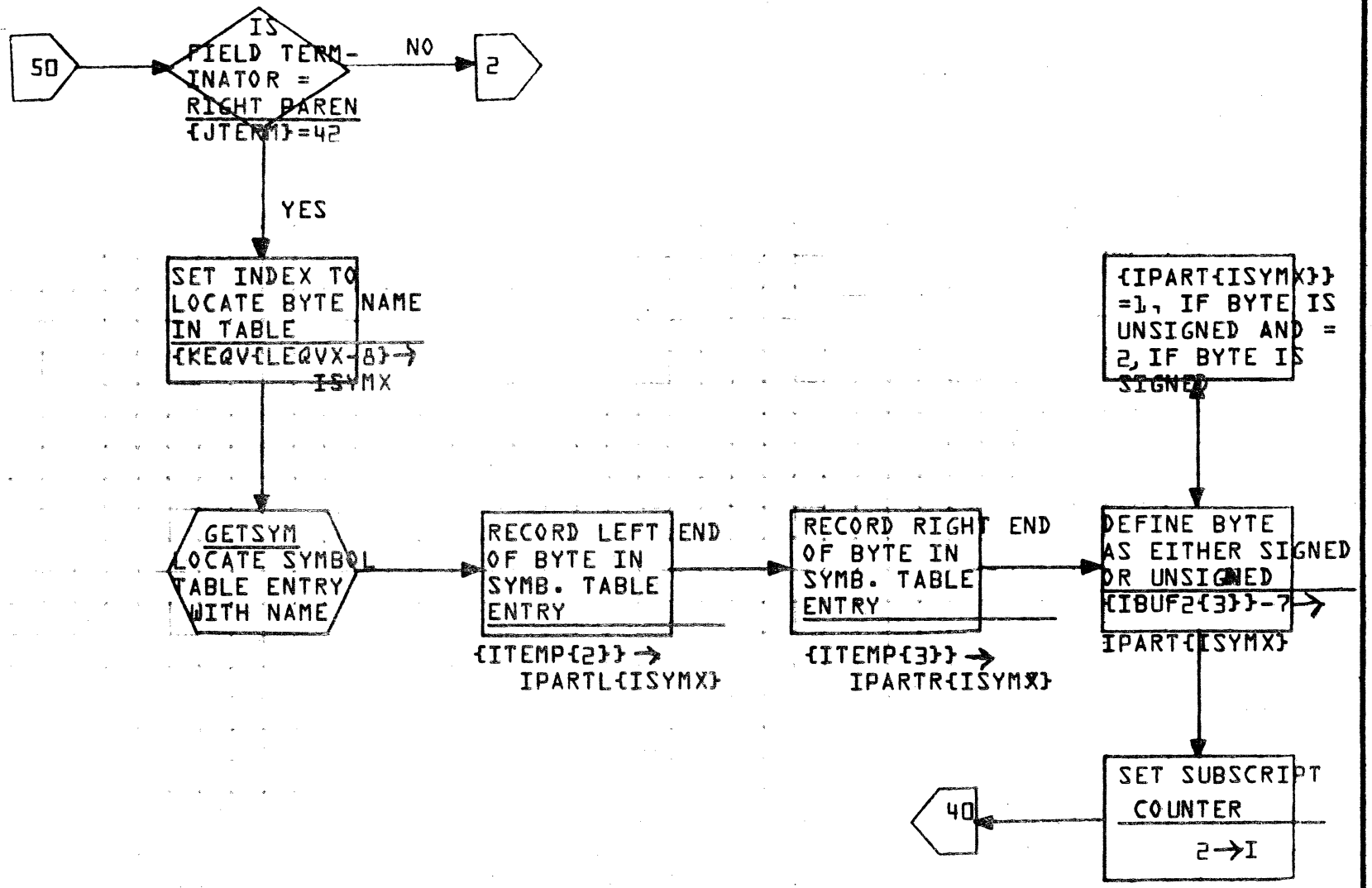
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BYEQPR		PROJECT MGR.		
		PAGE 11 OF 15	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



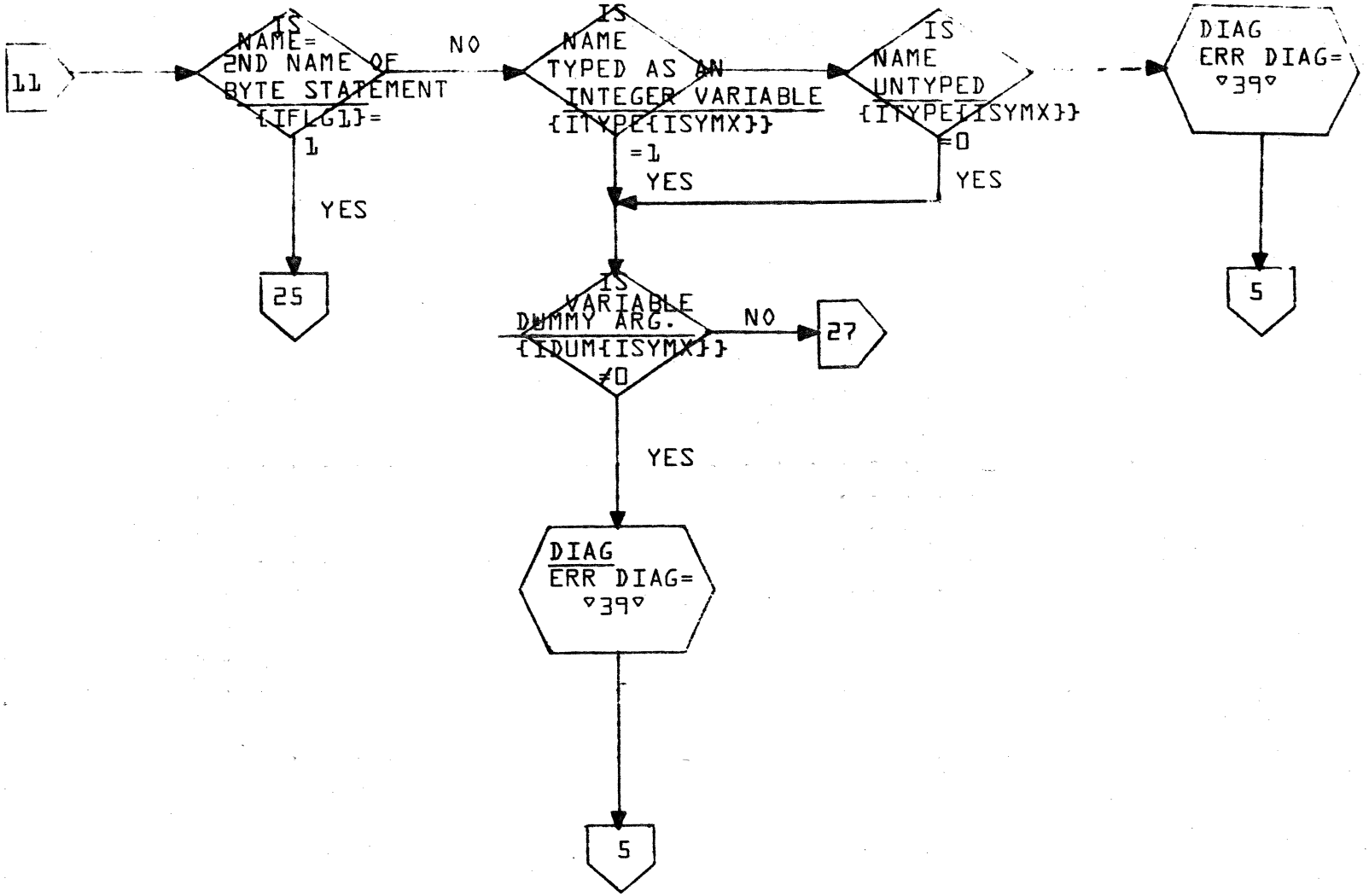
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BYEQPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 12 OF 15	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1388 (FORMERLY CA127-1)

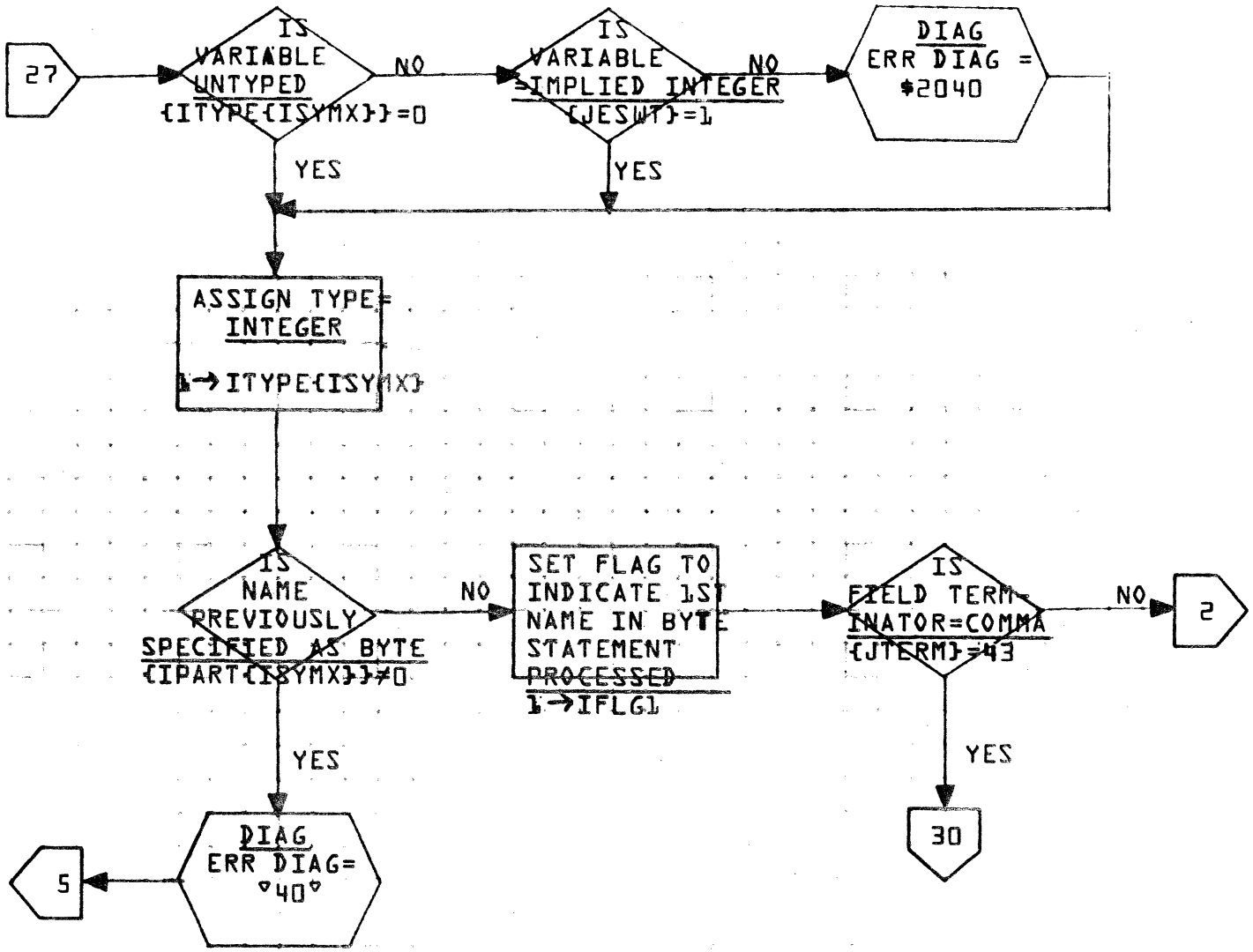
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BYEQPR	PROJECT MGR.			
	PAGE 13 OF 15	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BYEQPR	PROJECT MGR.			
	PAGE 14 OF 15	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



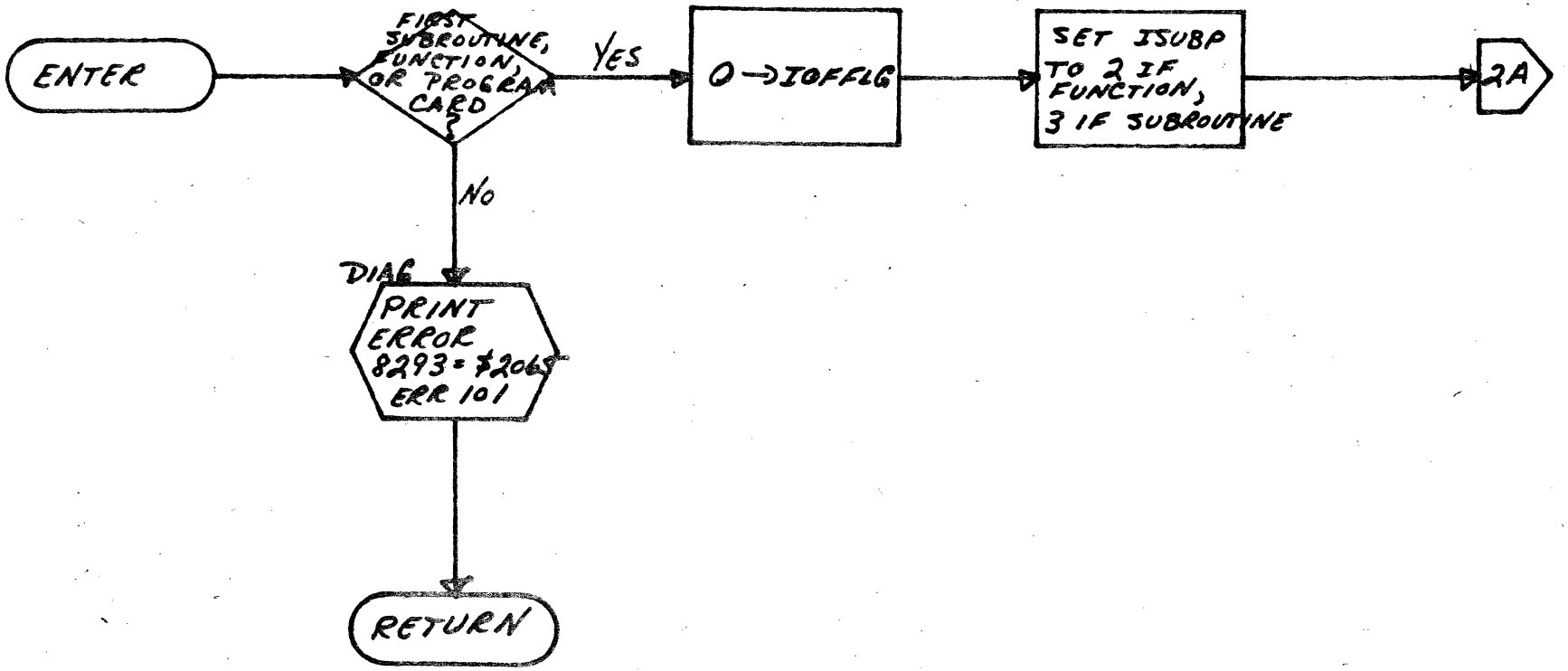
CONTROL DATA CORPORATION	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE BYE@PR		PROJECT MGR.			
SAMPLE CODE ☐		PAGE 15 OF 15	PROJECT NAME			
FLOWCHART ☒	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

2.4.5 SUBROUTINE or FUNCTION Statement - Subroutine SUBPPR

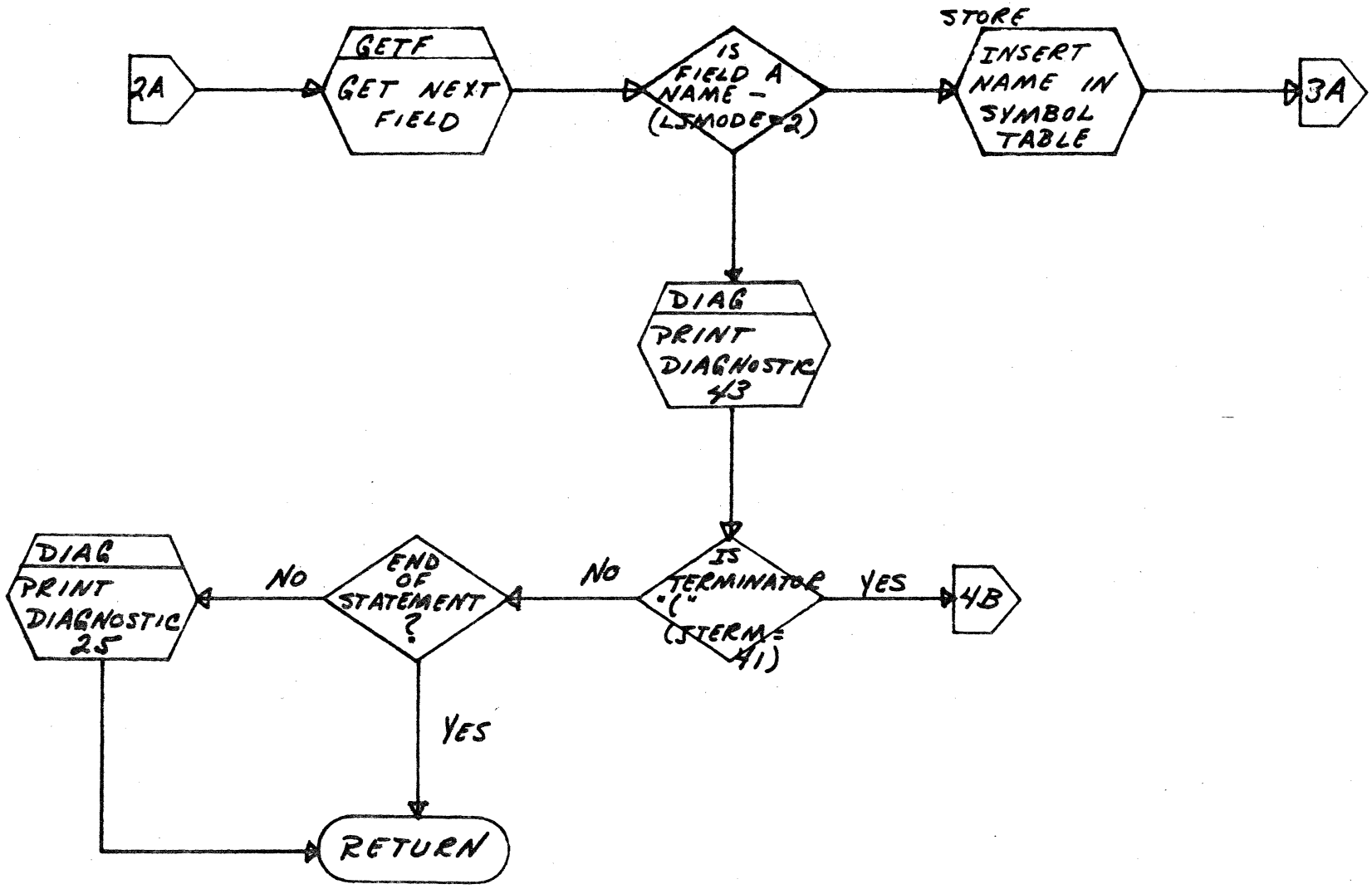
SUBPPR processes SUBROUTINE or FUNCTION statements and is called directly by PHASE A It sets a flag to distinguish between a subroutine and a function. (2 = function, 3 = subroutine.)

The next field is checked to see if it is a name (JMODE = 2). The name is checked for legality. If legal, it is put in the symbol table. If not, a diagnostic is printed. The terminator is checked. If not a left parenthesis, SUBPPR returns to PHASE A If left parenthesis, fields are checked until a right parenthesis is found. Each name encountered is checked for legality and entered in the symbol table. The symbol table pointer goes to output buffer.

SUBPPR

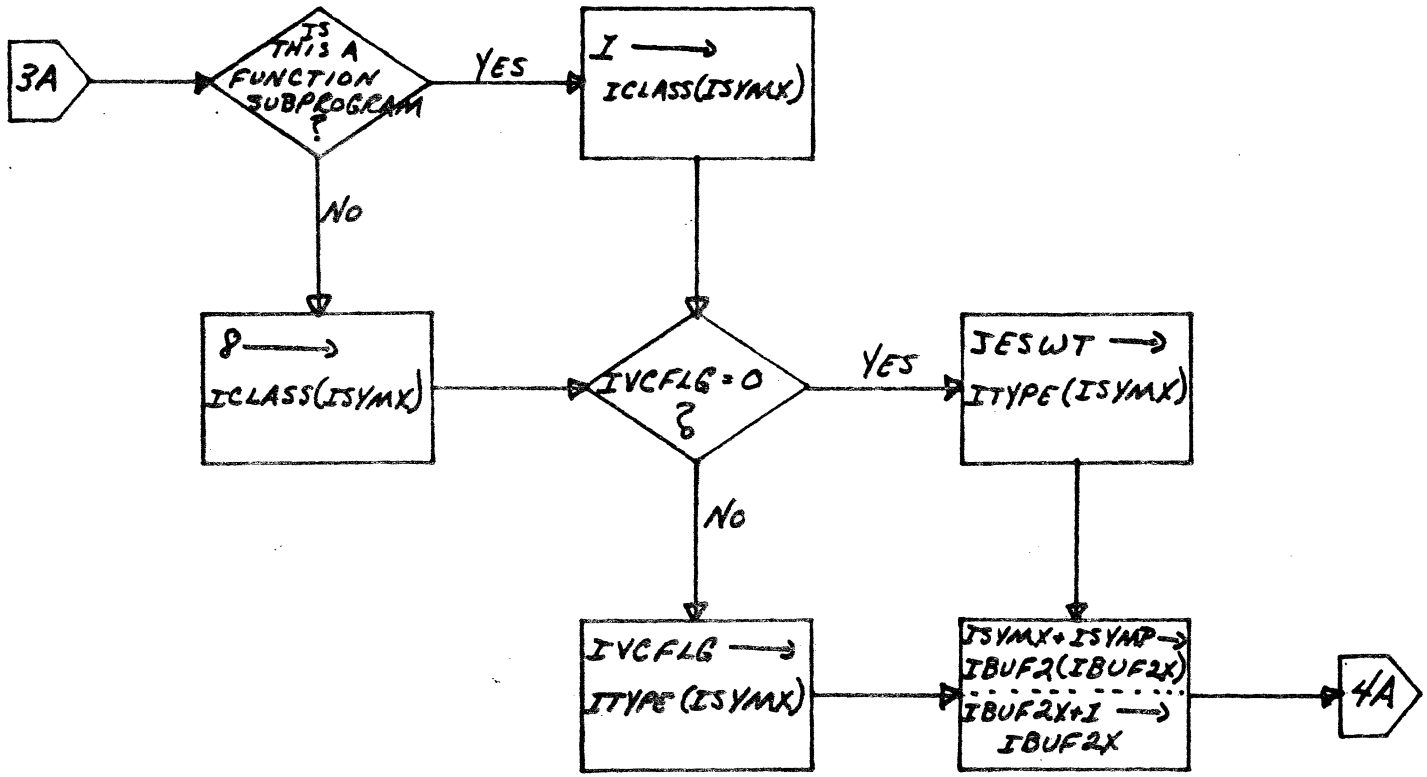


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBPPR	PROJECT MGR.			
	PAGE 1 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

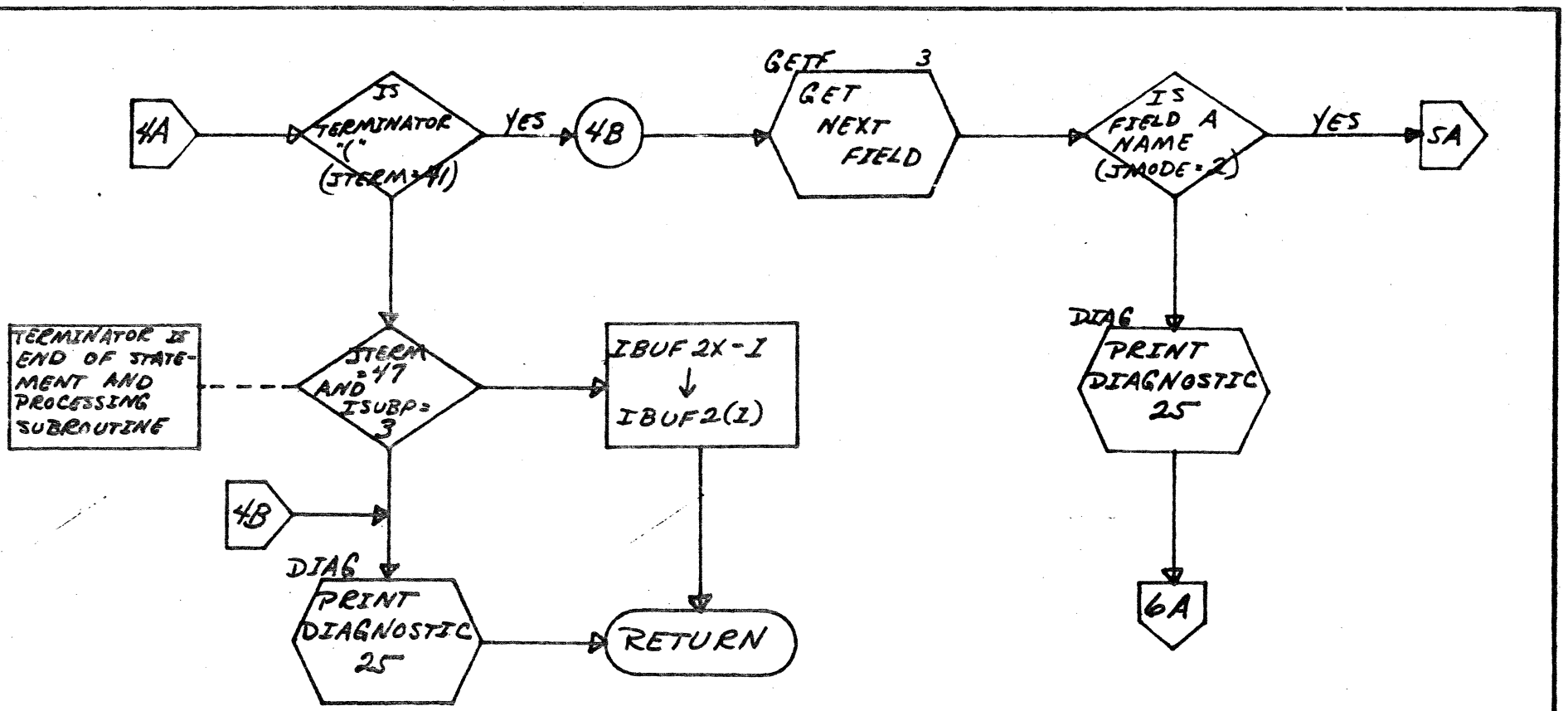


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SUBPPR		PROJECT MGR.		
		PAGE 2 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

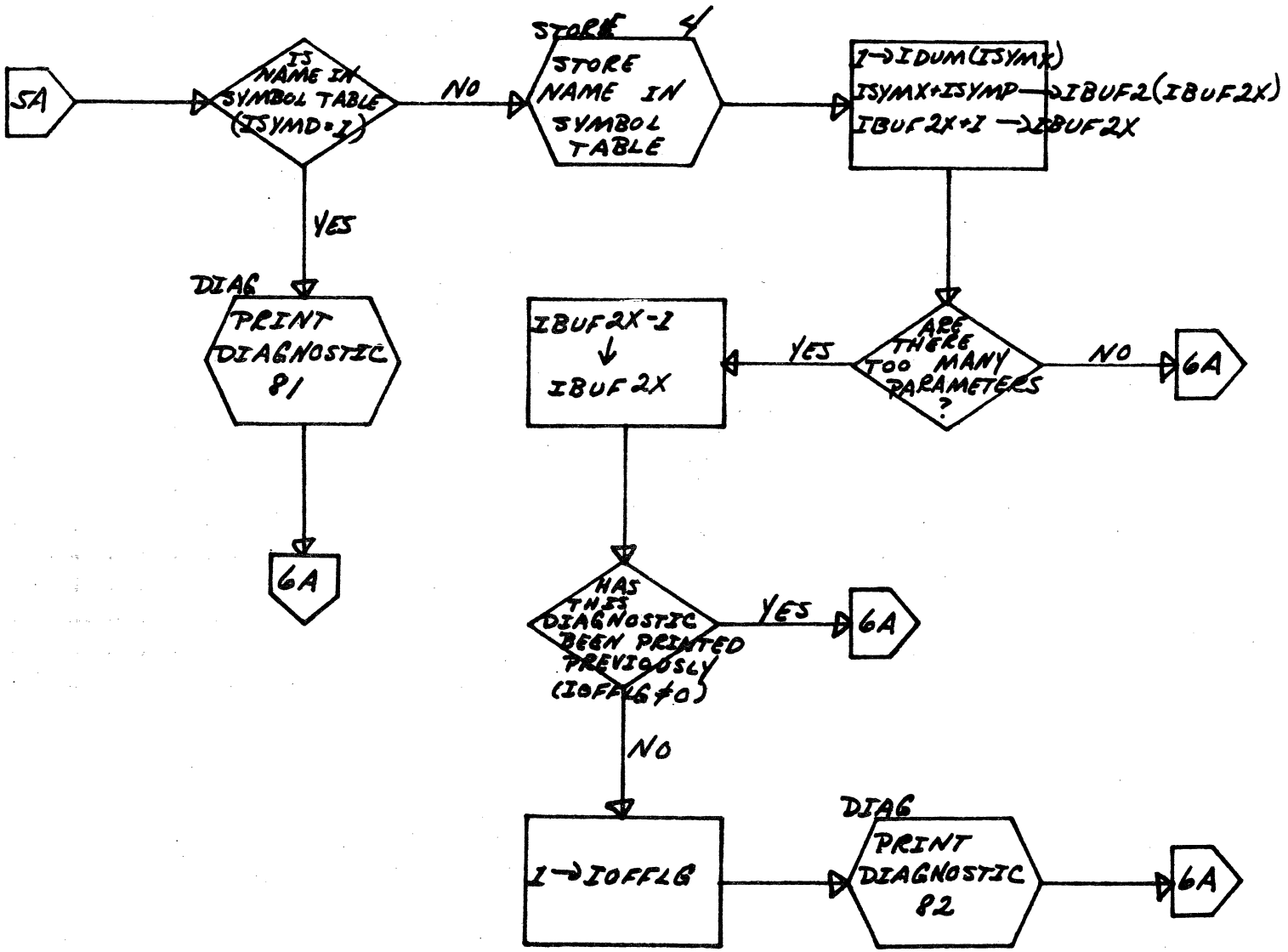


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBPPR		PROJECT MGR.			
	PAGE 3 OF 6		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

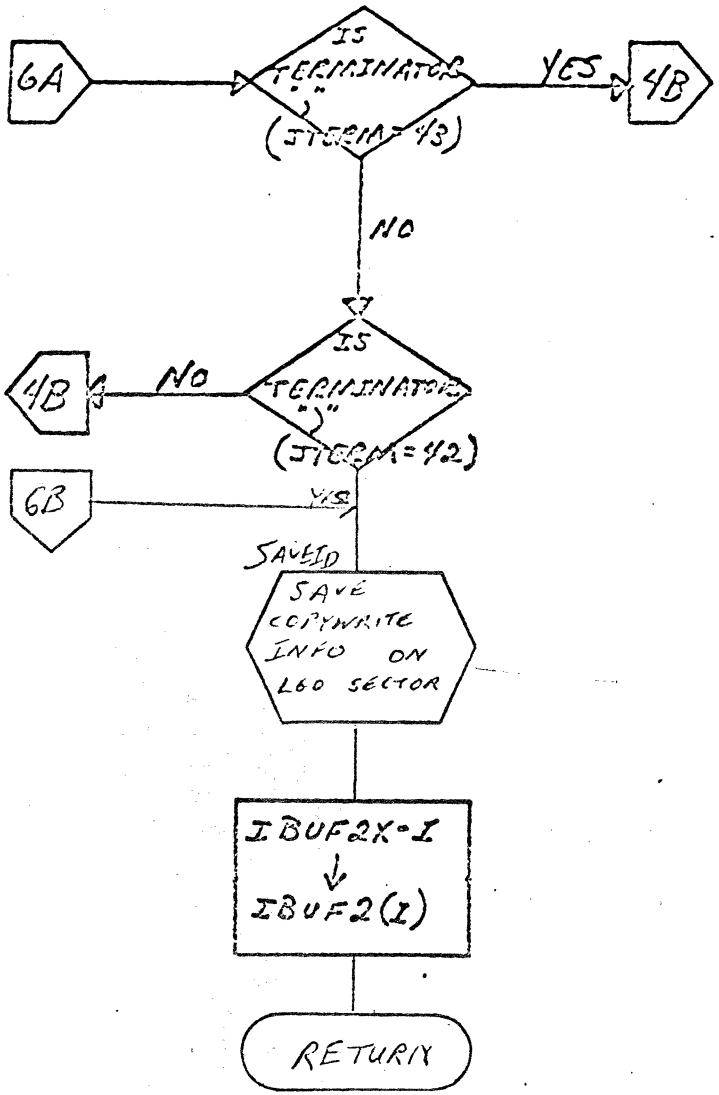


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE SUBPPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 4 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBPRR	PROJECT MGR.			
	PAGE 5 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED	DATE
	DOCUMENT TITLE SUDPPR		PROJECT MGR.			
	PAGE 6 OF 6		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

2.4.6 DATA Statements - Subroutine DATAPR

The DATAPR subroutine is used to process DATA statements. Intermediate language will be produced in the output buffer for the names and constants which appear in the source statement. The output buffer is named IBUF2. The tally register that holds the word count for the information placed in the output buffer is IBUF2X. IBUF2X is set to an initial value prior to entering the DATAPR subroutine. Intermediate language generated for each member of a name list in the source statement is inserted in the output buffer beginning at

IBUF2{IBUF2X}.

The tally register IBUF2X is increased by the number of words of intermediate language generated for the member being processed. Upon exit from DATAPR, IBUF2X contains the word length of the output entry.

2.4.6.1 Processing Name List

The DATAPR subroutine calls the GETF subroutine to extract a member of the name list from the source statement. A member of the name list must be alphanumeric. If the name does not appear in the symbol table, it is recorded therein by a call to the STORE subroutine. If the name is untyped, an indicator is set in the symbol table entry to the type number for the name:

{JESWT} → ITYPE{ISYMX}

where JEWST is set as a result of the call to GETF. Another indicator is set in the symbol table entry to show that this name appeared on a data statement:

1 → IDATAS{ISYMX}

If the symbol is unclassified, the indicator in the symbol table entry is set to show that this symbol is classified as a variable name:

1 → ICLASS{ISYMX}

If the entry containing this name indicates that the name is classified, it must be classified as a variable name. The variable may not be used as a byte, a dummy variable, or variable assigned to blank common.

If the name is an element of an array, a call is made to the CONSUB routine in order to compute the increment from the subscripts of the element. Upon return from

CONSUB, the value in IVCFLG indicates that the element is the IVCFLGth member of the array. If each element of the array occupies one storage cell,

{IVCFLG} - 1

is the increment in words which must be added to the base address of the array in order to reference the element in the array. Each element of the array occupies one storage cell if either -

1. The name is typed as an integer, or
2. the ASA option has been chosen and the name is typed as a single.

In other words

either {ITYPE{ISYMX}} = 1, or {IK} ≠ 0 and {ISNGL{ISYMX}} ≠ 0.

If each element of the array occupies 2 storage cells -

{IVCFLG}*2-2

is the increment in words which must be added to the base address of the array in order to reference the element in the array. Each element of the array occupies 2 storage cells if either -

1. The name is typed as real, or
2. the ASA option is chosen but the integer element is not typed as a SINGLE.

In other words -

ITYPE{ISYMX} = 2 or {ITYPE{ISYMX} = 1 and either {IK} = 0, or {IK} ≠ 0 and {ISNGL{ISYMX}} = 0

If each element of the array occupies 3 storage cells -

{IVCFLG}*3-3

is the increment in words which must be added to the base address of the array in order to reference the element in the array. Each element of the array occupies 3 storage cells if it is double precision regardless of the ASA option.

The members of the list of names must be separated by commas and the last name in the list must be followed by a slash. When DATAPR encounters the slash, it terminates processing the list of names and proceeds to process the list of constants. {Refer to item 2.4.6.2} Members of the name list may be variables, array elements, arrays, and implied DO's. When an array name is found, it is reduced into a series of array elements. When an

implied DO is analyzed, a series of array elements is also generated.

2.4.6.1.1 Intermediate Language Generated as a Result of Processing the Name List

The starting position in IBUF2 of the intermediate language is recorded in ITEMP1 prior to processing the 1st name in the list. As each member of the list of names is processed, the following intermediate language is generated:

WORD 1: {IBUF2{IBUF2X}} = pointer to the name in the symbol table.

WORD 2: {IBUF2{IBUF2X+1}} =

1. 0 if the variable is not an element of an array,
2. {IVCFLG}-1 if the variable is an element of an array where each element of the array occupies 1 storage cell,
3. IVCFLG*2-2 if the variable is an element of an array where each element of the array occupies 2 storage cells, and
4. IVCFLG*3-3 if the variable is an element of an array where each element of the array occupies 3 storage cells.

WORD 3: {IBUF2{IBUF2X+2}} = the number of storage cells the variable occupies {according to its type}.

The tally register is increased by the number of words of intermediate language to be generated for this name

{IBUF2X}+2+{IBUF2{IBUF2X+2}} → IBUF2X

When the DATAPR subroutine encounters a slash indicating the end of the name list, an end of output mark is generated in the intermediate language buffer by

-1 → IBUF2{IBUF2X}

2.4.6.1.2 Error Checking While Processing Name List

Diagnostics will be given for the following variables:

1. A member of the name list is not alphanumeric.
2. A member of a name list appears in the symbol table as something other than a class 1 variable.

3. The member of the name list appears in the symbol table, but an indicator is set to show this symbol is a byte.
4. The member of the name list appears in the symbol table, but an indicator is set to show this symbol is either in numbered common or used as a dummy variable.
5. Either the members of the name list are not separated from each other by commas, or the final member in the list is not followed by a slash.

Errors 1 and 5 will cause processing of the source statement to be terminated by an exit from DATAPR. For the remaining errors, processing of the source statement is resumed following the output of the error indication.

2.4.6.2 Processing Constant List

In a DATA statement, there must be a one to one correspondence between the members of the constant list and the members of the preceding name list. The members of the constant list may be numeric and literal. A literal constant must be contained by single quotes. Numeric constants may be double, real, or integer and may have a leading algebraic sign of "+" or "-". If the value of the constant is to be assigned to consecutive variables in the preceding name list, it may be preceded by a repeat count equal to the number of times it is to be used. The repeat count for the constant must be an integer, and it must be separated from the constant by an asterisk. In order to comply with the one to one correspondence rule as stated above, the following condition must be satisfied when a member of a constant list is preceded by a repeat count:

C+RC=V

where

1. C represents the number of constants in the list without repeat counts,
2. RC represents the sum of the repeat counts occurring in the constant list, and
3. V represents the number of variables in the preceding name list.

When using literal constants, the number of characters must be taken into consideration. If the variable type is:

Single or Integer	it holds	2 characters
Real	it holds	4 characters
Double Precision	it holds	6 characters

Trailing blanks are used to fill a variable entry that has been started and an end of literal quote is encountered, that is:

D is a double precision variable.

```
DATA D/'ABC'/
```

The variable D will be initialized as if the DATA statement were:

```
DATA D/'ABCbbb'/     b = blanks
```

Since a numeric constant may be double precision, real or integer, it must agree in type with the variable in the preceding name list to which it is assigned.

The members of the constant list must be separated from each other by commas. The final member of the list must be followed by a slash which is followed by either a comma or an EOS. If the list terminates with a slash and an EOS, processing of DATA statement terminates when processing of the current constant list terminates. If the list terminates with a slash and a comma, DATAPR proceeds to process the list of names which follow in the manner describe by item 2.4.6.1.

2.4.6.2.1 Completing the Intermediate Language in the Output Buffer

The tally register IBUF2X is set to the value recorded in ITEMP1 prior to processing the list of names preceding the list of constants. (See item 2.4.6.1.1) The DATAPR subroutine calls GETF to extract a member of the list of constants from the source buffer. Prior to calling GETF, the repeat ICOUNT is set to 1. Upon return from GETF, one of the following conditions has occurred:

CONDITION 1: The operand is not preceded by a repeat count, or a quote, or a leading algebraic sign. Upon return from GETF -

JTERM = comma, slash, or EOS
JMODE = 3 for integer constant, 5 for real constant and 6 for double precision constant
JSYM(1) = value of integer constant
JSYM(1) and JSYM(2) = value of real constant
JSYM(1), JSYM(2) and JSYM(3) = value of double precision constant

The numeric constant and the name from the preceding list to which the constant is assigned must agree in type such that -

CONSTANT	JMODE		IBUF2{IBUF2X+2}
double precision	6	agrees with	3
real	5	agrees with	2
integer	3	agrees with	1

The constants are entered into the intermediate language by -

integer	JSYM{1} → IBUF2{IBUF2X+2}
real	JSYM{1}, JSYM{2} → IBUF2{IBUF2X+2}, IBUF2{IBUF2X+3}
double precision	JSYM{1}, JSYM{2}, JSYM{3} → IBUF2 {IBUF2X+2}, IBUF2{IBUF2X+3}, IBUF2 {IBUF2X+4}

The tally register is increased by 3, 4, 5 accordingly.

CONDITION 2: The operand is preceded by a leading algebraic sign. Upon return from GETF -

JTERM = either "+" or "-"
JMODE = 0 indicating a null field

A second call is made to GETF to extract the numeric constant from the source statement. If the algebraic sign was a "-", the value for the constant is complemented by:

- JSYM{1} → JSYM{1},
- JSYM{2} → JSYM{2},
and - JSYM{3} → JSYM{3}.

If the leading algebraic sign was a "+", the sign is ignored. The constant is then processed as if it had no leading algebraic sign. {See condition 1}

CONDITION 3: The constant is preceded by a repeat count. The repeat count must be an integer separated from the constant by an asterisk. Upon return from GETF -

JTERM = "*", JMODE = 3, and JSYM{1} = value of repeat count

The repeat count is recorded in ICOUNT by -

JSYM{1} → ICOUNT

A second call is made to GETF to obtain the constant. If it is a literal constant, it is processed as condition 5. The numeric constant is assigned to a number of

consecutive names from the preceding name list until ICOUNT = 0. The constant must match "type wise" each of the consecutive names to which it is to be assigned. The constant is inserted into the intermediate language in IBUF2 for each of the names in the manner described by condition 1. IBUF2X is increased by the number of words of intermediate language for each entry made in IBUF2. The repeat count is reduced by 1 for each entry made into IBUF2. When ICOUNT = 0, the repeat conditions are satisfied. The repeat count must not be so large that it is not zero when the end of the name list is encountered in IBUF2. The end of the intermediate language is marked by an entry in which:

IBUF2{IBUF2X} = -1

A flag IVBUF is checked to see if the name list has been exhausted. If IVBUF = 1 then more intermediate language will be generated and IBUF2 will be written out and refilled. If IVBUF = 0 and the end of the intermediate language is encountered, then the end of the name list will have been reached. {See item 2.4.6.1.1}

CONDITION 4: The numeric constant is preceded both by a repeat count and has a leading algebraic sign. First, the repeat count is processed according to condition 3. Then the leading algebraic sign is processed according to condition 2.

CONDITION 5: The operand is preceded by a single quote, indicating a literal constant. Literal constants must have a terminating single quote. Upon return from GETF -

JTERM = "'" single quote
JMODE = 0 indicating a null field

The routine GETC is used to extract characters from the source statement. Characters are extracted until a terminating quote is encountered.

The characters are entered into the intermediate language according to the following conditions.

Type of NAME	No. of CHARACTERS
Integer	2
Real	4
Double precision	6

Characters are assigned to consecutive names until the literal is terminated. If assignment has begun and the literal terminates, the remainder of the name is assigned with blanks. (That is, if a name, whose type is real, was assigned a character and the end of literal was encountered 3 blanks are assigned to fill up the entry.) The intermediate language pointer is updated after each name assignment by:

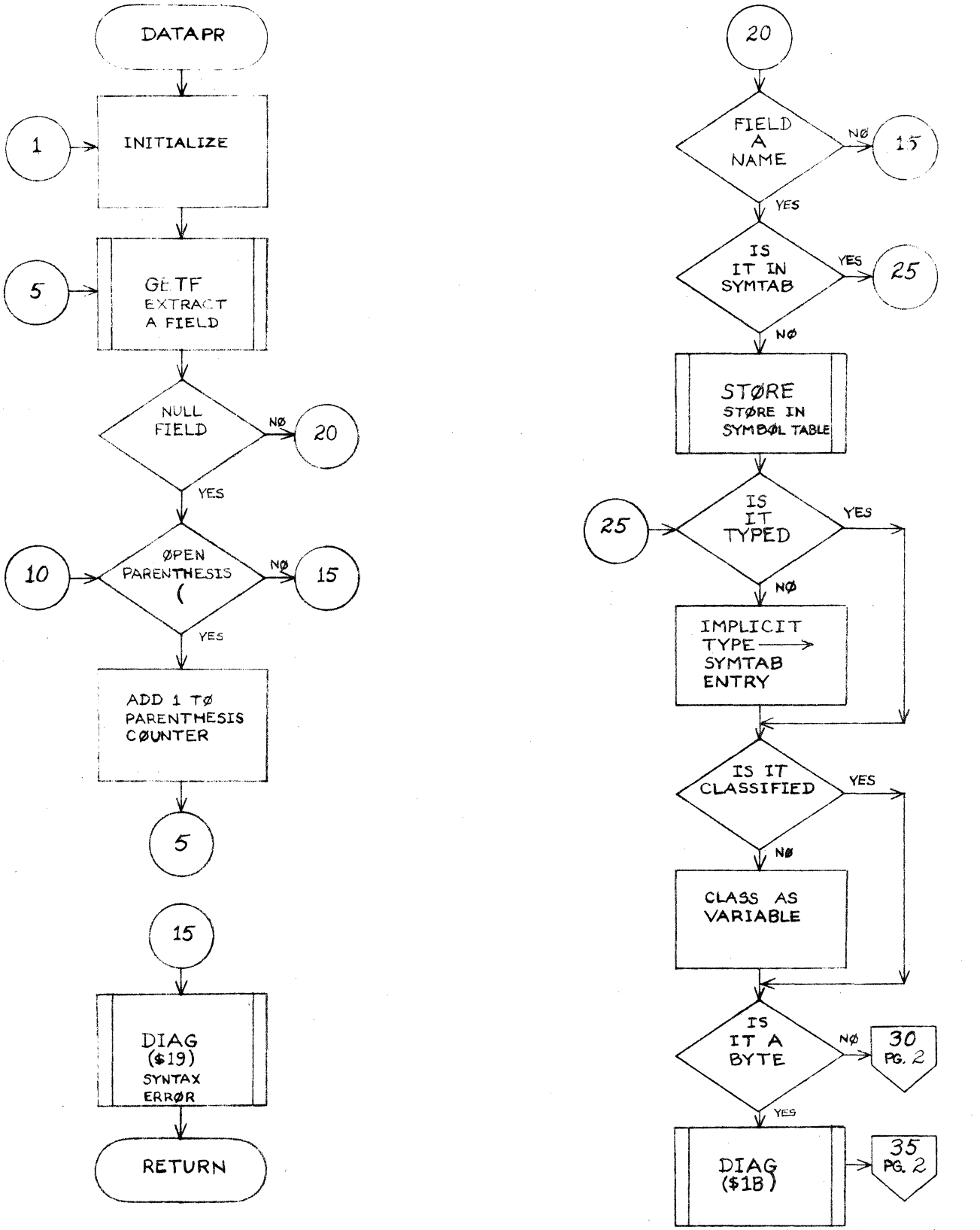
Type of Name → IENTRY
IBUF2X+IENTRY+2 → IBUF2X

2.4.6.2.2 Error Conditions Check for While Processing Constant List

Diagnostics will be given for the following errors:

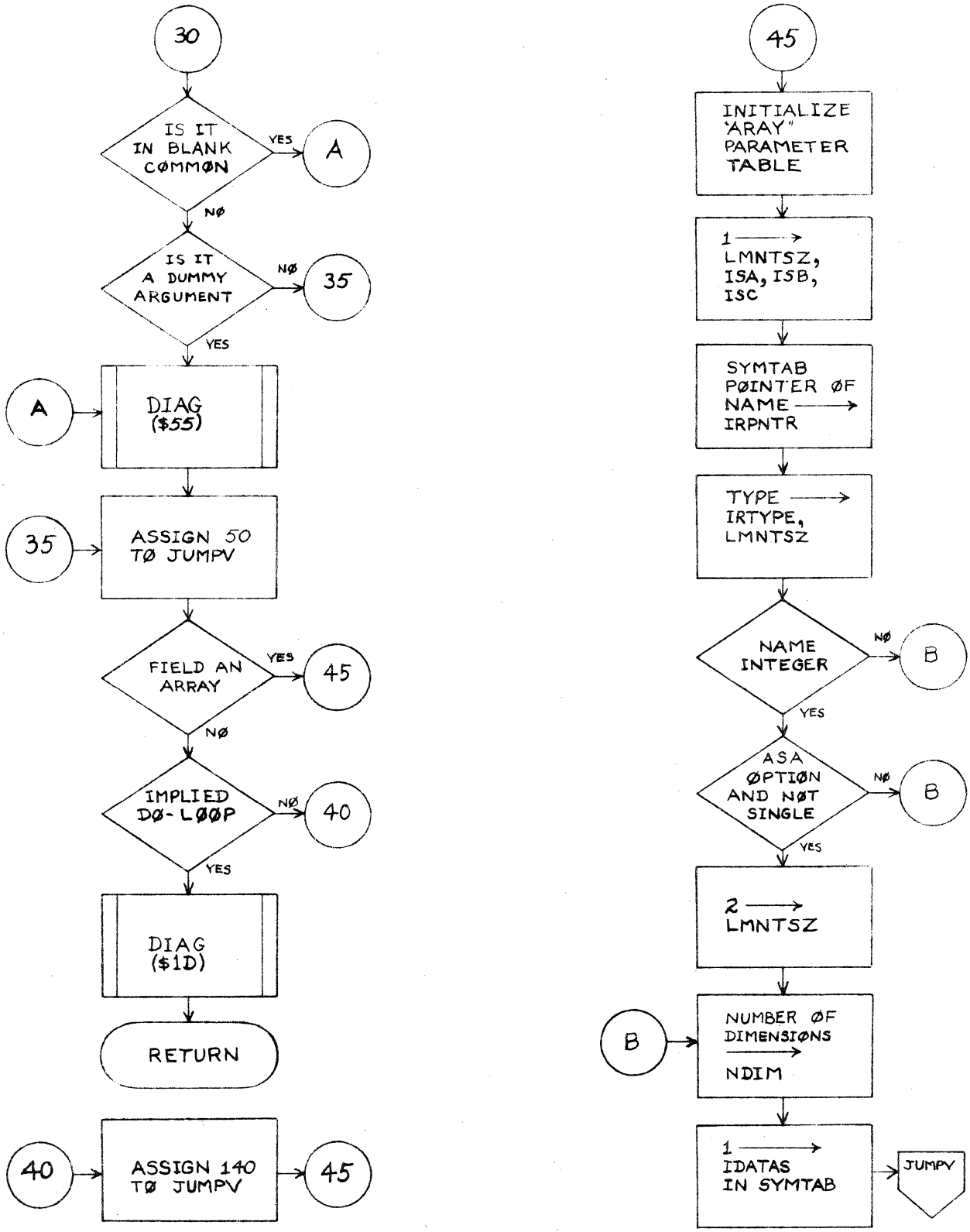
1. The repeat counter is either non-integer or is ≤ 0 or both.
2. A member of the constant list is both non-numeric and not a literal.
3. The numeric constant does not agree in type with the name (from the preceding name list) to which it is assigned.
4. There is not a one to one correspondence between the members of the name list and the members of the constant list. This could be caused by a repeat count which is excessive in value, it could also be caused by an incorrect number of characters in the literal constant.
5. The last member of the constant list is not followed by either a "/" or a "/EOS".
6. The members of the constant list are not separated by commas.
7. A member of the constant list is followed by an EOS not preceded by a slash.

Errors 1, 4, 5, 6 and 7 will cause processing of the source statement to be terminated by an exit from DATAPR. Processing of the source statement will be resumed following the printing of the error indication for errors 2 and 3. For illegal numeric constants, zeros will be substituted in their place.



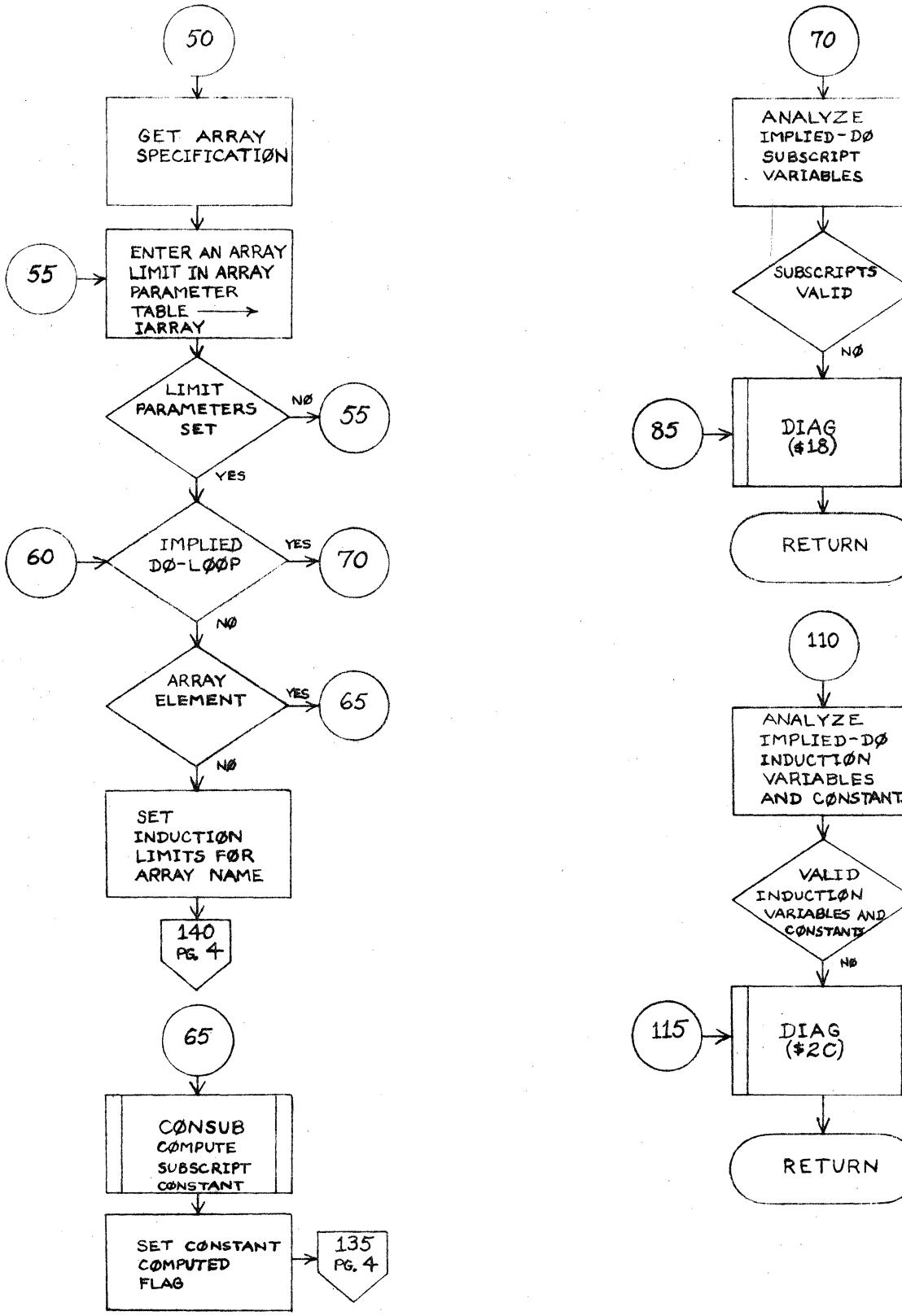
TITLE DATAPR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 1 OF 11

CSD 203



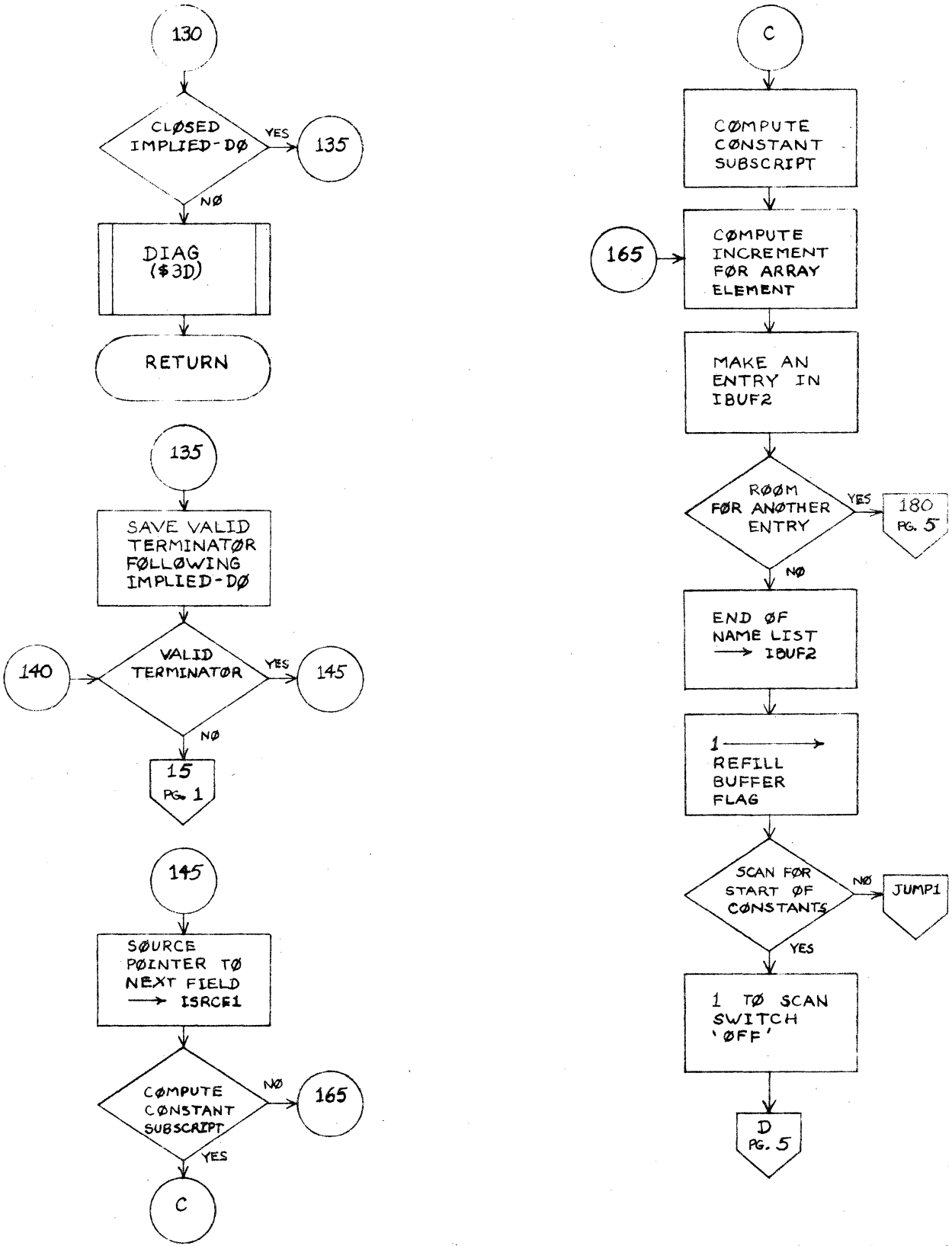
TITLE	DATAPR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2 OF 11

CSD 203



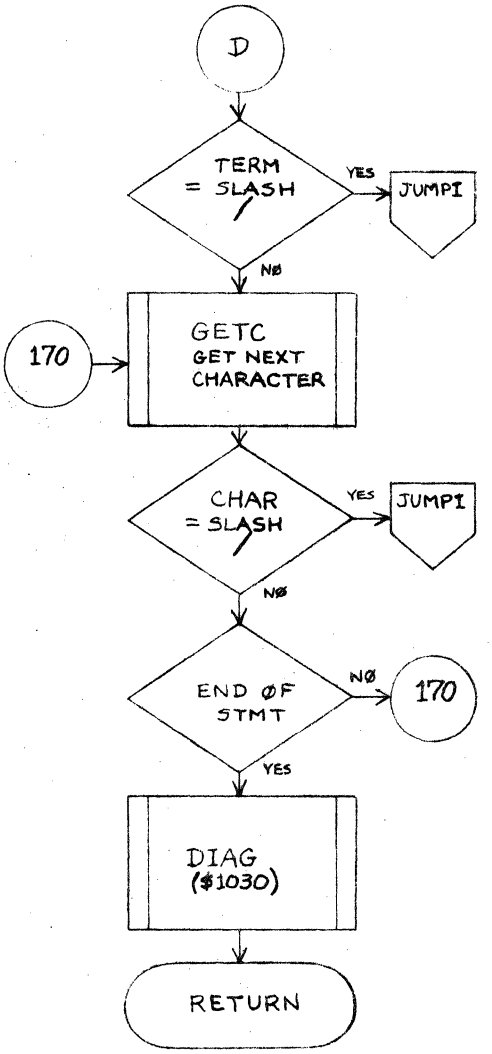
TITLE	DATAPR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF 11

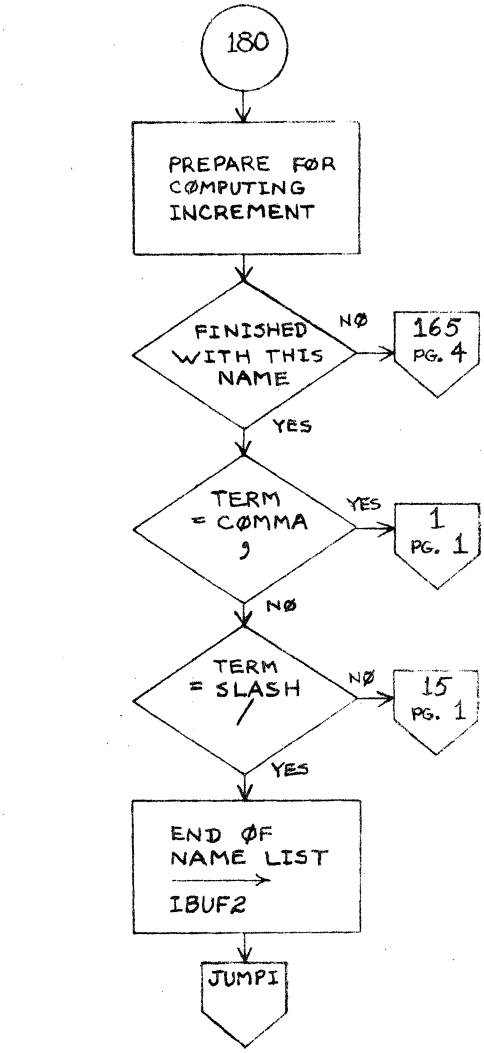
CSD 203

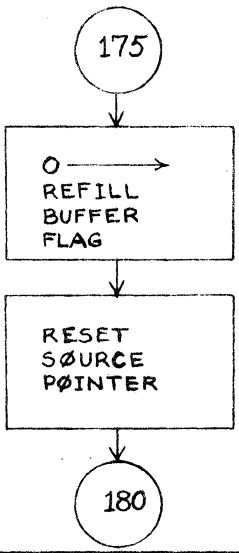


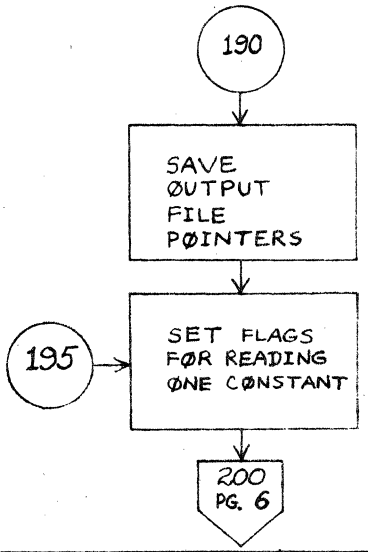
TITLE	DATAPR	DRG. NO.
		REVISION
DRAWN BY	PROJ.	DATE
		SHEET 4 OF 11

CSD 203





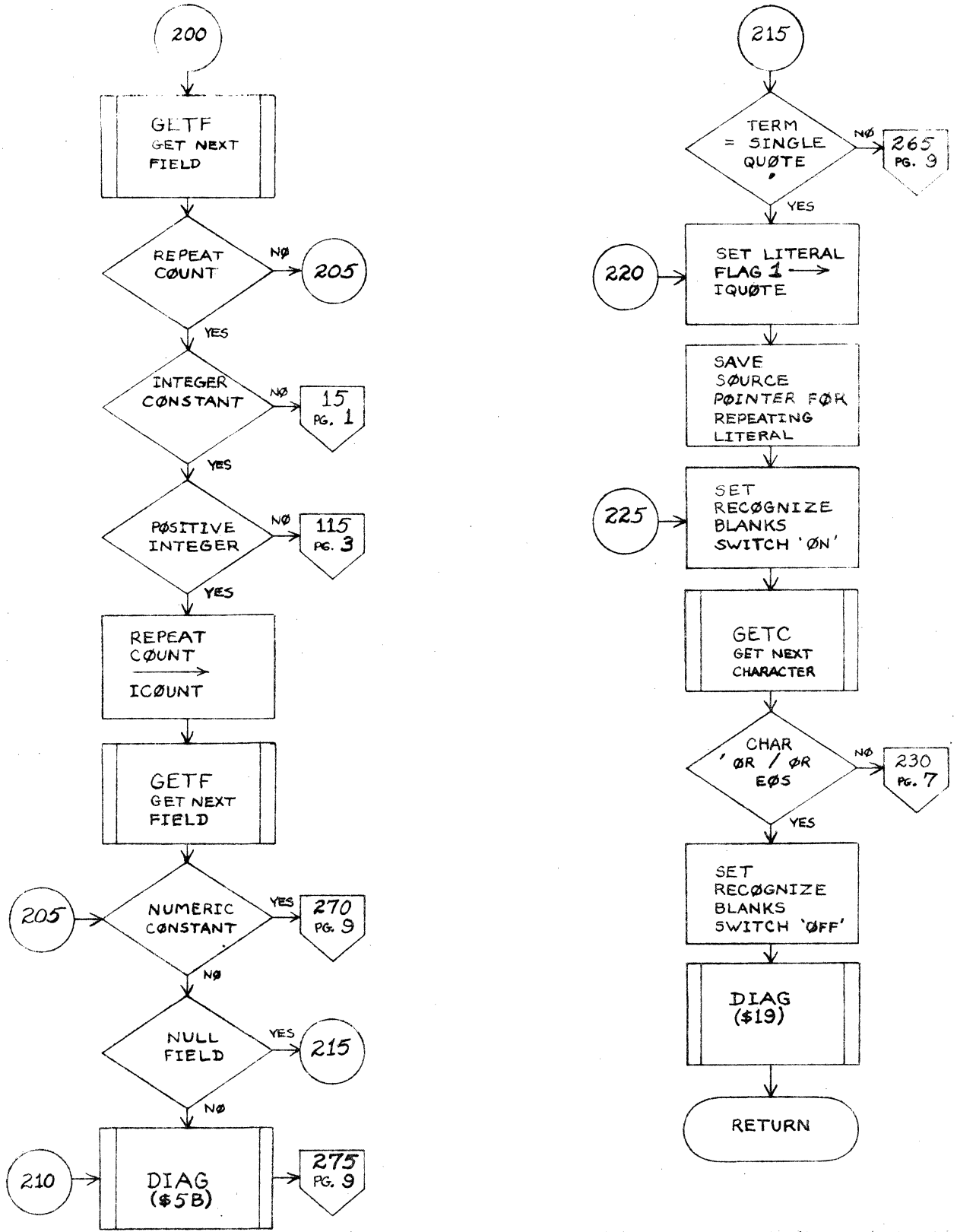




TITLE	DATAPR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 5 OF 11

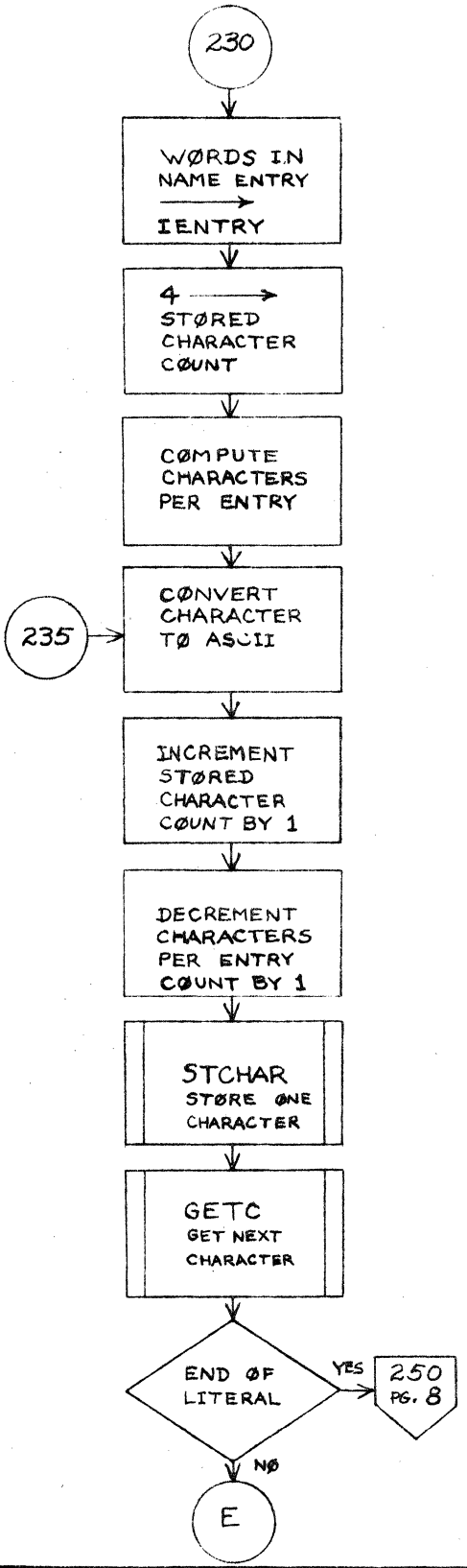
CONTROL DATA

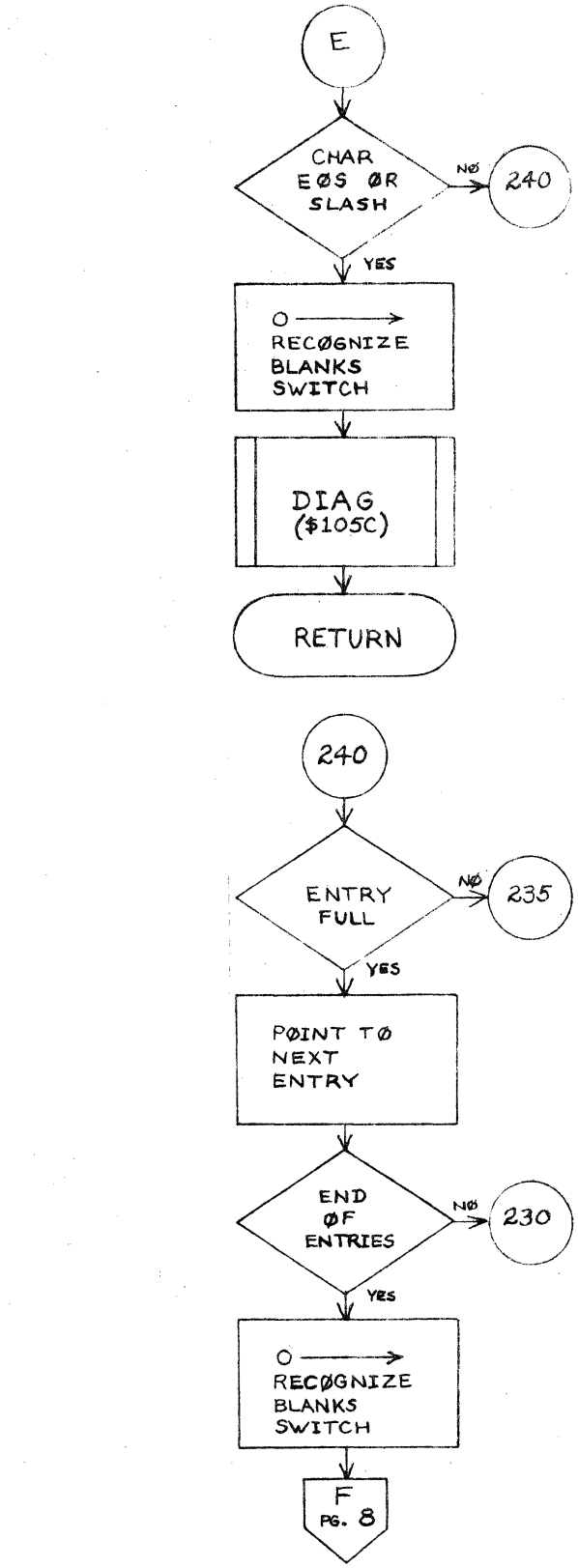
LA JOLLA FACILITY



TITLE	DATAPR	DRG. NO.
		REVISION
DRAWN BY	PROJ.	DATE
		SHEET 6 OF 11

CSD 203



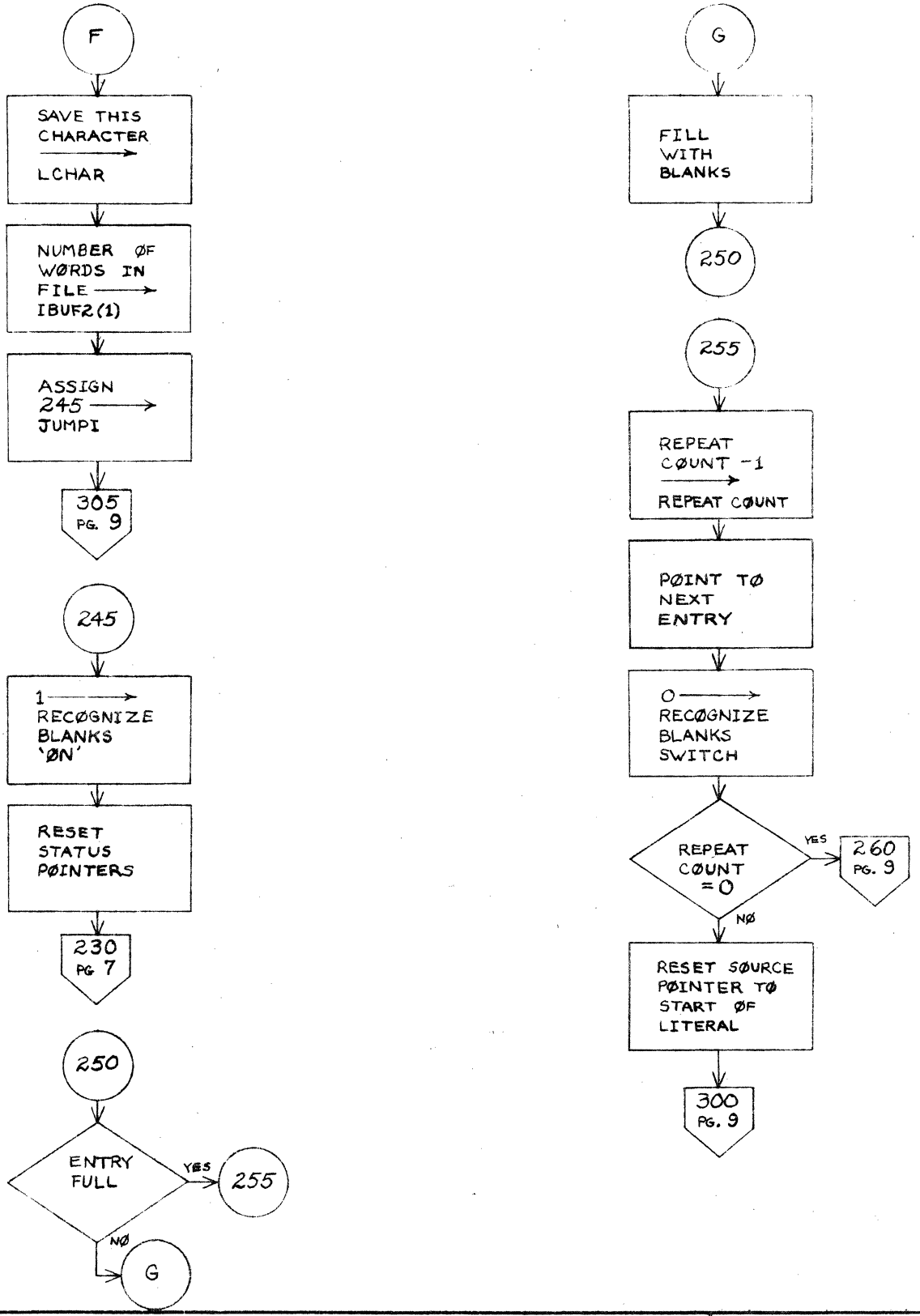


TITLE	DATAPR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 7 OF 11

CSD 203

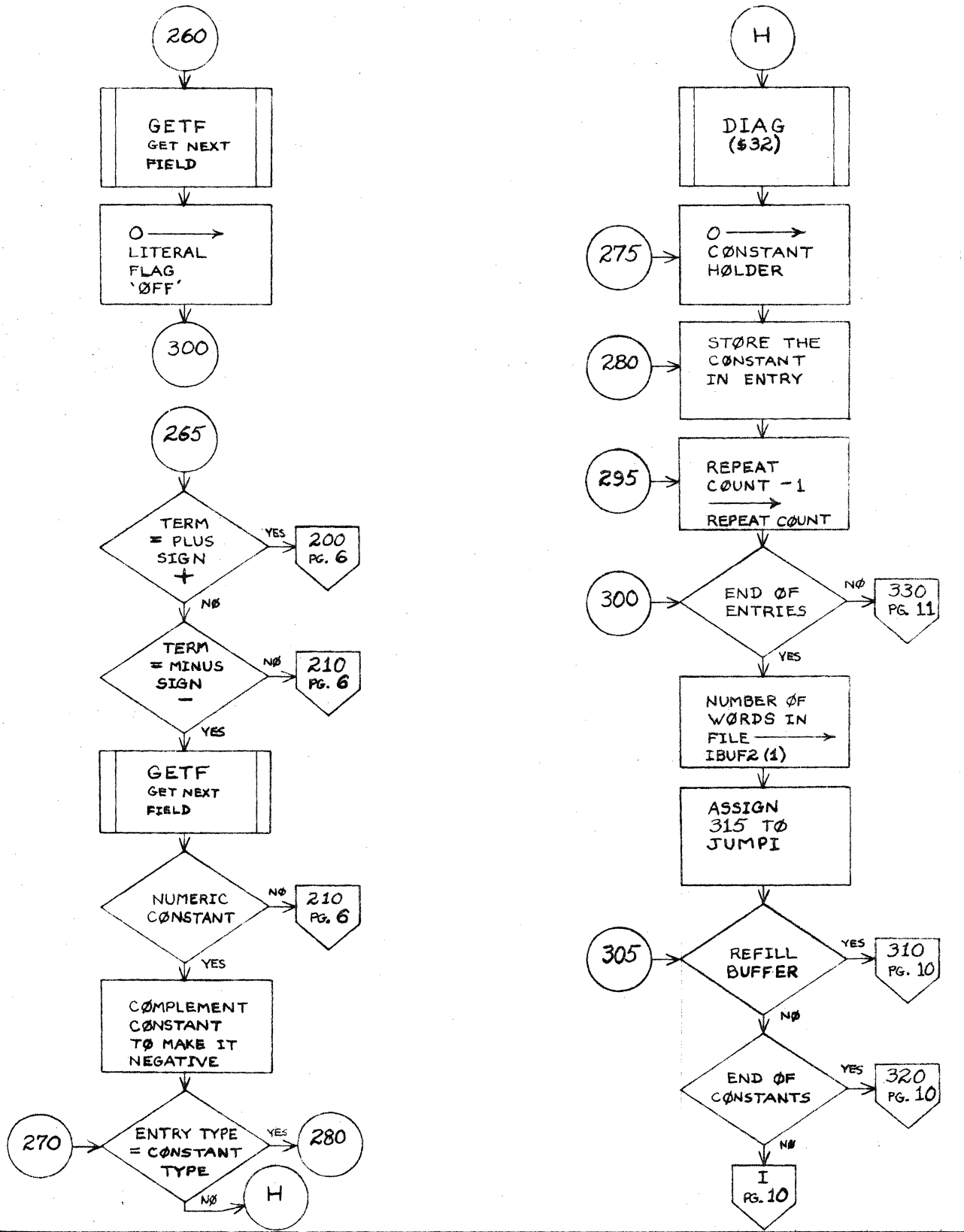
CONTROL DATA

LA JOLLA FACILITY



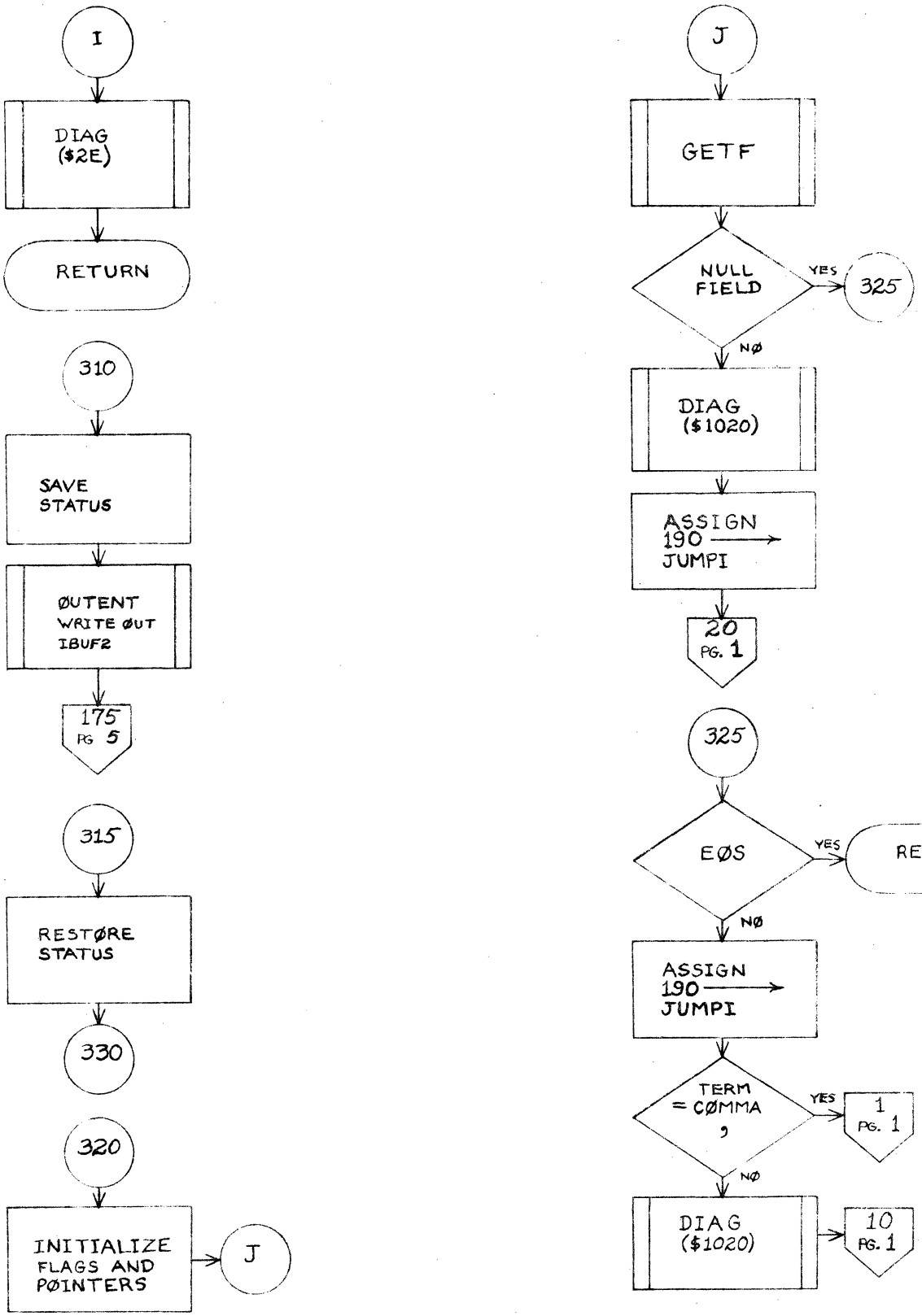
TITLE			DRG. NO.
DATAPR			REVISION
DRAWN BY	PROJ.	DATE	SHEET 8 OF 11

CSD 203



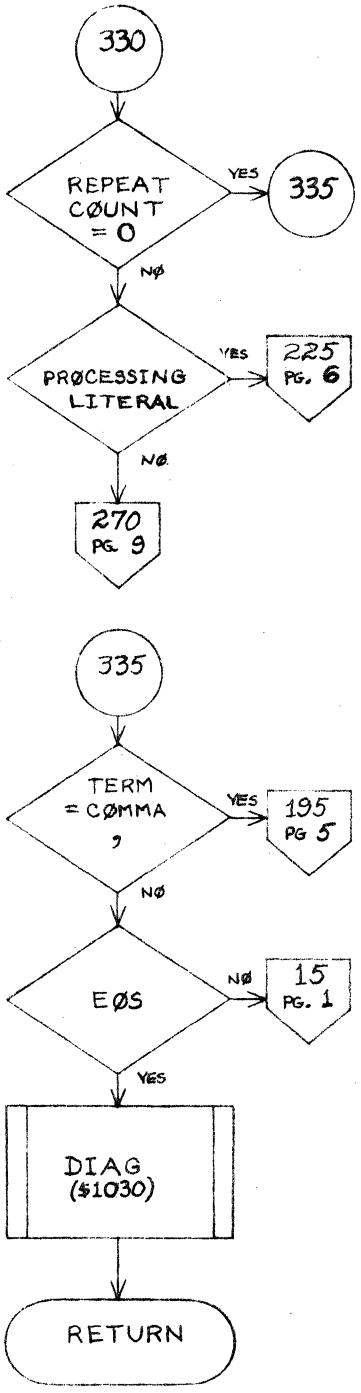
TITLE	DATAPR	DRG. NO.
		REVISION
DRAWN BY	PROJ.	DATE	SHEET 9 OF 11

CSD 203



TITLE	DATAPR	DRG. NO.
		REVISION
DRAWN BY	PROJ.	DATE	SHEET 10 OF 11

CSD 203



TITLE	DATAPR			DRG. NO.
				REVISION
DRAWN BY	PROJ.	DATE	SHEET 11	OF 11

CSD 203

2.4.7 FORMAT Statements

2.4.7.1 Subroutine CHECKF

The routine CHECKF is entered. The statement is syntax checked, but remains a string of ASCII characters. It is noted whether E or F conversion is used so entry points for these conversions will or will not be created.

CHECKF is a routine to syntax check a FORMAT statement. If it discovers an error in the format specifications CHECKF calls DIAG, the diagnostic routine.

INPUT

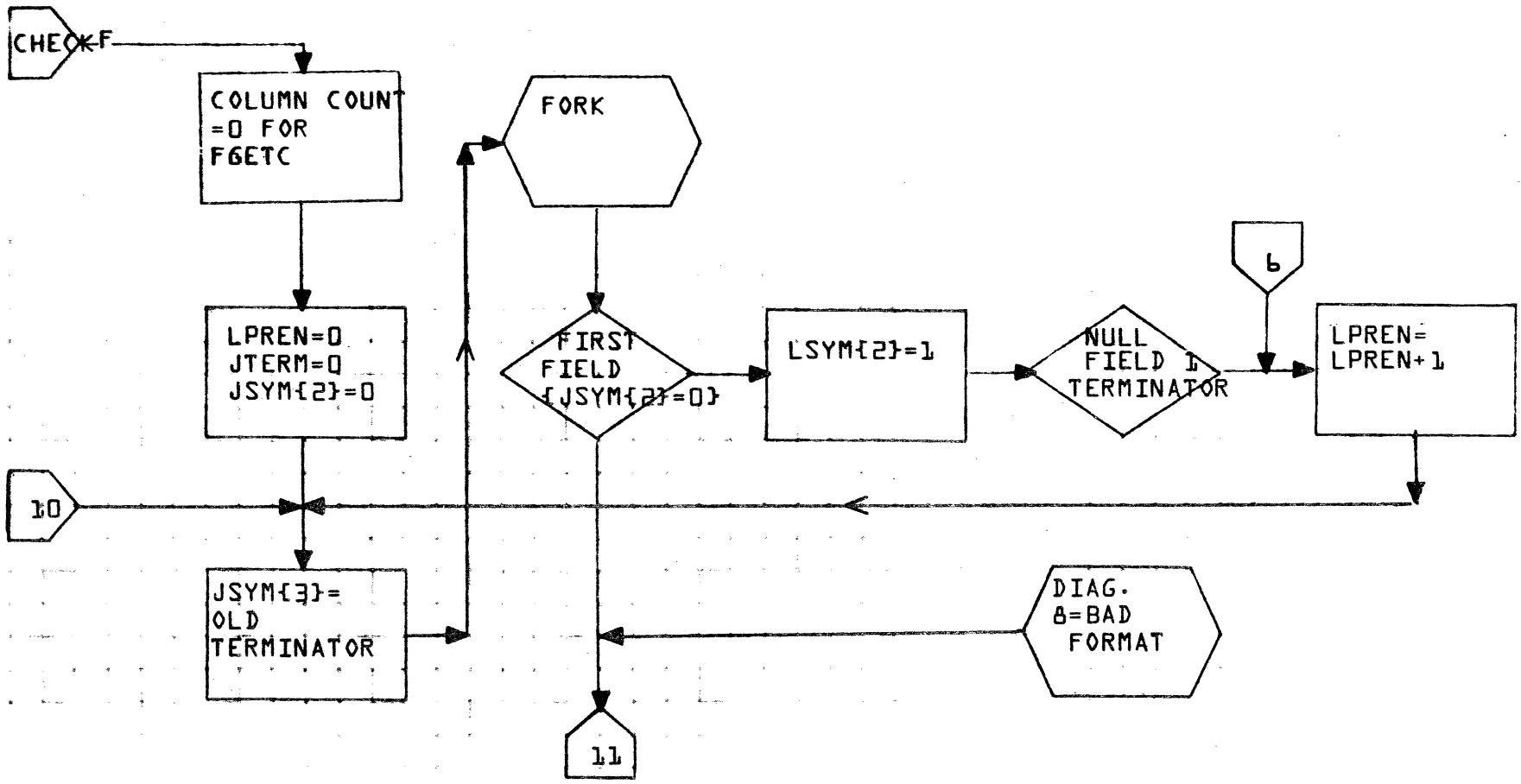
CHECKF uses the FORK routine to check fields in the FORMAT statement. The first field it receives should always be null with a (terminator (the leftmost left parenthesis). The field preceding the last must always terminate with); the last field must be null with an end of statement terminator. CHECKF assumes that each successive call to FORK picks up the next field scanning left to right, in the FORMAT statement.

OUTPUT

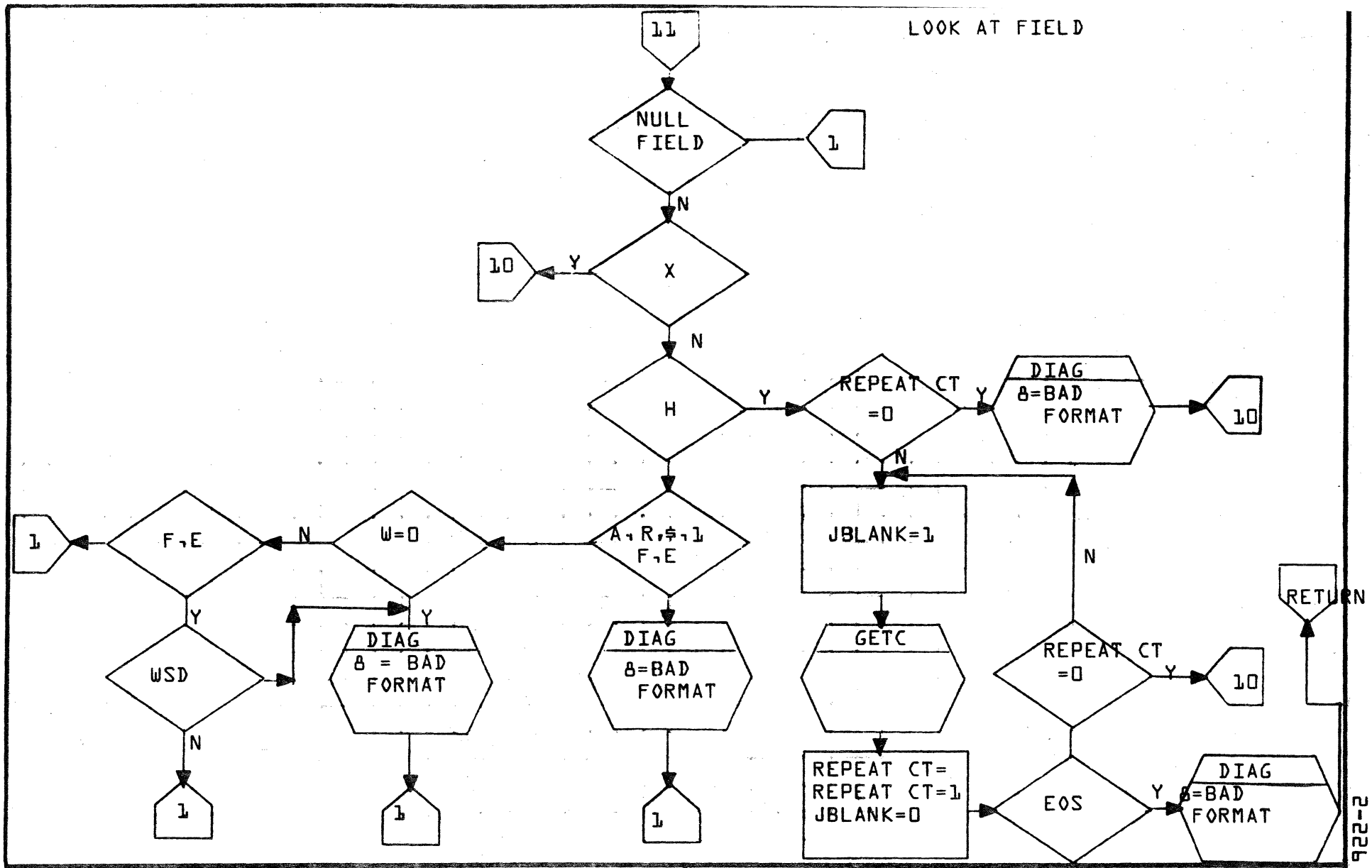
As the check proceeds FORK (using FGETC) converts the FORMAT statement to external code (BCD or ASCII) and places it into IBUF2 starting at IBUF2(6). The last character placed in IBUF2 (banning an error) will be the terminating right parenthesis. The FORTRAN statement placed in IBUF2 will not include extraneous blanks or the EOS character.

Upon exit IBUF2(X) will be at the number of words in IBUF2.

Subroutines used: FORK, FGETC, GETC, DIAG



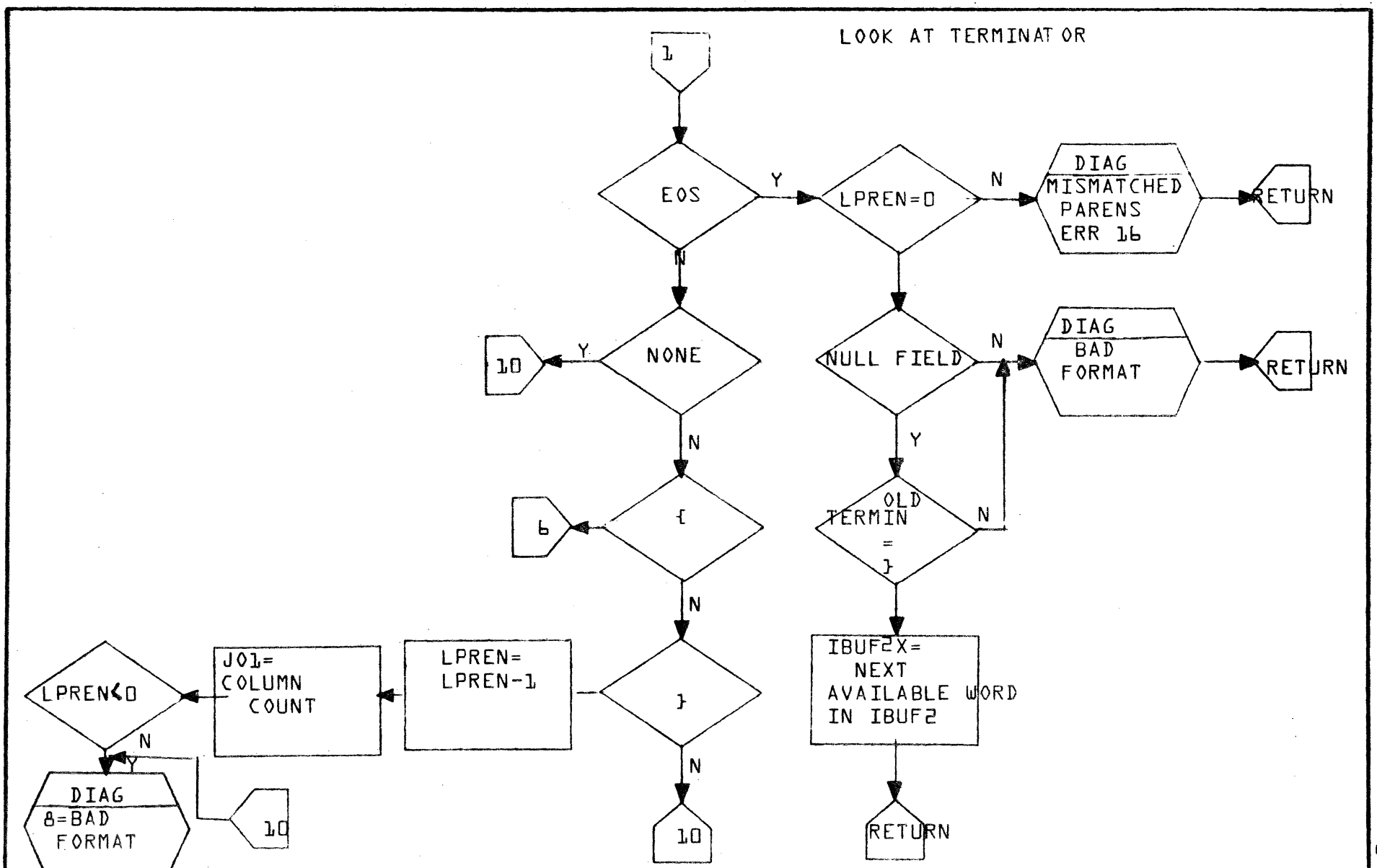
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE CHECKF	PROJECT MGR.			
	PAGE 1 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS — MACH. TYPE: 1700	PROJECT NO.	REV / APPROVED / DATE
	DOCUMENT TITLE: CHECKF	PROJECT MGR.	
	PAGE 2 OF 3	PROJECT NAME	
	NUMBER — ISSUE DATE	TASK NO.	
	DRAWN BY — DATE	TASK NAME	

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: CHECKF		PROJECT MGR.		
		PAGE 3 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

2.4.7.2 Format Handling Routines - FORK

Syntax checks a format statement. Input parameters: holders for output.

Output parameters:

1. I1 field type

   1 = null or repeat count only
   2 = X
   3 = H
   4 = A
   5 = R
   6 = $
   7 = I
   8 = f
   9 = E
  10 = Error

2. I2 field terminator

   0 = none (X,H fields)
   1 = (
   2 =)
   3 = ,
   4 = /
 -13 = E05

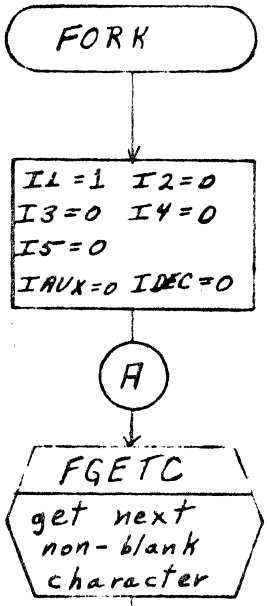
3. I3 repeat count

4. I4 = W

5. I5 = D

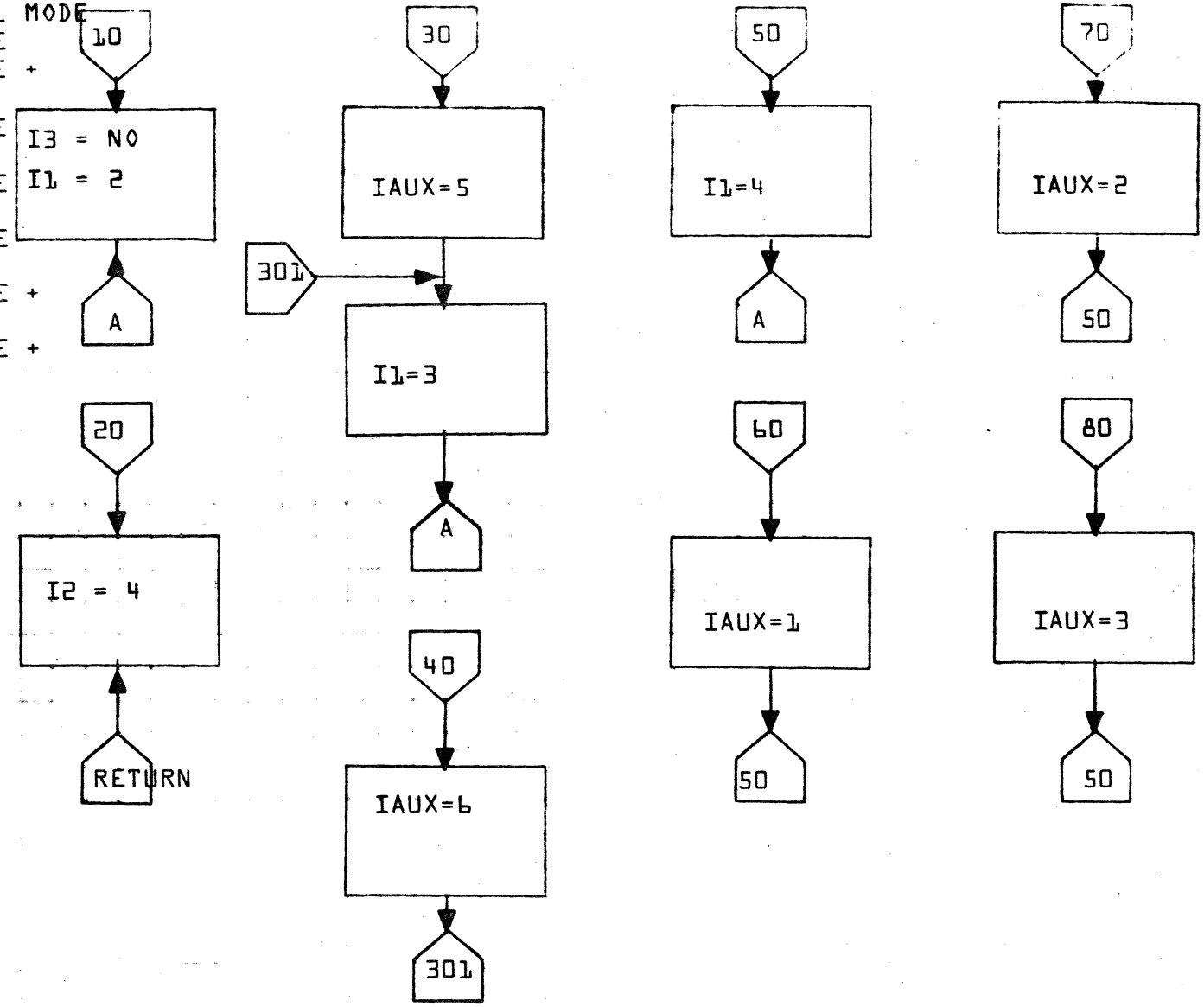
Modes: 1 = initial
2 = repeat
3 = f, E aux switch = 5,6
4 = A,R,$,I
5 = error aux switch = 0,1,2,3

Subroutines used: FGETC



0-9	EOS	F	F	A	R		$	I	X	H	.	(	)	,
10	20	30	40	50	60	INITIAL (I1=1)	70	80	90	100	120	150	140	A1
150	160	30	40	50	60	REPEAT (I1=2)	70	80	90	100	120	130	180	190
200	220	120	120	120	120	F, E (I1=3)	120	120	120	120	260	270	280	290
240	230	120	120	120	120	A, R, $, I (I1=4)	120	120	120	120	120	130	140	210
A	230	A	A	A	A	ERROR (I1=5)	A	A	A	A	A	130	140	210

10. NUMBER IN INITIAL MODE
20. / IN INITIAL MODE
30. A IN INITIAL MODE + REPEAT MODE
40. E IN INITIAL MODE + REPEAT MODE
50. A IN INITIAL MODE + REPEAT MODE.
60. R IN INITIAL MODE + REPEAT MODE
70. $ IN INITIAL MODE + REPEAT MODE
80. I IN INITIAL MODE + REPEAT MODE



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FORK		PROJECT MGR.			
		PAGE 2 OF 8	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

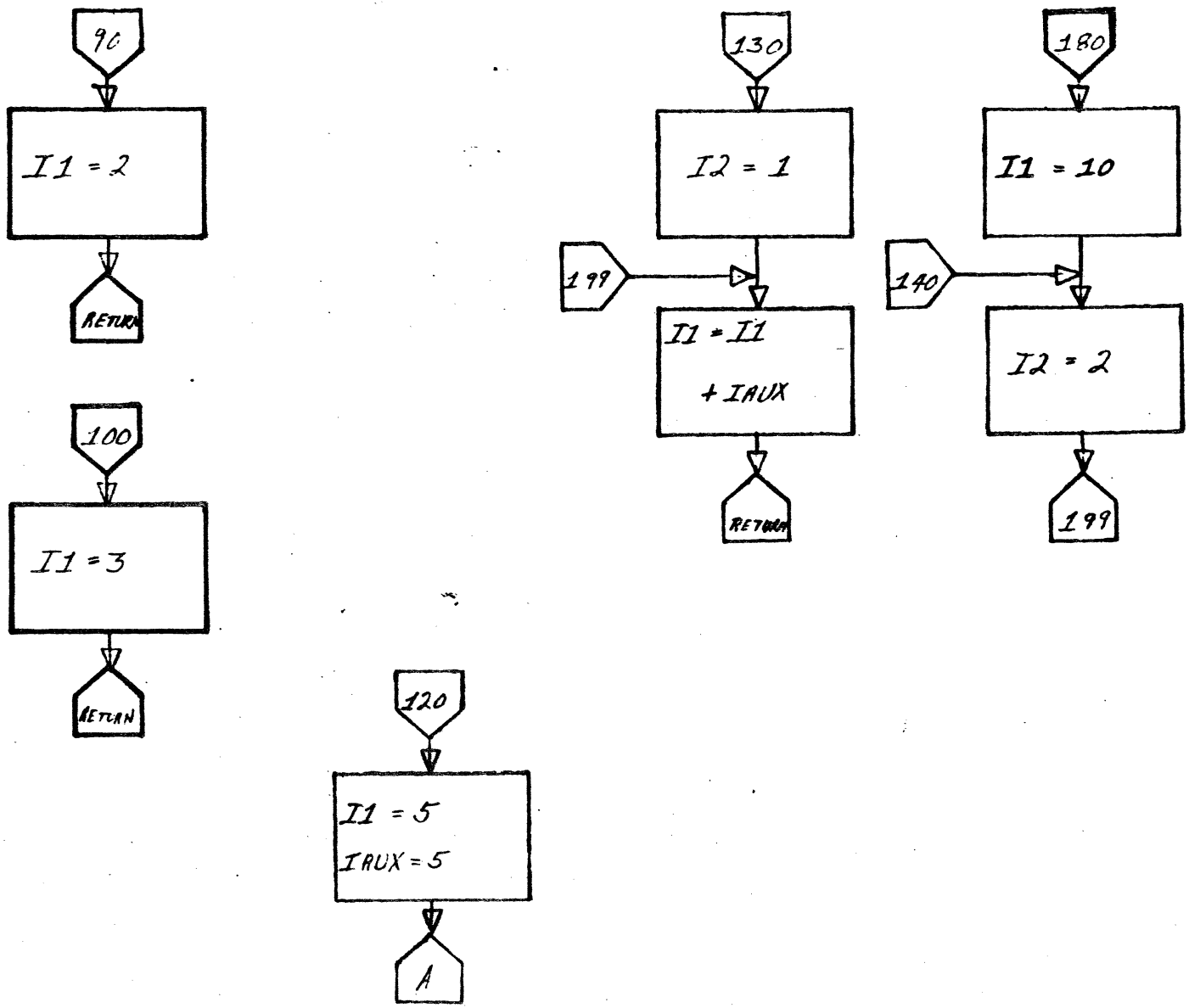
90. X IN INITIAL MODE + REPEAT MODE

100. H IN INITIAL MODE + REPEAT MODE

120. ERROR

130. (ALL MODES EXCEPT F, E

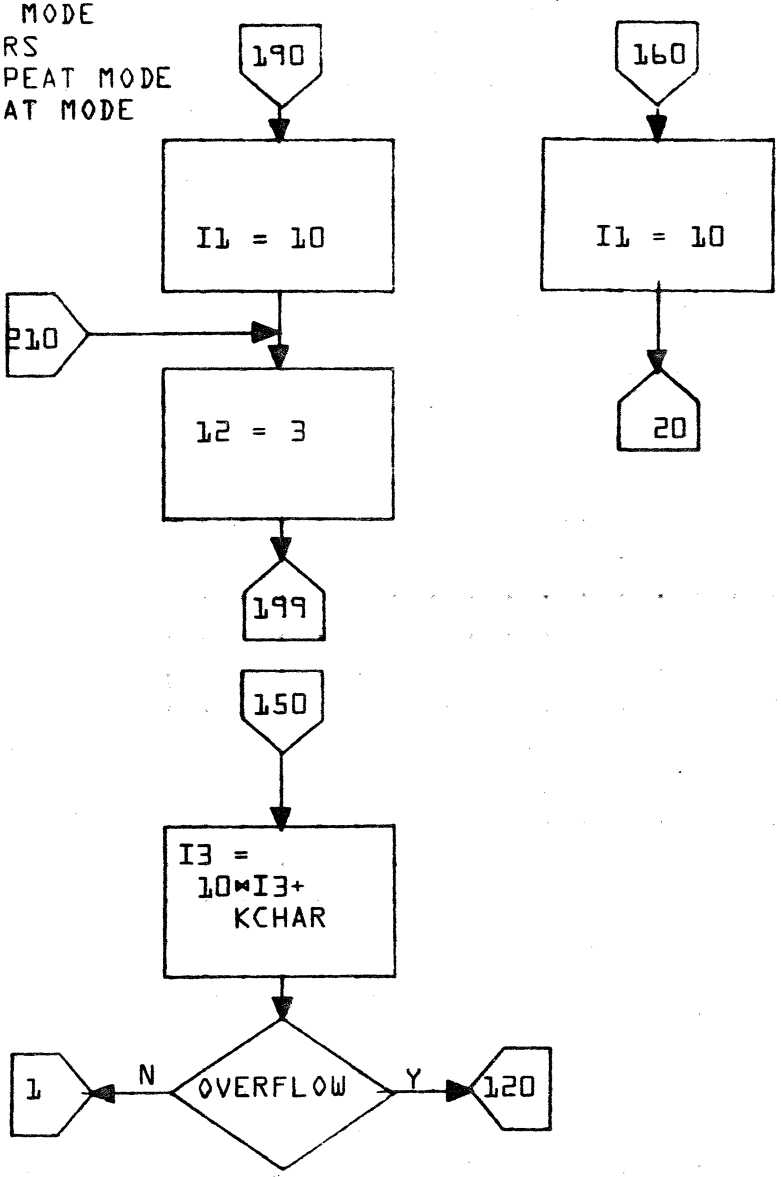
180.) REPEAT MODE
) ALL OTHERS EXCEPT F, E



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☑ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE FORK		PROJECT MGR		
		PAGE 3 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

1231-2

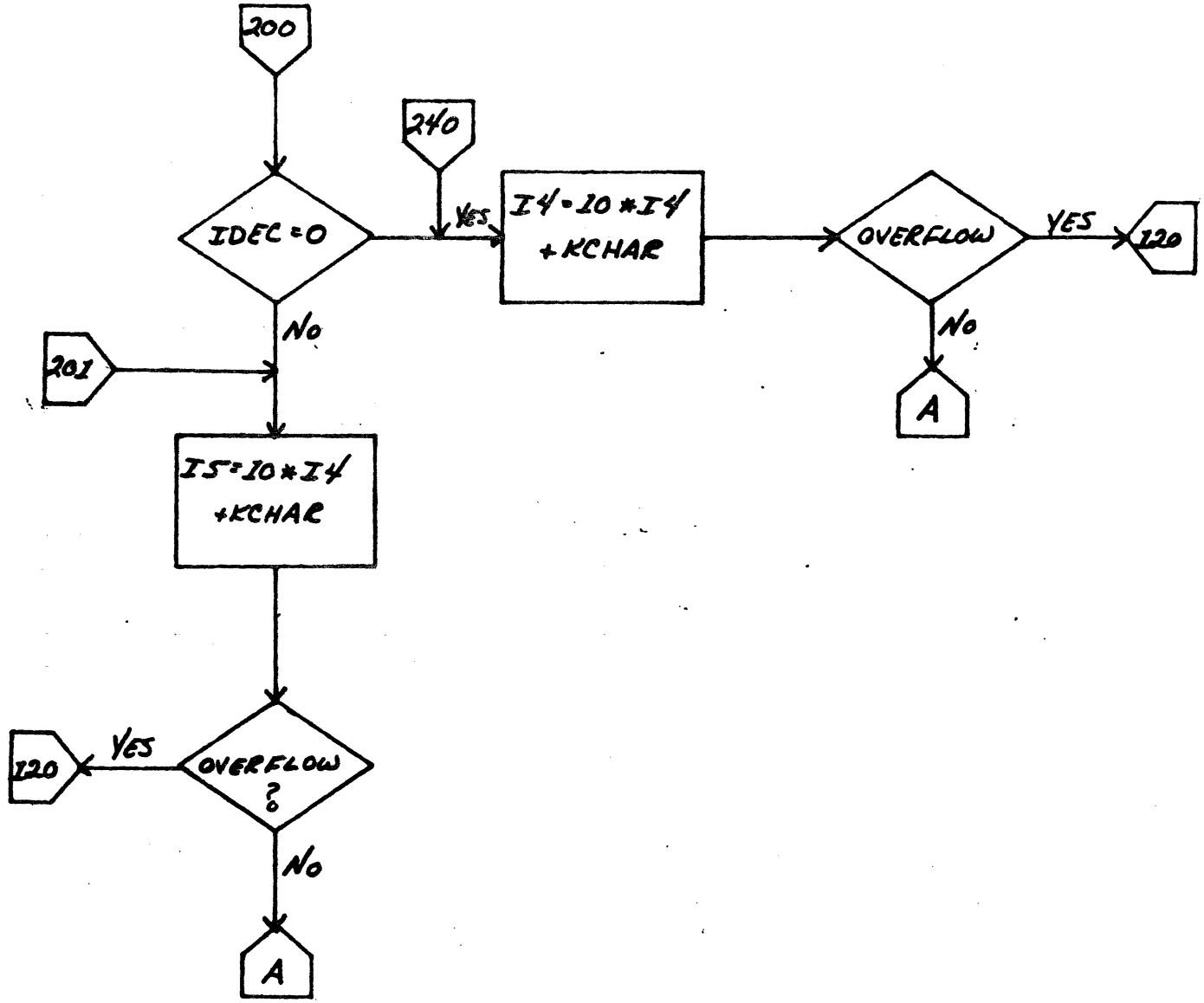
190. }IN REPEAT MODE
}ALL OTHERS
150. 0-9 IN REPEAT MODE
160. / IN REPEAT MODE



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE FORK		PROJECT MGR.		
		PAGE 4 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1388 (FORMERLY CA127-1)

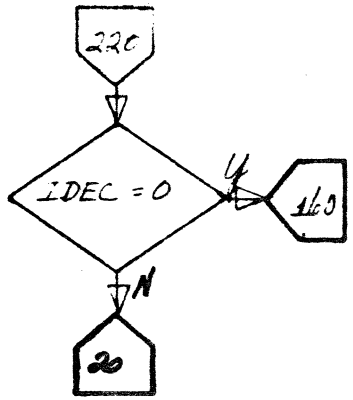
PRINTED IN U.S.A.

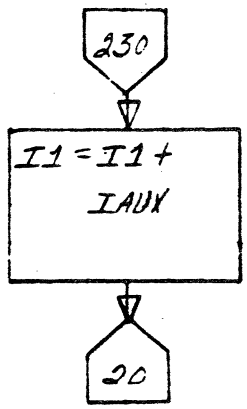


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE FORK		PROJECT MGR.			
FLOWCHART ☒		PAGE 5 OF 8	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

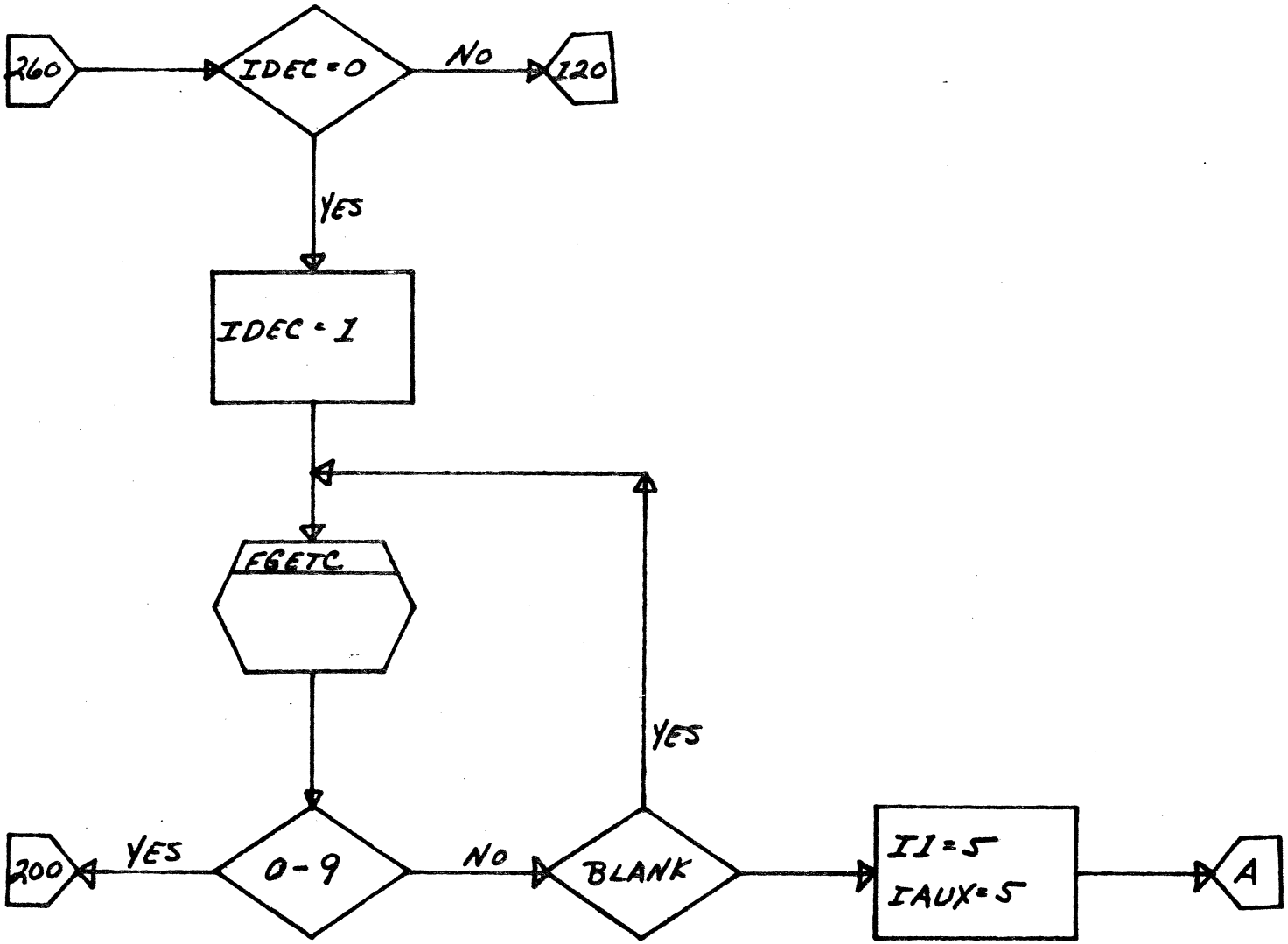
220. / IN F,E & EOS IN F,E

230. / IN A,R,$,I AND ERROR
EOS in A,R,$,I, and ERROR



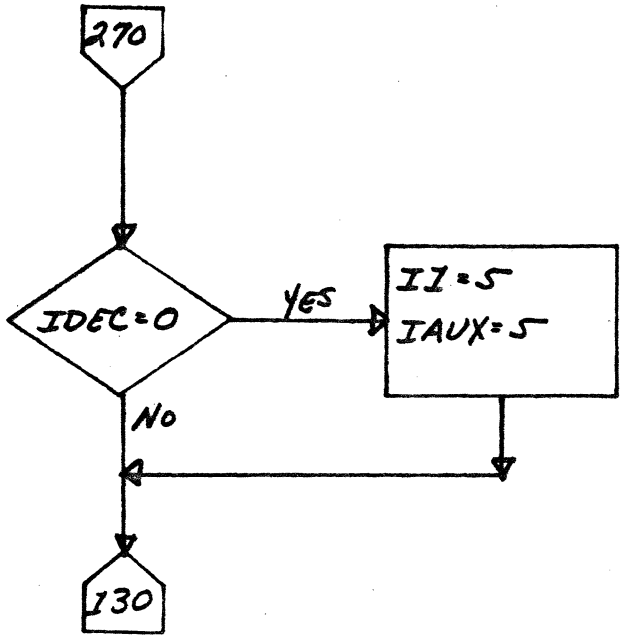


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FORK	PROJECT MGR.			
	PAGE 6 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FORK	PROJECT MGR.			
	PAGE 7 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

270. (IN FILE

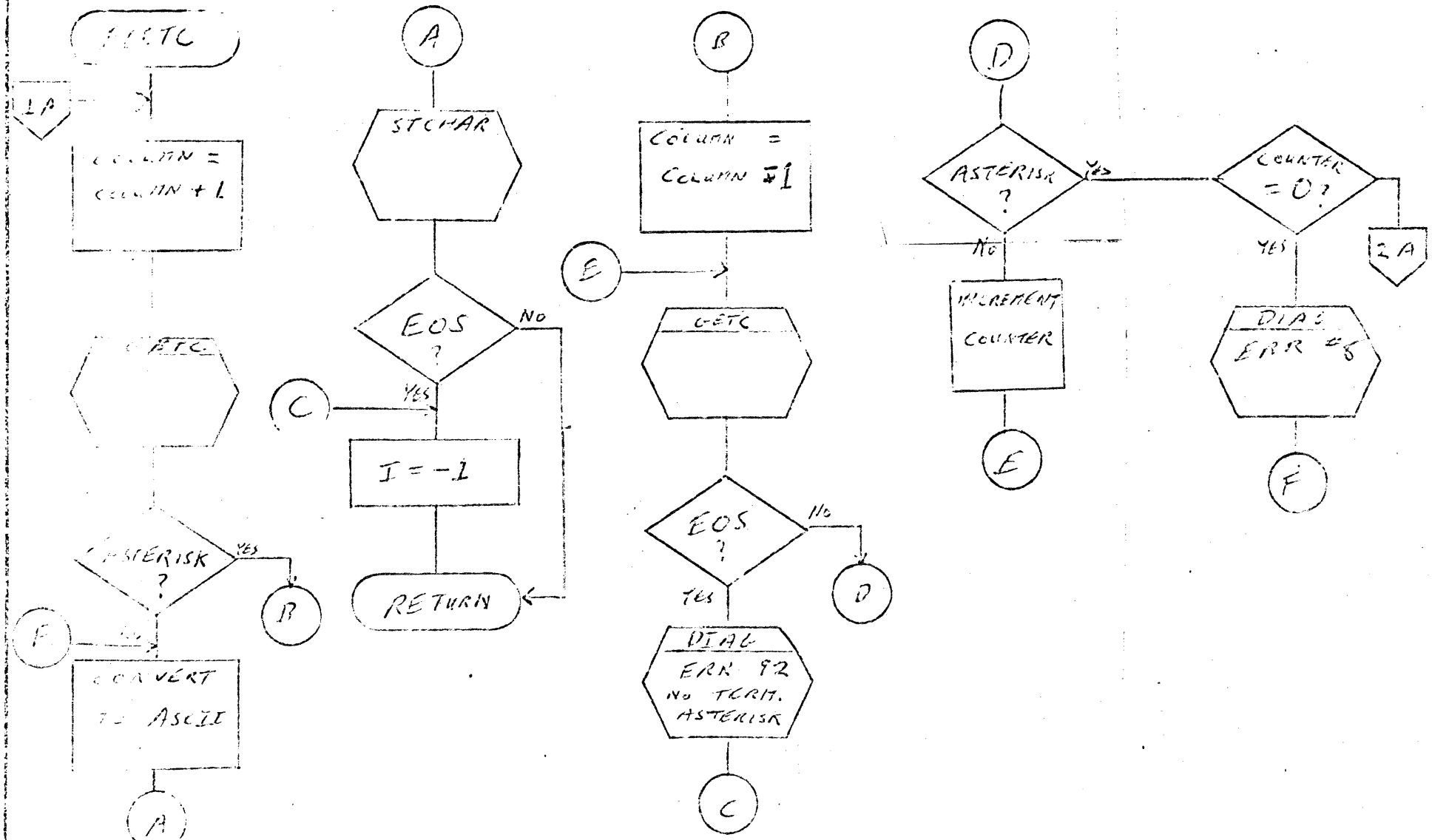


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE FORK		PROJECT MGR.			
FLOWCHART ☒		PAGE 8 OF 8	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

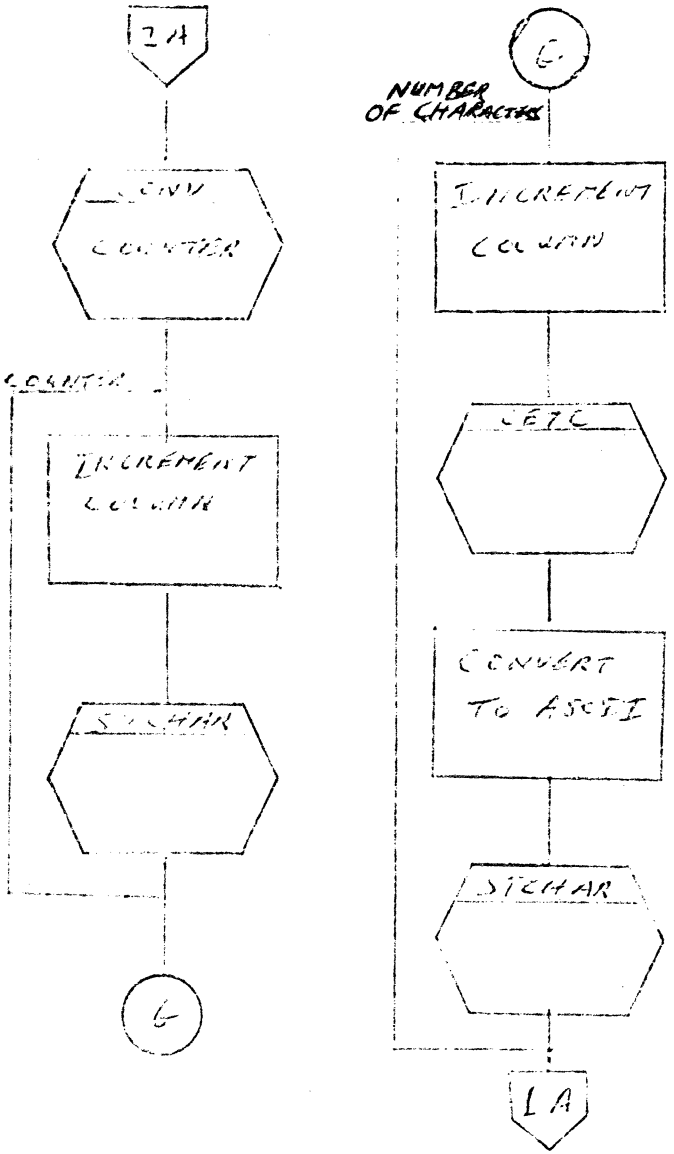
2.4.7.3 Subroutine FGETC

Subroutine FGETC usually delivers one character from ISORS. The character is converted to ASCII and output. However, if the character is an asterisk indicating a ▽free field▽ Hollerith string, FGETC counts the number of characters in the literal string and makes up an appropriate Hollerith declaration.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED / DATE
	DOCUMENT TITLE FGETC		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA;385 (FOR Y CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TFS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FLETC	PROJECT MGR.			
	PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

2.4.8 ASSEM Statements - Subroutine ASEMPR

The ASEMPR subroutine will generate intermediate language for each member of a parameter list in an ASSEM statement. The meaning of the intermediate language generated for a particular constant depends on the type of constant the ASEMPR subroutine encounters in the parameter list. The constants may be either numeric constants, address constants of statement labels.

2.4.8.1 Output of Intermediate Language

The intermediate language is generated in the output buffer named IBUF2. The tally register used for the word length of the output entry is IBUF2X. IBUF2X is set initially prior to entering ASEMPR, and is increased by the number of words of intermediate language generated for a parameter when the parameter is processed. An i.l. entry generated for a parameter begins at IBUF2 (IBUF2X).

2.4.8.2 Numeric Constants

A numeric constant must be an integer of the form:

$hhhh,

implying a hexadecimal number. The GETF subroutine is called to extract the numeric constant from the source buffer. If, upon return from GETF, the operand is other than an integer, the flag JMODE will not contain a "3". An error diagnostic will be made followed by an exit from ASEMPR.

If the numeric constant is legal, the following 2 words of intermediate language will be generated in the output buffer:

WORD 1: (IBUF2(IBUF2X)) = 1; = the number which identifies the parameter as a numeric constant.

WORD 2: (IBUF2(IBUF2X+1)) = the value for the numeric constant.

2.4.8.3 Address Constants

An address constant may be either an alphanumeric operand in which case it is a name or it may be a numeric operand in which case it is regarded as a statement label. The name or label is extracted from the source buffer by a call to GETF. If the name or label does not appear anywhere in the symbol table, a call is made to the store routine to read the operand and its class.

Either name or label is assumed to be used with the relative mode of addressing unless the operand is preceded by a "+". It is then regarded as to be used with the absolute mode of addressing. If either operand is to have an increment, the increment must follow the operand and be enclosed with parentheses. The increment must be an integer constant. The GETF subroutine is used to extract the increment from the source buffer. If, upon return from GETF, either -

1. {JMODE} ≠ 3 indicating that the constant is not an integer, or
2. {JTERM} ≠ 42 indicating that the increment is preceded by a left parenthesis but not followed by a right parenthesis, an error diagnostic will begin.

Upon return from GETF, the value of the increment is recorded in JSYM{1}: If {1} the switch IK is set non-zero indicating that the ASA option is selected, {2} the parameter is a name, and {3} the name does not appear in the symbol table typed as SINGLE, the value of the increment is doubled. This is to accomodate the fact that if the ASA option is selected, integer and real names require 2 storage cells each while SINGLE names require 1. In other words: If {IK} ≠ 0, and ISNGL {ISYMX} = 0,

{JSYM{1}}*2 → JSYM{1}.

Triple size of increment if double precision element:

{JSYM{1}}*3 → JSYM{1}

The following 2 words of intermediate language will be generated for an address constant using absolute addressing and having no increment:

WORD 1: {IBUF2{IBUF2X}} = 2

WORD 2: {IBUF2{IBUF2X+1}} = pointer to name or label in symbol table.

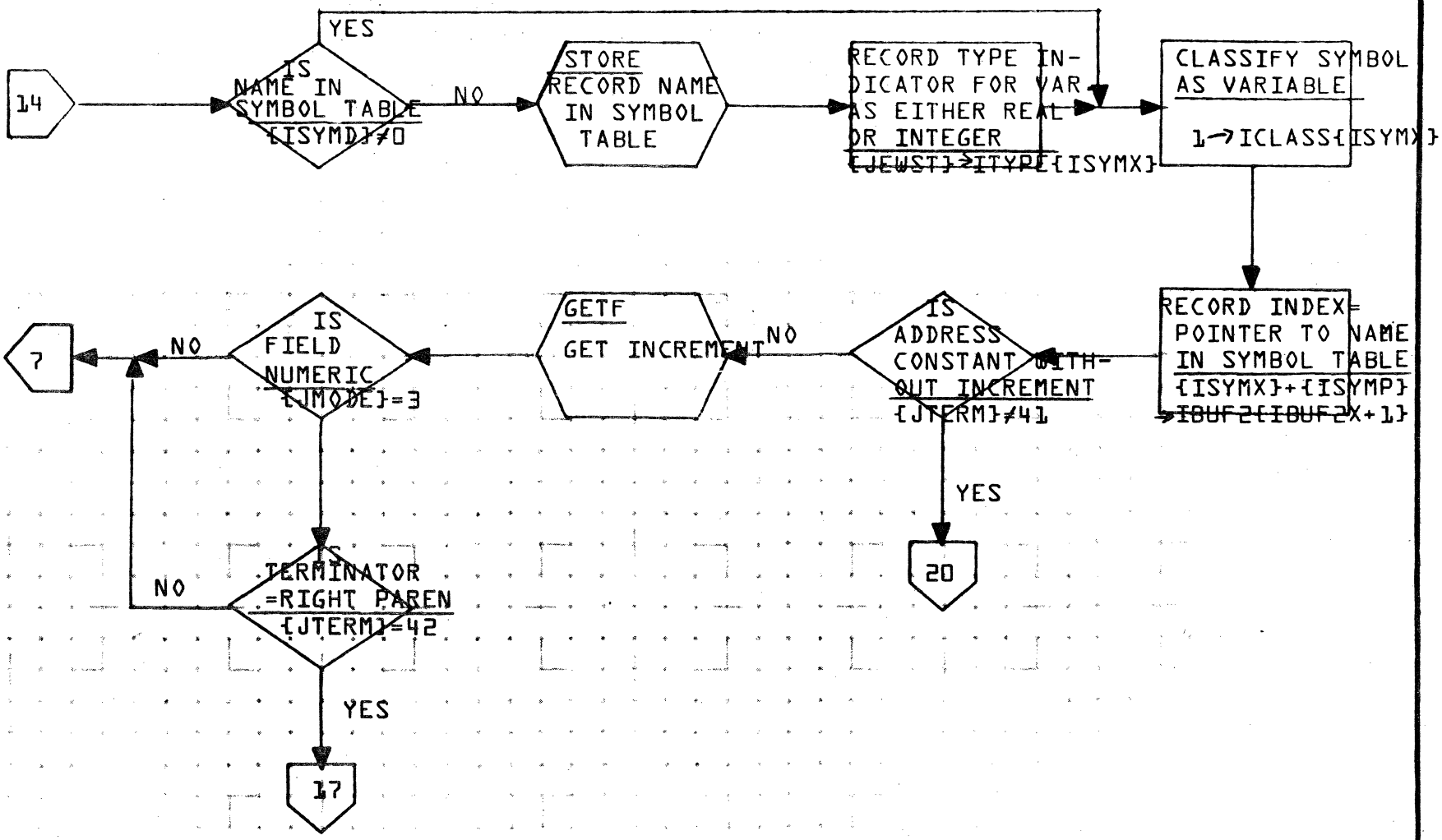
The following 3 words of intermediate language will be generated for an address constant using the absolute mode of addressing and having an increment:

WORD 1: {IBUF2{IBUF2X}} = 4

WORD 2: {IBUF2{IBUF2X+1}} = pointer to name or label in symbol table.

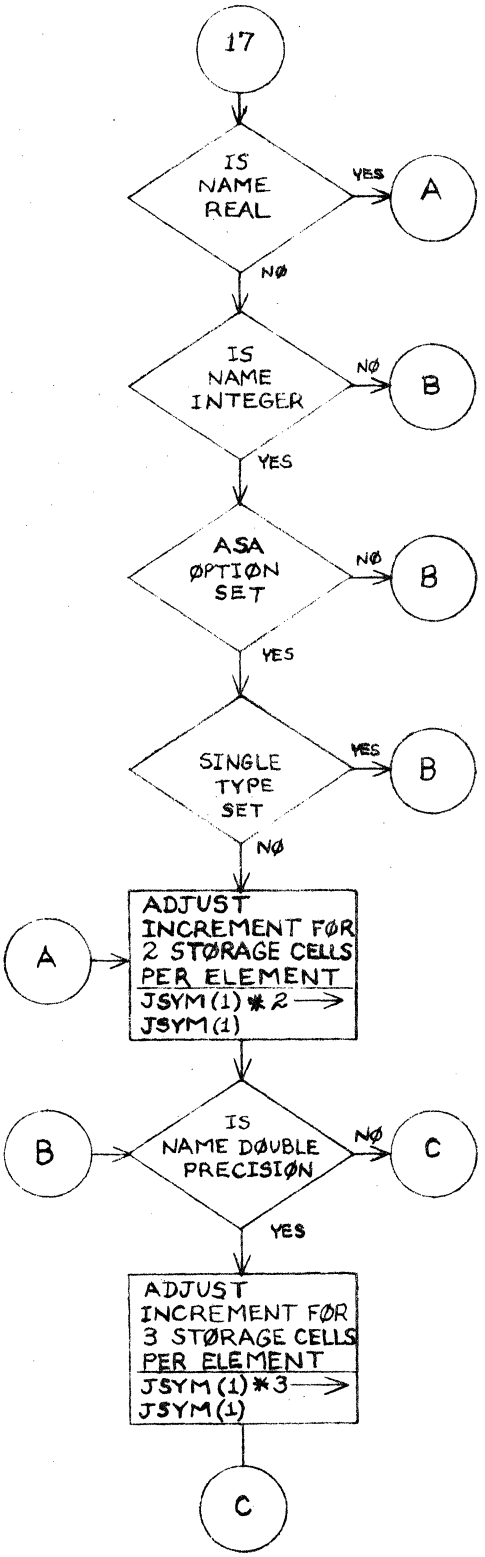
WORD 3: {IBUF2{IBUF2X+2}} = value for increment recorded in JSYM{1}.

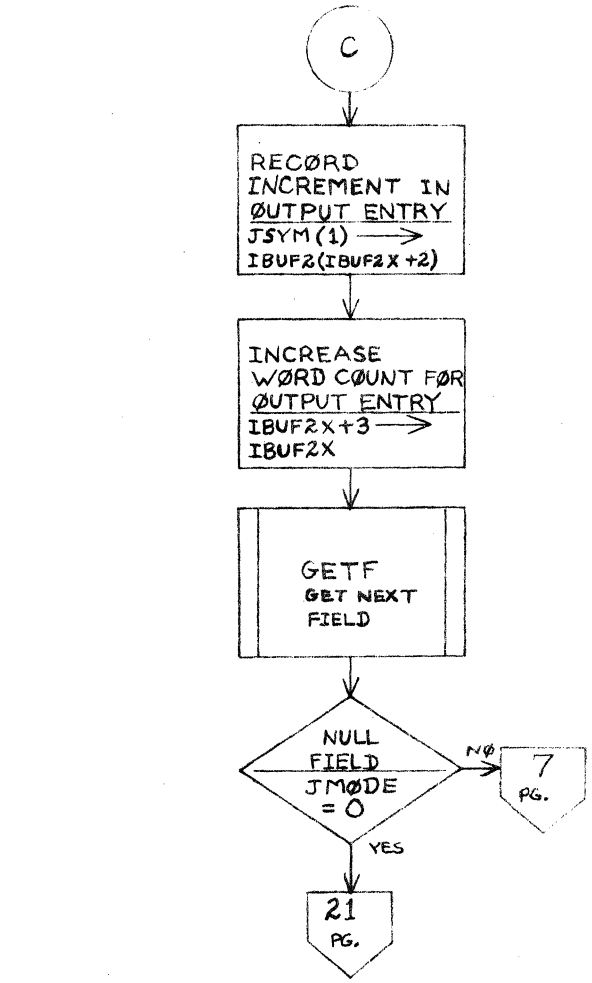
The following 3 words of intermediate language will be generated for an address constant using the relative mode of addressing and having an increment:



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ASEMPR		PROJECT MGR.		
		PAGE 2 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

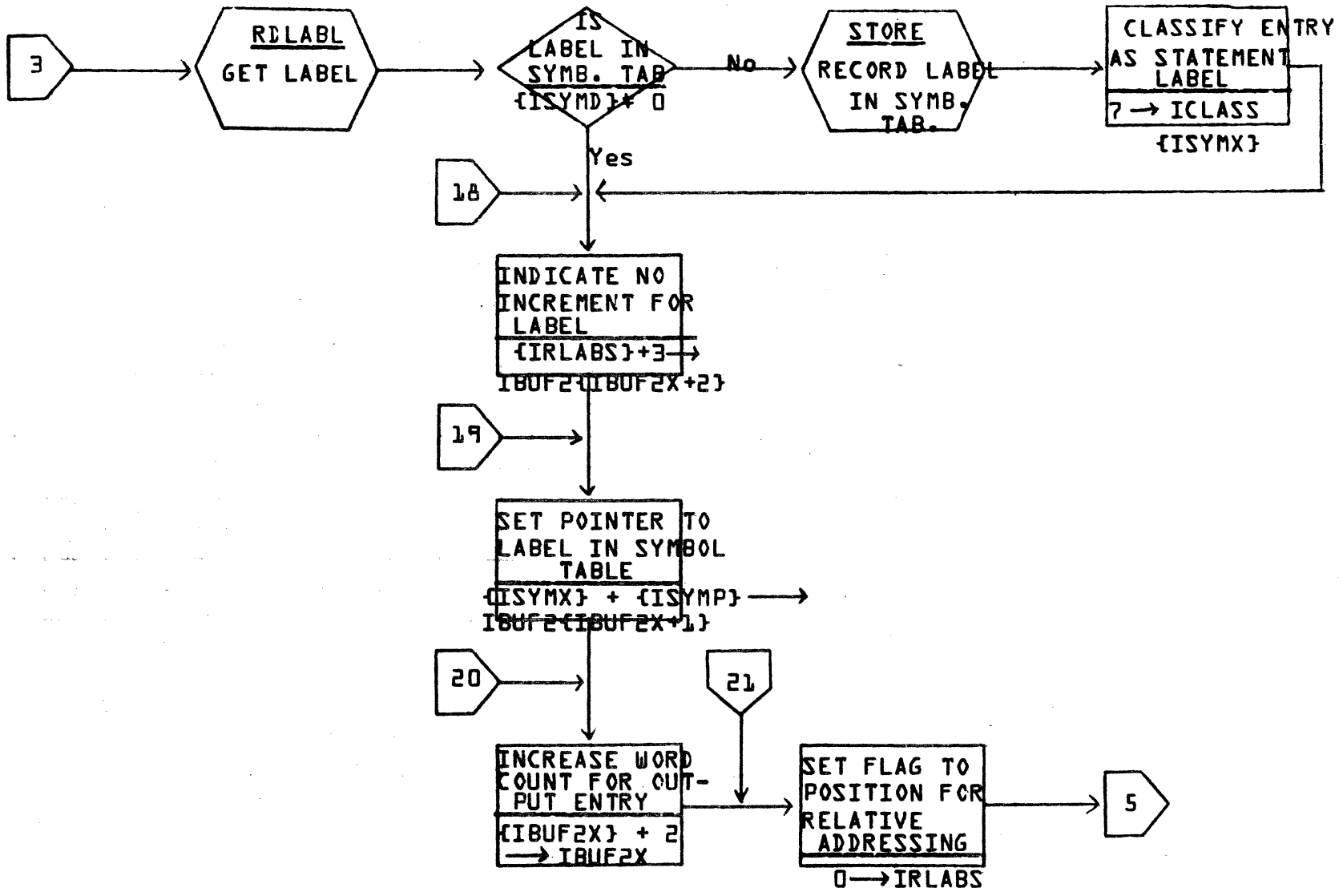
Eh2-2



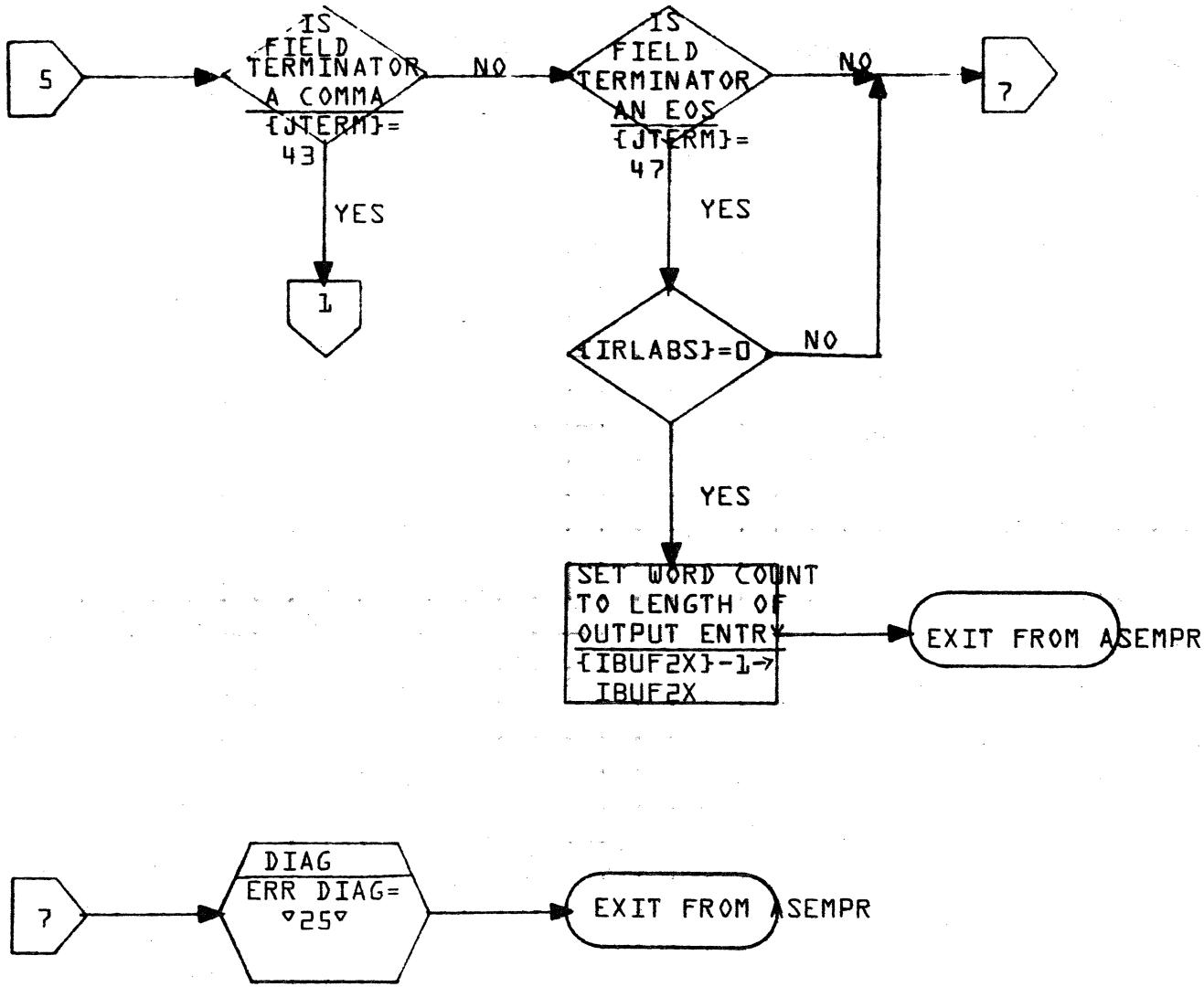


TITLE ASEMPR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF 3

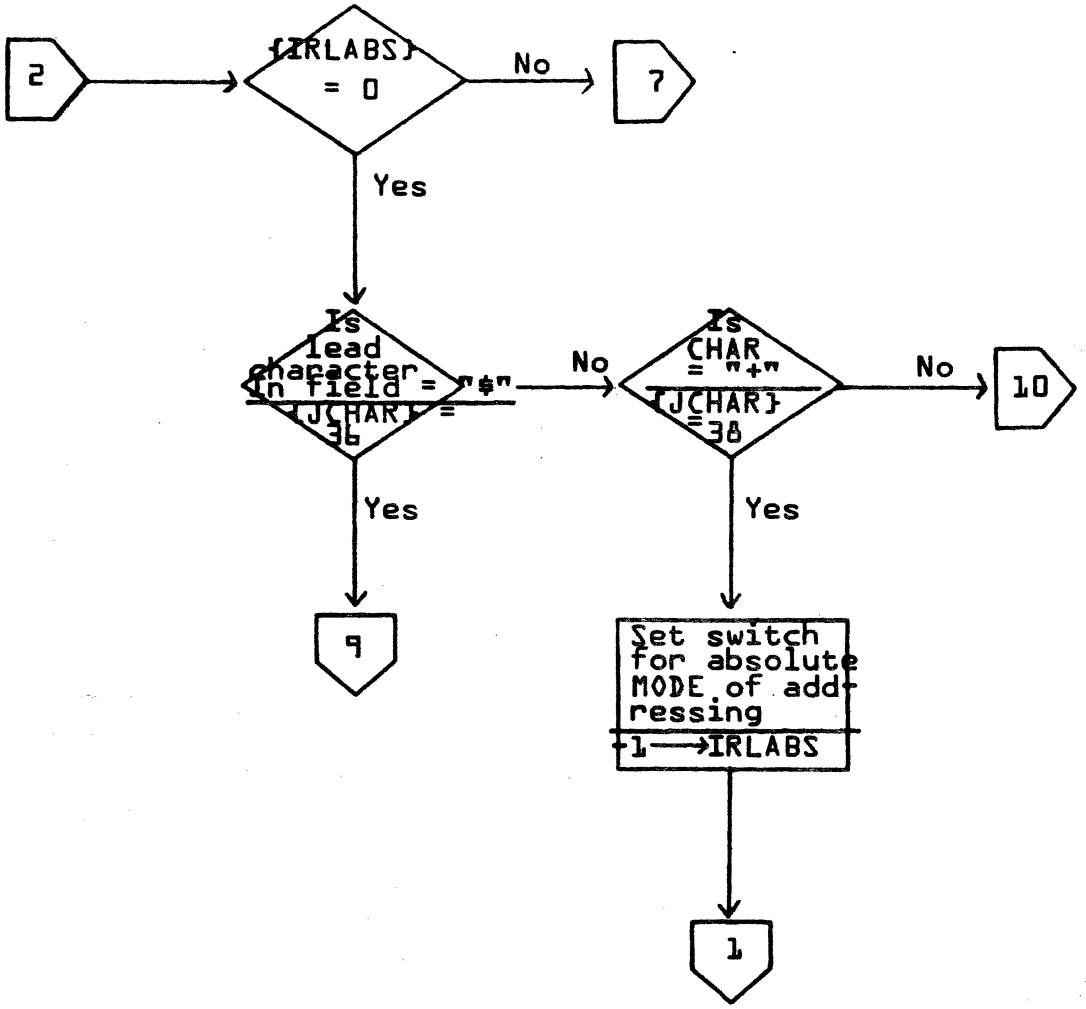
CSD 203



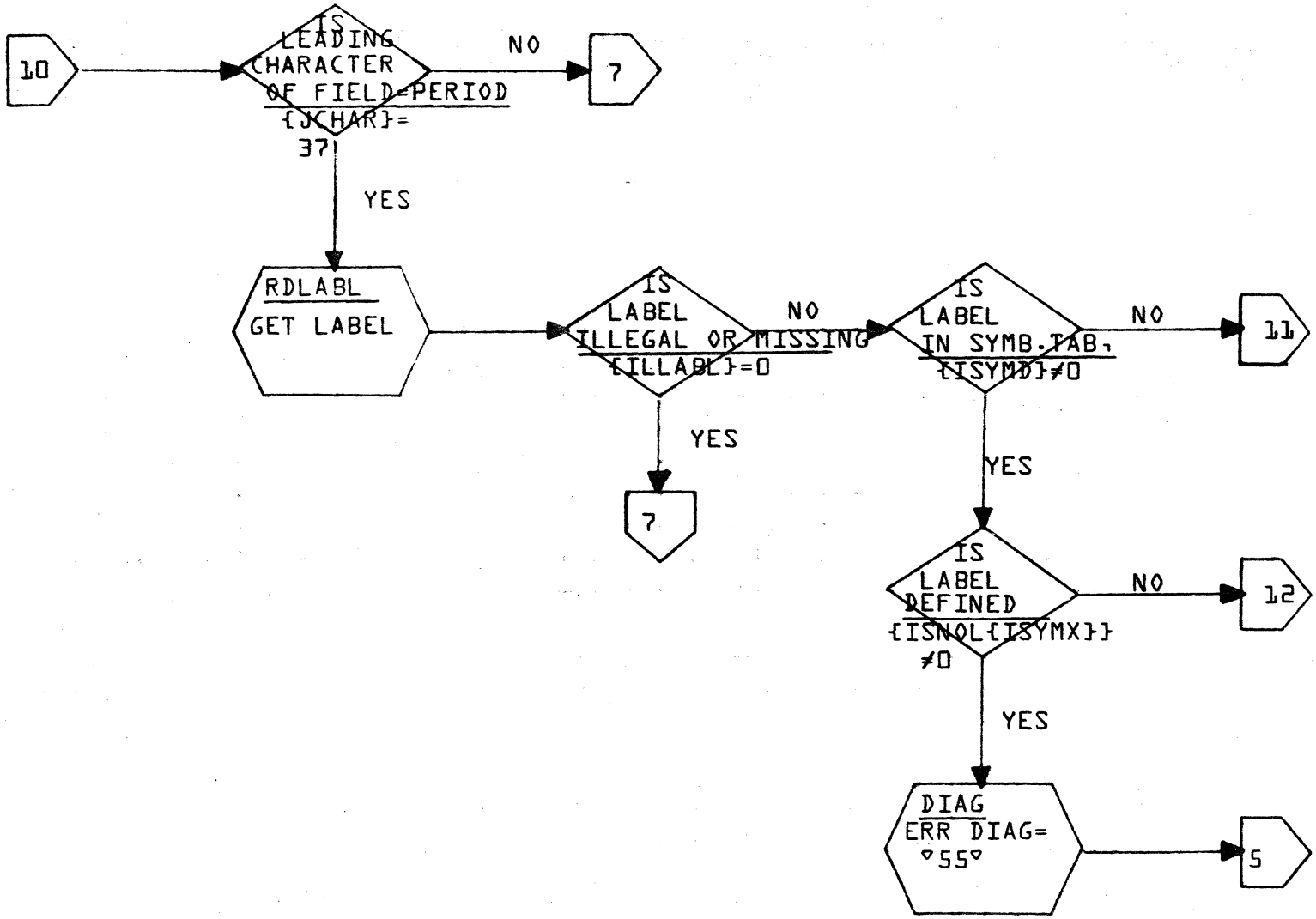
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ASEMPR		PROJECT MGR.			
		PAGE 4 OF 8	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ASEMPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 5 OF 8	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



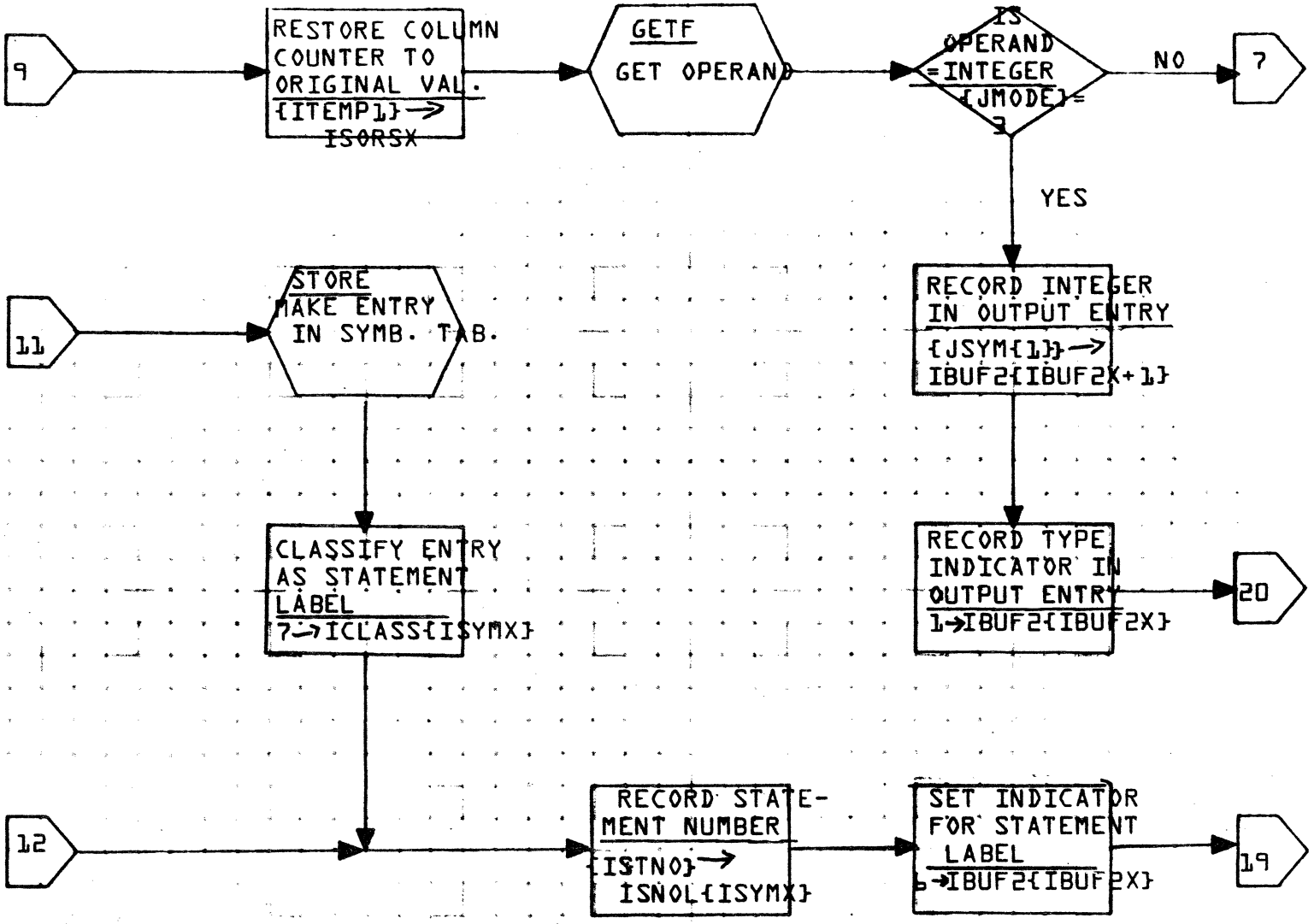
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ASEMPR		PROJECT MGR.			
		PAGE 6 OF 8	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ASEMPR		PROJECT MGR.		
		PAGE 7 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

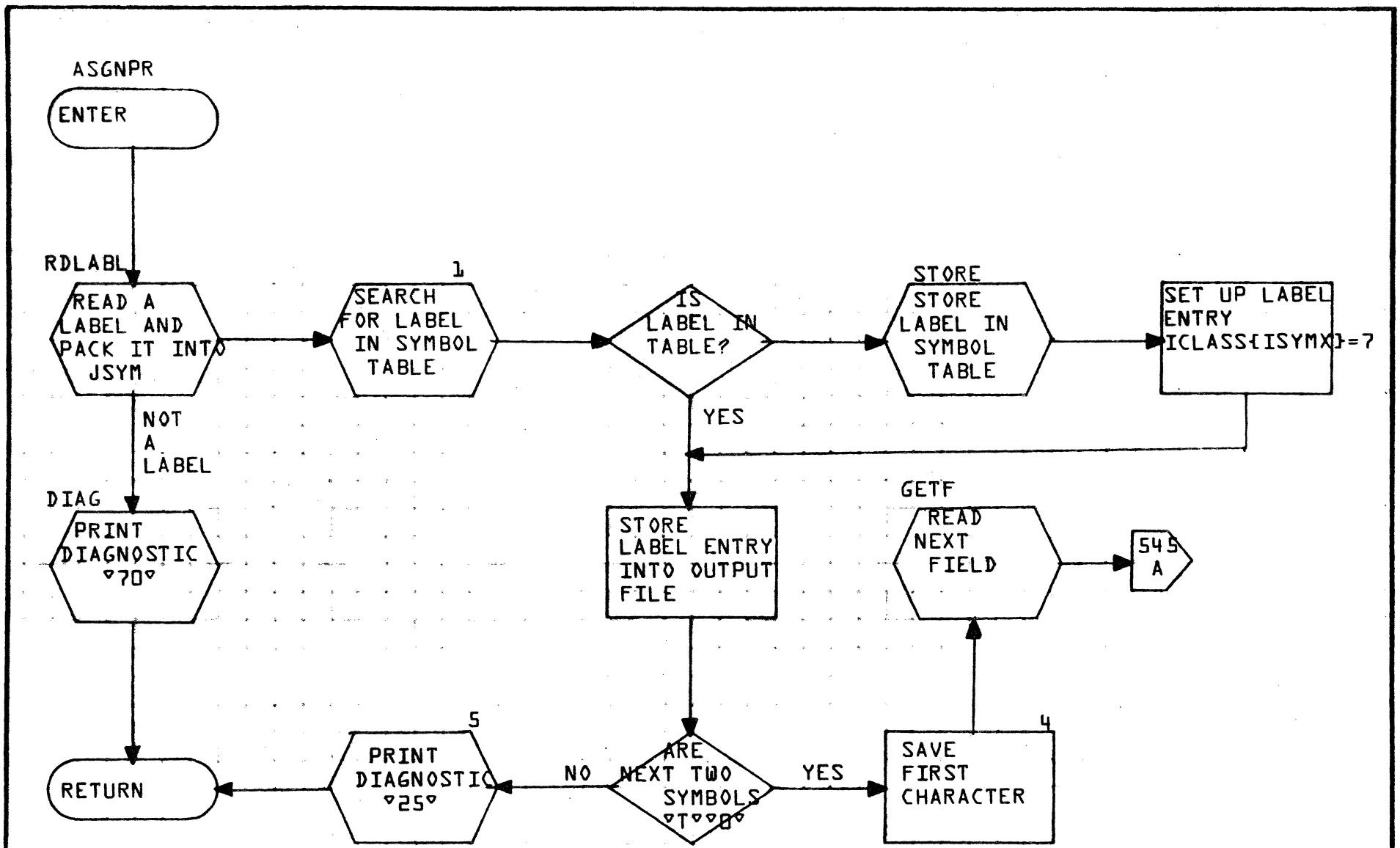
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ASEMPR	PROJECT MGR.			
	PAGE 8 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

2.4.9 ASSIGN Statement - Subroutine ASGNPR

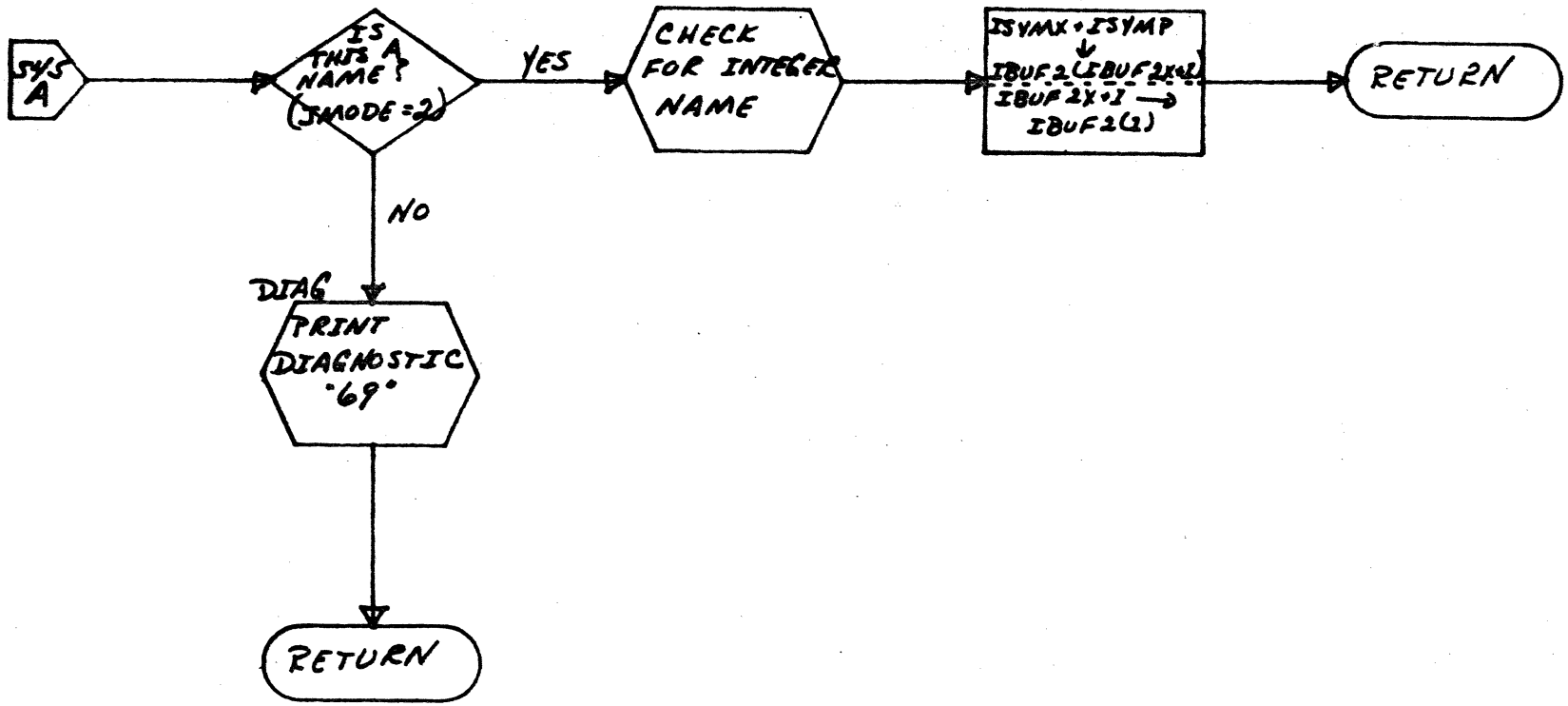
ASGNPR is called directly by PHASEA to process ASSIGN statements. It reads the next field in ISORS using RDLABL and ascertains that it is a label. ASGNPR locates the label in the symbol table and places this symbol table pointer into the output buffer. The next two characters of ISORS are checked for T, 0 - if they are not, a diagnostic is printed. ASGNPR reads the next field and checks that it is an integer variable name. If so, the name is located in the symbol table and its symbol table pointer is placed in the output buffer. The number of words in the record is put into the first word of the output record and ASGNPR exits to PHASEA.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ASGNPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 2	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

21251

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ASGNPR		PROJECT MGR.		
		PAGE 2 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

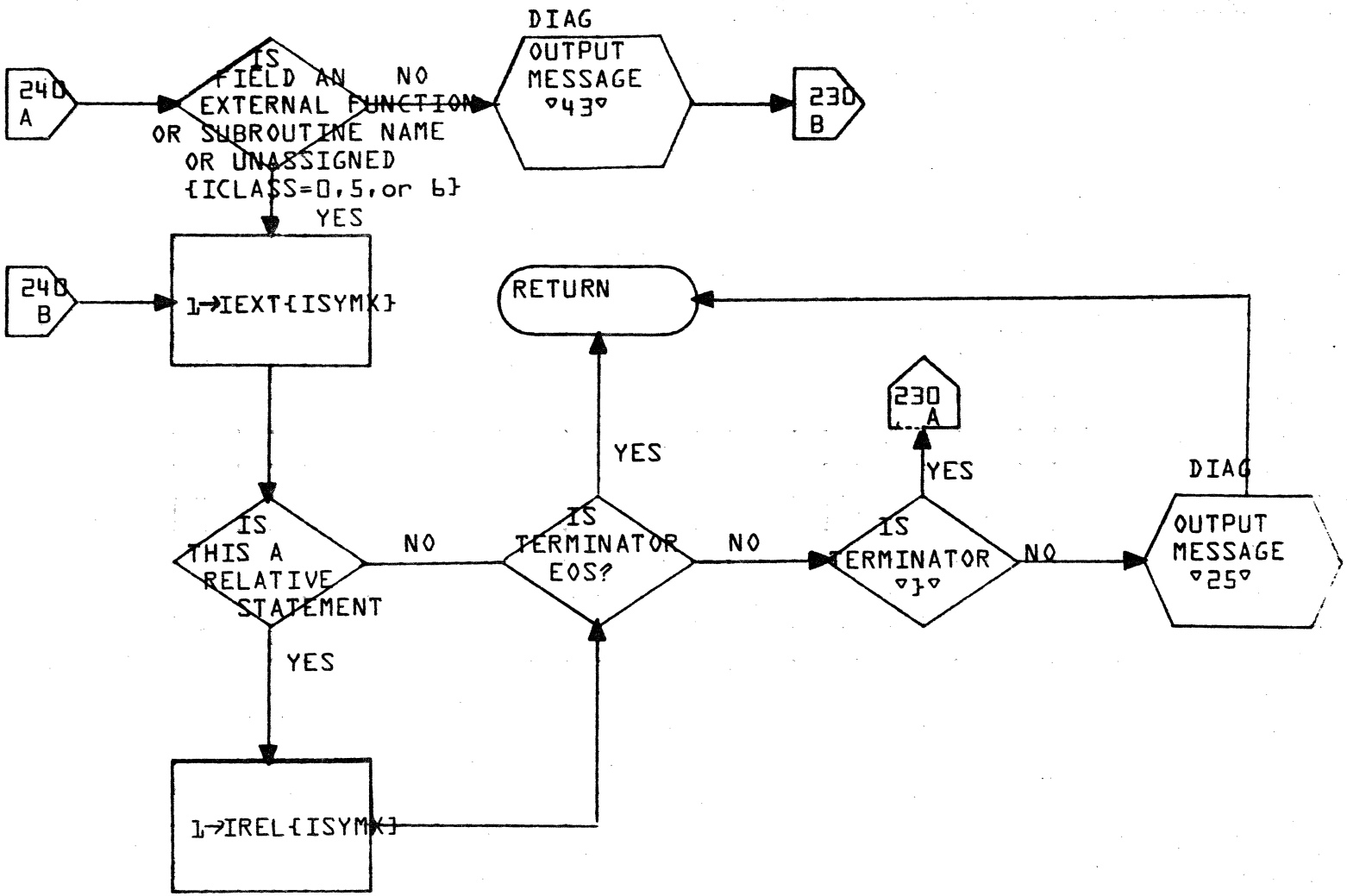
21252

AA1385 (FORMERLY CA127-1)

2.4.10 EXTERNAL and RELATIVE Statements - Subroutine EXRLPR

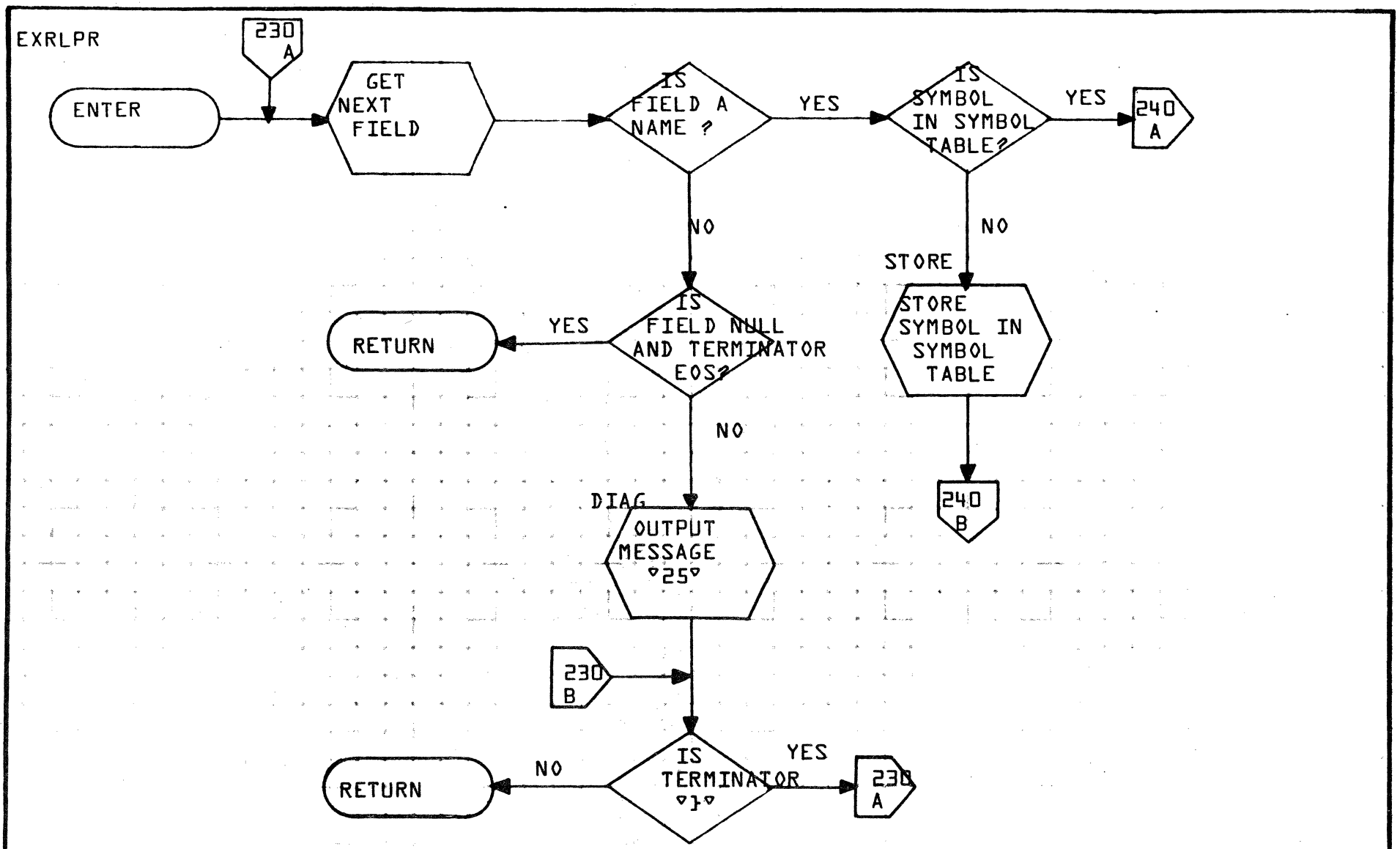
EXRLPR is called by PHASEA to process EXTERNAL and RELATIVE statements. It scans ISORS, checking the legality of each name it picks up, putting the name in the symbol table, and checking whether it has previously been declared as other than an external name. If it has been declared as other than an external, an error is output. If not, IEXT in the symbol table is set to one and, if EXRLPR is processing a relative statement, IREL is also set to one. Upon encountering the end of the statement, EXRLPR exits to PHASEA .

EXRLPR



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE EXRLPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 2 OF 2	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

21254

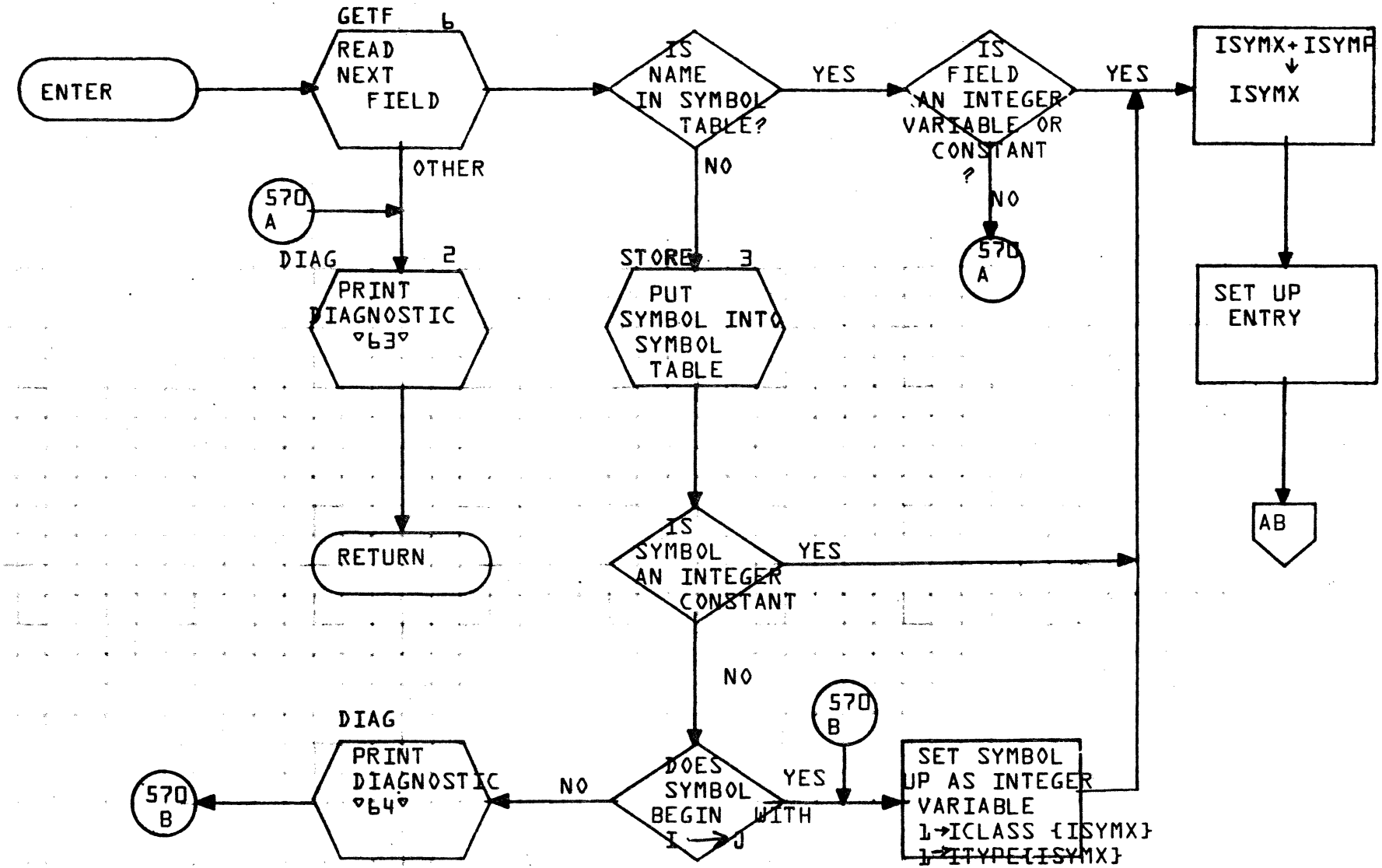


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE EXRLPR		PROJECT MGR.			
		PAGE 1 OF 2	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

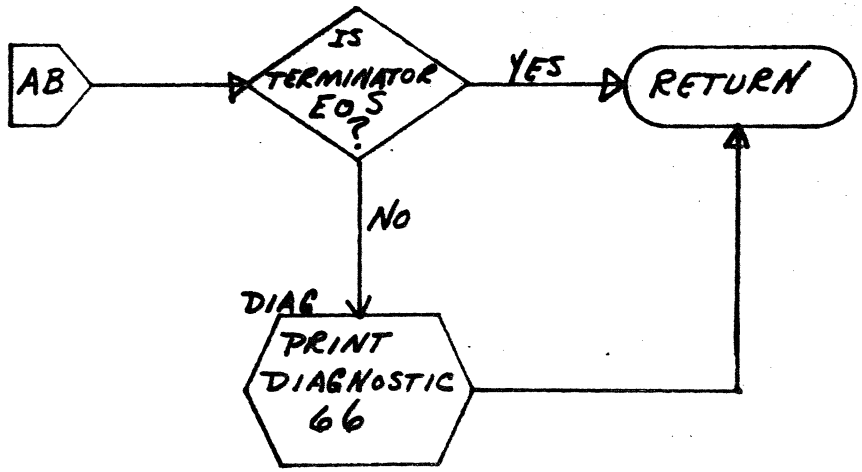
2.4.11 REWIND, ENDFILE, BACKSPACE Statements - Subroutine ERBPR

ERBPR is called directly by PHASEA to process REWIND, BACKSPACE, or ENDFILE statements. It reads the next field of ISORS and verifies that it is an integer constant or variable name. After verification ERBPR sets up the output entry and return.

In the 3.0A compiler the length of ERBPR is crucial. Any increase in ERPBR must be accompanied by a corresponding decrease in the BSS buffer at the end of DUMYA6. {See page 2-9}



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ERBPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 2	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ERBPR		PROJECT MGR.		
		PAGE 2 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

2-1256

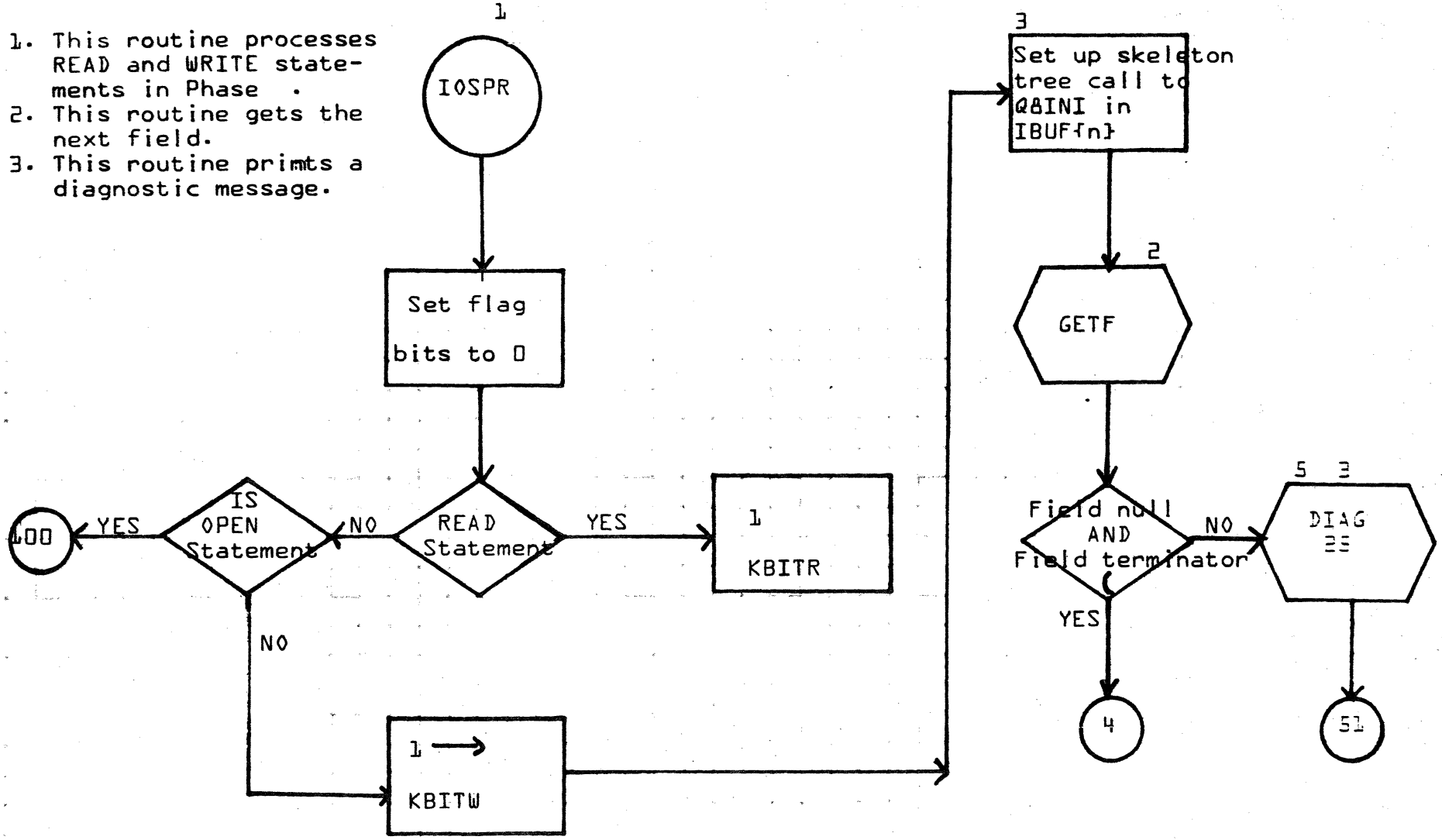
2.4.12 IOSPR - Tree Builder for I/O Statements

This routine is called by PHASEA upon encountering a READ or WRITE. It builds the trees for the calls to Q8QINI, Q8QX, and Q8QEND, as required in IBUF2. Calls to Q8QX are built as new source statements which will be processed by ARITH. Other calls are built in the form of ARITH output and will be processed next by PHASEB

For a READ, WRITE statement the first tree generated is for Q8QINI. If there is a list, a Q8QX call is generated for each element, and a call to Q8QEND is generated after the last element.

NOTES:

1. This routine processes READ and WRITE statements in Phase .
2. This routine gets the next field.
3. This routine prints a diagnostic message.

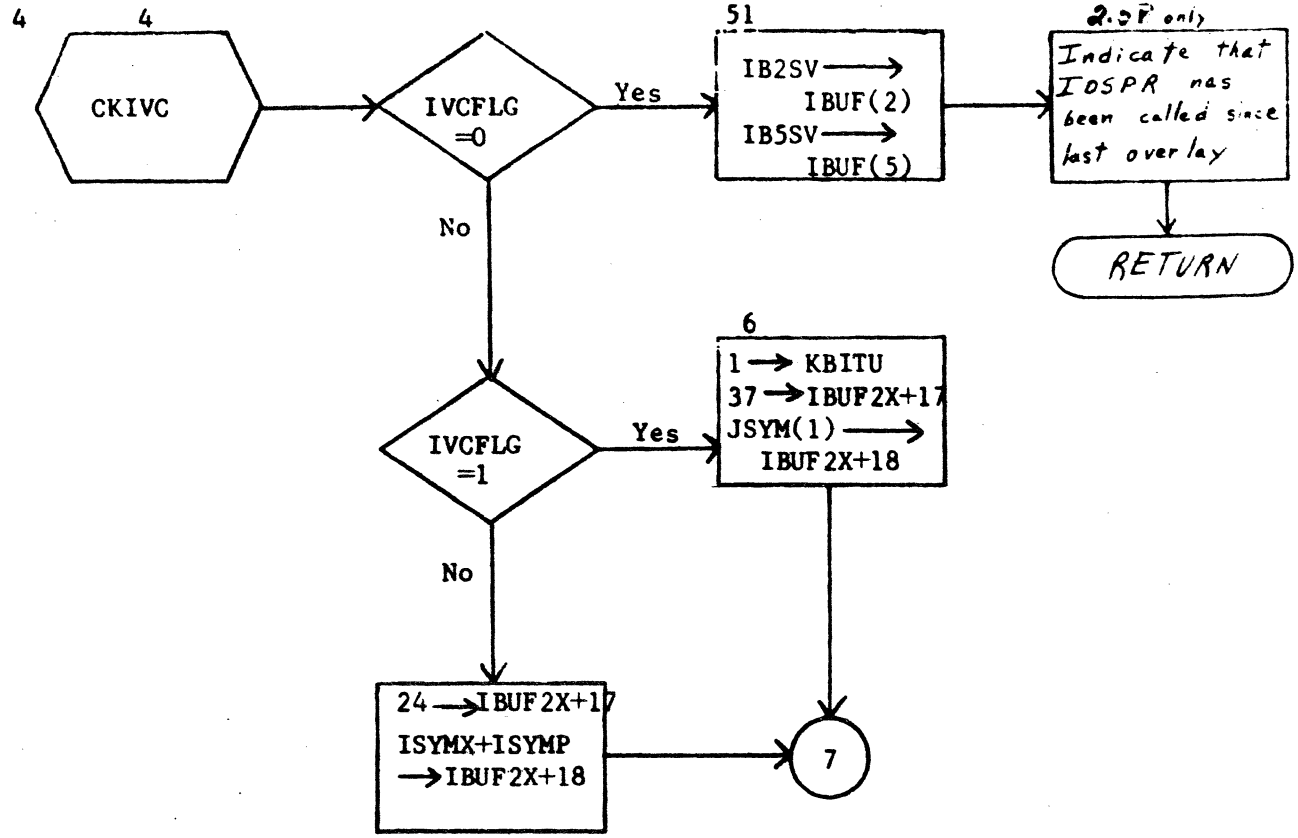


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE IOSPR	PROJECT MGR.			
	PAGE 1 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

21260

NOTES:

4. Checks for integer variable or constant - prints DIAG 63 if not.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE IOSPR	PROJECT MGR			
	PAGE 2 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

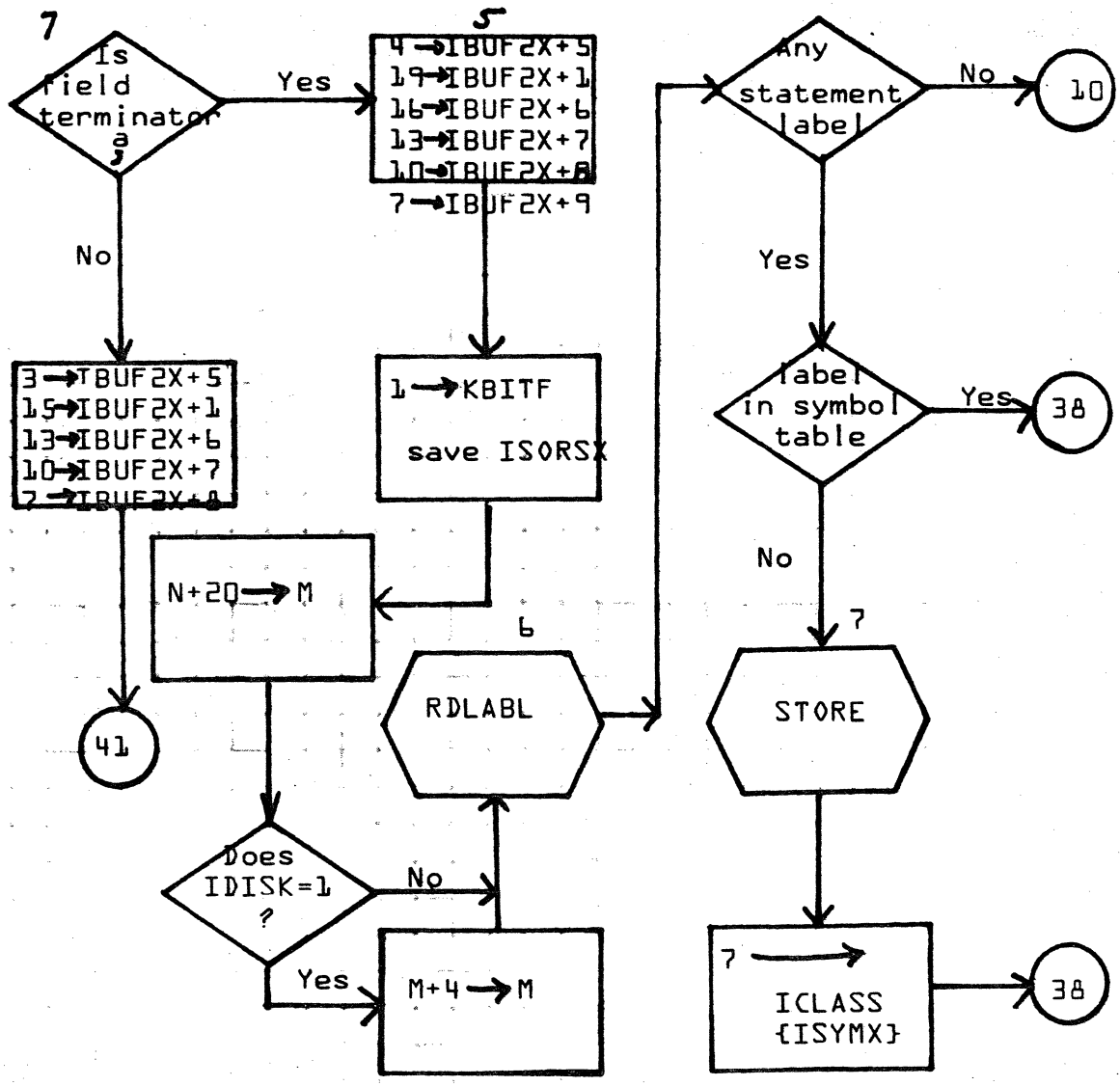
NOTES:

5. Set up formatted I/O.

6. Read statement label.

7. Put label in symbol table.

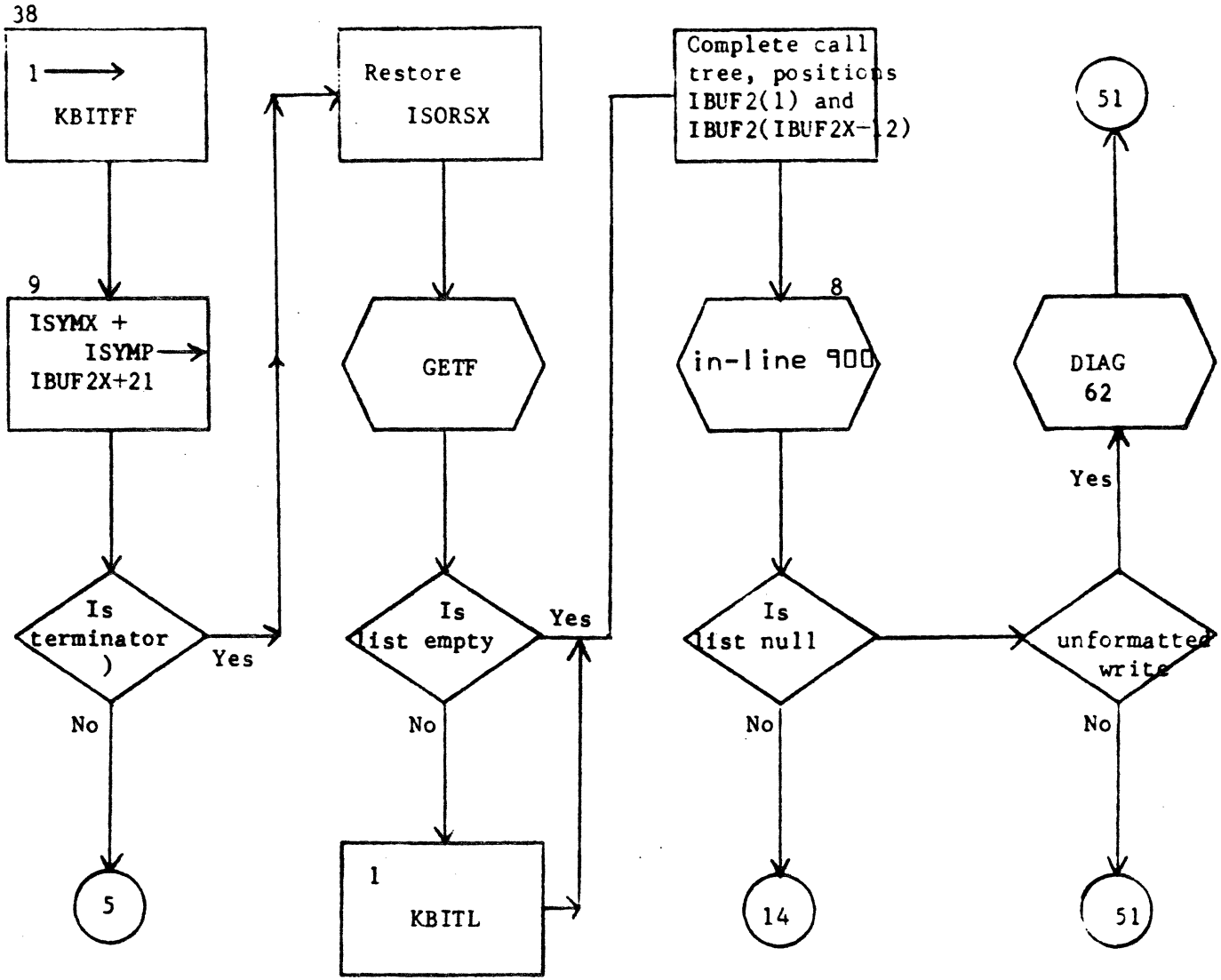
8. For IBUF2X+5 means IBUF2{IBUF2X+5}.



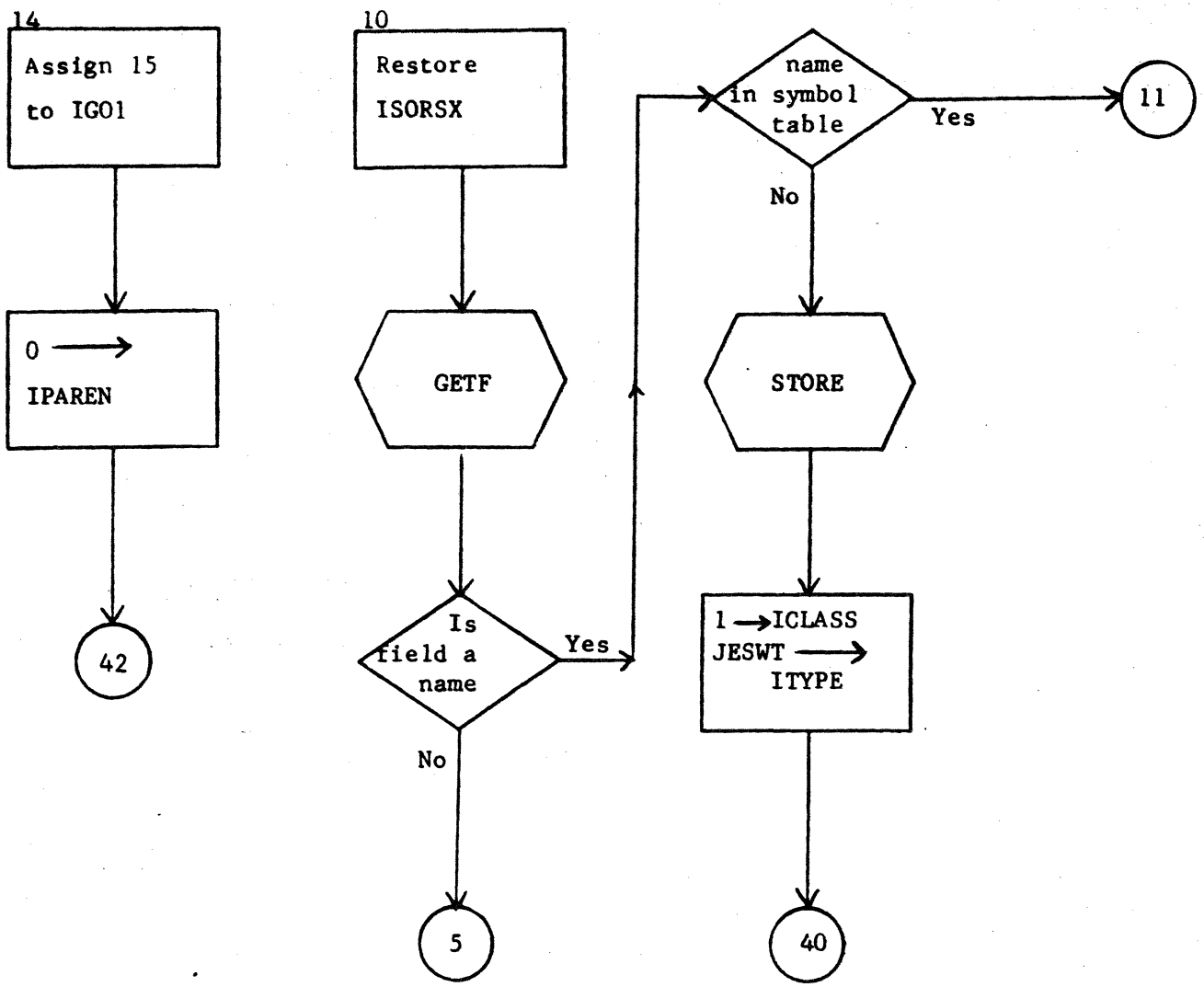
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE IOSPR		PROJECT MGR.		
		PAGE 3 OF 16	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

NOTES:

8. Output the statement.



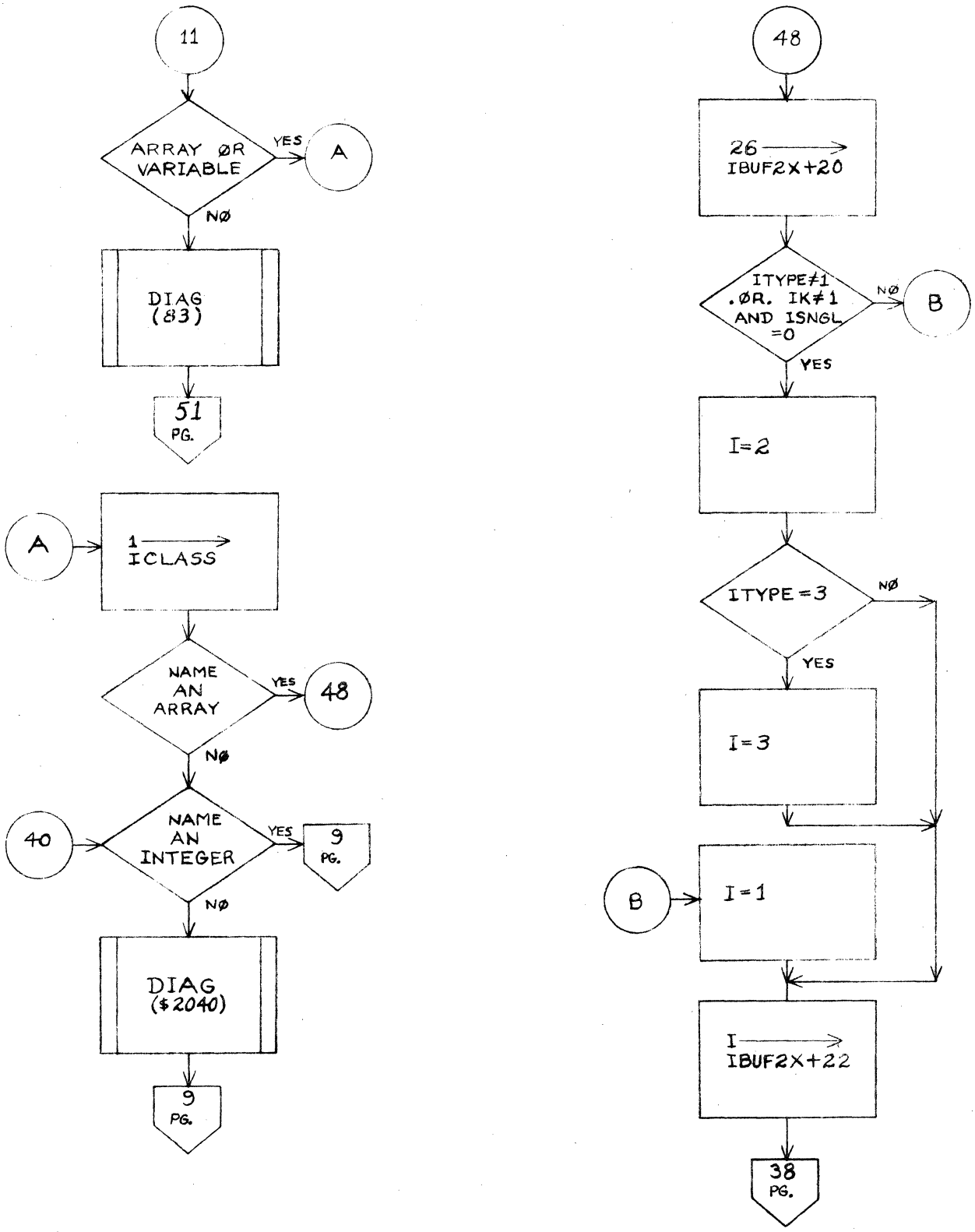
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE / FLOWCHART / DECISION TABLE / OTHER	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE: IOSRS		PROJECT MGR		
		PAGE 4 OF 16	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE 5/4/77	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE IOSAR		PROJECT MGR.			
		PAGE 5 OF 16	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 5-6-7	TASK NAME			

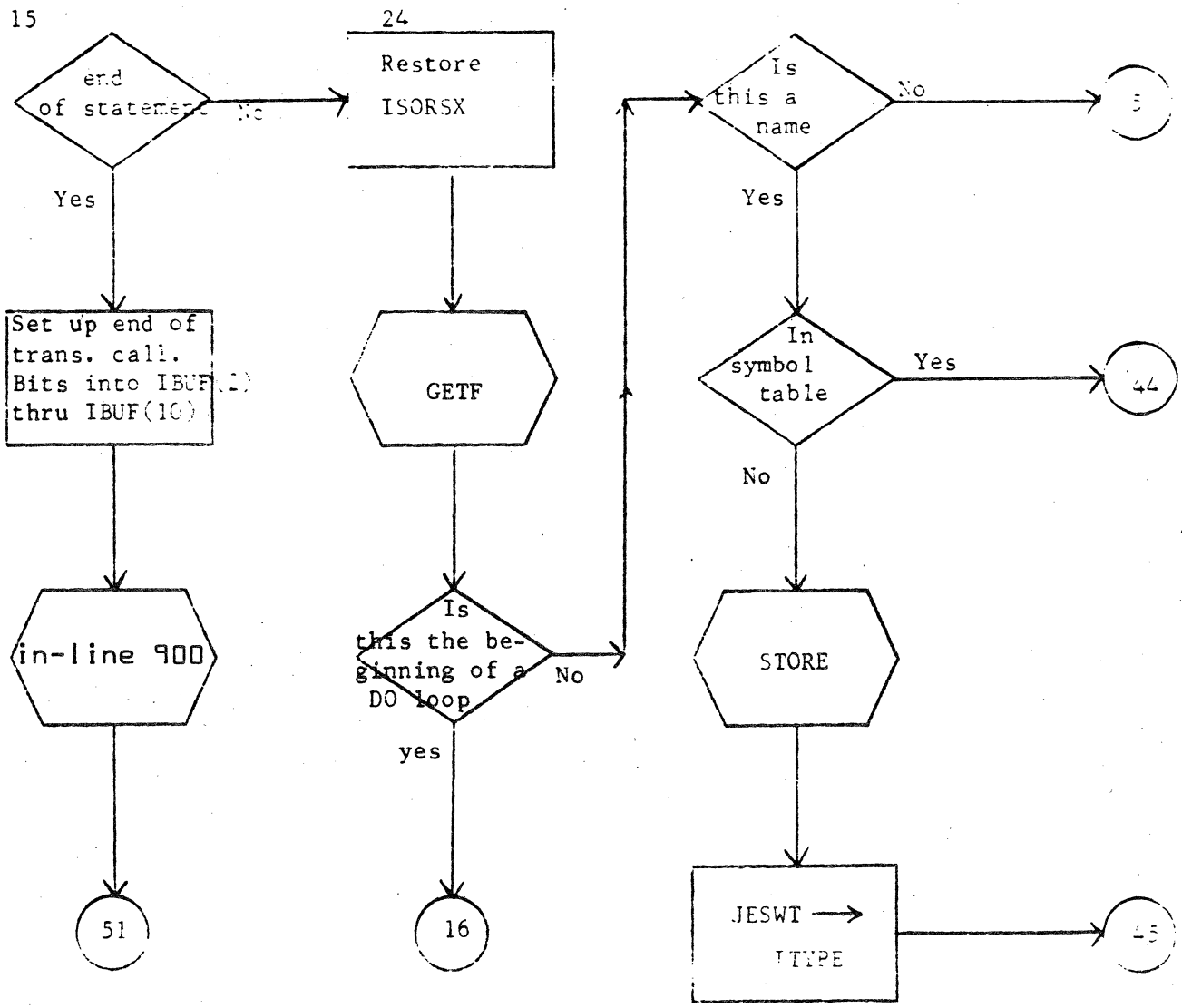
CONTROL DATA CORPORATION

LA JOLLA FACILITY

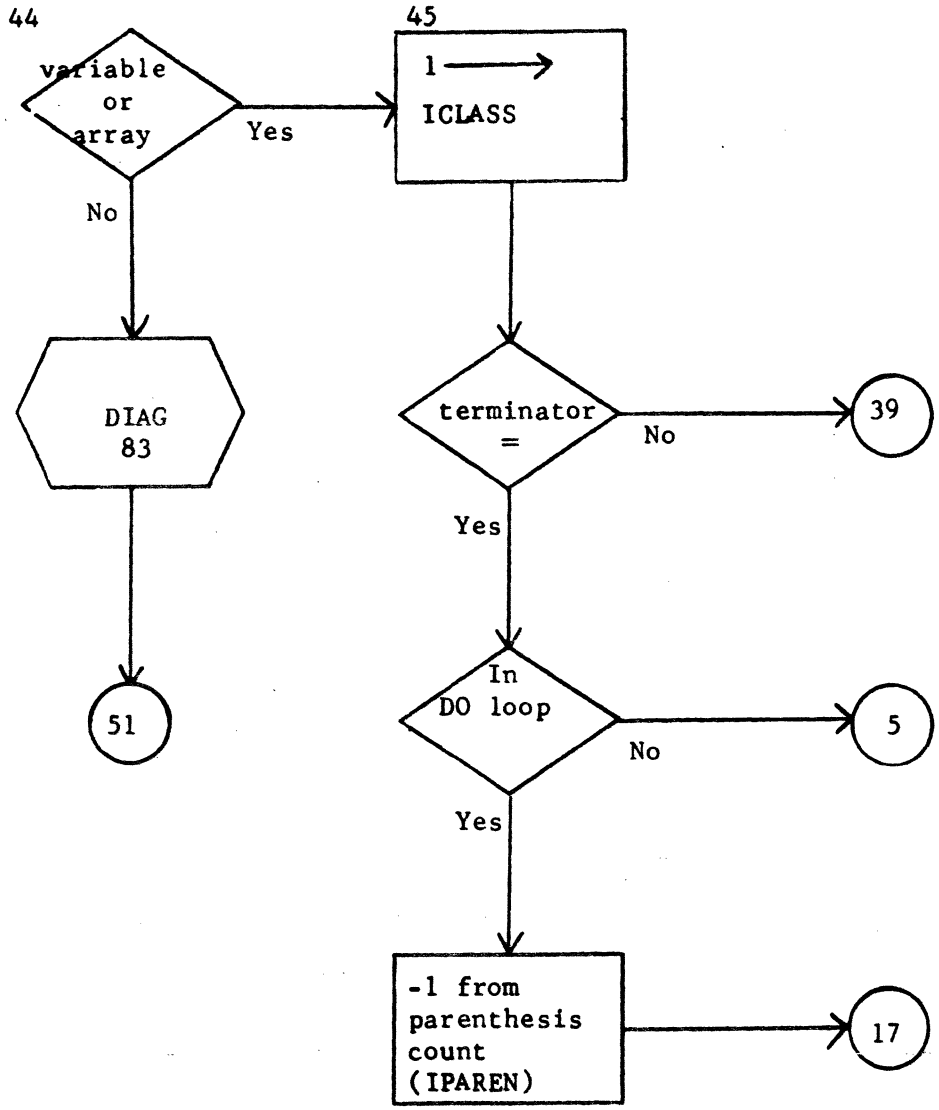


TITLE	ASEMPR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 4 OF 8

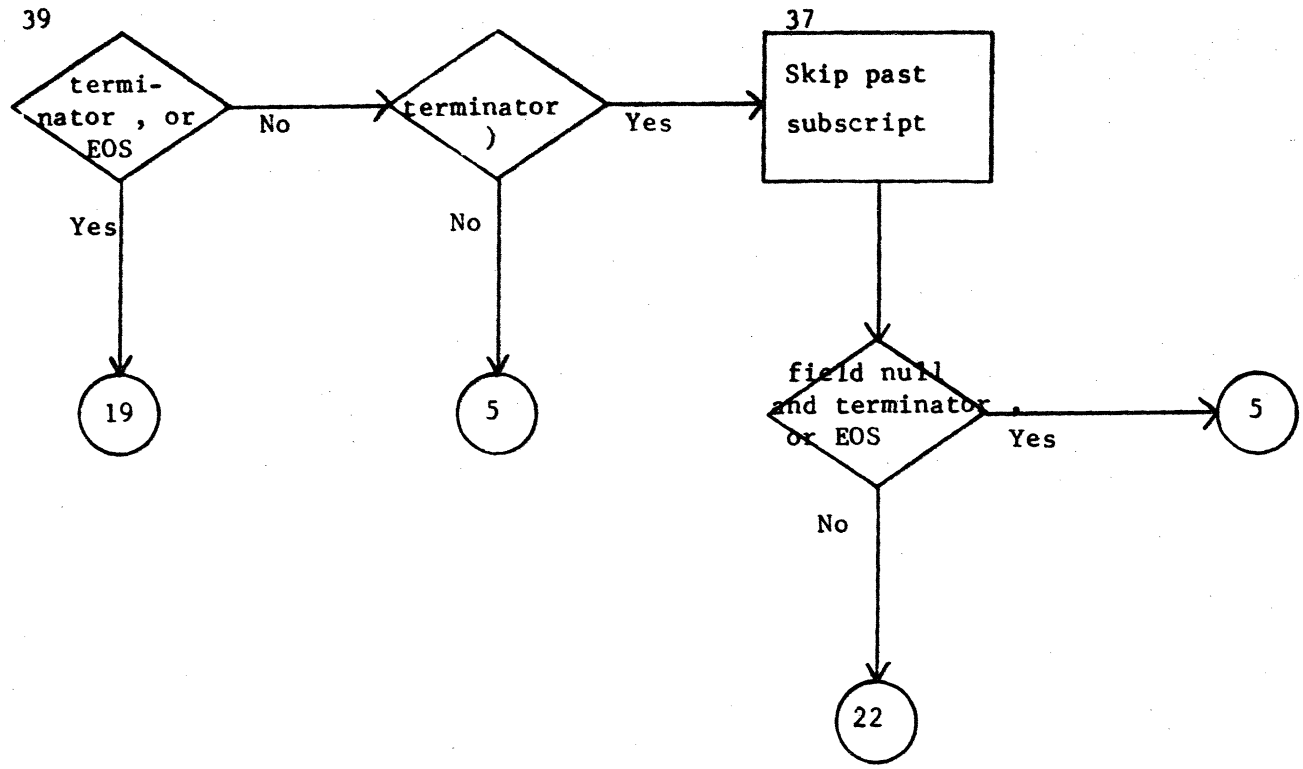
CSD 203



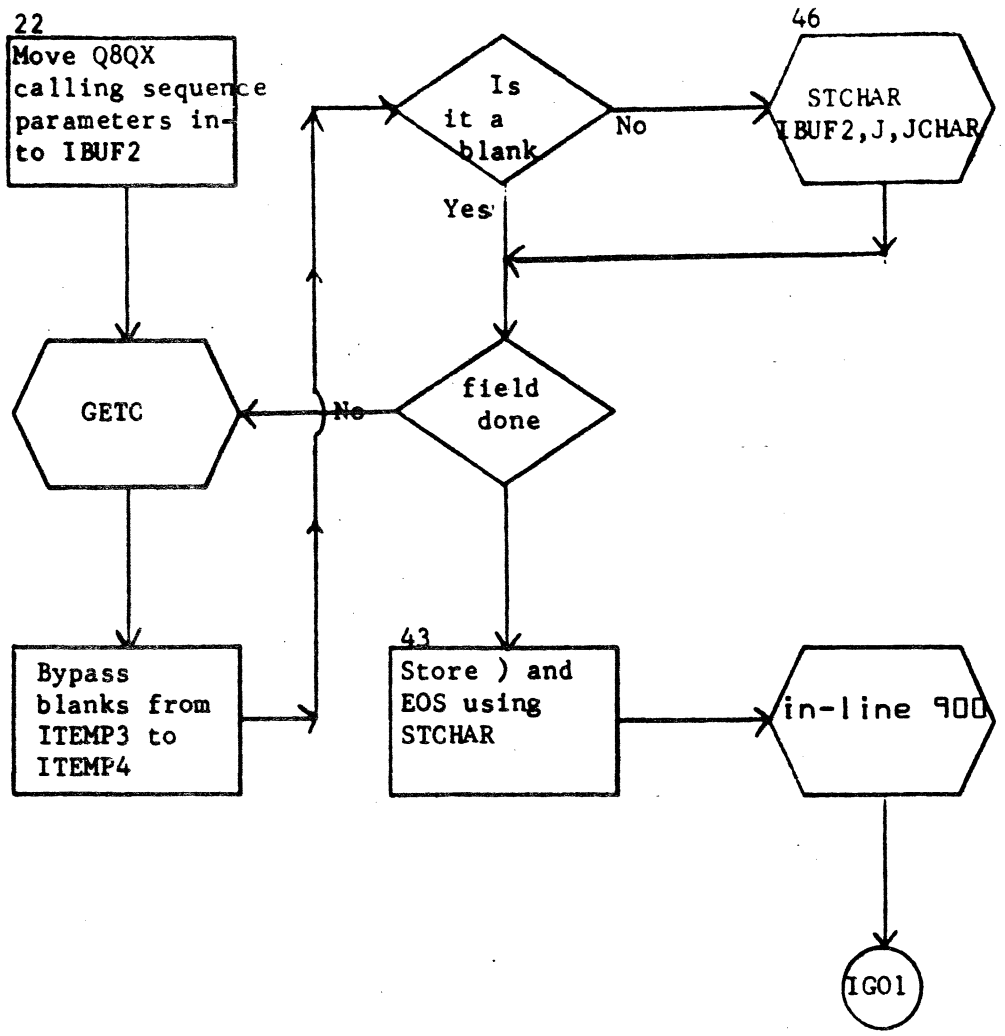
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE IOSRX		PROJECT MGR		
		PAGE 7 OF 16	PROJECT NAME		
	NUMBER	SERIES DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		



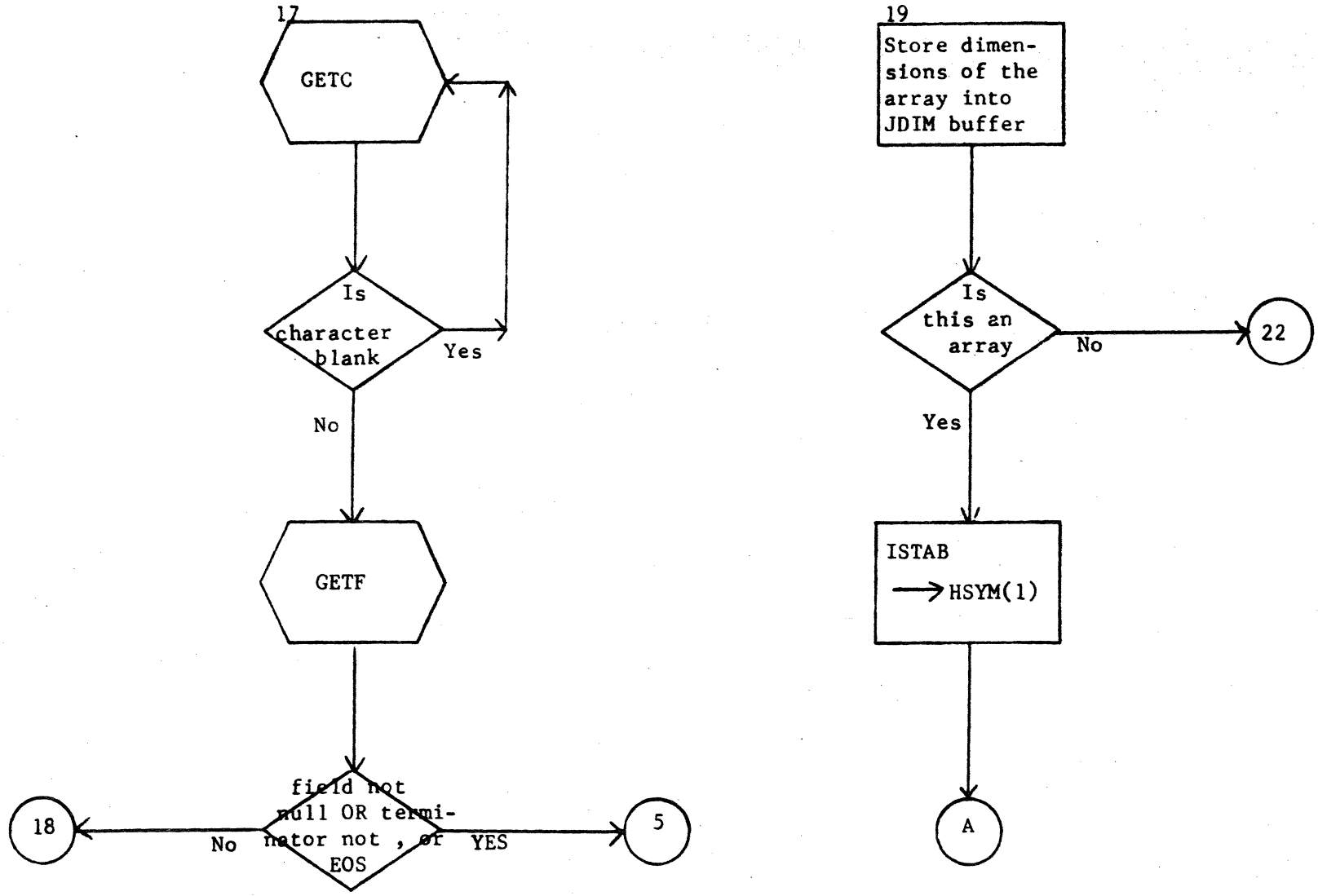
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE IOSAR		PROJECT MGR.		
FLOWCHART ☒		PAGE 8 OF 16	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		



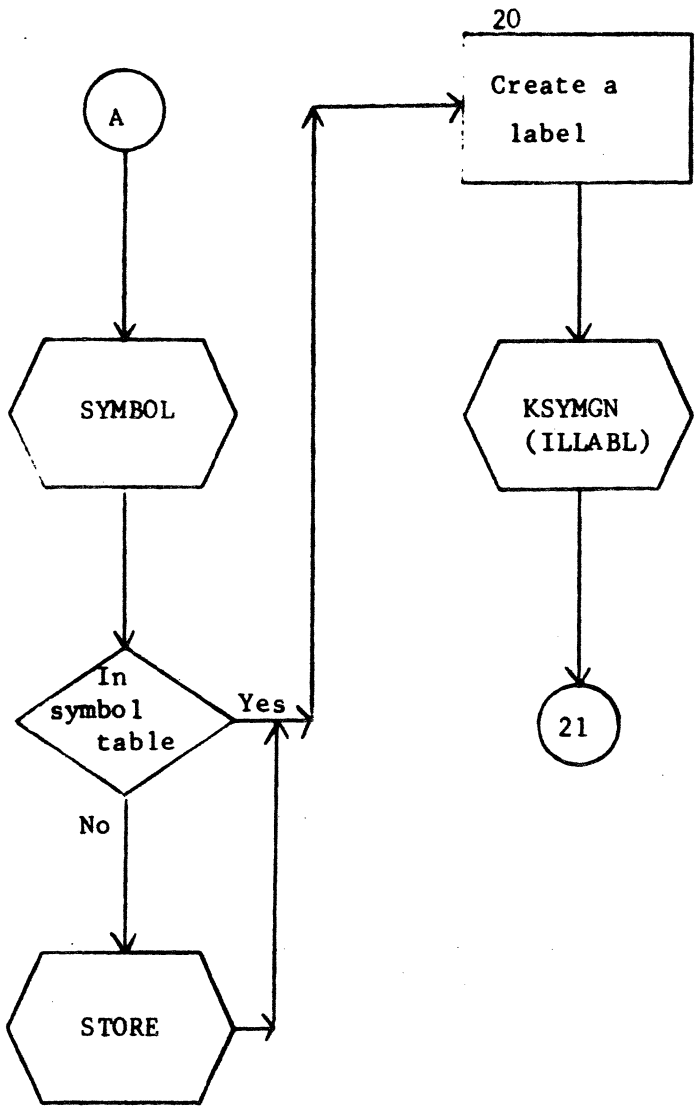
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE IOSPR		PROJECT MGR.			
FLOWCHART ☒		PAGE 9 OF 16	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE IOSAR PAGE 10 OF 16 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO. PROJECT MGR PROJECT NAME TASK NO TASK NAME	REV	APPROVED	DATE



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE IOSAP		PROJECT MGR.			
FLOWCHART ☒		PAGE 11 OF 16	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



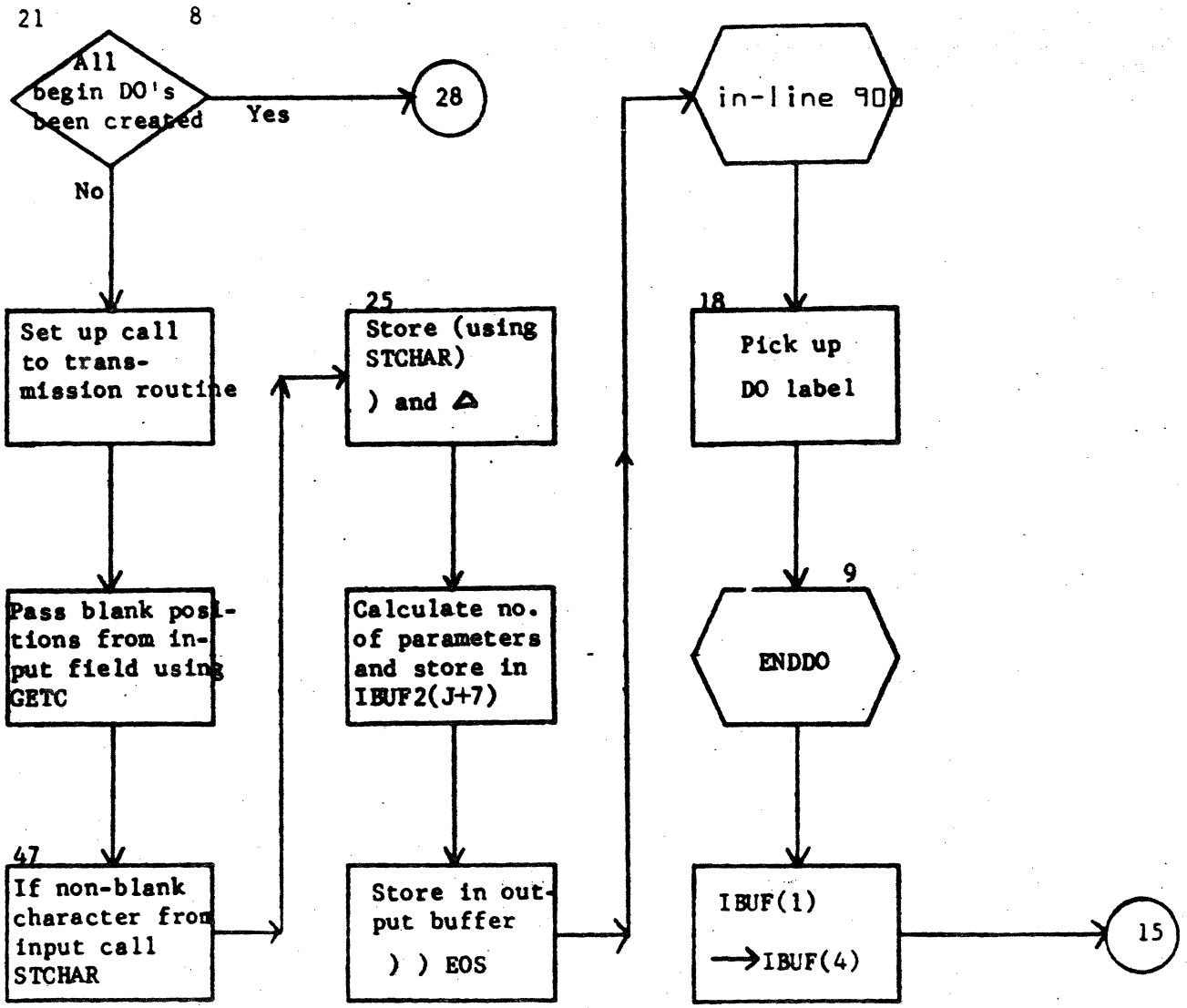
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE IOSAP	PROJECT MGR.			
	PAGE 12 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

T22-2

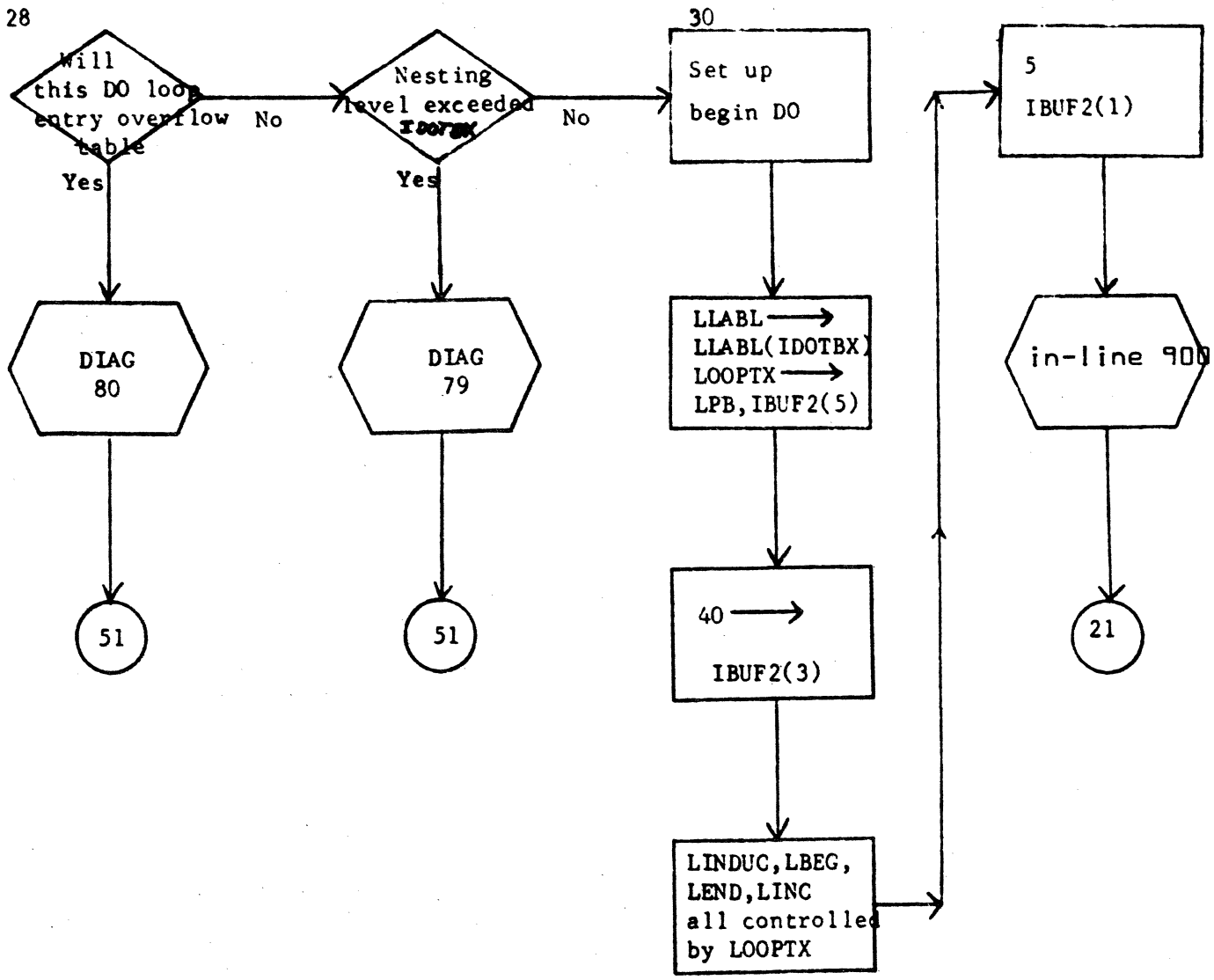
NOTES:

8. ITEMPB>0

9. Create an end DO constraint.



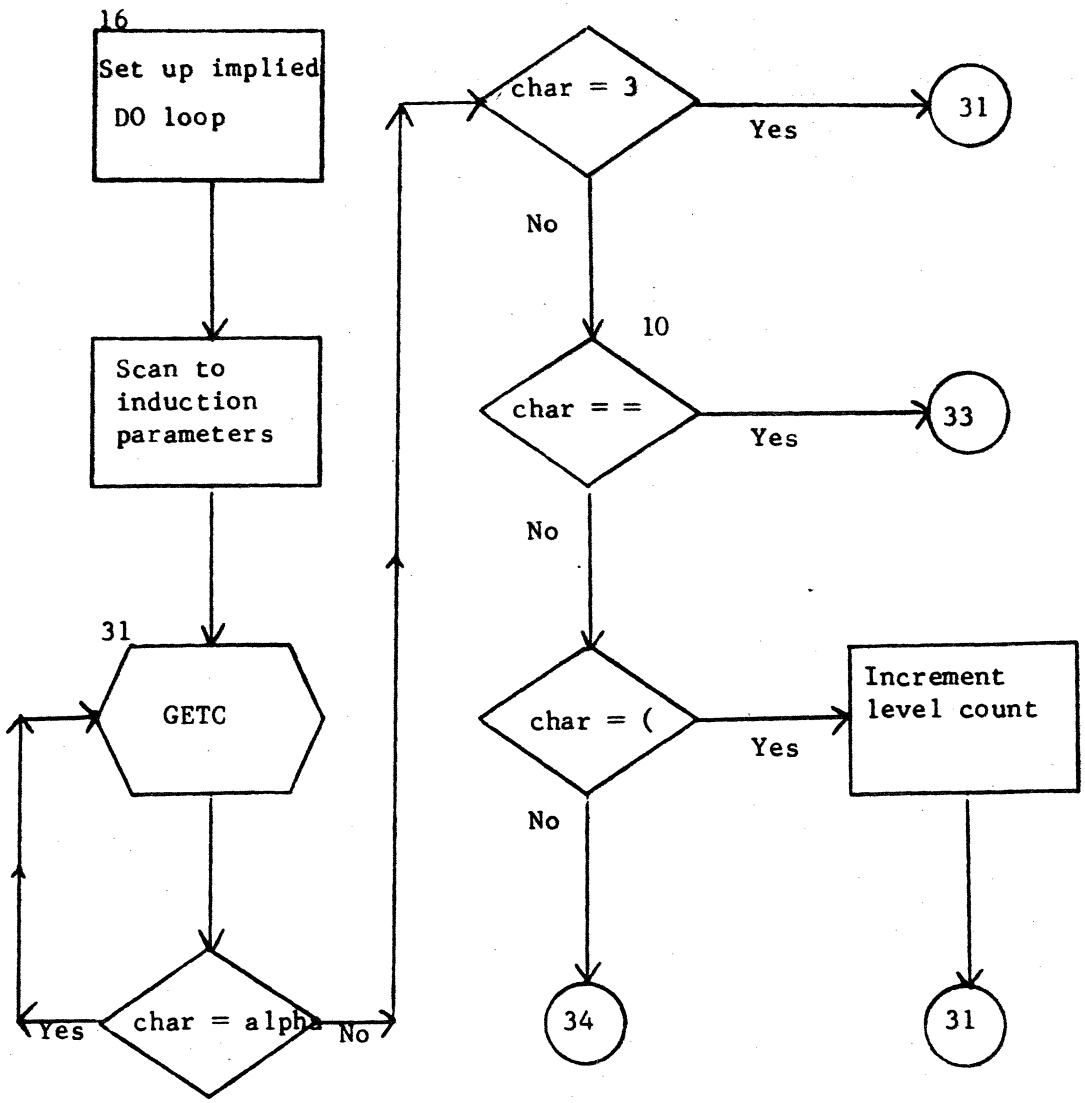
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE IOSAP		PROJECT MGR.			
FLOWCHART ☒		PAGE 13 OF 16	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 3-67	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE IOSPR		PROJECT MGR.		
		PAGE 14 OF 16	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

NOTES:

10. At zero parenthesis level.

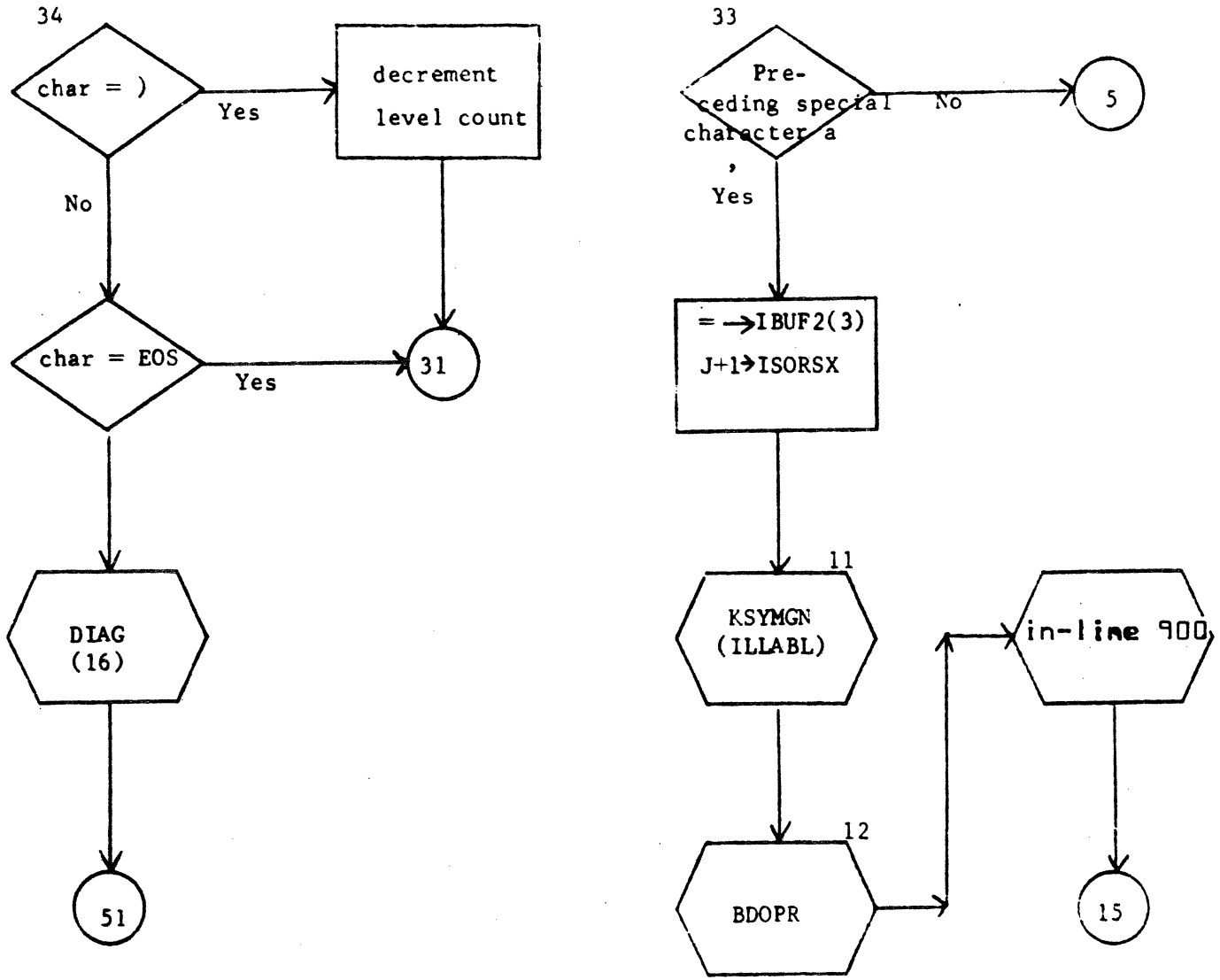


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE IOSPR	PROJECT MGR.			
	PAGE 15 OF 16	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

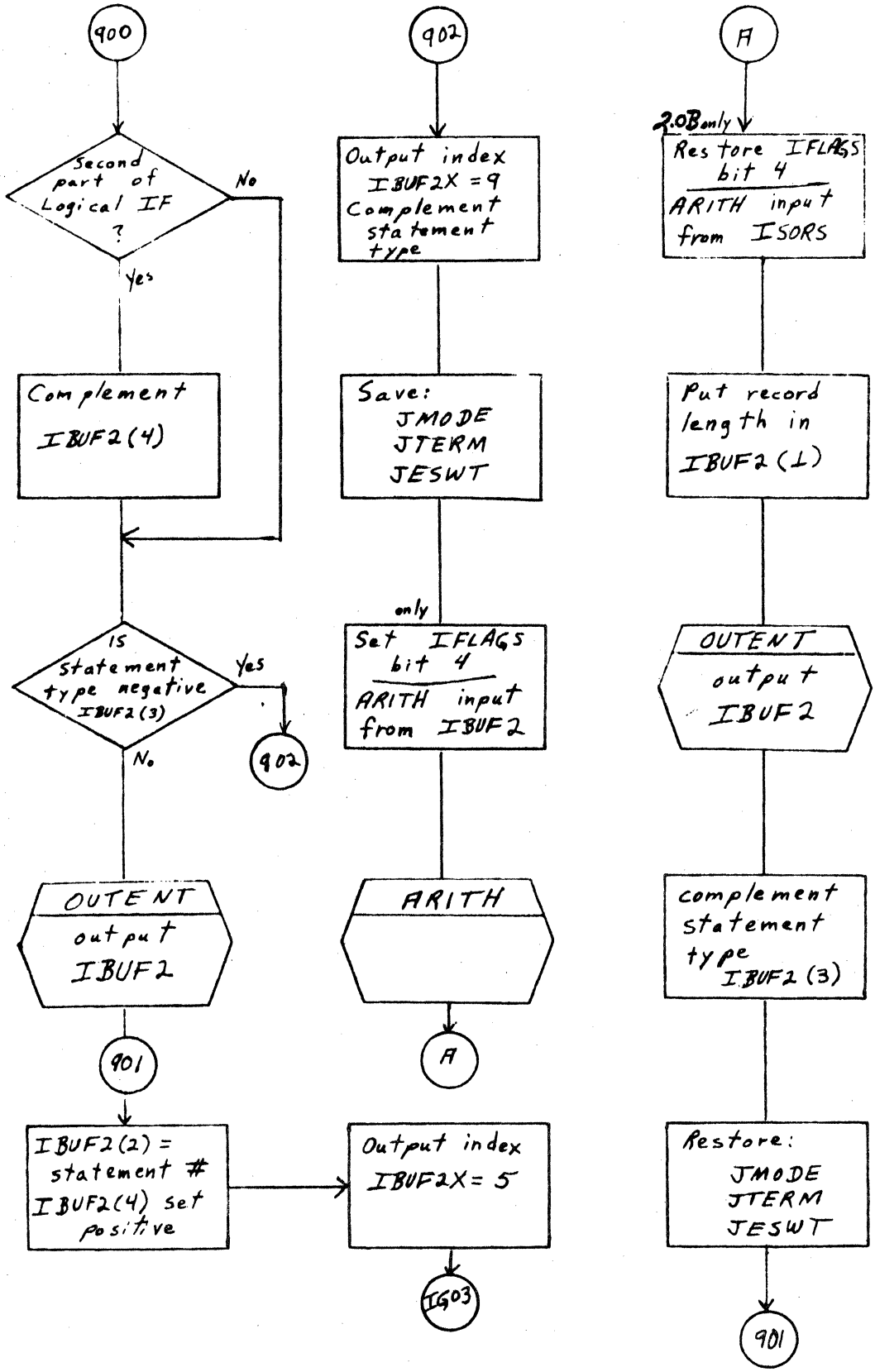
NOTES:

11. Allocate one symbol table entry.

12. Create begin DO.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE / FLOWCHART / DECISION TABLE / OTHER	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE IOSPR		PROJECT MGR		
		PAGE No OF 16	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		


900
Second part of Logical IF ?
No
Yes
Complement IBUF2(4)
IS statement type negative IBUF2(3)
Yes
902
No
OUTENT output IBUF2
901
IBUF2(2) = statement # IBUF2(4) set positive
902
Output index IBUF2X = 9 Complement statement type
Save: JMODE JTERM JESWT
only
Set IFLAGS bit 4 ARITH input from IBUF2
ARITH
A
Output index IBUF2X = 5
IG03
A
2.0B only
Restore IFLAGS bit 4 ARITH input from ISORS
Put record length in IBUF2(1)
OUTENT output IBUF2
complement statement type IBUF2(3)
Restore: JMODE JTERM JESWT
901

2.4.13 BDOPR

The BDOPR subroutine is called to process DO statements of the form:

DO label induc = lbeg, lend, incr

The parameters for this statement are as follows:

1. "label" is the statement label assigned to the final statement of the do loop.
2. "induc" is the induction parameter.
3. "lbeg" is the beginning parameter.
4. "lend" is the end parameter.
5. "incr" is the increment parameter.

2.4.13.1 "label"

A call is made to the RDLABL subroutine in order to extract the label from the source language input. Upon return from RDLABL the following is true:

1. Either (ILLABL) = 0 if either -

   a. the label is missing, or -
   b. the label is illegal,

   or (LLABL) ≠ 0 if the label is present and legal.

2. Either (ISYMD) = 0 if the label does not appear in the symbol table, or (ISYMD) ≠ 0 if an entry for the label is present. If (ISYMD) = 0, a call is made to the STORE routine to insert this label in the symbol table.

3. If (ISYMD) ≠ 0, the statement label must either be undefined, or it must occur subsequent to the DO statement. If -

   (ISNOL(ISYMX)) = 0,

   the label is undefined, and if

   (ISNOL(ISYMX)) > (ISTNO)

   the label occurs subsequent to the DO statement where -

   (ISTNO) = the statement number for the source statement currently being processed.

2.4.13.2 "induc"

The restrictions on the induction parameter are as follows:

1. It must be an integer variable.
2. It may not be preceded by an algebraic sign.
3. It must be followed by an equal sign.
4. This parameter must not be used as any parameter in another DO statement if the do-loop for that statement contains the do-loop of the statement currently being processed.

A call is made to the CKNAME subroutine to extract the parameter from the source language input.

Upon return from CKNAME either -

1. (IVCFLG) =2 if the parameter is an integer variable, or -
2. (IVCFLG) ≠ 2 if not.

If 2 is the case an error diagnostic is given prior to the return from CKNAME. Therefore, following the return from CKNAME, only an exit is made from BDOPR.

2.4.13.3 "lbeg"

The restrictions on the use of the beginning parameter are as follows:

1. It must be either an integer variable or an integer constant.
2. It may not be preceded by an algebraic sign.
3. It must be separated from the preceding parameter by an equal sign.
4. It must be followed by a comma.

A call is made to the CKIVC subroutine to extract this parameter from the source buffer. Upon return from CKIVC, either -

1. (IVCFLG) = 0 if the parameter is non integer,
2. (IVCFLG) = 1 if the parameter is an integer constant, or -
3. (IVCFLG) = 2 if the parameter is an integer variable.

If (IVCFLG) = 0, an error diagnostic is given by the CKIVC subroutine. Therefore, after returning from CKIVC, only an exit is made from BDOPR.

2.4.13.4 "lend"

The restrictions for the use of this parameter are the same as those for the "lbeg" parameter. The CKIVC subroutine is used to extract this parameter from the source buffer the same as for the "lbeg" parameter.

2.4.13.5 ▽incr▽

The restrictions for the use of this parameter are the same as for the ▽lbeg▽ parameter with one exception:

> The increment parameter may be preceded by a minus sign. This parameter is also followed by an EOS. This parameter is optional and may be omitted. If so, the ▽lend▽ parameter is followed by an EOS. A value of 1 is substituted for the increment parameter.

2.4.13.6 Output of BDOPR

The BDOPR subroutine will generate a 5 word output entry for the do-loop table and a 1 word output entry for the output buffer.

2.4.16.1 Do-Loop Table Entry

▽LOOPT▽ = name of table
{LOOPTS} = capacity of the table
{LOOPTB} = length of a table entry
{LOOPTX} = pointer to the entry in the table

A DO-LOOP table entry is five words long. The format of the DO-LOOP table is as follows:

WORD1:
- LEND = bits 0-15 of LOOPT{4}.
- {LEND} = pointer to symbol table entry containing the ▽lend▽ parameter.

WORD2:
1. LID = bit 15 of LOOPT{5}.
   {LID} = 1 if the ▽incr▽ parameter is preceded by a minus sign.
2. LINDUC = bits 0-15 of LOOPT{1}.
   {LINDUC} = pointer to the symbol 1 table entry containing the ▽induc▽ parameter.

WORD3:
- LBEG = bits 0-15 of LOOPT{2}.
- {LBEG} = pointer to the symbol table entry containing the ▽lbeg▽ parameter.

WORD4:
- LINC = bits 0-15 of LOOPT{3}.

{LINC} = pointer to symbol table entry containing the ▽incr▽ parameter.

WORD5: LLABL = bits 0-14 of IDOTB{5}.
{LLABL} = pointer to symbol table entry containing the ▽label▽ parameter.

2.4.13.6.3 Output Buffer Entry

An entry made in the output buffer consists of 1 word, a pointer to the LOOPT entry for this DO statement.

{LOOPTX} ⟶ IBUF2{IBUF2X}

2.4.13.7 Errors

Error diagnostics are given by BDOPR for the following:

1. An illegal label or a missing label.
2. The label occurs prior to the DO statement.
3. Overflow of the DO-LOOP table.
4. Overflow of the NESTED-DO-LOOP table.
5. The induction parameter is currently in use as an induction variable in a do-loop containing the do-loop for the statement currently being processed.
6. Illegal characters separating the parameters in the DO statement.

Errors 1, 2 & 5 will cause termination of the DO statement processing, and an exit will be made from BDOPR following the printout of the error diagnostic. Following the printout of the error diagnostic for errors 3, 4 & 6, processing of the DO statement will be resumed.

2.4.13.8 Induction Variables

Indicators are to be set in each of the symbol table entries containing the "lbeg", "lend", "induc" and "incr" parameters to show that these are induction variables. These indicators are reset when the statement label is encountered that marks the end of the do-loop. Such an indicator is set by -

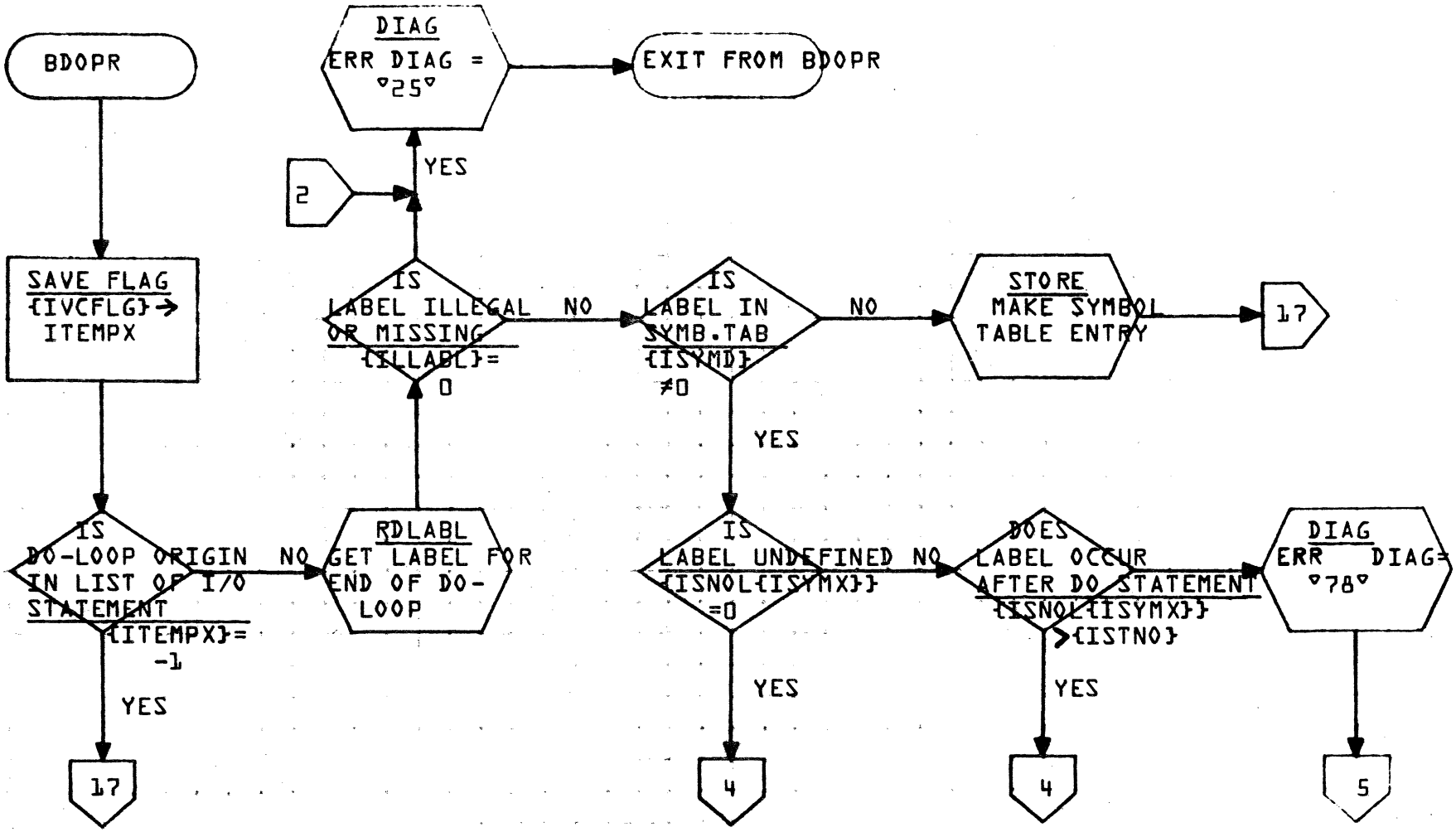
1 ⟶ INDUCV(ISYMX)

2.4.13.9 Do-Loops Originating in I/O Lists

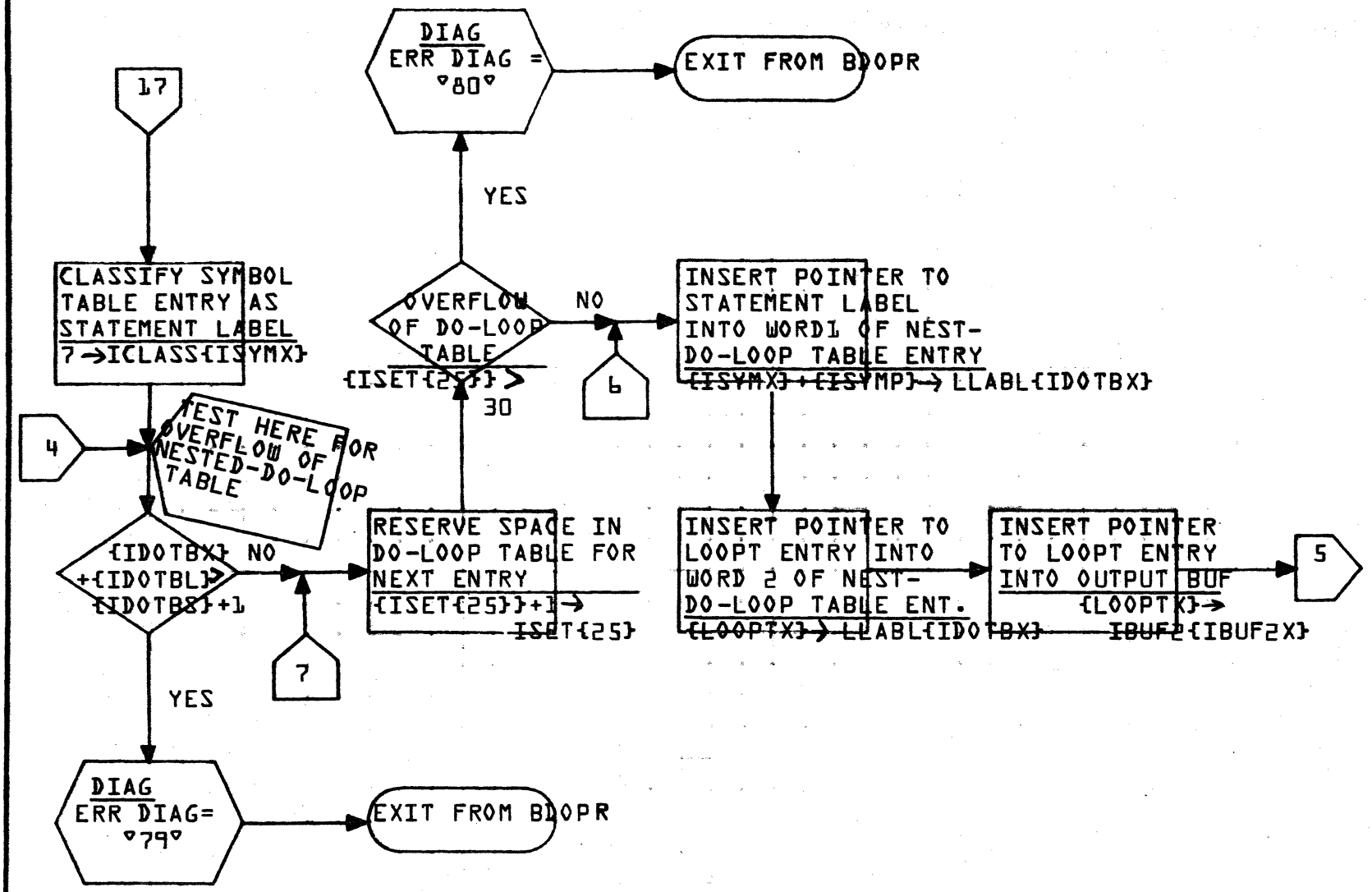
A DO statement in the source buffer is terminated by an EOS. A do loop may originate in an I/O list such as -

WRITE (unit, format statement label)(A(I),I=1,10),...

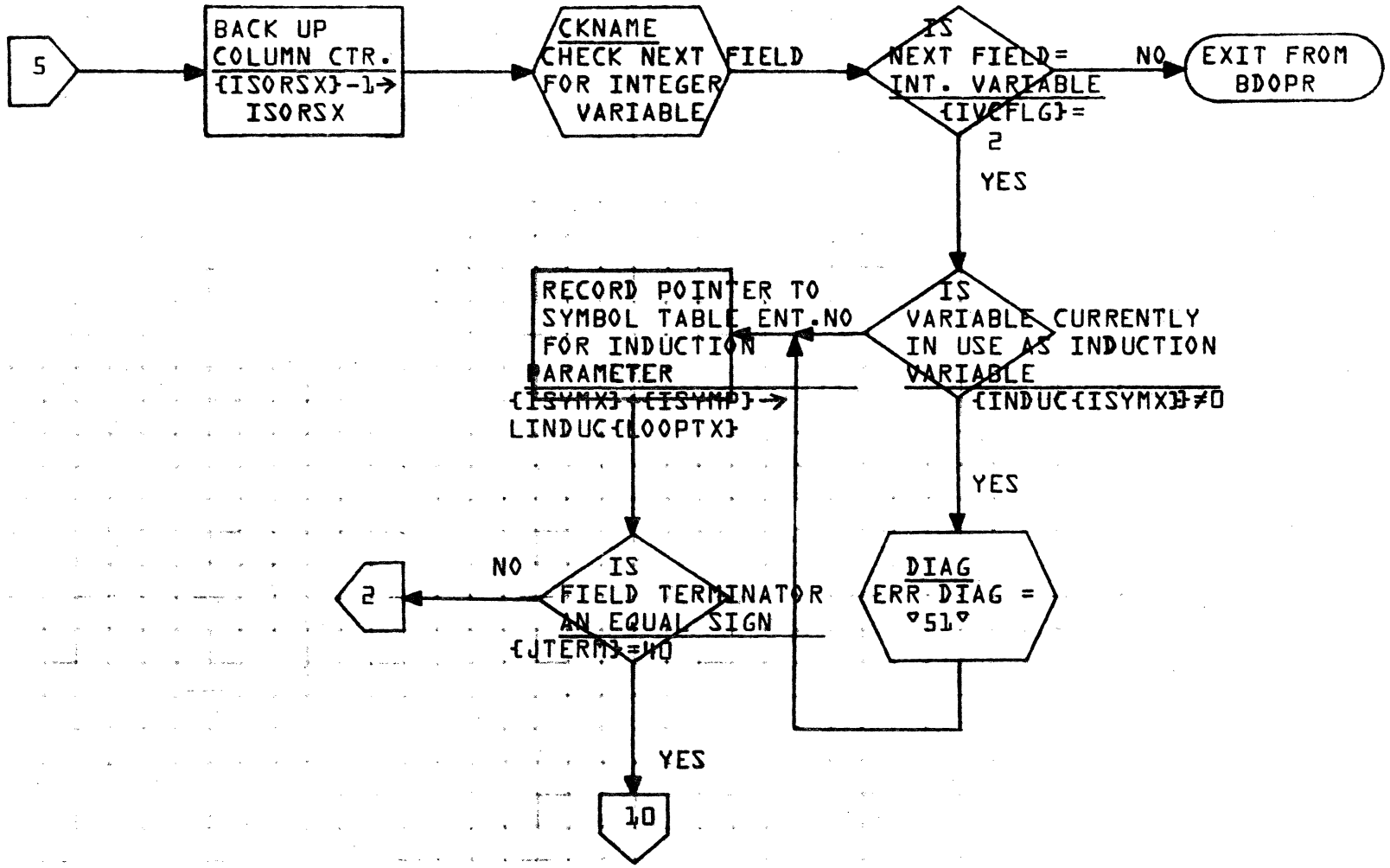
If so, an indicator is set prior to entering BDOPR. This indicator is recorded in ITEMPX. If the origin of the do loop is in an I/O list, (ITEMPX) = -1. If this is the case, a right parenthesis replaces an EOS as the end of the statement in the processor.



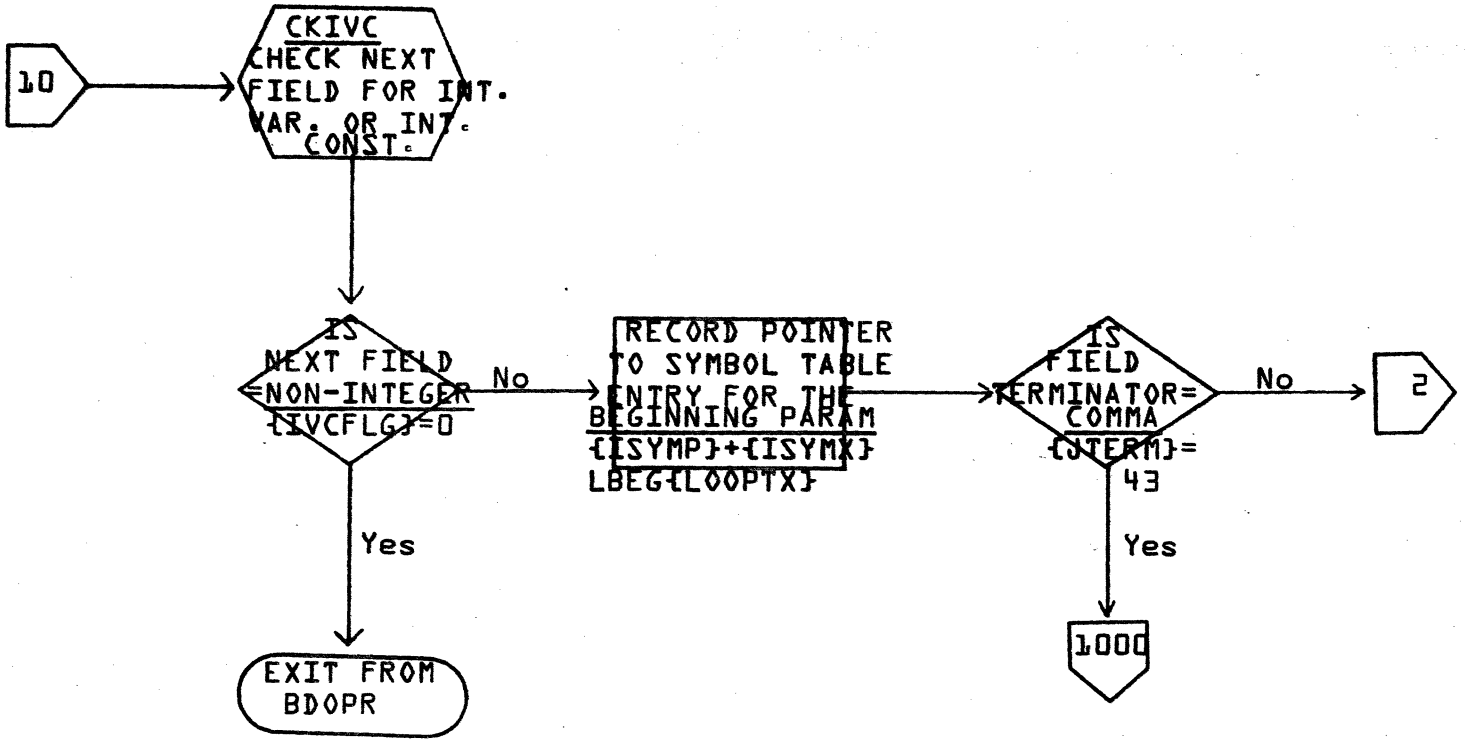
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BDOPR		PROJECT MGR.		
		PAGE 1 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



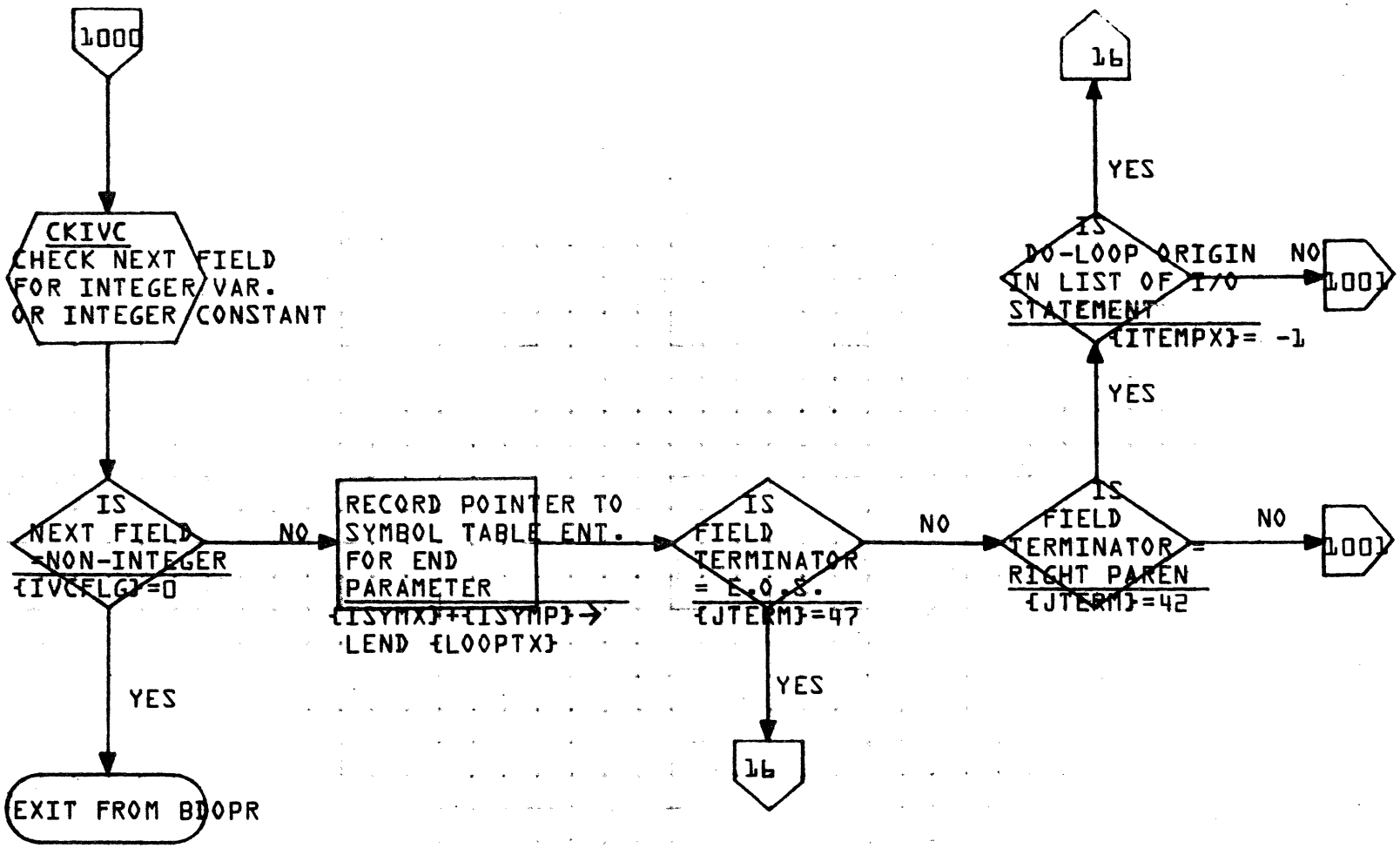
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BDOPR		PROJECT MGR.		
		PAGE 2 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



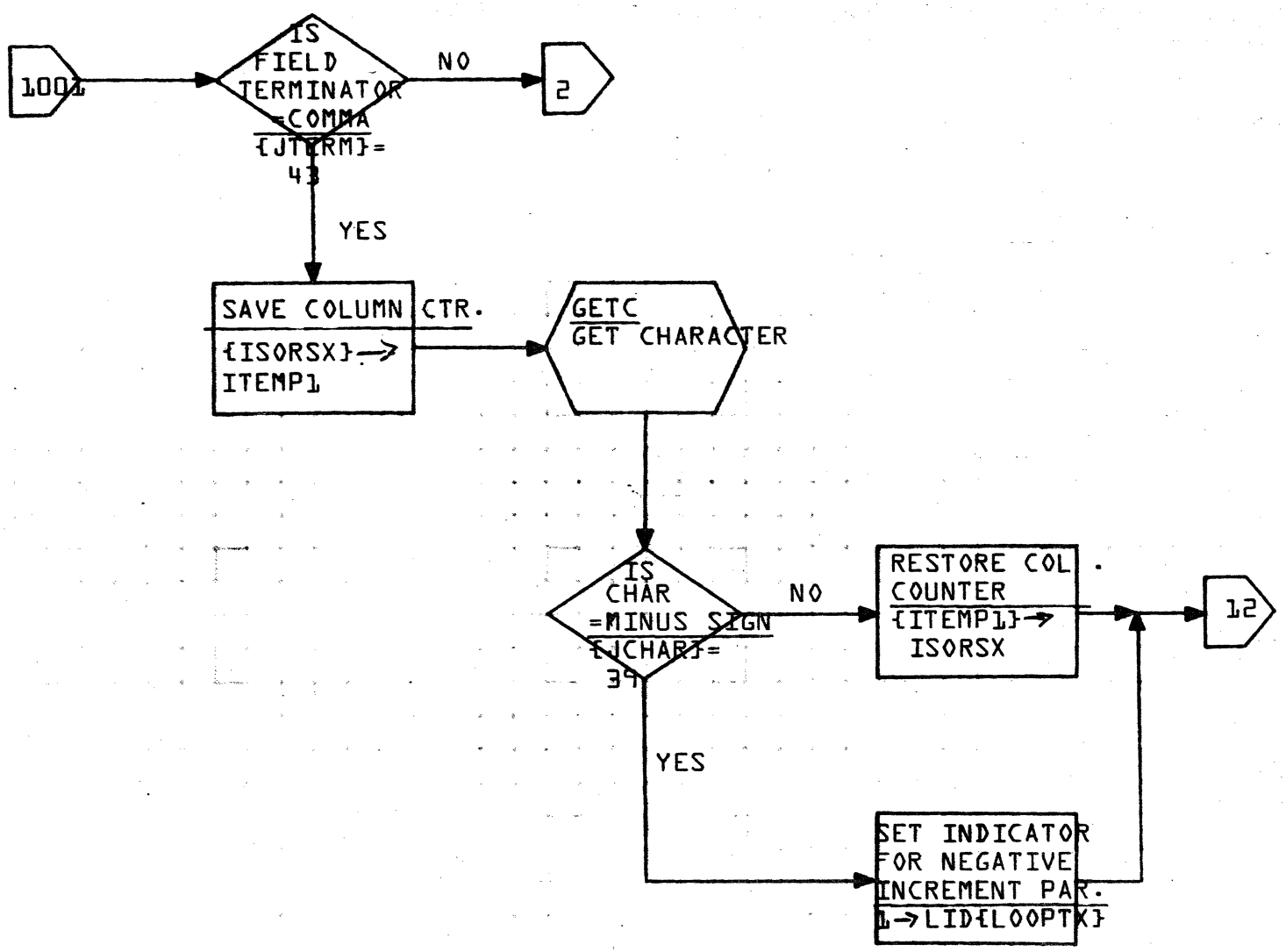
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BDOPR	PROJECT MGR.			
	PAGE 3 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BDOPR		PROJECT MGR.			
		PAGE 4 OF 8	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

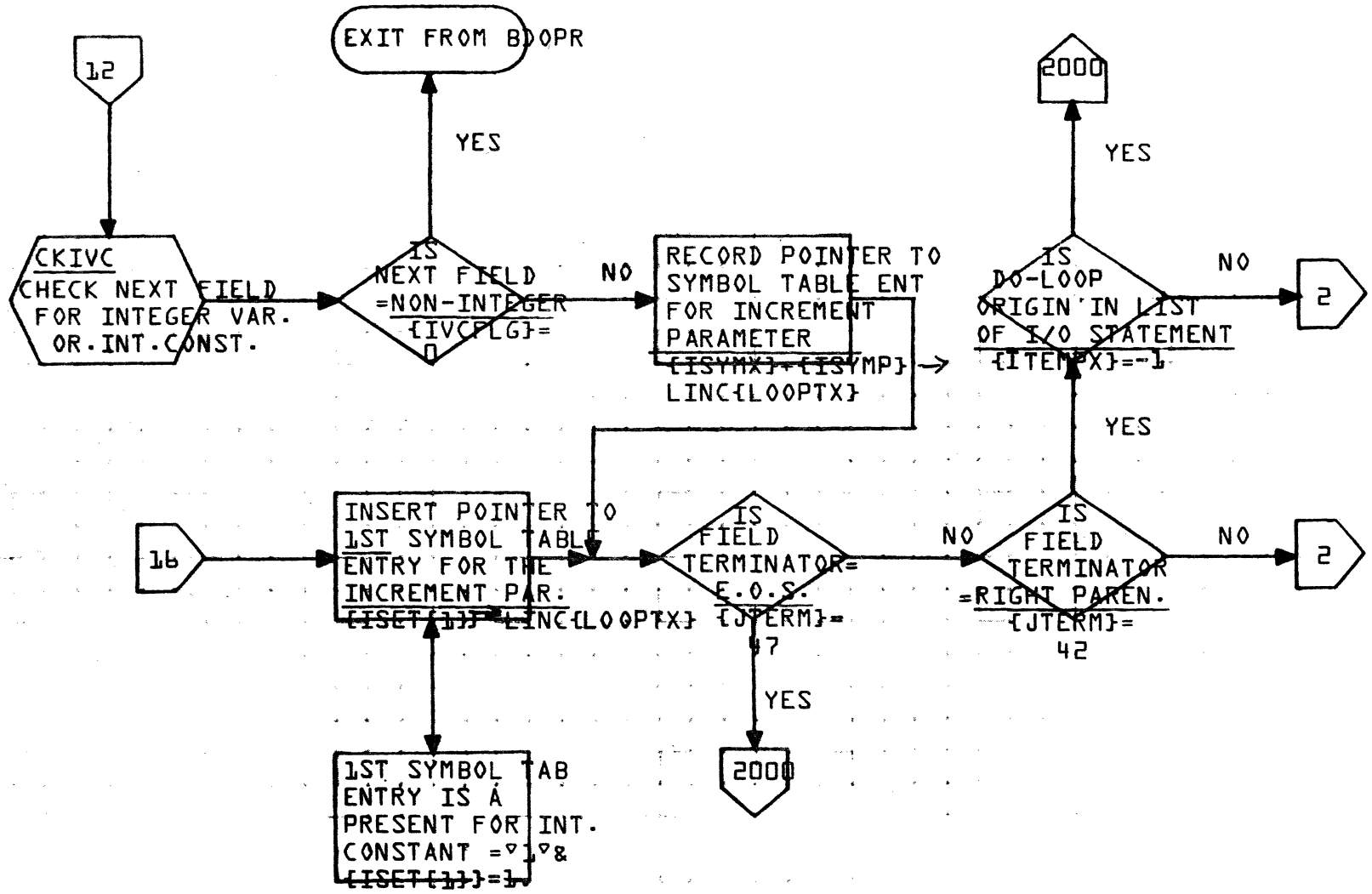


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BDOPR	PROJECT MGR.			
	PAGE 5 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

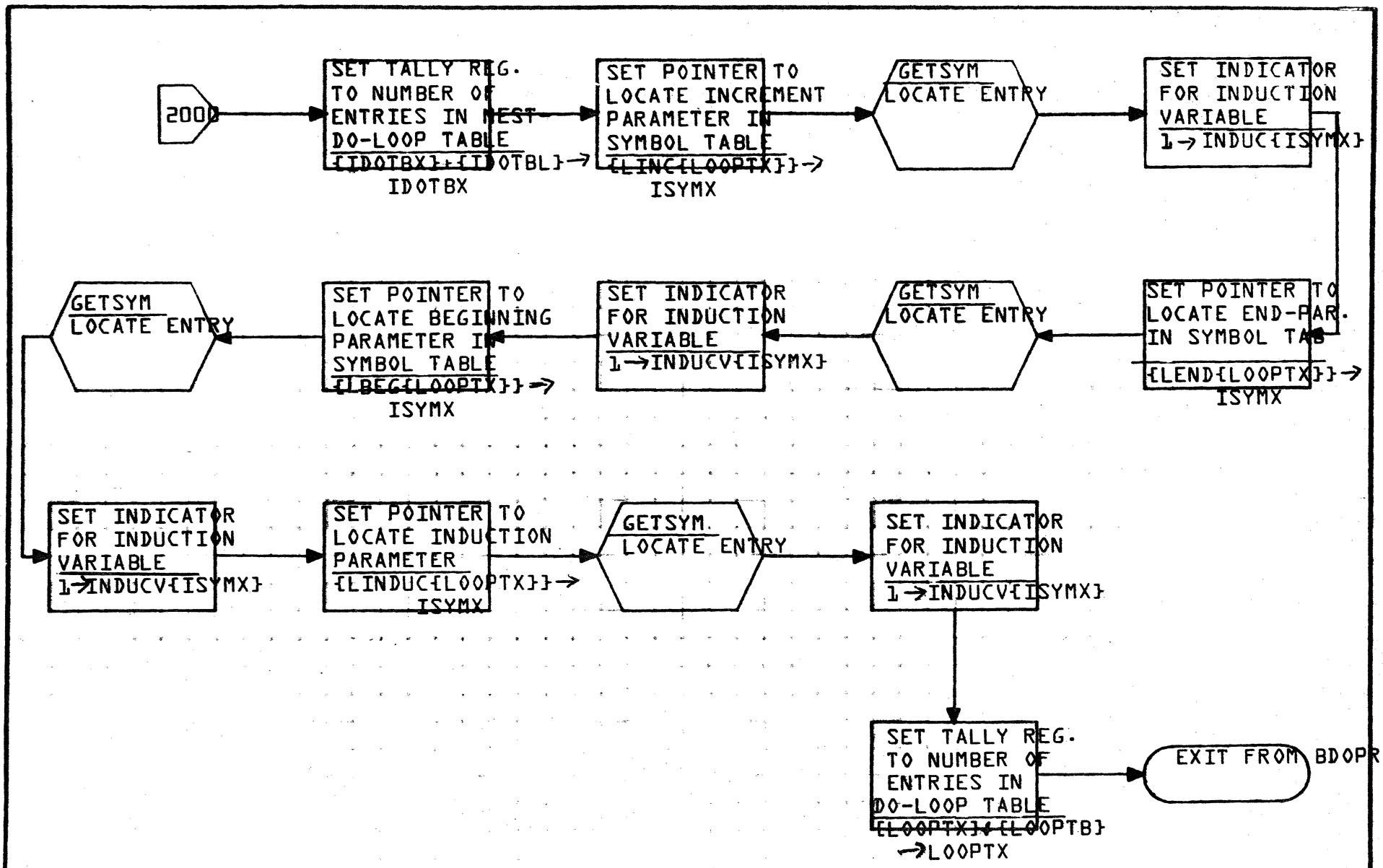


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BDOPR		PROJECT MGR.		
		PAGE 6 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

2-288



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BDOPR	PROJECT MGR.			
	PAGE 7 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BDOPR		PROJECT MGR.			
		PAGE 8 OF 8	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

2-290

3.3 Special Subroutines for Common Allocation

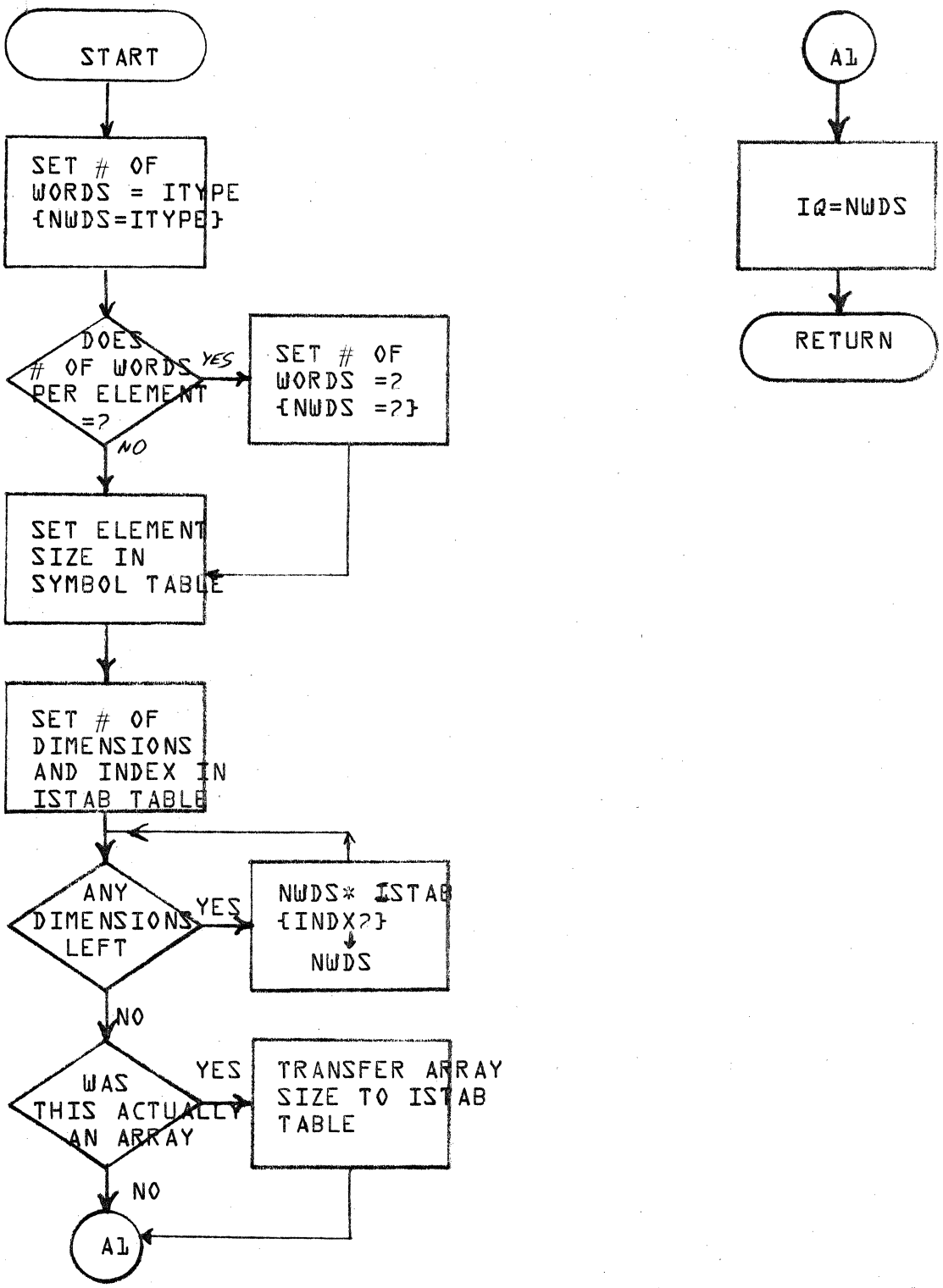
3.3.1 Subroutine ARAYSZ

ARAYSZ is called to determine the number of words in any given array. The number of words is determined by the number of elements in the array times the number of words per element. If the array is real or integer with ASA options selected, the number of words per element will equal 2; if the array is double precision, the number of words per element will equal 3. ARAYSZ determines the number of words per element and sets the element size in the symbol table (KELSIZ). It then picks up the number of dimensions and the index into the ISTAB table. The number of words in the given array is calculated and the array size is transferred to the ISTAB table at the first entry for the array.

CONTROL DATA CORPORATION

LA JOLLA FACILITY

ARAYSZ

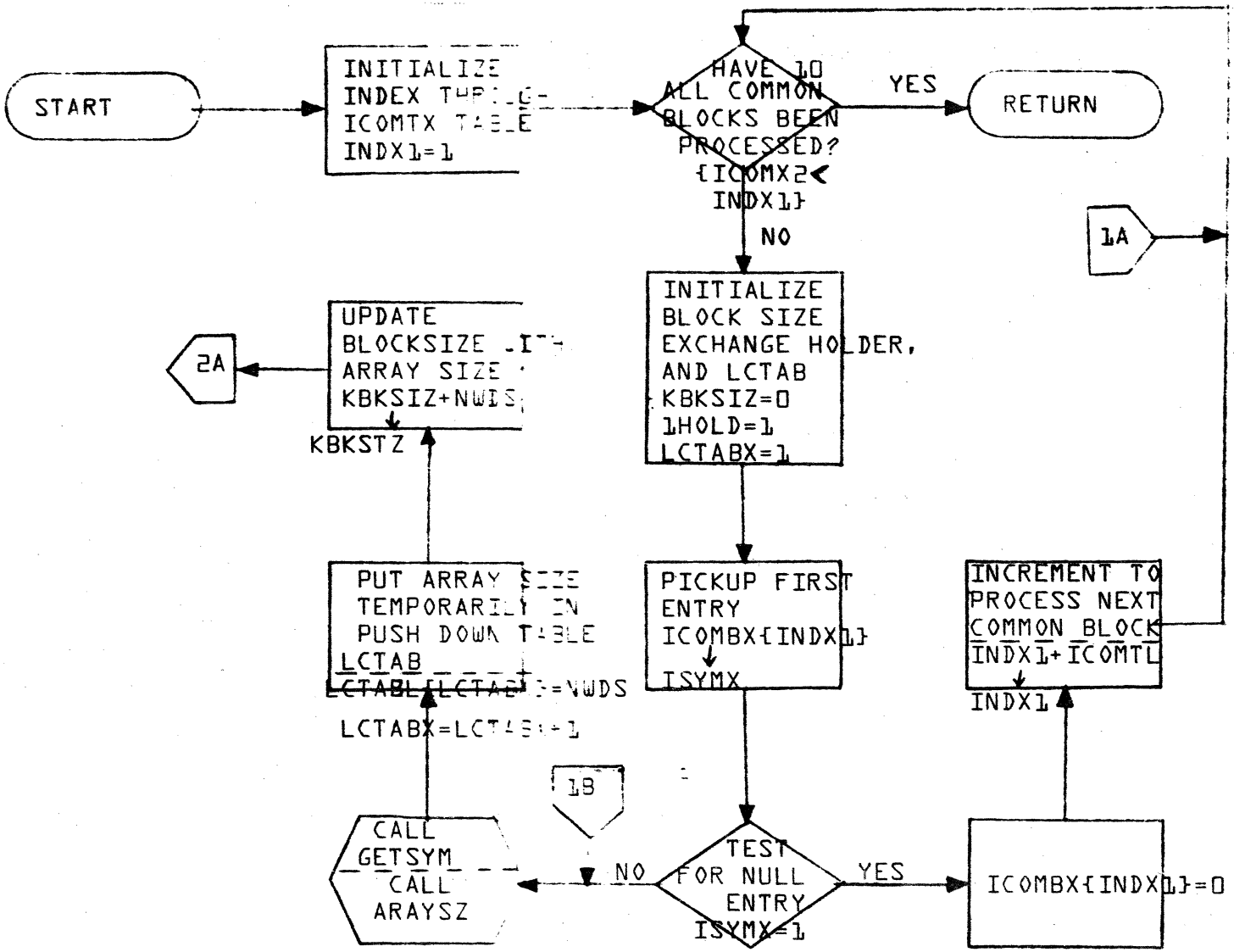


TITLE	ARAYSZ		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET / OF 3

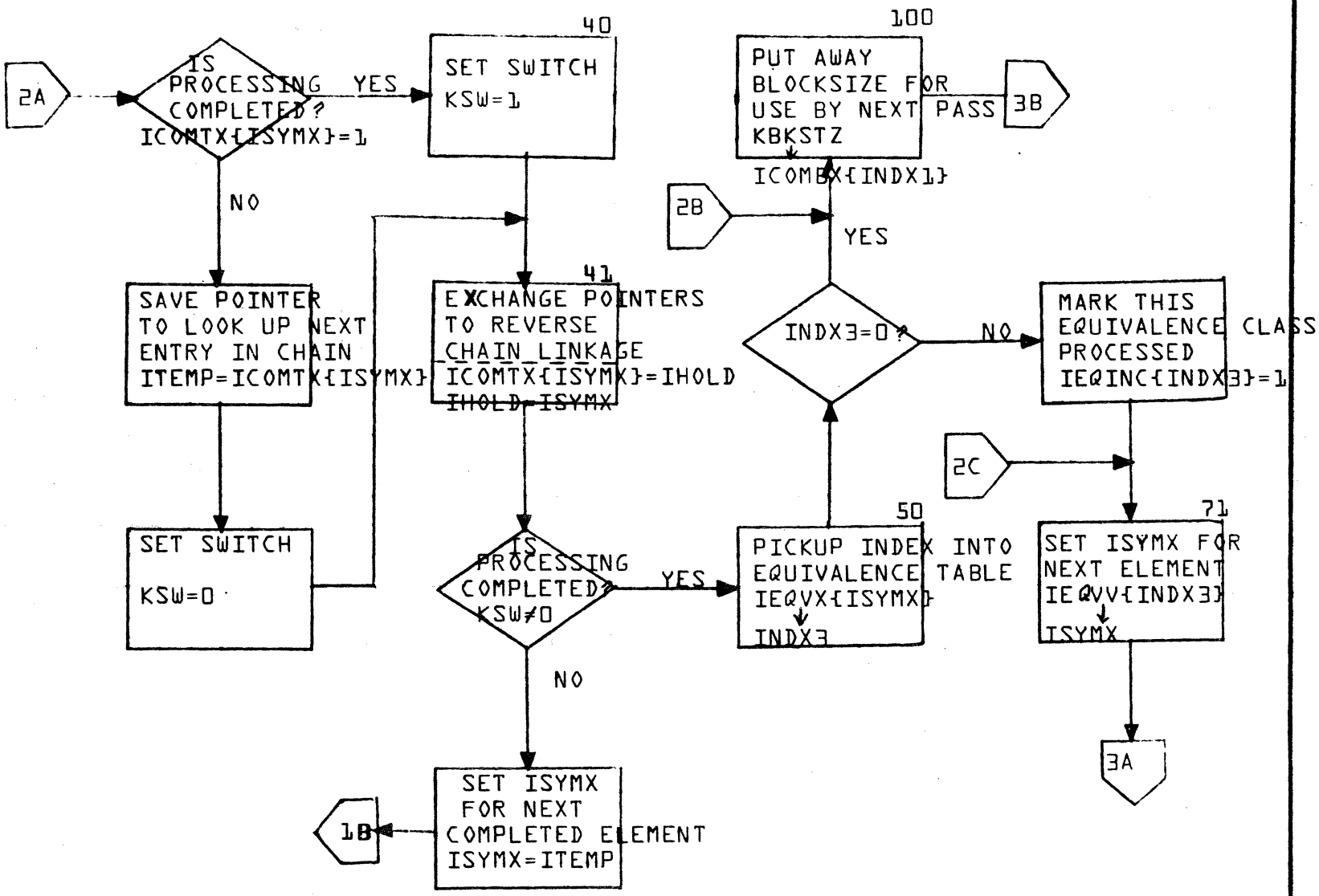
CSD 203

3.3.2 Subroutine CPLOOP

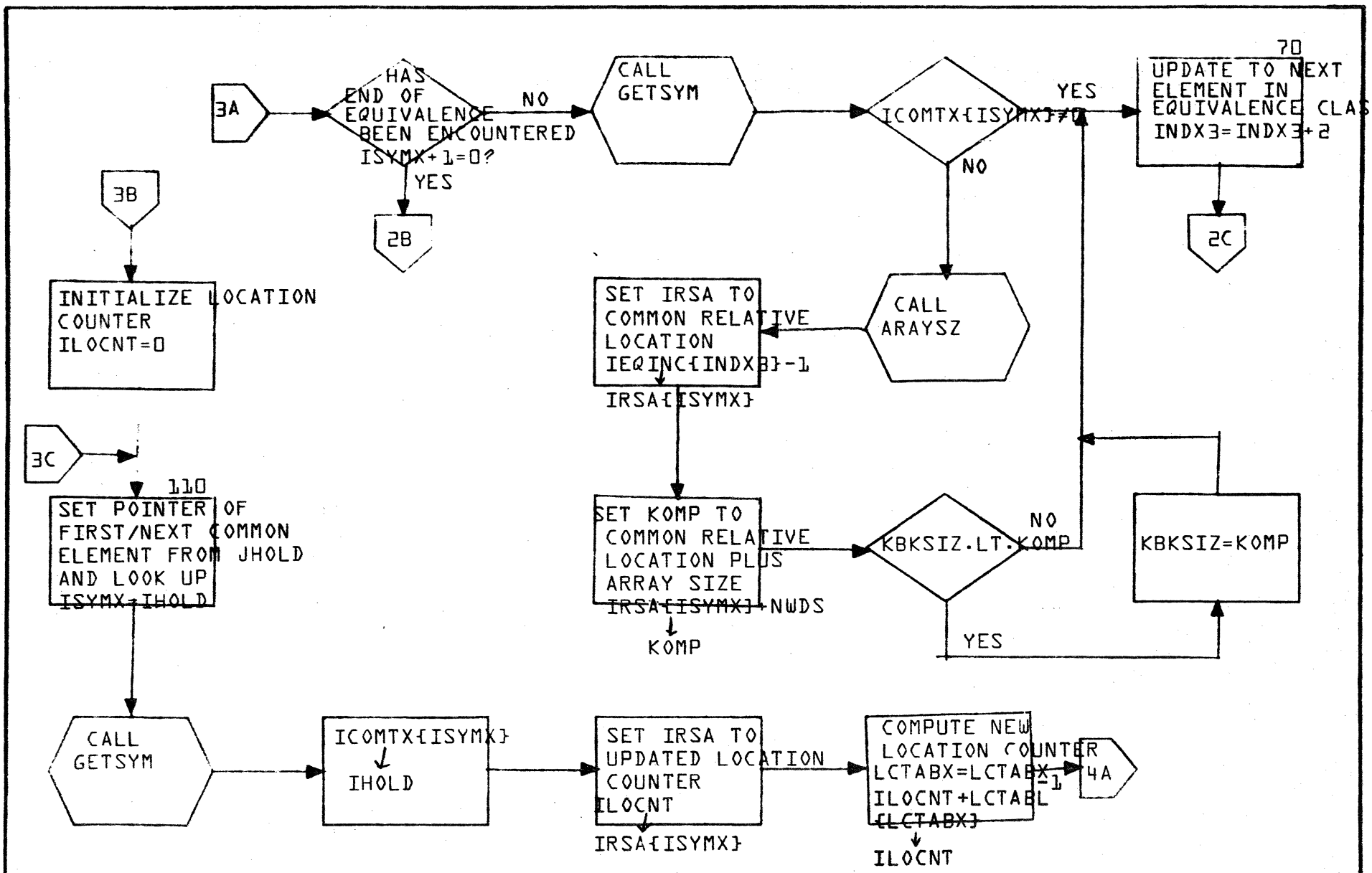
CPLOOP is called by PHASE^{A} It allocates, in consecutive order, the space needed for each common block. CPLOOP operates in two passes, the first of which determines the size of each array in common, and the second of which allocates space. Two passes are necessary because common is strung in the reverse direction from which the COMMON statements came in. CPLOOP takes into account any EQUIVALENCE statements to variables and/or arrays in common, and lengthens the COMMON BLOCK if necessary based on these EQUIVALENCES.



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE CPLXIF		PROJECT MGR.		
FLOWCHART ☒		PAGE 1 OF 4	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		



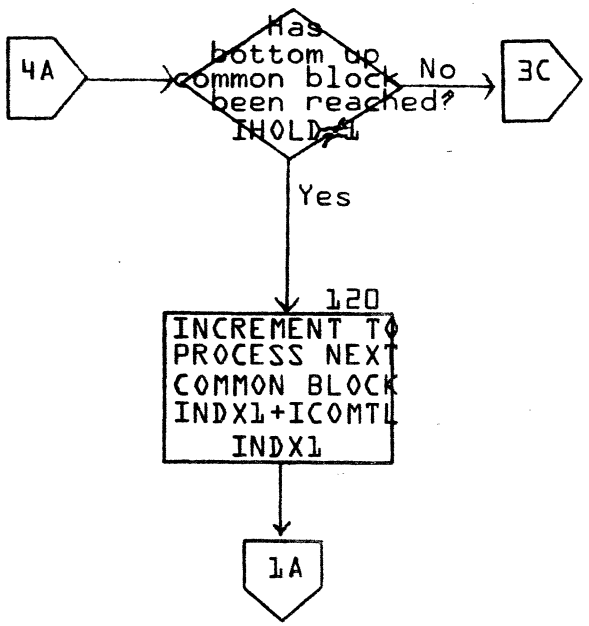
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE CPLOOP		PROJECT MGR.			
		PAGE 2 OF 4	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE CPLOOP		PROJECT MGR.		
		PAGE 3 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE CPLOOP	PROJECT MGR			
	PAGE 4 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

3.4.0 Arithmetic Expression Processors

3.4.1 Subroutine ARITH

ARITH is the Phase A subroutine which itemizes and syntax checks arithmetic expressions.

ARITH processes an expression found in ISORS (ISORSX)* to either end of statement or zero parenthesis level, dependent on the setting of ISTOP.

The itemization and syntax check are done concurrently and an itemized reconfiguration of the source input is built in IBUF1. If syntax errors are encountered, error diagnostics are output.

If a subscript of a variable is encountered, a call to SUBSCR is made to expand the subscript to a polynomial.

When the scan of ISORS* is complete, IERR is checked. If IERR ≠ 0, ARITH exits. If IERR = 0, ARITH calls subroutine TREE to reconfigure the items in IBUF1 into a tree structure representing the arithmetic expression. In the 2.0A version, ARITH passes a mixed-mode flag to TREE in the common variable LS1.

Upon return from TREE, ARITH exits to its caller. In the 2.0B version, ARITH calls MODMXP just before exit if mixed mode arithmetic has been encountered.

Input parameters:

ISORS*	-	buffer in which to find statement to analyze
ISORSX*	-	Indexer into ISORS
ISTOP	-	Switch signalling when to stop analysis 0 = end of statement 1 = zero parenthesis level (If statement 2 = process single variable (READ,WRITE)
IBUF2	-	Set to the next available word (relative to start) in the output buffer (IBUF2)
ISTNO	-	Current statement number

Output parameters:

IBUF2	-	Contains the expression in tree form
IBUF2X	-	Set to next available word in IBUF2
IERR	-	Switch signalling syntax error
ISORSX	-	Updated to reflect end of expression or character following zero parenthesis level.

Subroutines used:

GETF	-	Get one field from ISORS
GETSYM	-	Get symbol table page
SYMBOL	-	Search symbol table
STORE	-	Store symbol in symbol table

* If AND (IFLAGS,$10) is not zero, input is in IBUF2 (IBUF2X). In 2.0A, there are two versions of ARITH.

ARITH Switches and Tables

Switches and tables for Syntax check

IBUF1	Table where reconfigured expression is built
IBUF1X	IBUF1 index
LS1	Prior symbol 0 = beginning of expression -1 = illegal
IERR	Syntax error switch
LPREN	Parenthesis counter
ISORS	Table containing input expression
ISORSX	ISORS indexer
LSWITC	Subroutine or function call in effect switch
ISTOP	Input switch signalling where to stop analysis 0 = end of statement 1 = zero parenthesis level 2 = after processing one variable
MODE	Mode of current expression
IWORK	Mode table for exponentiation or function calls. 2 word entry. Word 1 = mode Word 2 = LPREN setting for return to mode in word 1
IMSW	IWORK index
ITEM	Temporary holder
LO	Logical operator switch
LRELRQ	Relational operator required switch
LRELEN	Relational operator encountered switch

Operators

GROUP	LEVEL	GROUP MEMBERS
comma	0	comma
OR	1	OR
AND	2	AND
NOT	3	NOT
Relational	4	LT, GT, LE, GE, EQ, NE
-	5	+,-
*	6	*,/
**	7	**
(	8	(,), function

Function is a special variable treated like a set of parentheses. It and its parameters are not reorderable, and can be looked at only just as it sits.

INVERSE operands

If operator:	Operand is --
*	divisor
**	thing to exponentiate (coefficient)

SYNTAX LEGALITY TABLE (P - permissible)

FIRST \	REL	OR	AND	NOT	VAR	*/**	+-	(	)	,	EOS	CONST.	FUNC
RELAT	-	-	-	-	P	-	P	P	-	-	-	P	P
OR	-	-	-	P	P	-	P	P	-	-	-	P	P
AND	-	-	-	P	P	-	P	P	-	-	-	P	P
NOT	-	-	-	-	P	-	P	P	-	-	-	P	P
VARIABLE	P	P	P	-	-	P	P	P	P	P	P	-	-
*/**	-	-	-	-	P	-	-	P	-	-	-	P	P
+-	-	-	-	-	P	-	-	P	-	-	-	P	P
(	-	-	-	P	P	-	P	P	-	-	-	P	P
)	P	P	P	-	-	P	P	-	P	P	P	-	-
,	-	-	-	-	P	-	P	P	-	-	-	P	P
CONSTANT	P	P	P	-	-	P	P	-	P	P	P	-	-
BEGINNING	-	-	-	P	P	-	P	P	-	-	P	P	P
FUNCTION	-	-	-	-	-	-	-	P	P	P	-	-	-

3.4.2 Subroutine SUBSCR

SUBSCR expands a subscript of a variable into one of the following polynomials:

one dimensional array:

subscript = (C1*X1 ± D1) *E

two dimensional array:

subscript = (C1*X1 ± D1 + A*C2*X2 ± A*D2 ± A) *E

three dimensional array:

$$\text{subscript} = (C1*X1 + A1*C2*X2 + A1*A2*C3*X3 \pm D1 \pm A1*D2 - A1 \pm A1*A2*D3 - A1*A2) *E$$

Where A_i is the dimension of the variable being subscripted, E is the number of words in an array element, and the permissible form of a subscript dimension is:

$$C_iX_i \pm D_i$$

to a maximum of three dimensions.

Depending on the type of resulting polynomial when all constant terms are collected, the variable is set to incrementally subscripted (the polynomial has no variable terms), variable subscripted (the polynomial has only a single simple variable), variable and increment subscript (the polynomial has a single simple variable and a constant term), or complex subscript. The subscript is moved to IBUF1.

Subroutines used: SYMBOL
STORE
GETF

Switches and Tables

ISS	3 word table containing dimensions of subscripted variable
ISSX	current dimension
IV	IBUF1 index of variable associated with subscript
IVM	Word length of variable associated with subscript
ID1	3 word table of constant terms in subscript
IC1	3 word table of constant multipliers in subscript
IX1	3 word table of variables in subscript
ISFLAG	+, - flag for subscript terms

1. CALL statement.
2. Error in field.
3. Set to null field.

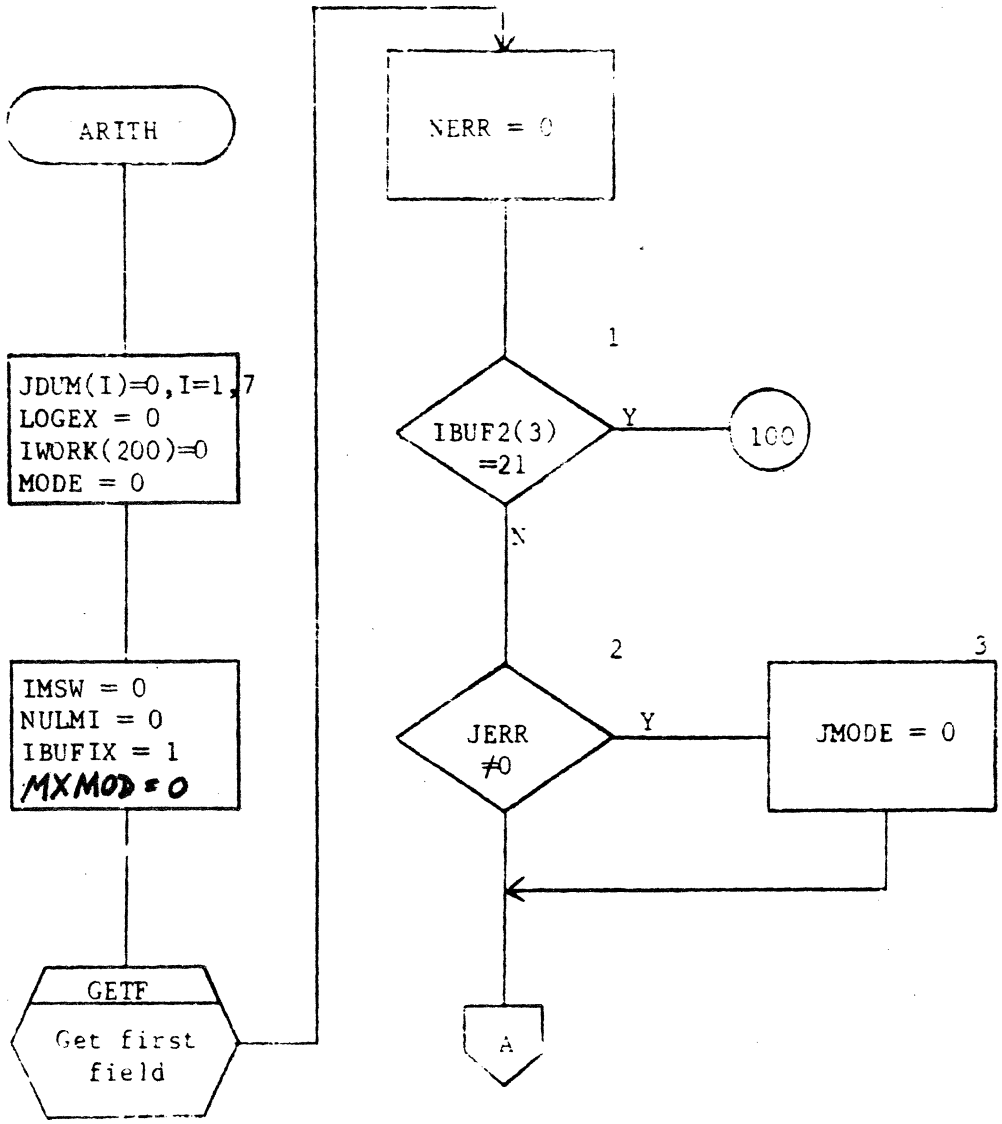
This routine syntax checks the source statement and stores it in pre-tree form in IBUF1 for TREE to process.

IBUF1 Form

WORD 1 Type

WORD 2 level

WORD 3 symbol table pointer (if necessary).



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART X DECISION TABLE OTHER	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR		
		PAGE 1 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE 7/1/	TASK NAME		

4. IF statement if ISTOP=1.

5. IF must start with "(" diagnostic 7.

6. Bump left parenthesis count and set "(" operand type.

7. Set operator level; set last symbol to "("; increase index by 2.

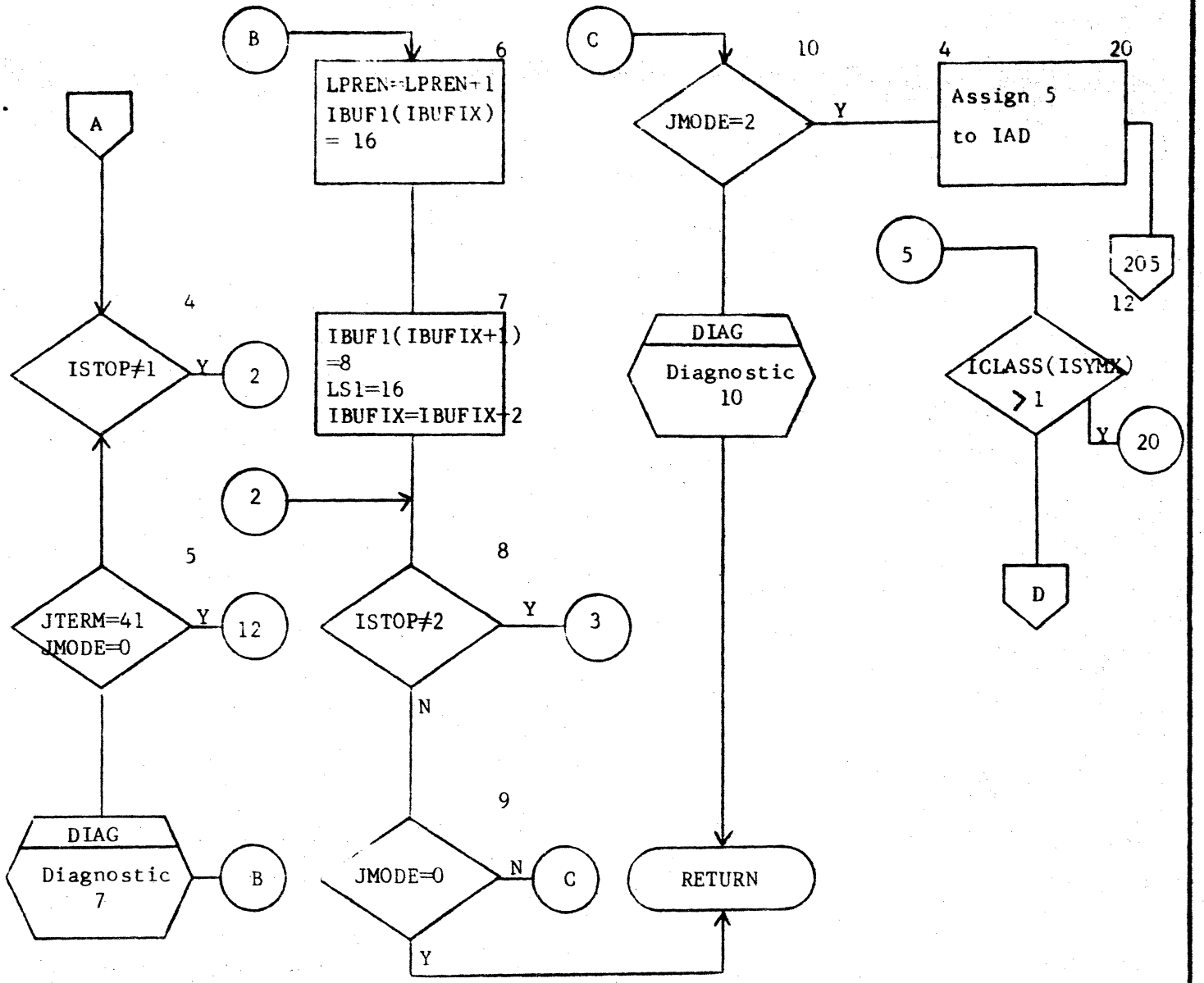
8. ISTOP=2 means process a variable.

9. Null character.

10. JMODE=2 means field is alphanumeric.

11. Illegal operator or operand.

12. 0 means unassigned 1 means variable or function definition name.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE [ ] FLOWCHART [X] DECISION TABLE [ ] OTHER [ ]	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BRITH	PROJECT MGR.			
	PAGE 20F 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

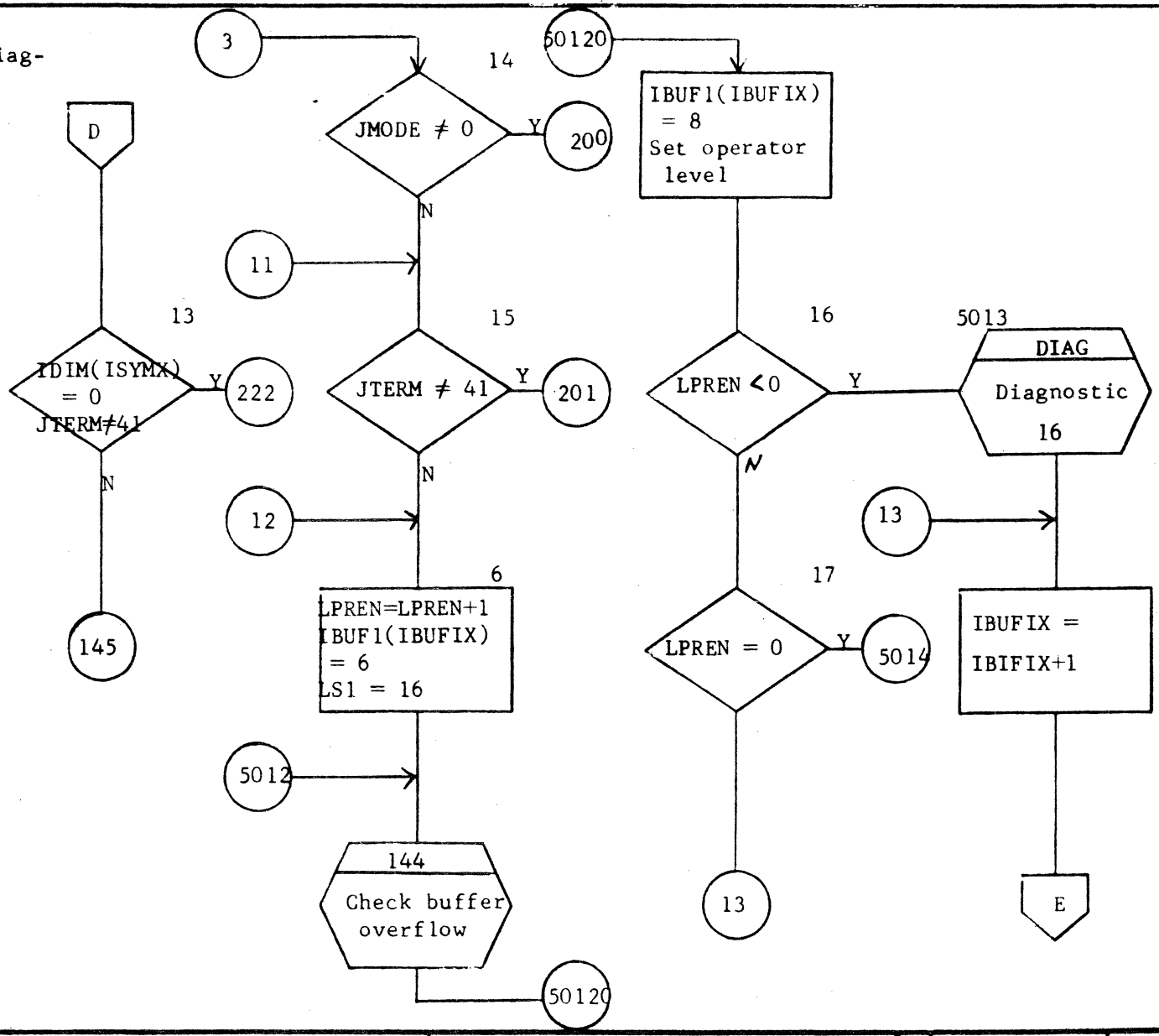
13. If dimension and "(", diagnostic and slew to ")". Otherwise class is a variable.

14. Not a null character; process the symbol.

15. Not "(", process other terminator.

16. Unmatched parenthesis.

17. Matched parenthesis - check for exit.



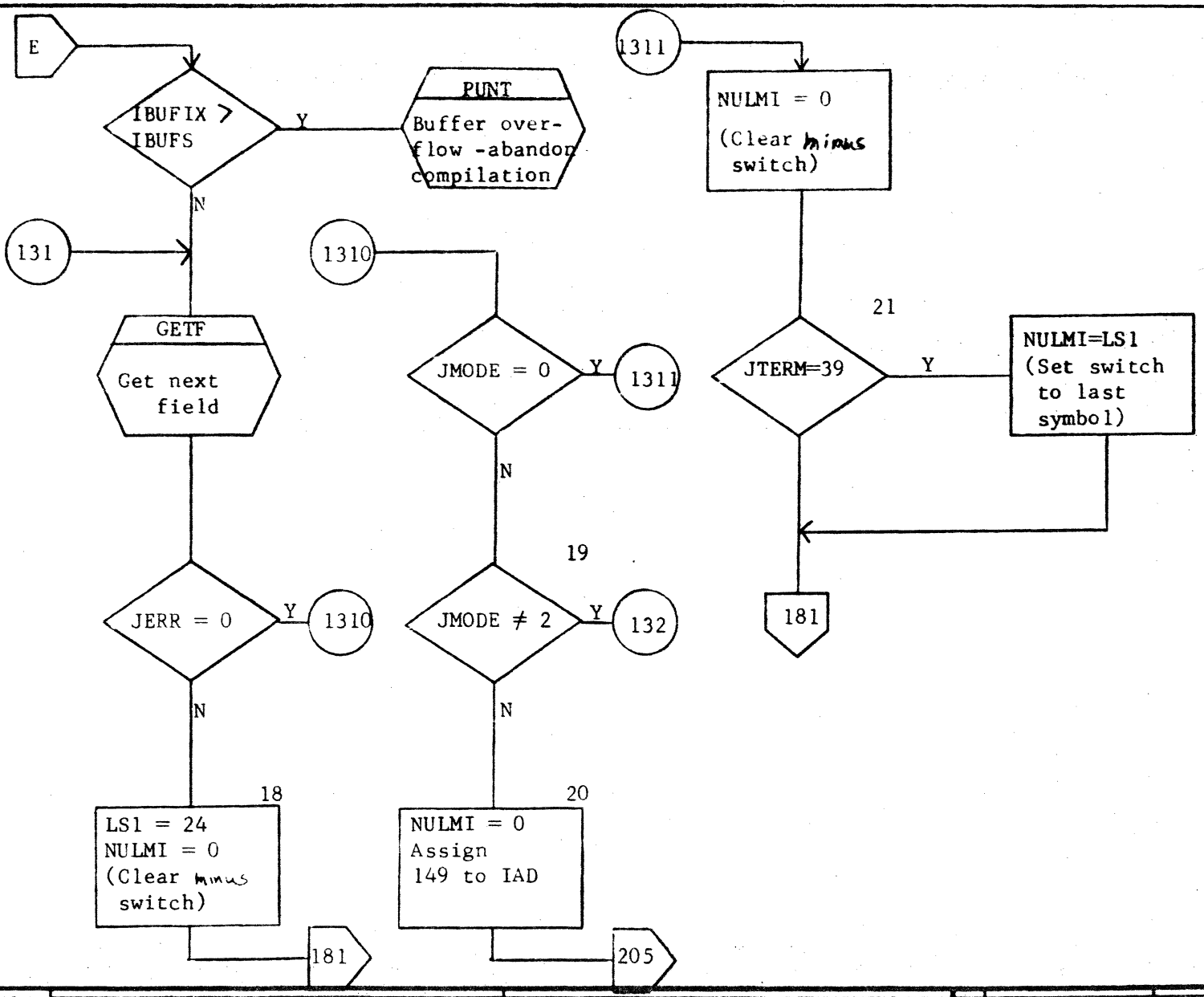
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.		
		PAGE 3 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

18. If error set last symbol to nonsubscripted variable and go to identify terminator.

19. Must be symbol field or go to constant processor.

20. Put symbol in symbol table if not already there.

21. Check for "-" terminator.



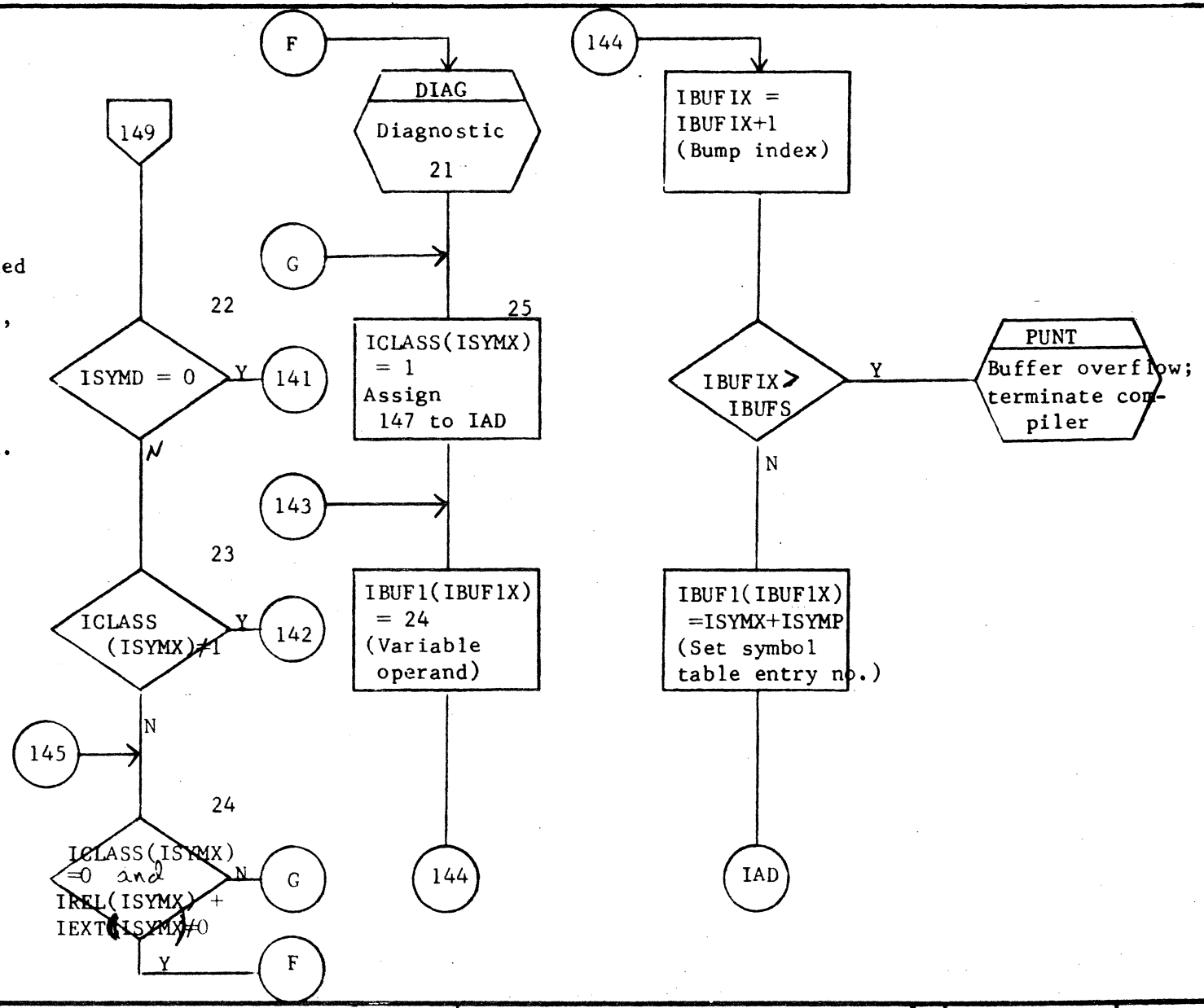
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ARITH		PROJECT MGR.			
FLOWCHART ☒		PAGE 4 OF 35	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

22. If the field is not in SYMTAB, check for "(".

23. If class is not a variable, must be "(", ")", or ",".

24. If class is unassigned and symbol is either relative or external, a variable has been used as a subprogram name.

25. Set class = variable.



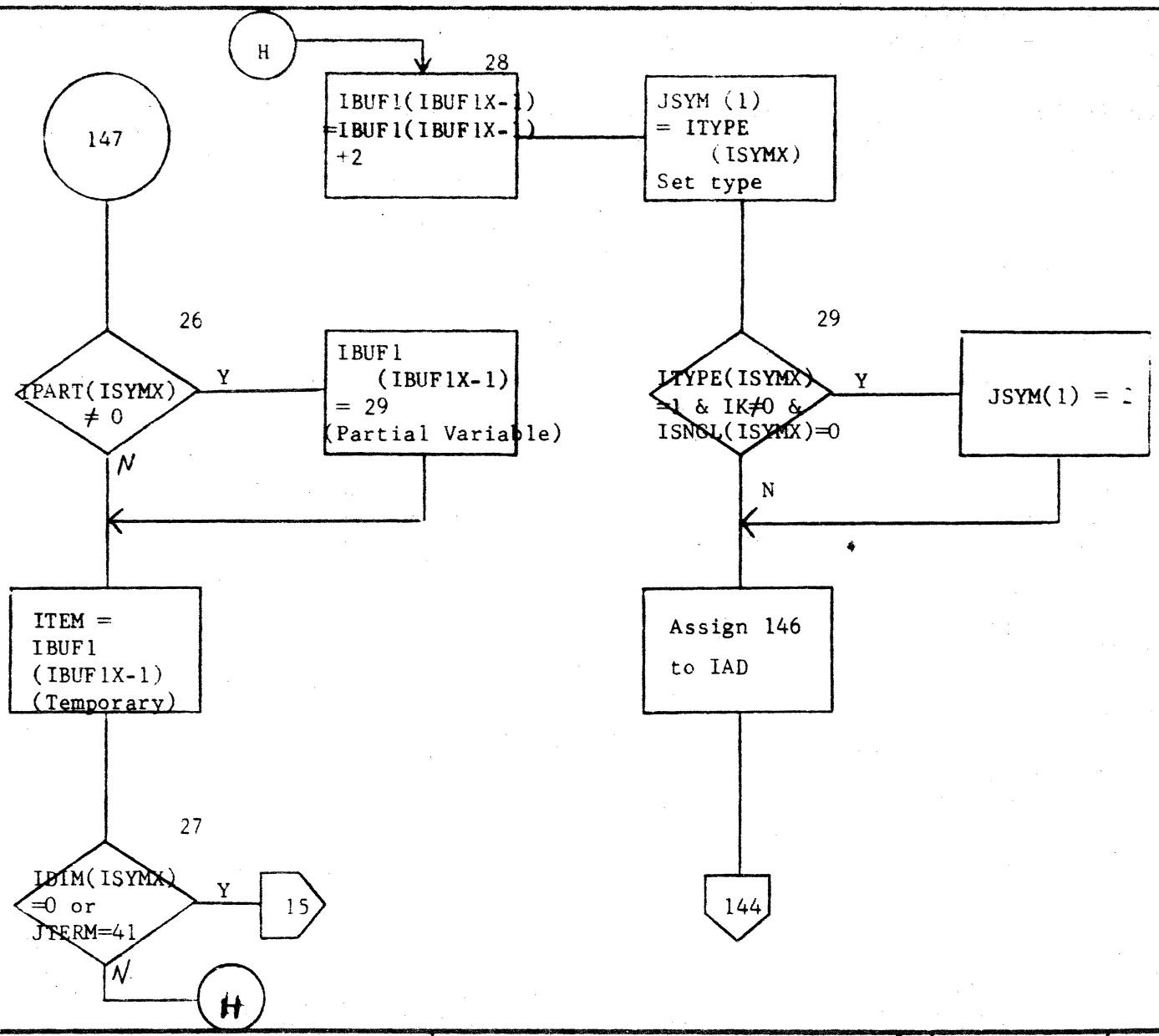
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ARITH	PROJECT MGR.			
	PAGE 5 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

26. Check partial variable.

27. If dimensional or "(" go check predecessor.

28. Dimensioned so set to increment only variable.

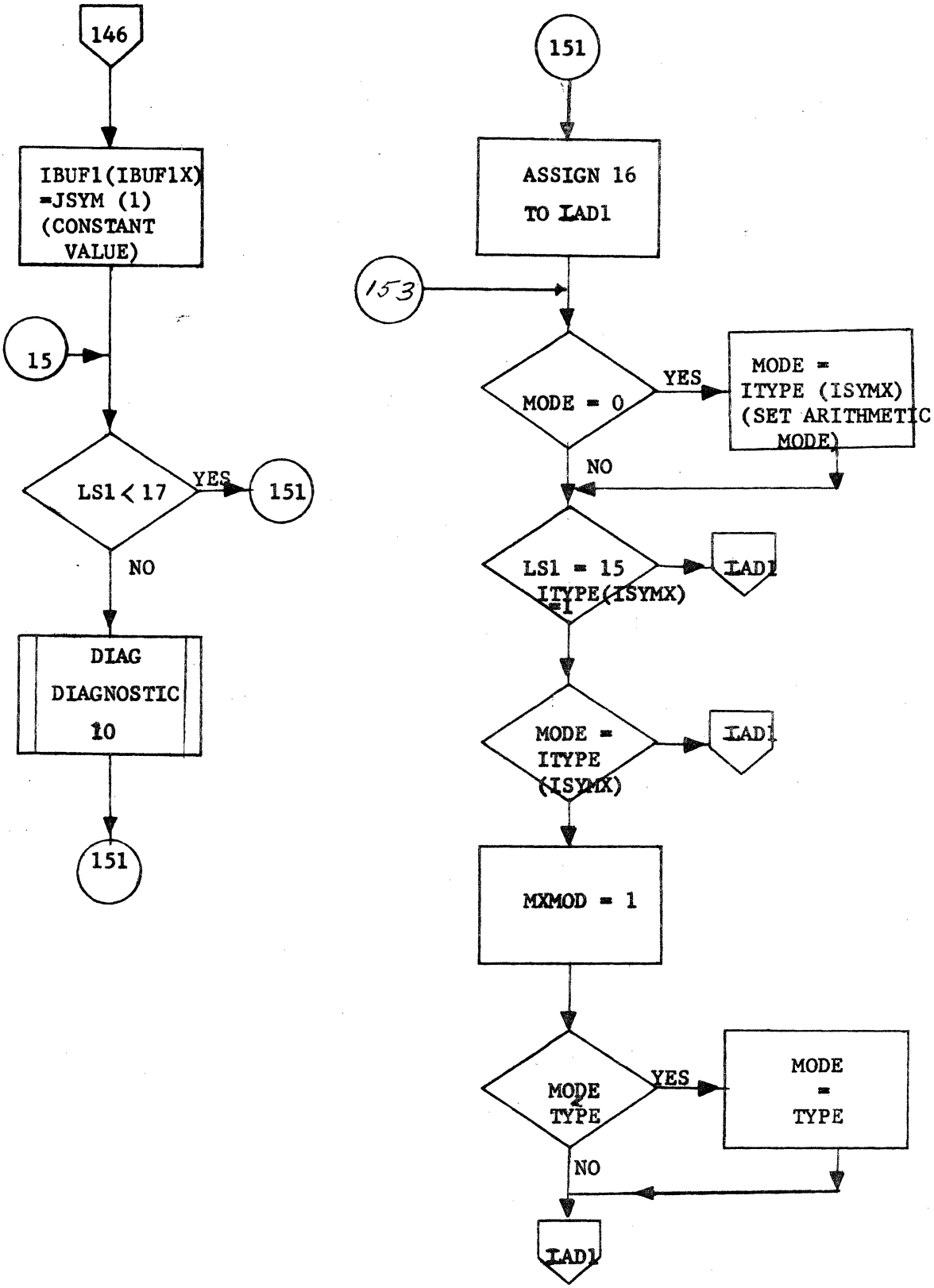
29. Two words/entry if non-integer or integer and ASA selected and SINGLE not selected.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.			
		PAGE 6 OF 35	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 3-1-1	TASK NAME			

CONTROL DATA

LA JOLLA FACILITY



TITLE	ARITH		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2 OF 3

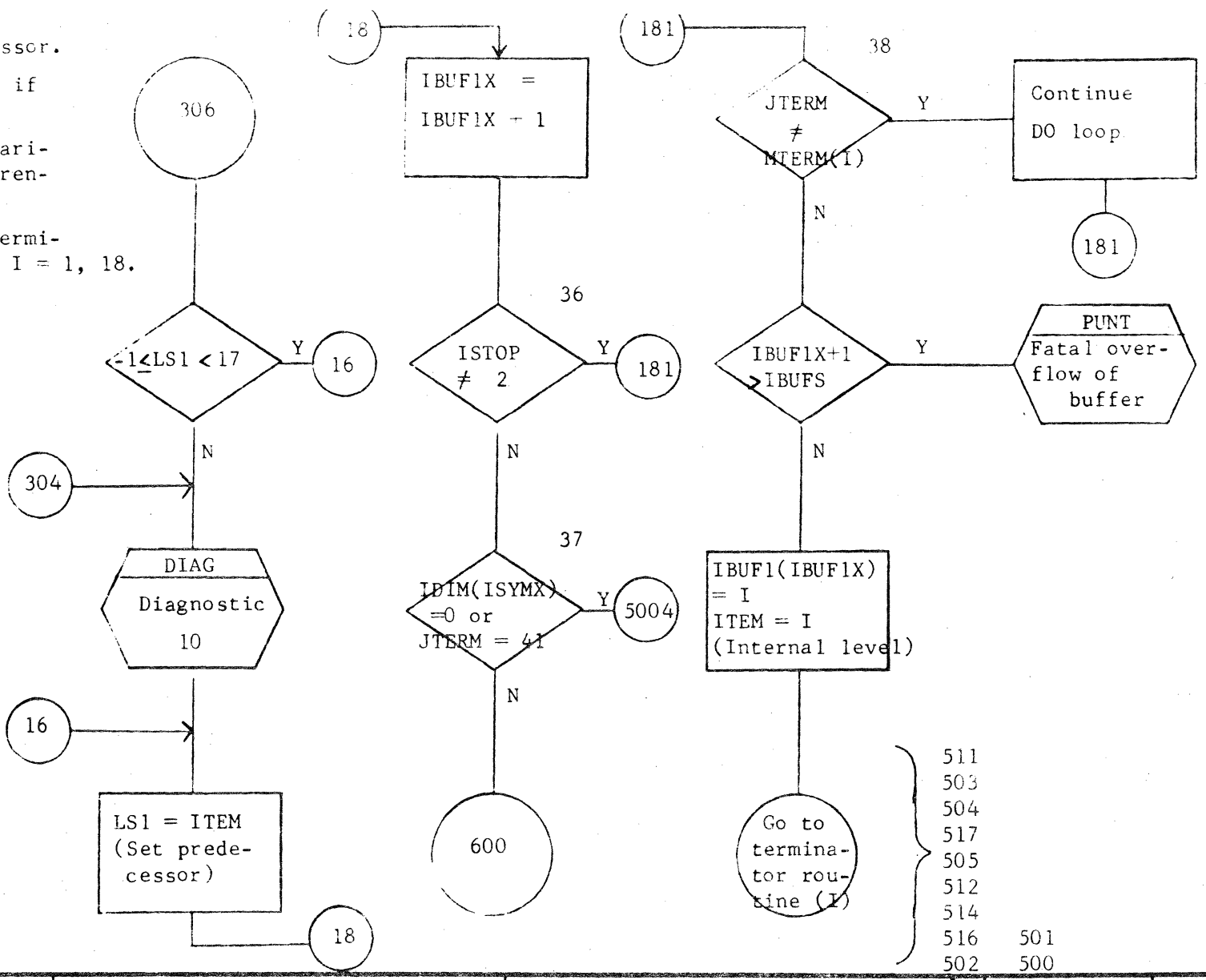
CSD 208

35. Illegal predecessor.

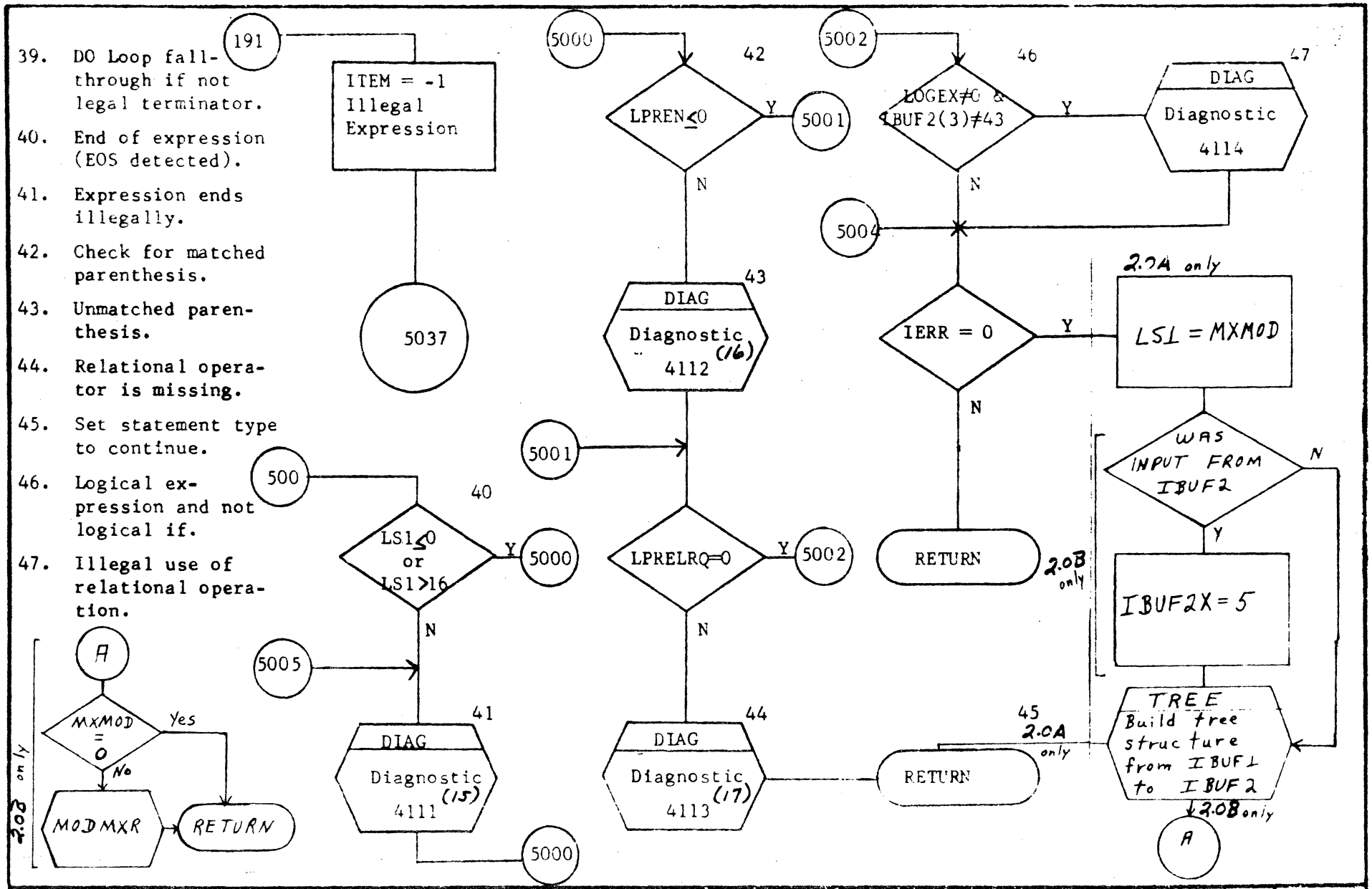
36. Single variable if ISTOP=2.

37. Undimensioned variable or left parenthesis.

38. Check against terminator table for I = 1, 18.



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ARITH		PROJECT MGR.		
FLOWCHART ☒		PAGE 8 OF 35	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR		
		PAGE 9 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE			

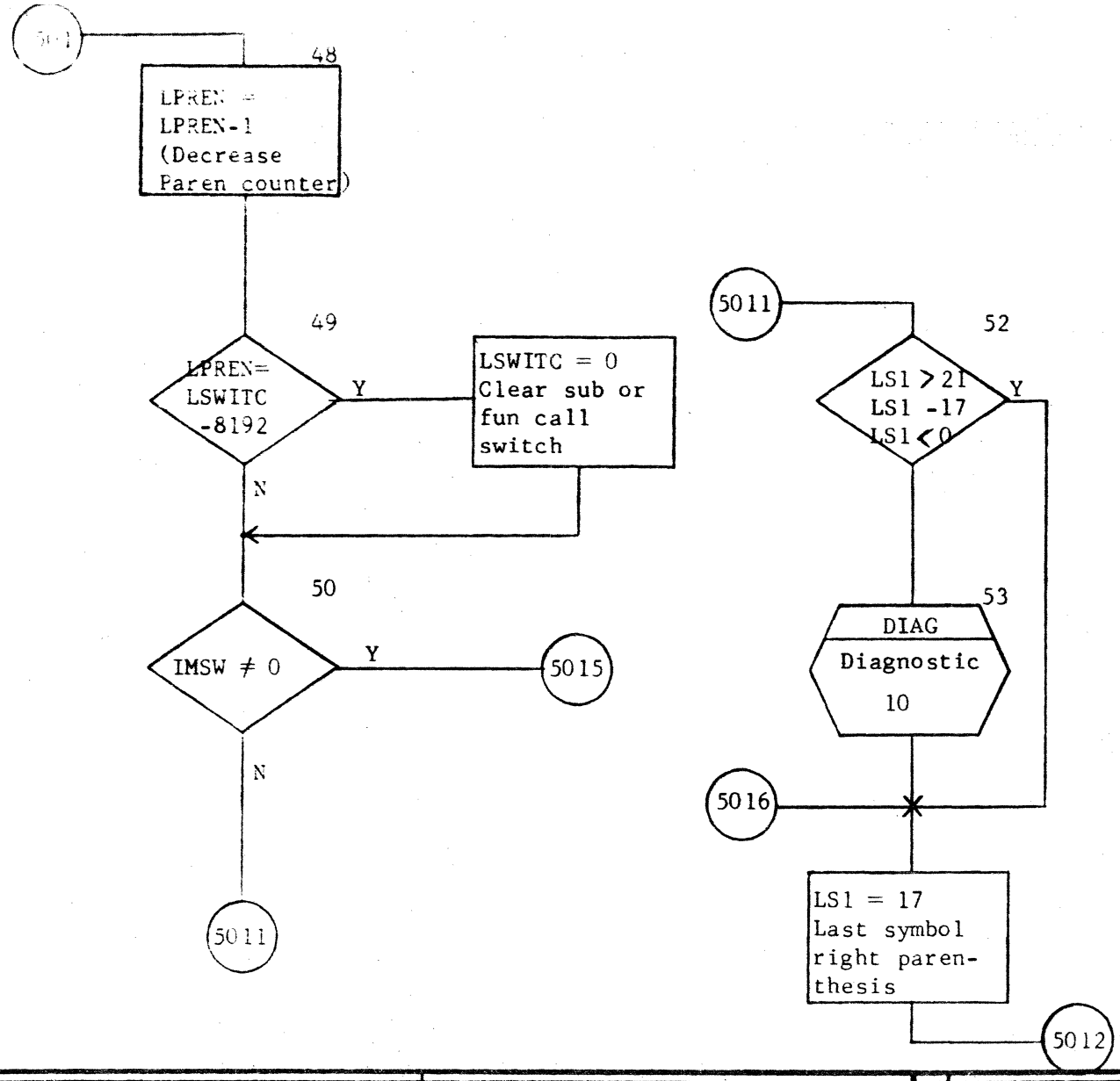
48. ")" detected.

49. Parenthesis level the same?

50. No entries in mode control table?

52. Check predecessor, variable, constant, no parameter function or subroutine legal.

53. Illegal operator or operand.



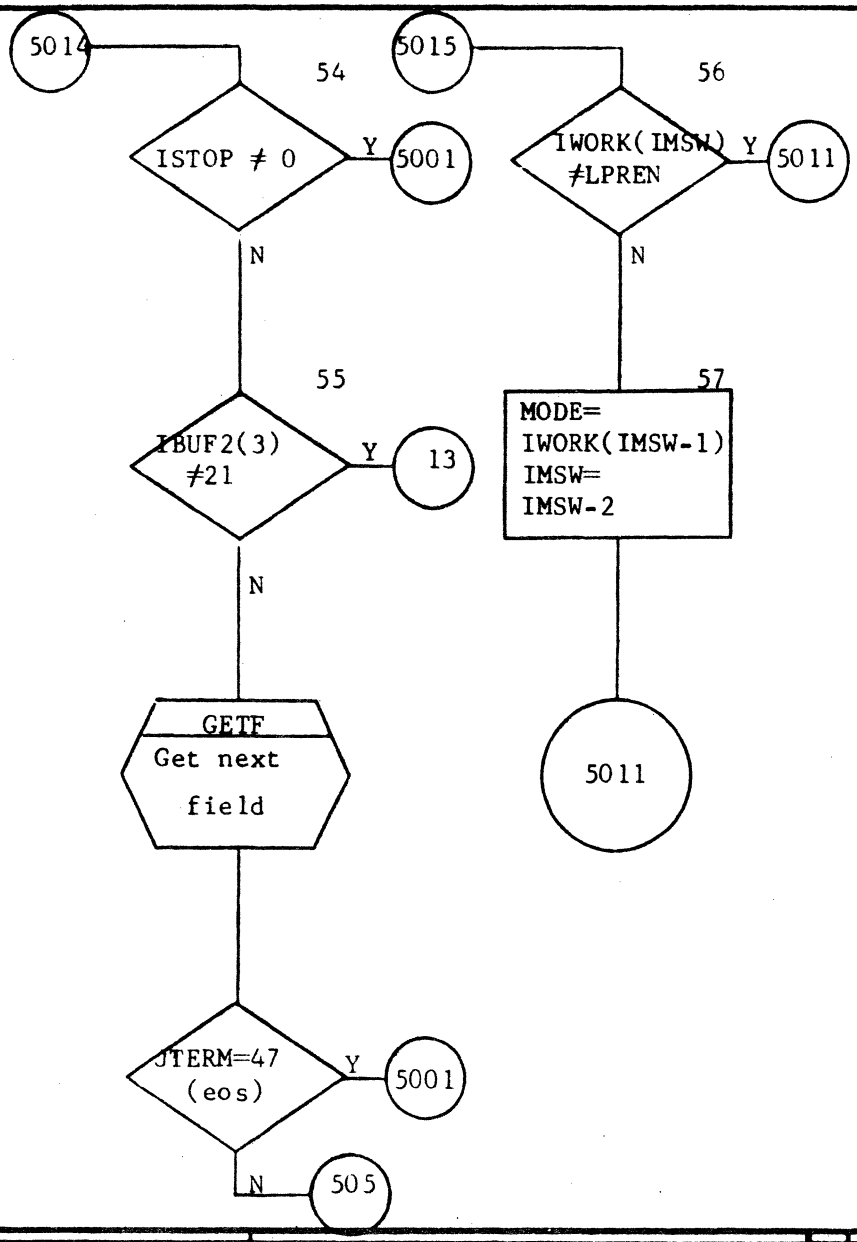
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ARITH	PROJECT MGR.			
	PAGE 10 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 4-7	TASK NAME			

54. Test for end of statement.

55. IBUF(3)=21 means CALL statement.

56. Parenthesis level the same?

57. Yes. Set mode and decrease IWORK index.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ARITH	PROJECT MGR.			
	PAGE 11 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

58. Left parenthesis processor.

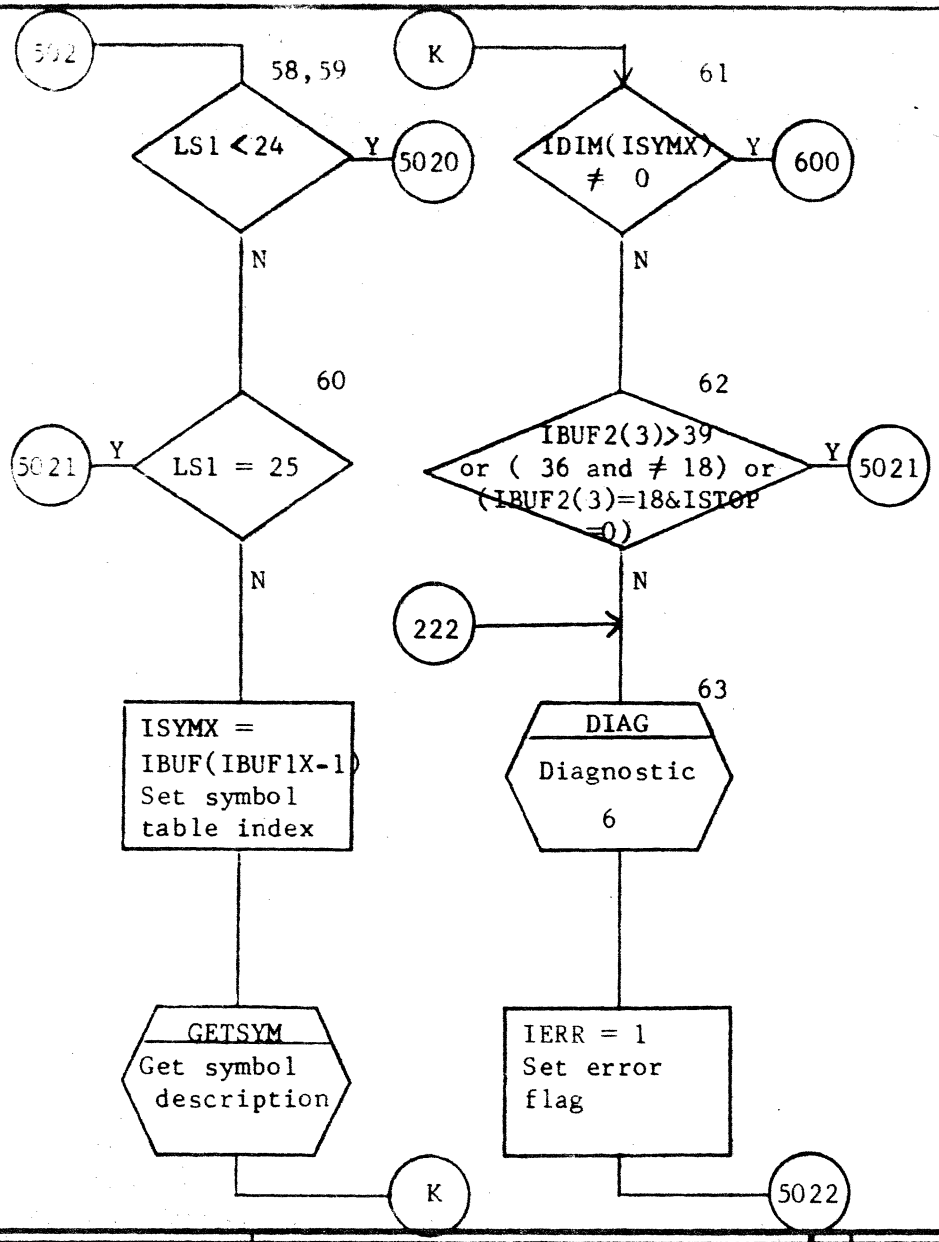
59. Check predecessor a variable.

60. Predecessor is a variable. Is it a numeric constant?

61. Dimensioned variable?

62. No. Check if not read, write, or replacement, or replacement and processing to end of statement.

63. Undimensioned subscripted variable.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED	DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.			
		PAGE 12 OF 35	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-11)

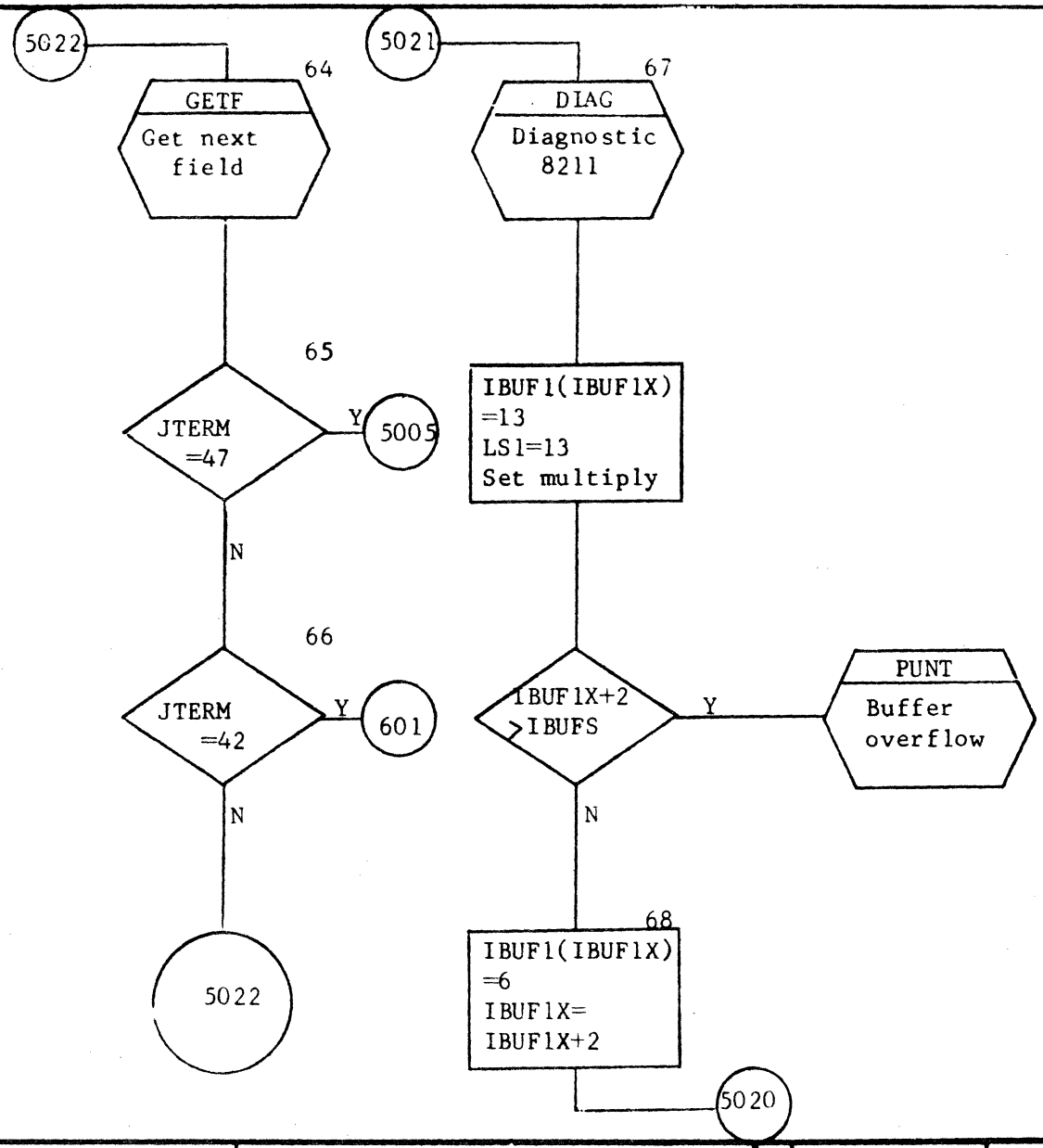
64. Slew to right parenthesis.

65. End of statement?

66. Right parenthesis.

67. * assumed.

68. Set level 6 and bump index.



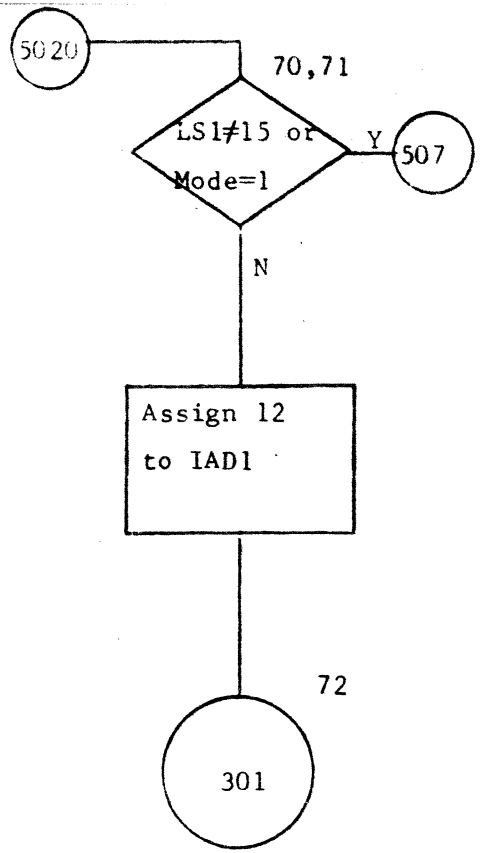
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.		
		PAGE 13 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

69.

70. Entry if nonvariable prececessor.

71. Check predecessor is ** or mode is integer.

72. Go make IWORK entry.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.		
		PAGE 14 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 7-1-7	TASK NAME		

AA1385 (FORMERLY CA127-1)

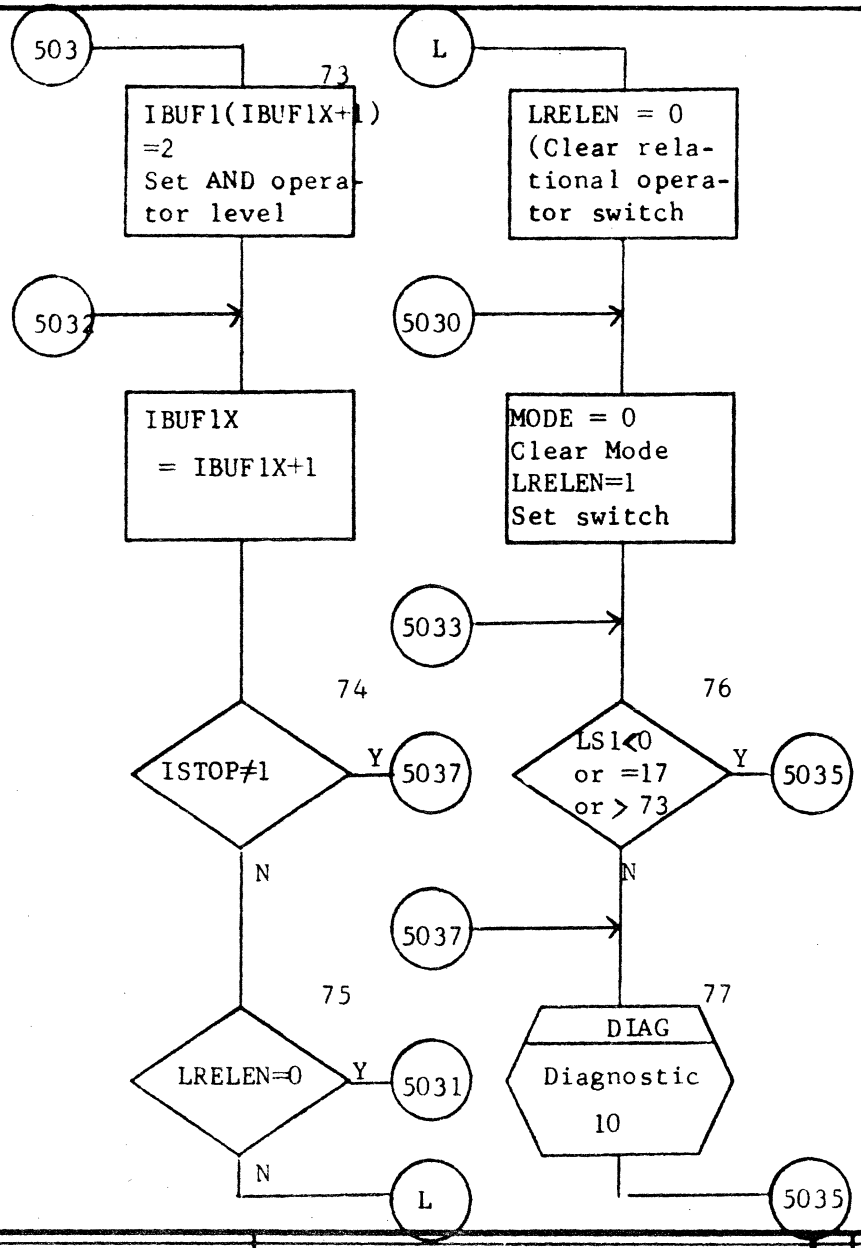
73. AND processor.

74. Only OK in logical if.

75. Relational operator encountered switch on?

76. Check if last symbol ")" or constant or variable.

77. Illegal operator or operand.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ARITH	PROJECT MGR.			
	PAGE 15 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

78. Relational operator missing.

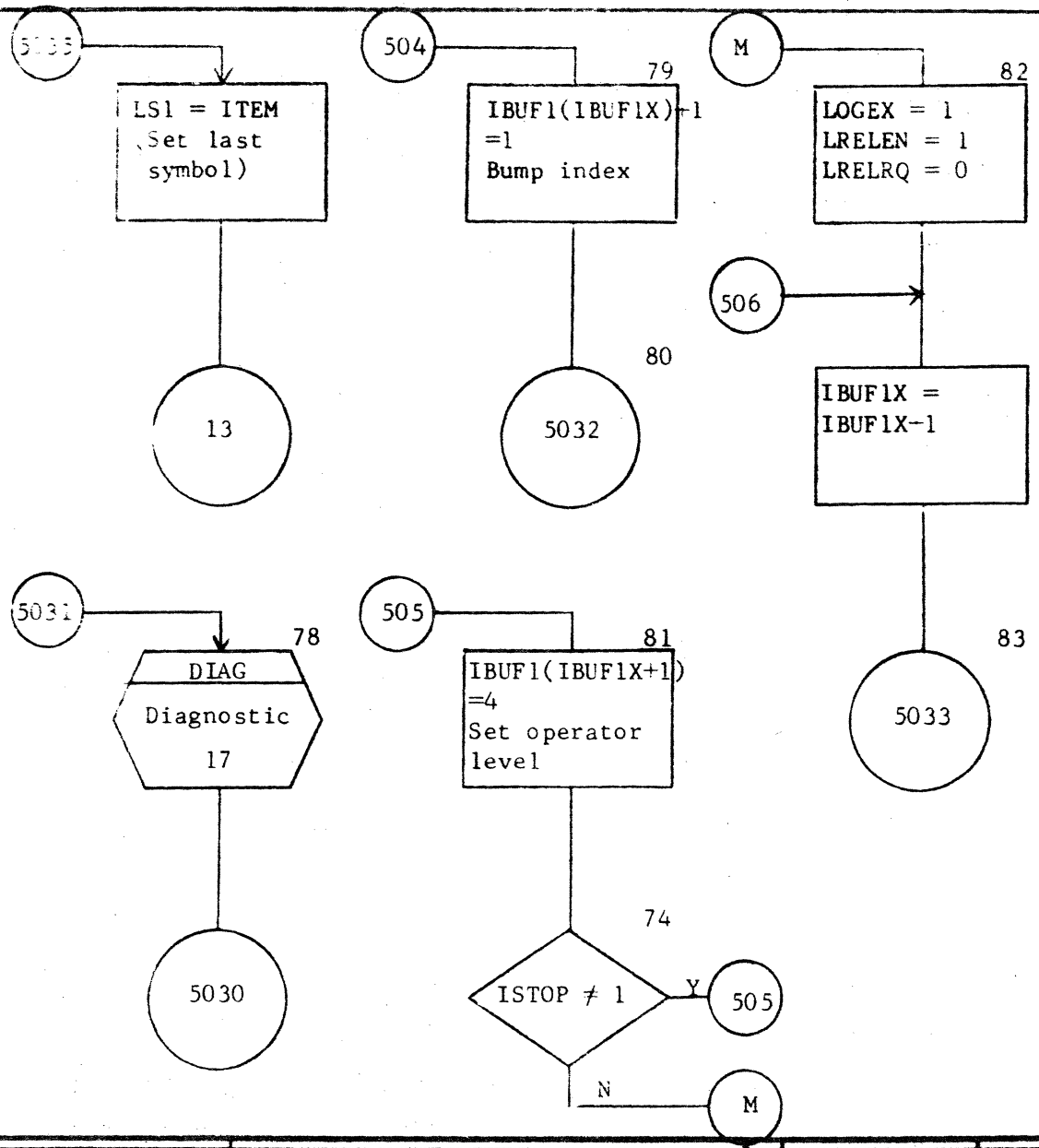
79. OR processor.

80. Same as AND.

81. >, <, ≤, ≥, =, ≠ Processor.

82. Set logical switch, Relational operator encountered switch. Clear relational operator required switch.

83. Same as AND and OR at this point.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ARITH	PROJECT MGR			
	PAGE 16 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

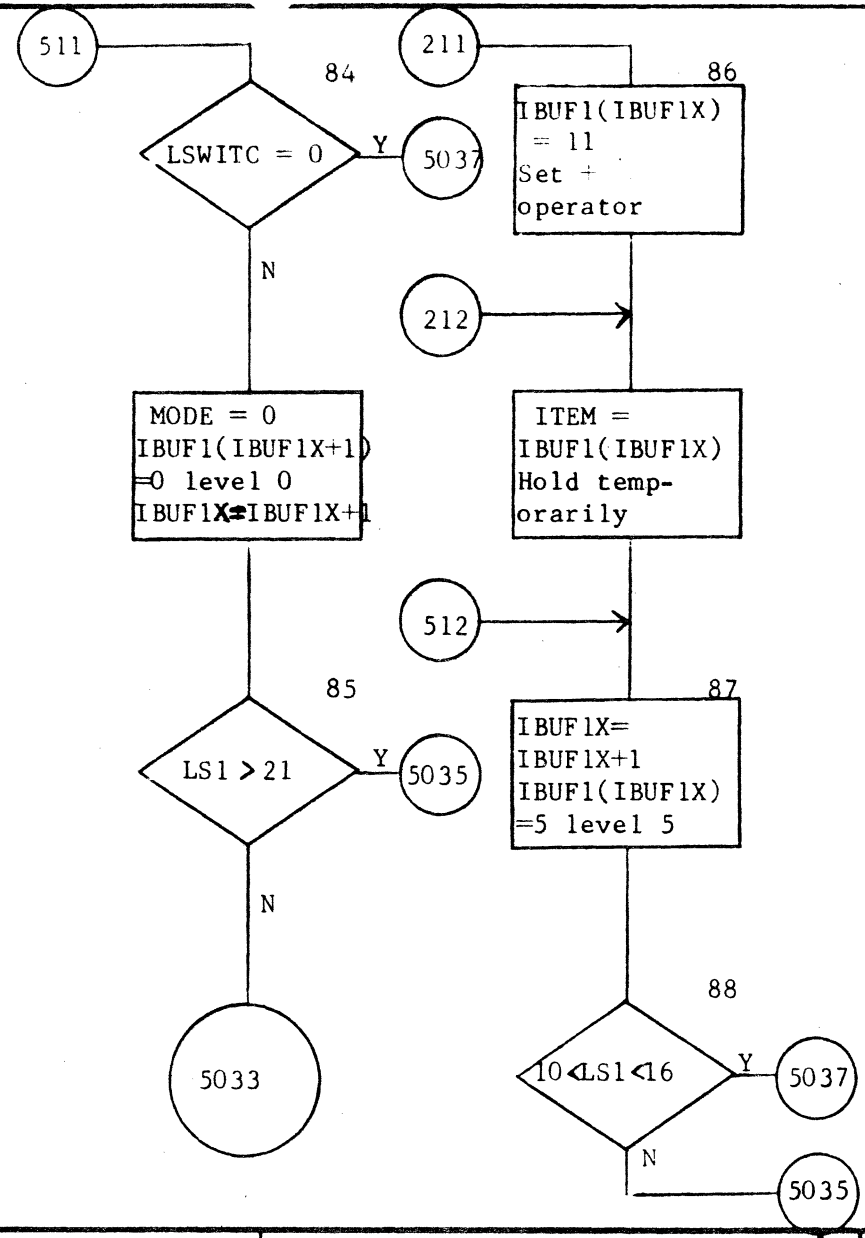
84. Comma Processor - Illegal if not function or subroutine.

85. Check for function or subroutine without argument.

86. Plus processor - first field.

87. "+, -" processor.

88. Check illegal predecessors - +, -, **, *, /.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR		
		PAGE 17 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

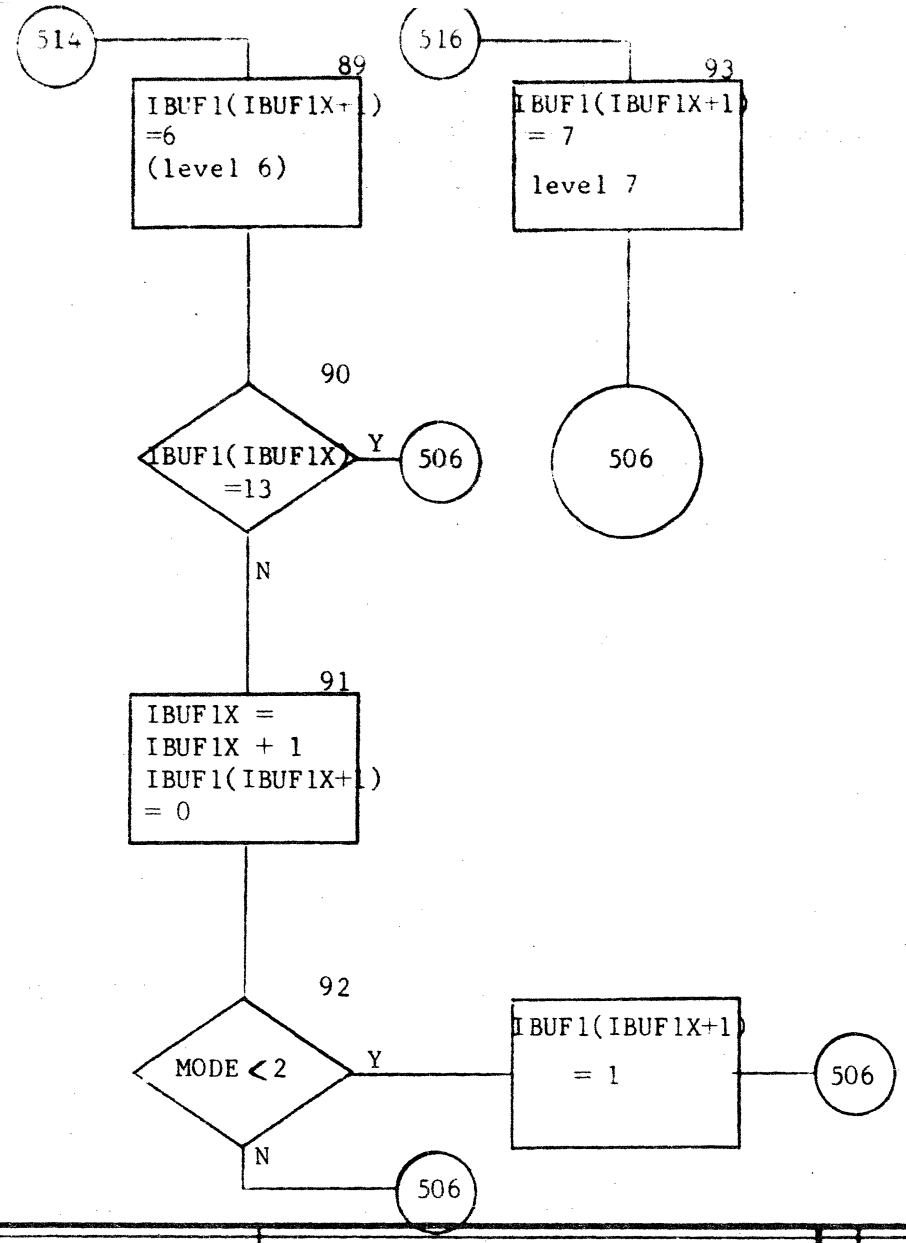
89. "*,/" processor.

90. Check for **.

91. Clear symbol table pointer.

92. Check mode unassigned or integer.

93. "**" processor.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ARITH		PROJECT MGR.			
FLOWCHART ☒		PAGE 18 OF 35	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

94. "NOT" processor.

95. Must be if statement.

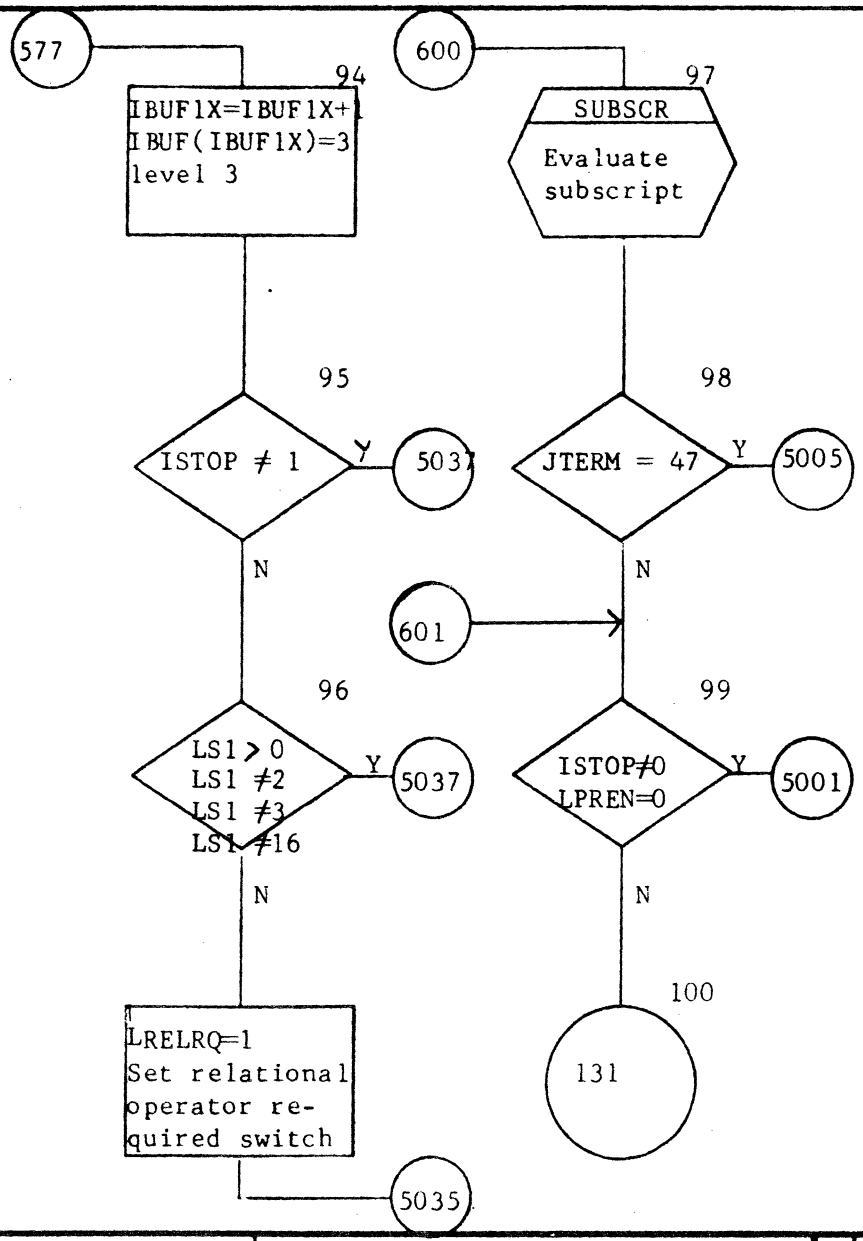
96. Legal predecessors are AND, OR, left parenthesis.

97. Dimensioned variable.

98. Check for end of statement.

99. End of parenthesis means put statement in TREE.

100. Go process next field.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ARITH		PROJECT MGR.			
FLOWCHART ☒		PAGE A OF 35	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

101. CALL statement processor.

102. Check for symbol.

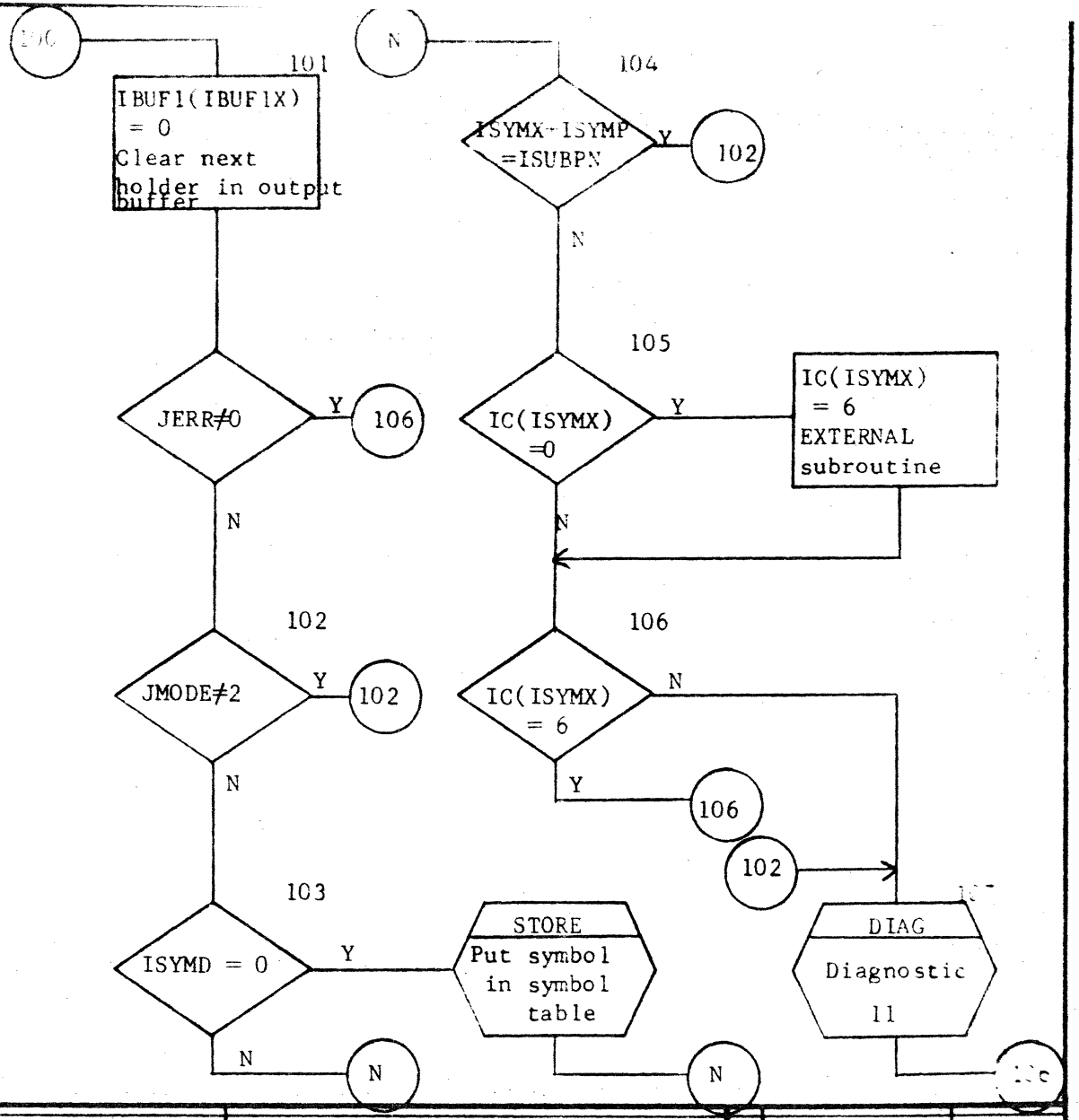
103. If SYMBOL not in symbol table, put it in.

104. Is CALL to same name as SUBROUTINE being compiled?

105. Check class unassigned.

106. Check class external.

107. Illegal subprogram reference.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ARITH		PROJECT MGR.			
FLOWCHART ☒		PAGE 30 OF 35	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

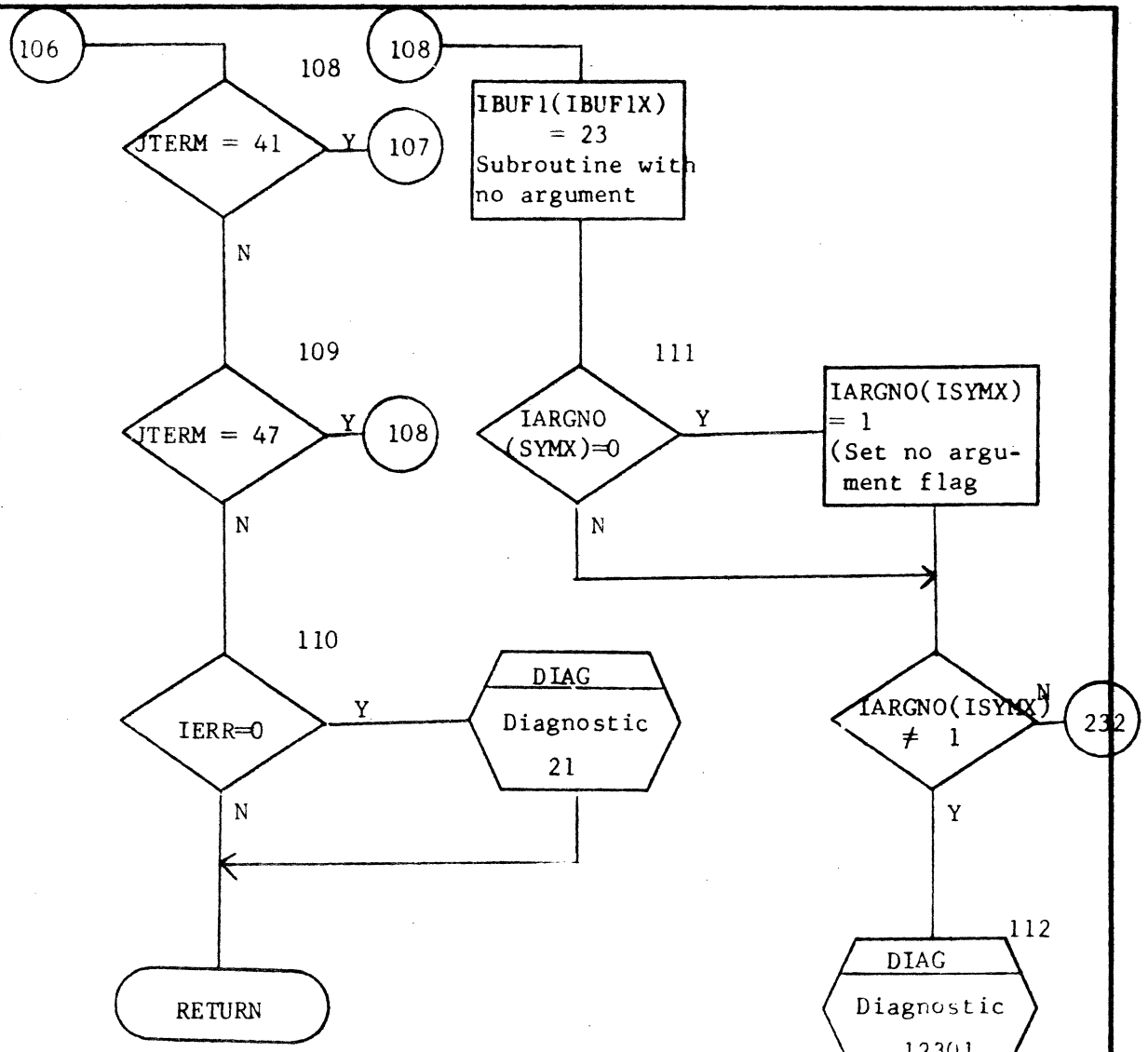
108. Check for next field starting with left parenthesis.

109. End of statement?

110. No previous error detected means a variable has been used as a subprogram name.

111. Check for argument flag set for no arguments.

112. Number of arguments differ in references to same subprogram.



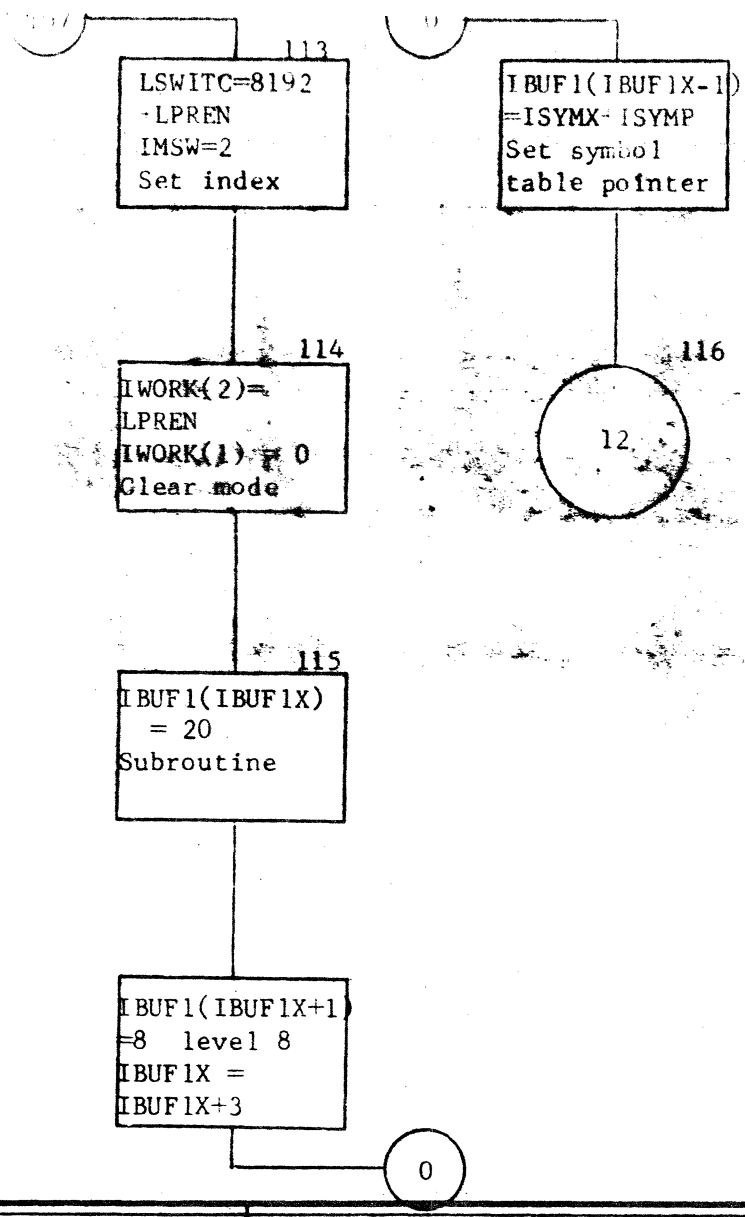
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BRITH		PROJECT MGR.		
		PAGE 21 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	PROGRAMMER	DATE	TASK NAME		

113. Make IWORK entry for this calling sequence parameter list. Set switch on and equal to 8192 + no. of left parentheses.

114. Set parenthesis level.

115. Subroutine opened

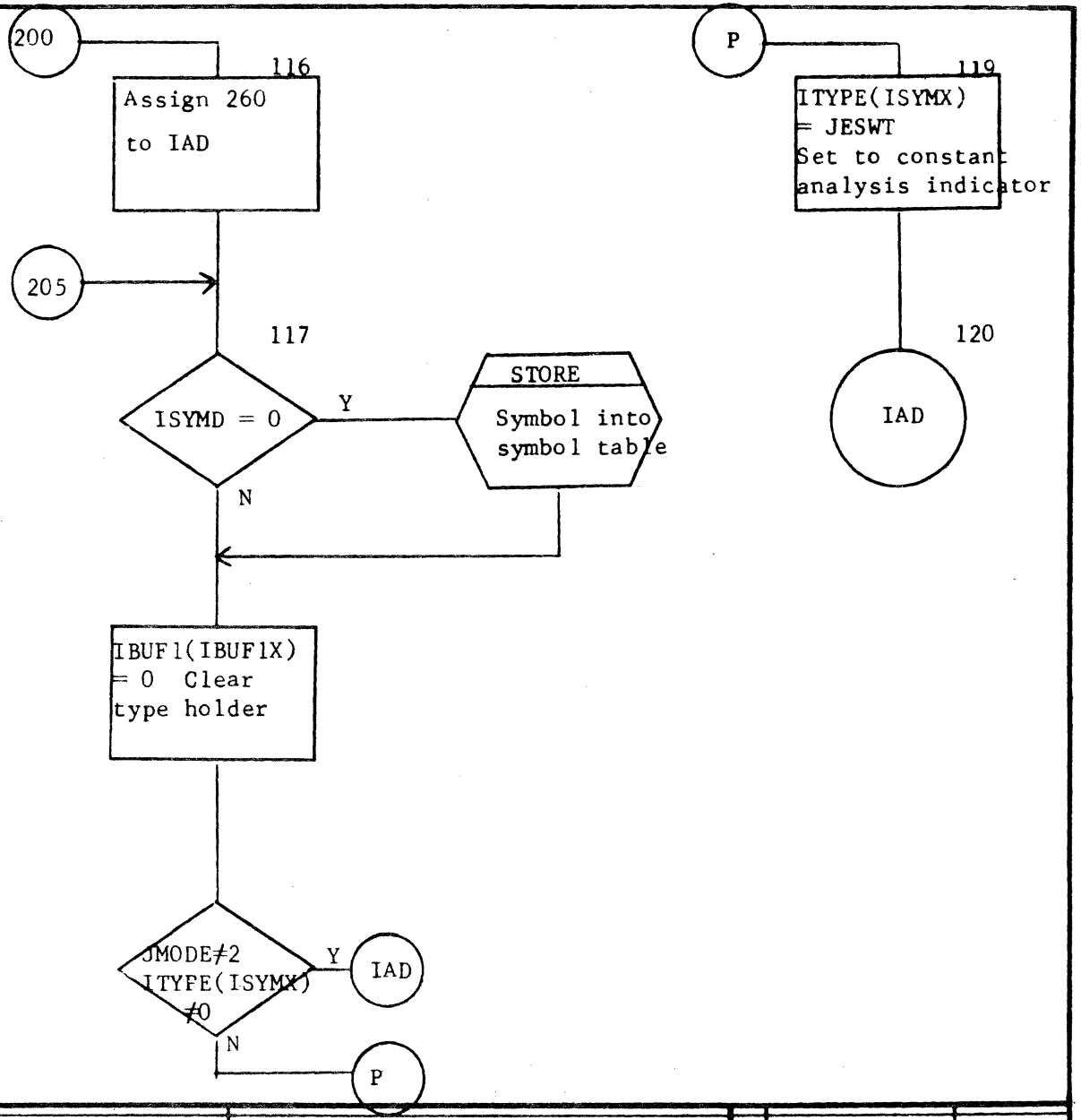
116. Go process "(".



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.		
		PAGE 22 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-6-7	TASK NAME		

AA1385 (FORMERLY CA127-1)

16. Process alphanumeric or numeric portion of first field.
17. Put symbol in symbol table if not already there.
18. Check not symbol or if unassigned.
19. Must be constant so set to constant analysis indicator (JESWT).
20. Set to 260 if 1st field, otherwise 149 is set in normal loop.

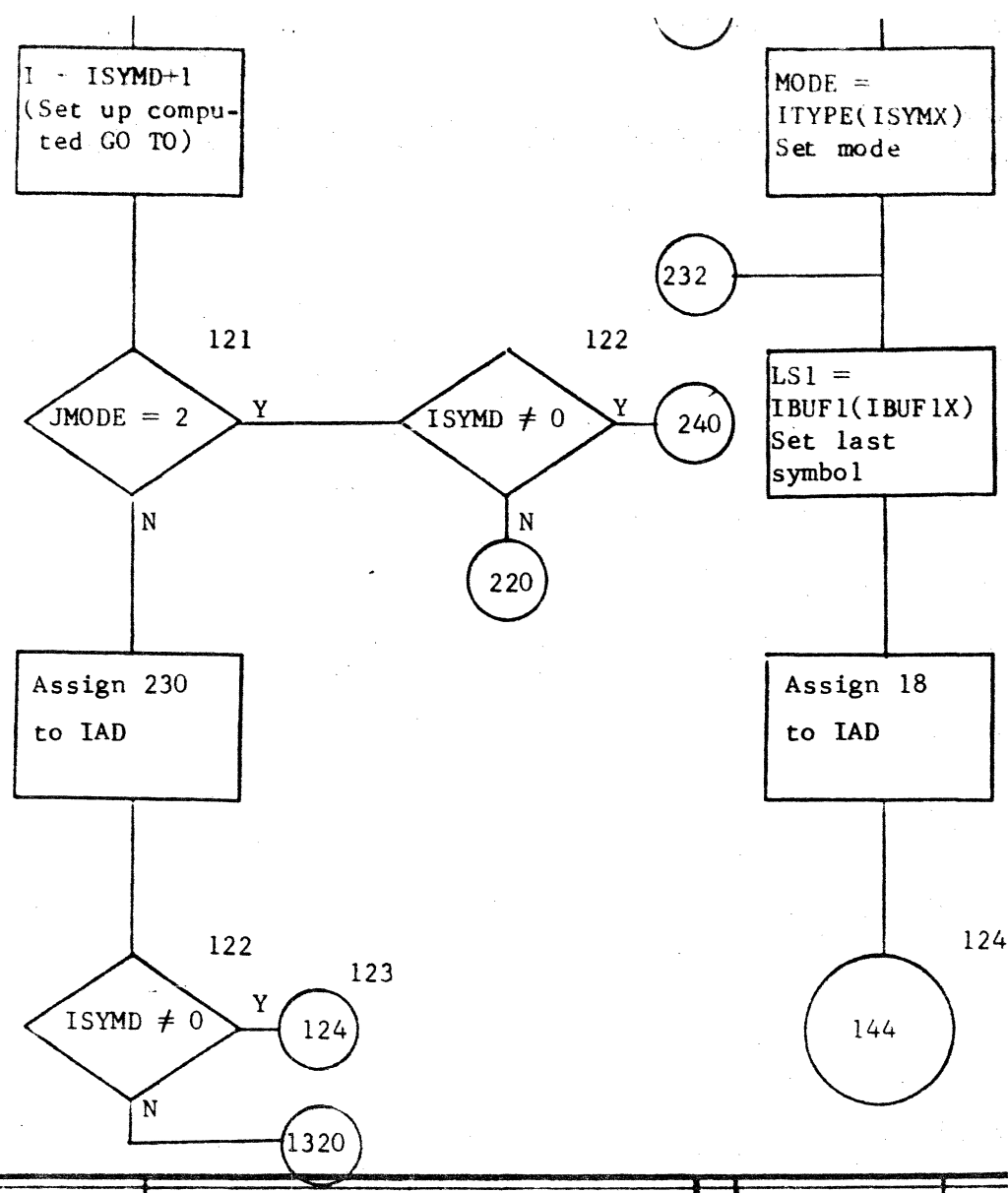


121. Check for symbol.

122. Computed go to based on symbol in symbol table.

123. Numeric constant.

124. Check for buffer overflow, then recycle.

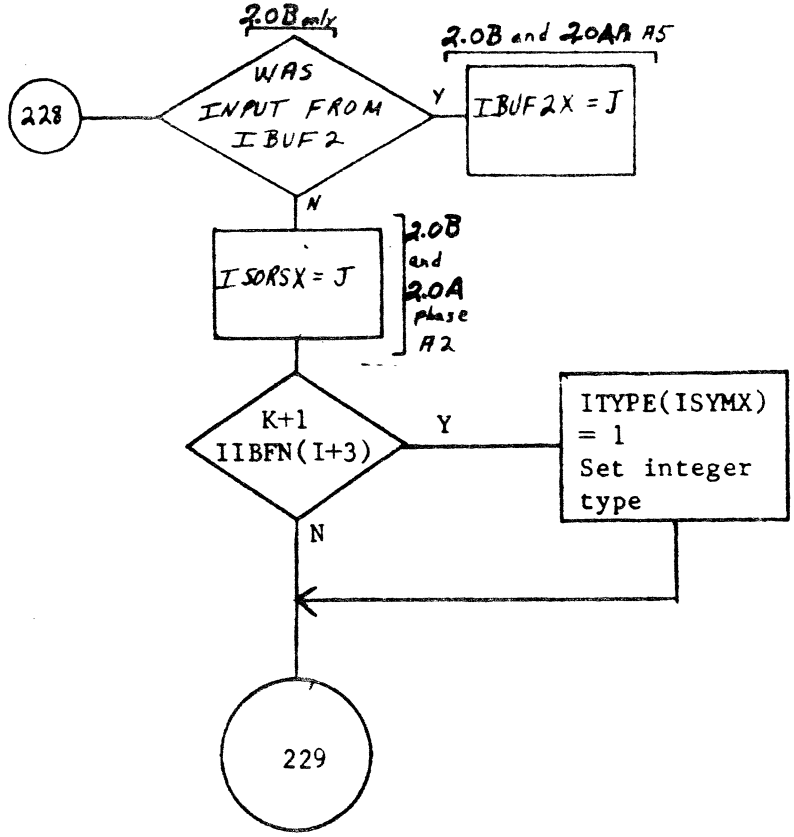


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS TMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ARITH		PROJECT MGR.		
FLOWCHART ☒		PAGE 34 OF 35	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE 3-69	TASK NAME		

AA1385 (FORMERLY CA127-1)

132. Name is in table of intrinsic functions.

133. Check if number of parameters the same.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE —	DOCUMENT TITLE ARITH		PROJECT MGR			
FLOWCHART X		PAGE 26 OF 35	PROJECT NAME			
DECISION TABLE —	NUMBER	ISSUE DATE	TASK NO			
OTHER —	DRAWN BY	DATE	TASK NAME			

125. Check for left parenthesis.

126. Set symbol table entry to function.

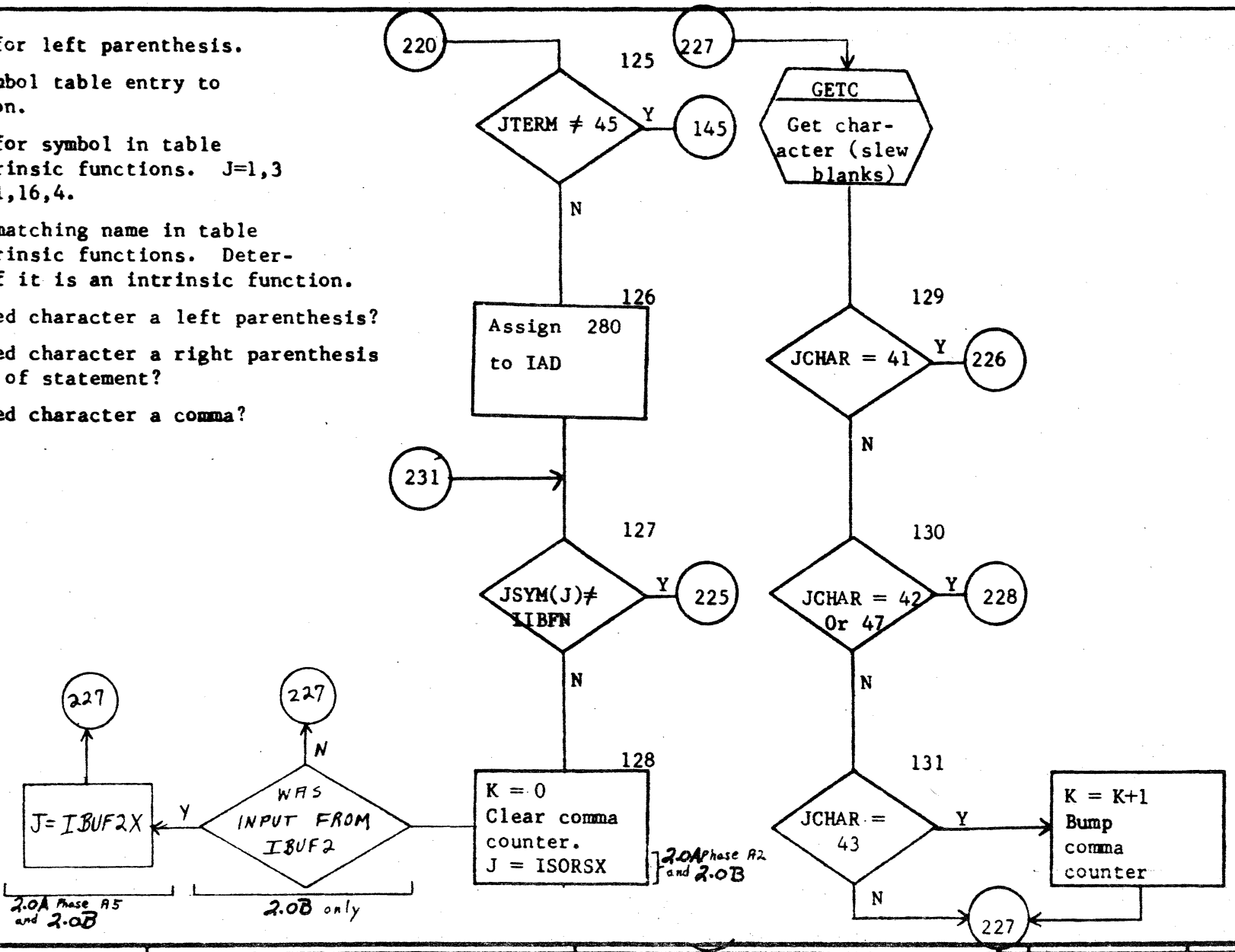
127. Check for symbol in table of intrinsic functions. J=1,3 and I=1,16,4.

128. Found matching name in table of intrinsic functions. Determine if it is an intrinsic function.

129. Imbedded character a left parenthesis?

130. Imbedded character a right parenthesis on end of statement?

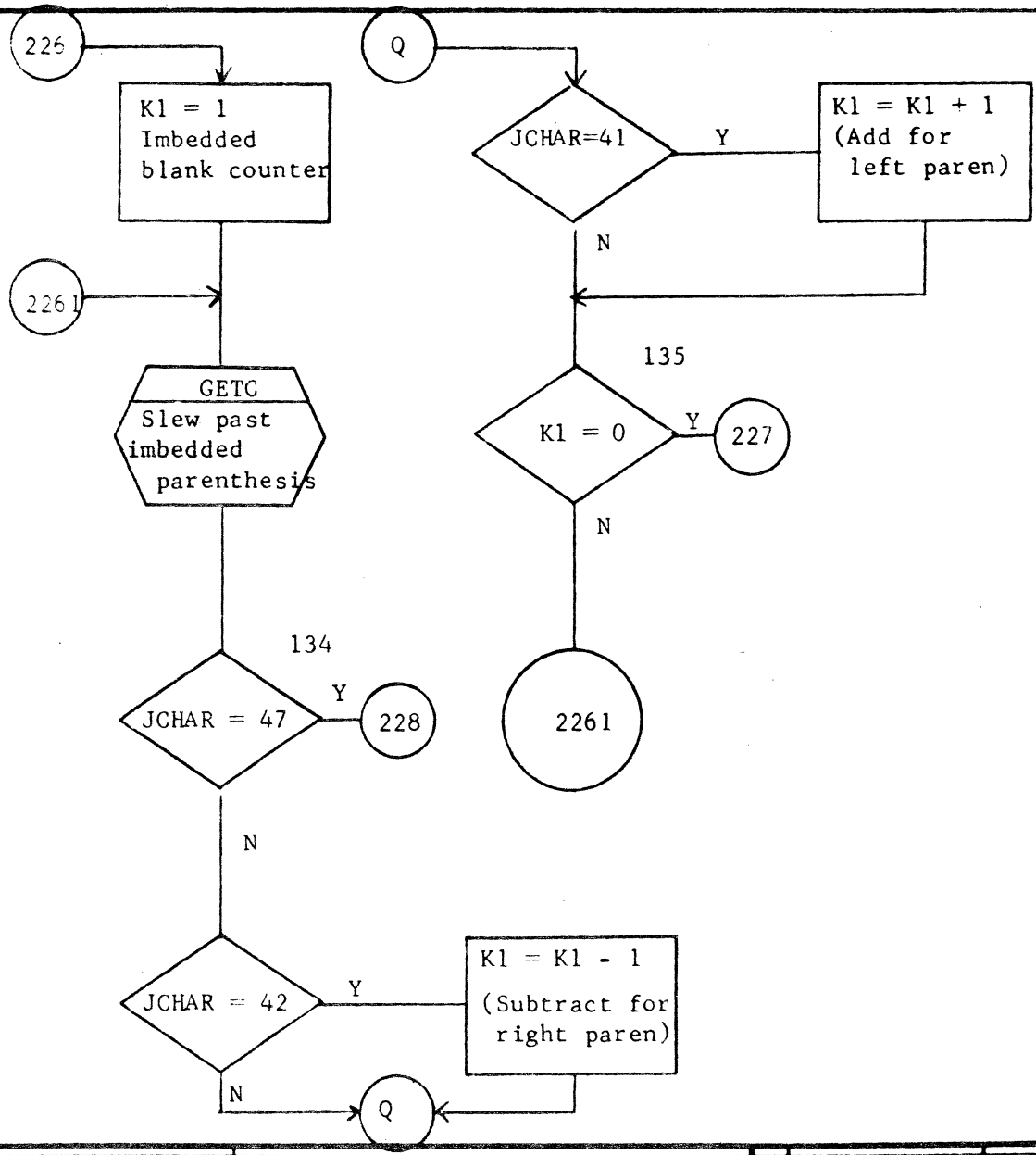
131. Imbedded character a comma?



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.		
		PAGE 35 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-4-7	TASK NAME		

134. Check for end of statement.

135. Check if past imbedded parenthesis.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER

DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE ARITH		PROJECT MGR.			
	PAGE 27 OF 35	PROJECT NAME			
NUMBER	SSLE DATE	TASK NO.			
DESIGN BY	DATE	TASK NAME			

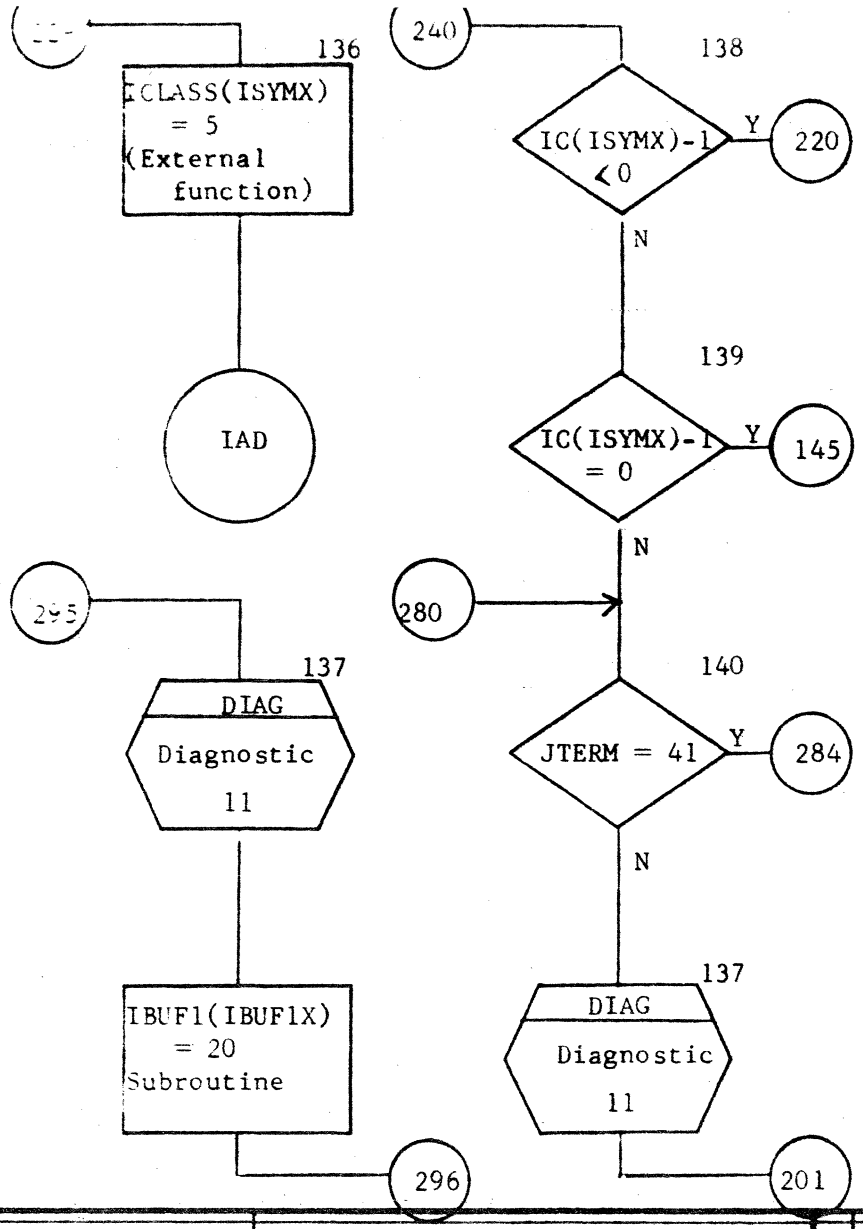
136. Set class.

137. Illegal subprogram reference.

138. Check if class unassigned.

139. Check if class variable.

140. Other check for left parenthesis.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.		
		PAGE 28 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

141. Check if terminator of first field +.

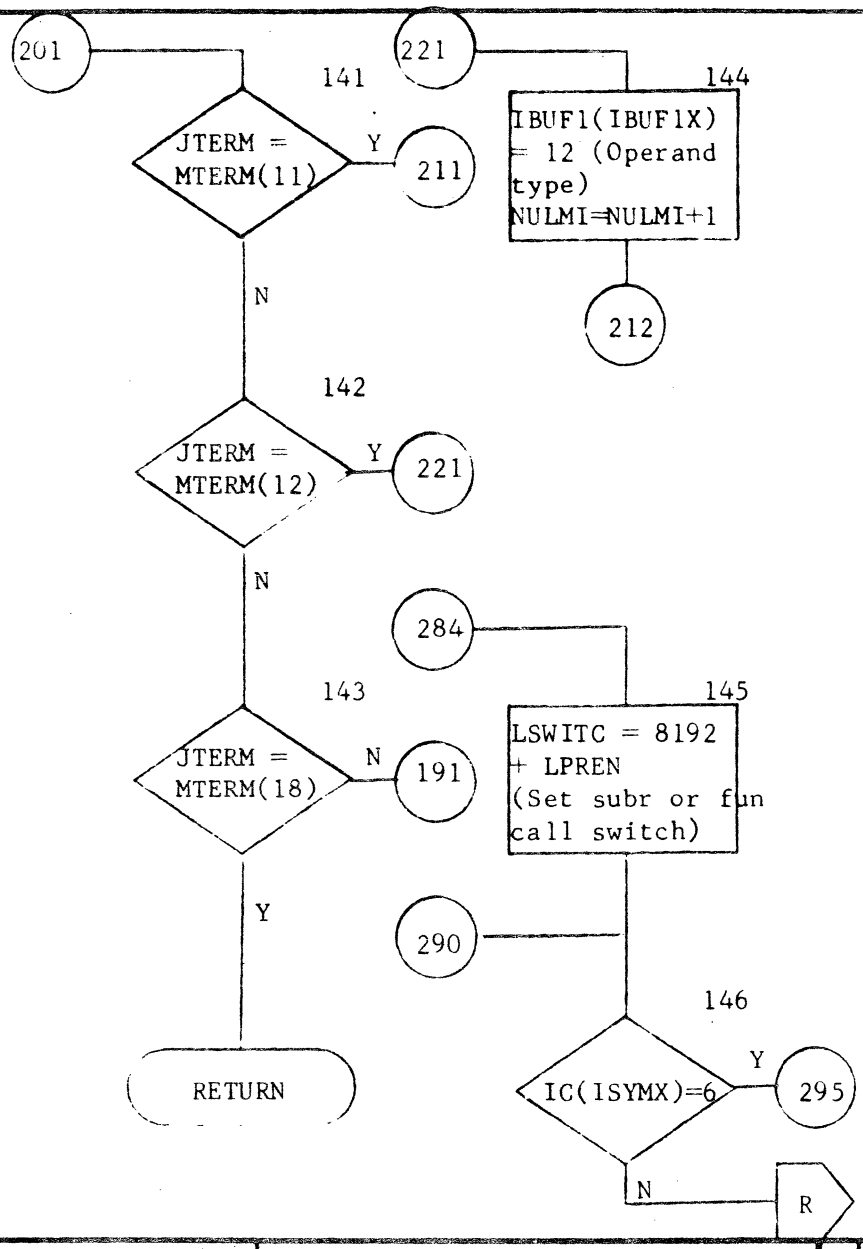
142. Check if terminator of first field -.

143. Check if terminator of first field end of statement.

144. Terminator a -.

145. Left parenthesis encountered.

146. Check for external.



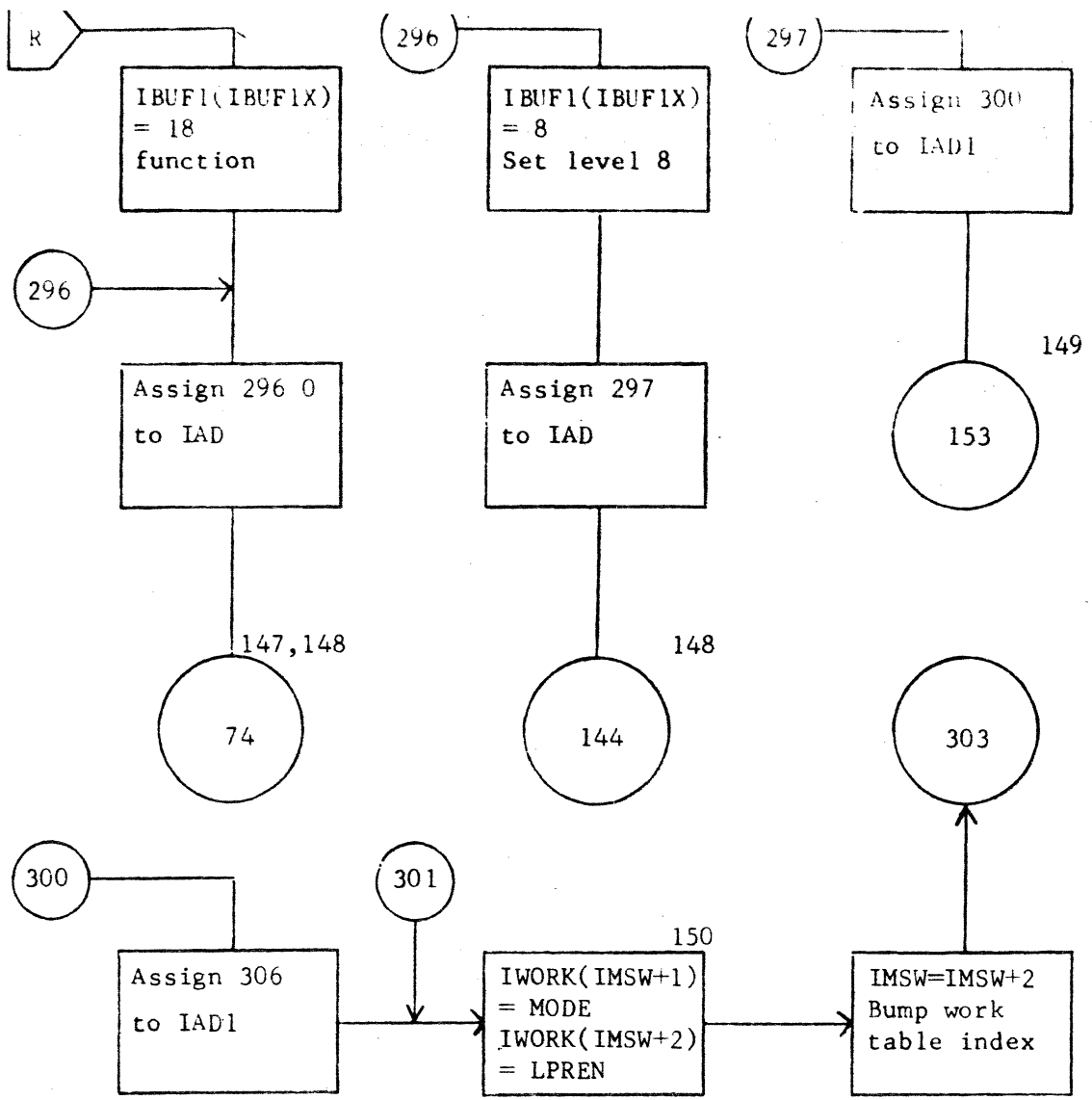
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE / FLOWCHART (X) / DECISION TABLE / OTHER	DOCUMENT CLASS: IMS — MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: ARITH	PROJECT MGR.			
	PAGE 29 OF 35	PROJECT NAME			
	NUMBER — ISSUE DATE	TASK NO.			
	DATE	TASK NAME			

147. Save type in ITEM.

148. Bump index, check for overflow and set SYMTAB entry number.

149. Check mode.

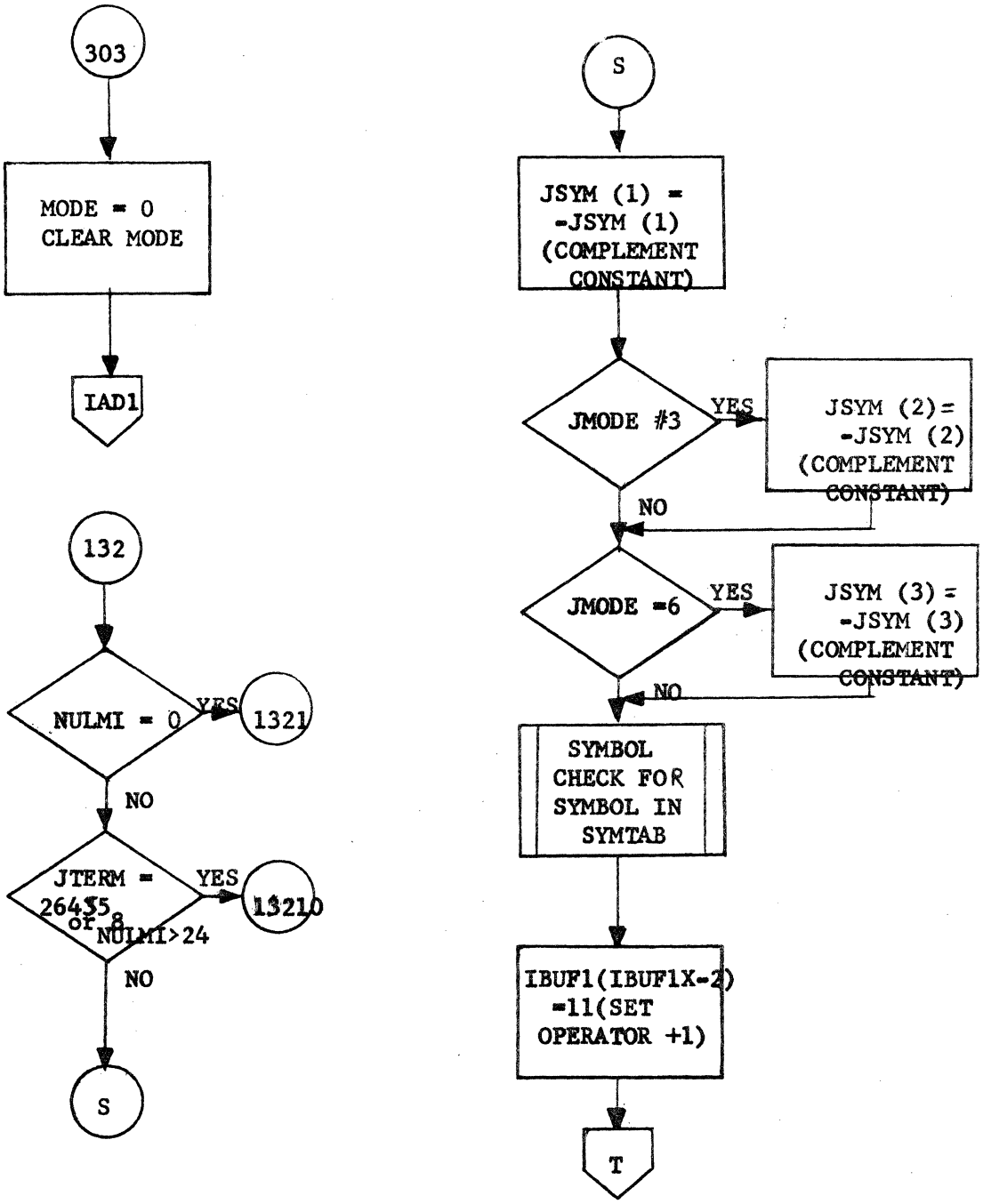
150. Set mode and level of parentheses in intrinsic function work table.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE / FLOWCHART (X) / DECISION TABLE / OTHER	DOCUMENT CLASS: IMS — MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: ARITH	PROJECT MGR.			
	PAGE 30 OF 35	PROJECT NAME			
	NUMBER — ISSUE DATE	TASK NO.			
	DRAWN BY — DATE 5-67	TASK NAME			

CONTROL DATA CORPORATION

LA JOLLA FACILITY



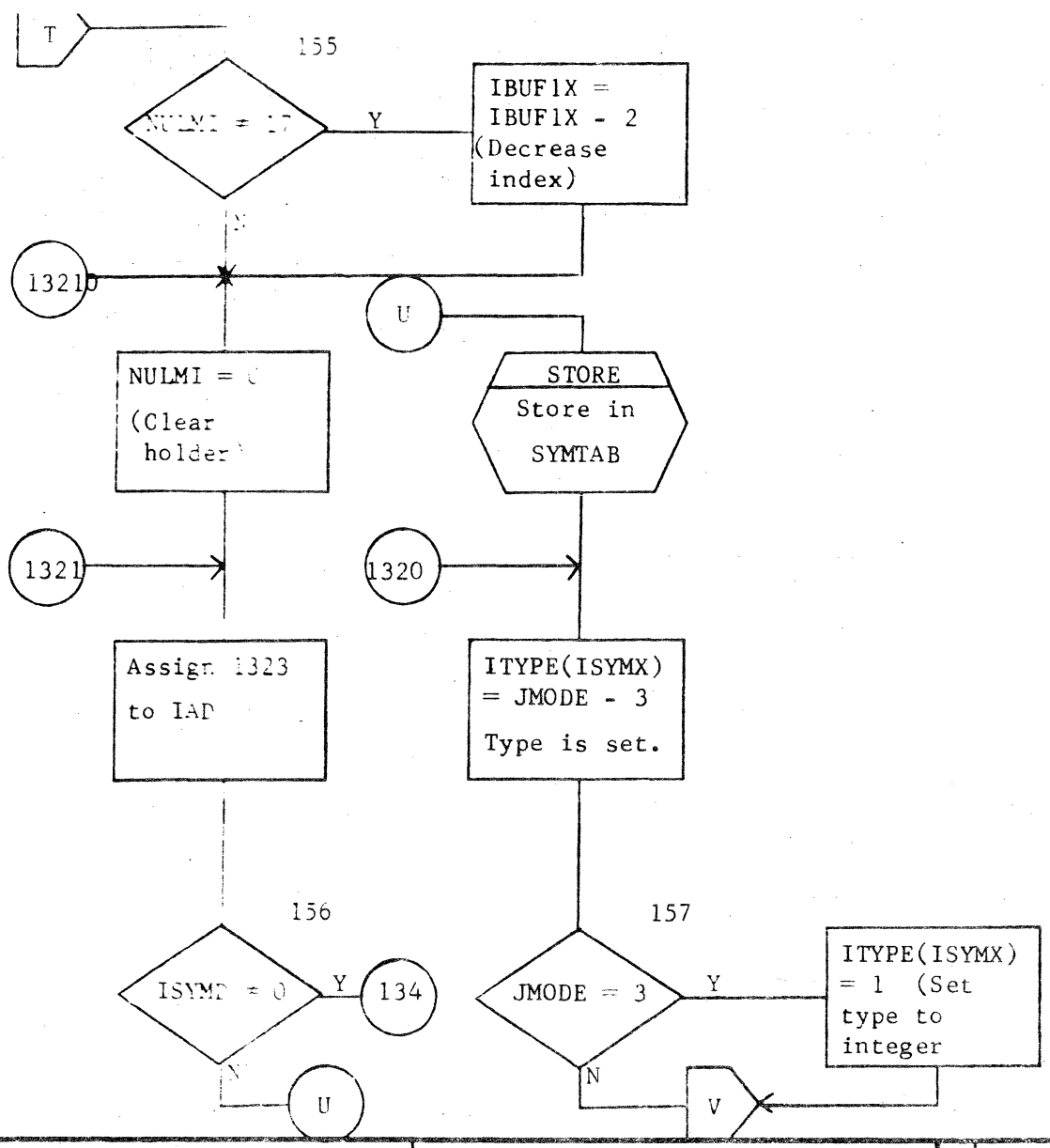
TITLE	ARITH		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF -

CSD 203

155. Last symbol not ")"?

156. Symbol not in symbol table?

157. Is type integer?



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE: ARITH		PROJECT MGR.			
	PAGE 32 OF 35		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 1/7	TASK NAME			

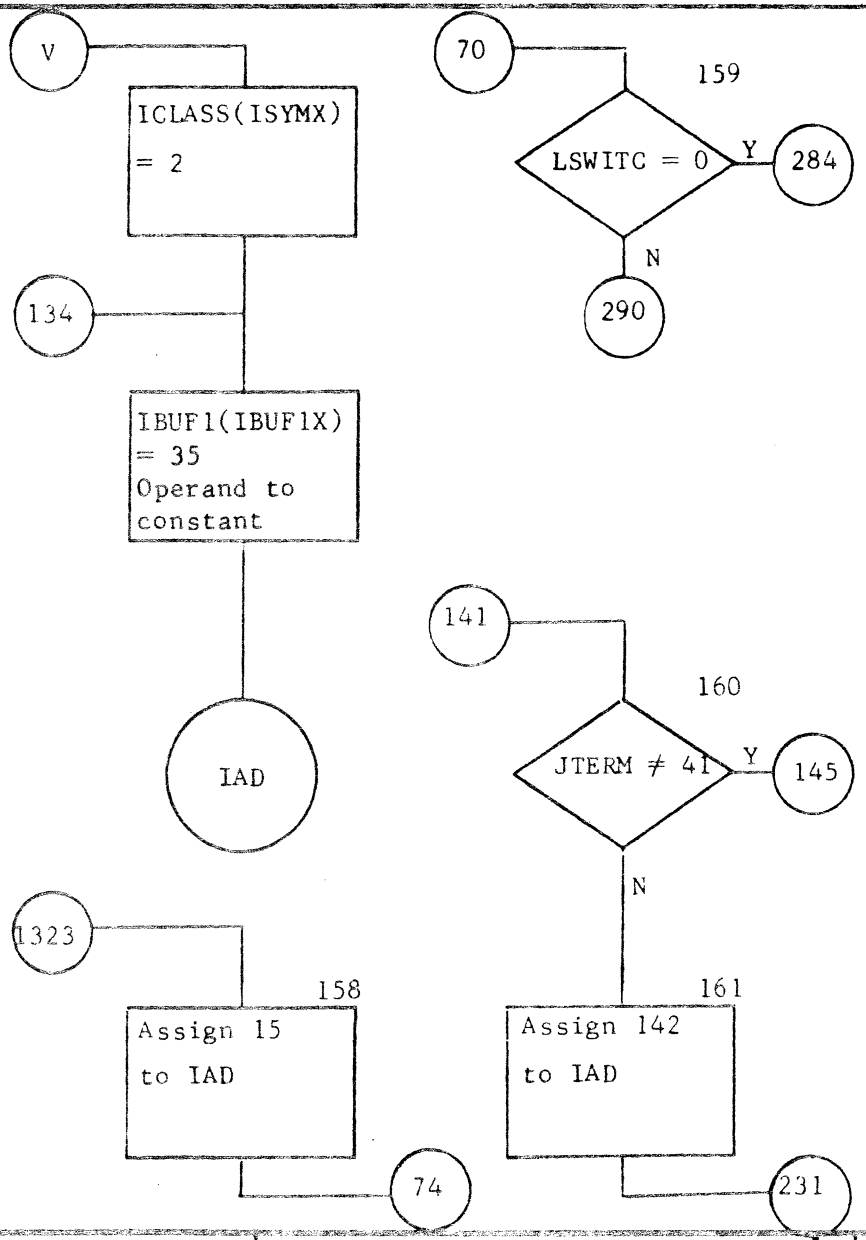
AA1385 (FORMERLY CA127-1)

158. Go check predecessor.

159. Class is function. Check for function indicator set.

160. Not left parenthesis? Yes, go to variable processor.

161. Process arguments as in subroutine call.



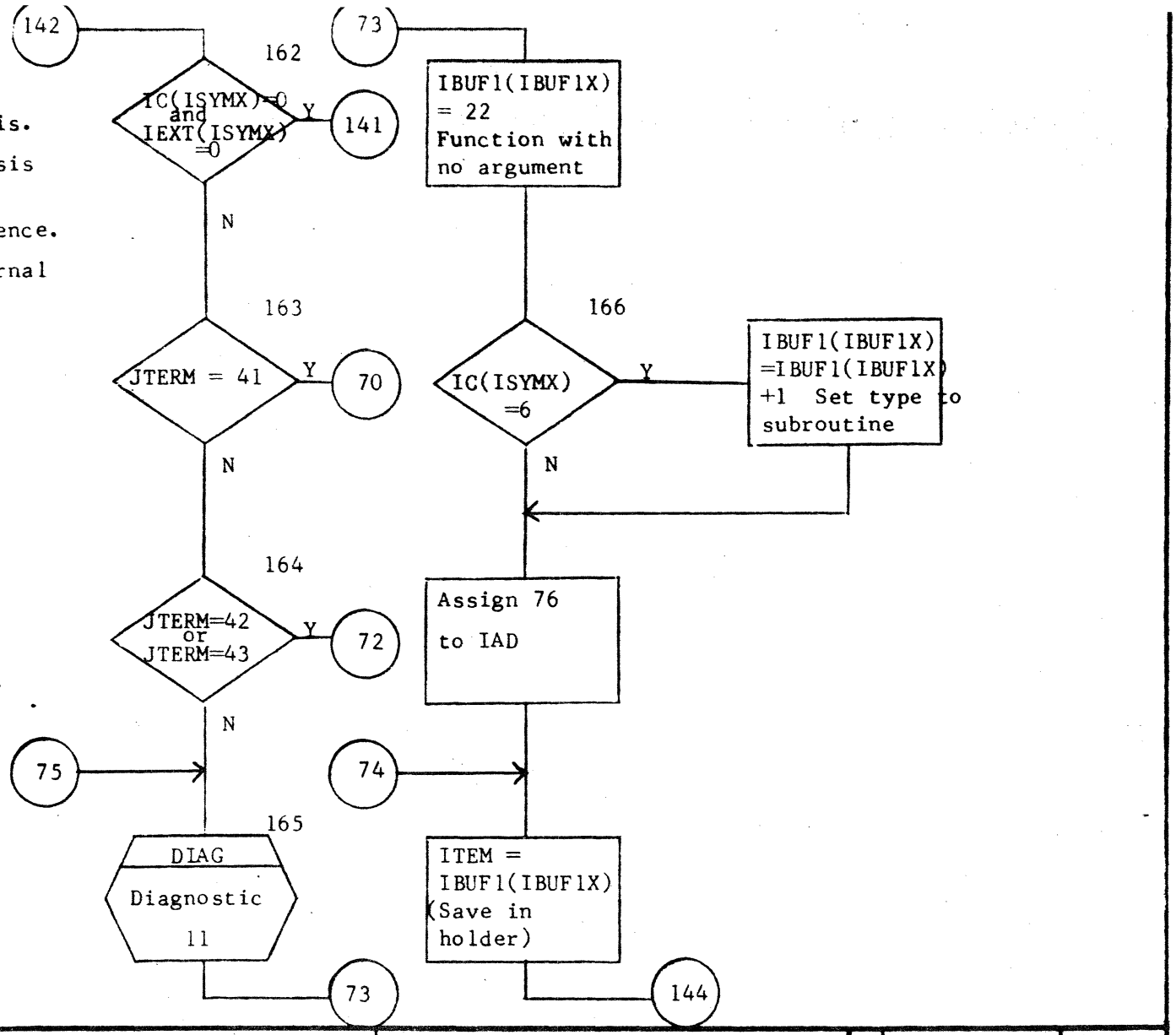
162. Determine if class is set and external is set.

163. Check for left parenthesis.

164. Check for right parenthesis or comma.

165. Illegal subprogram reference.

166. Check if class says external subroutine.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARITH		PROJECT MGR.		
		PAGE 34 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 7/1	TASK NAME		

9h-E

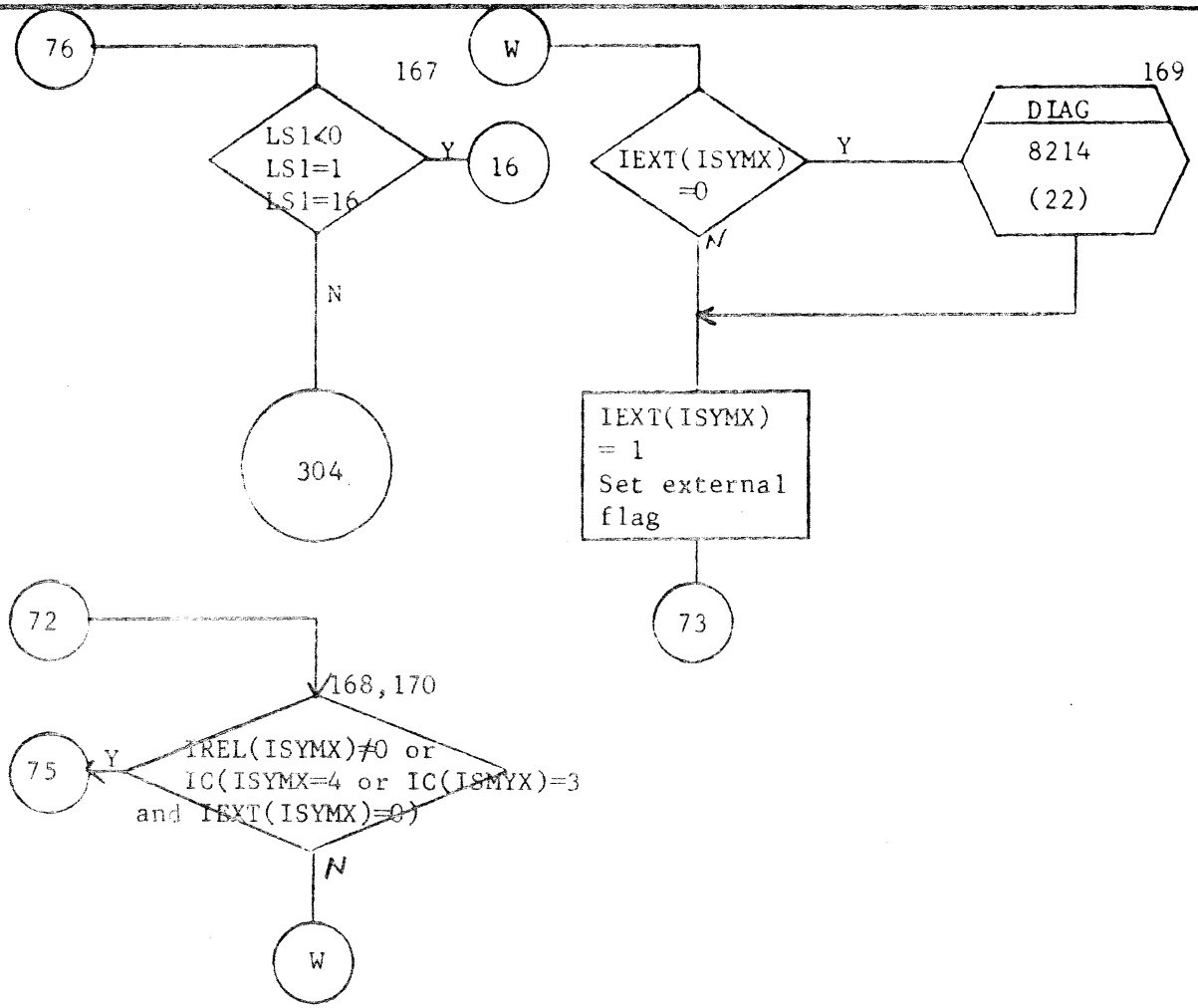
CA1385 (FORMERLY CA127-1)

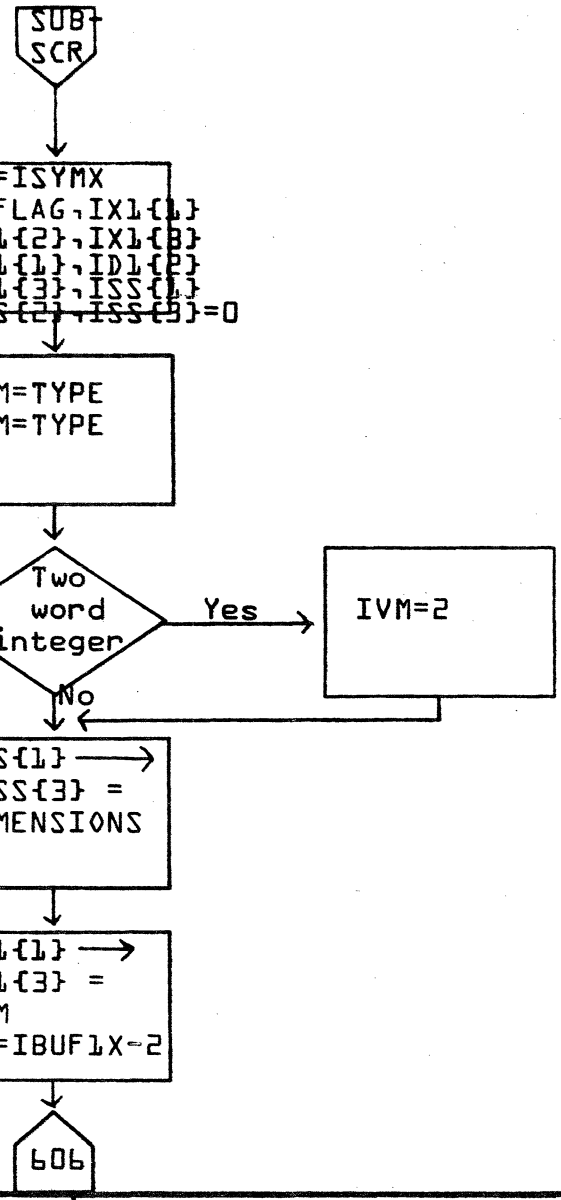
167. Check for predecessor a comma or right parenthesis.

168. Check for RELATIVE, statement function or Intrinsic function and external. Relative external is illegal as an argument.

169. Subprogram name does not appear in external statement.

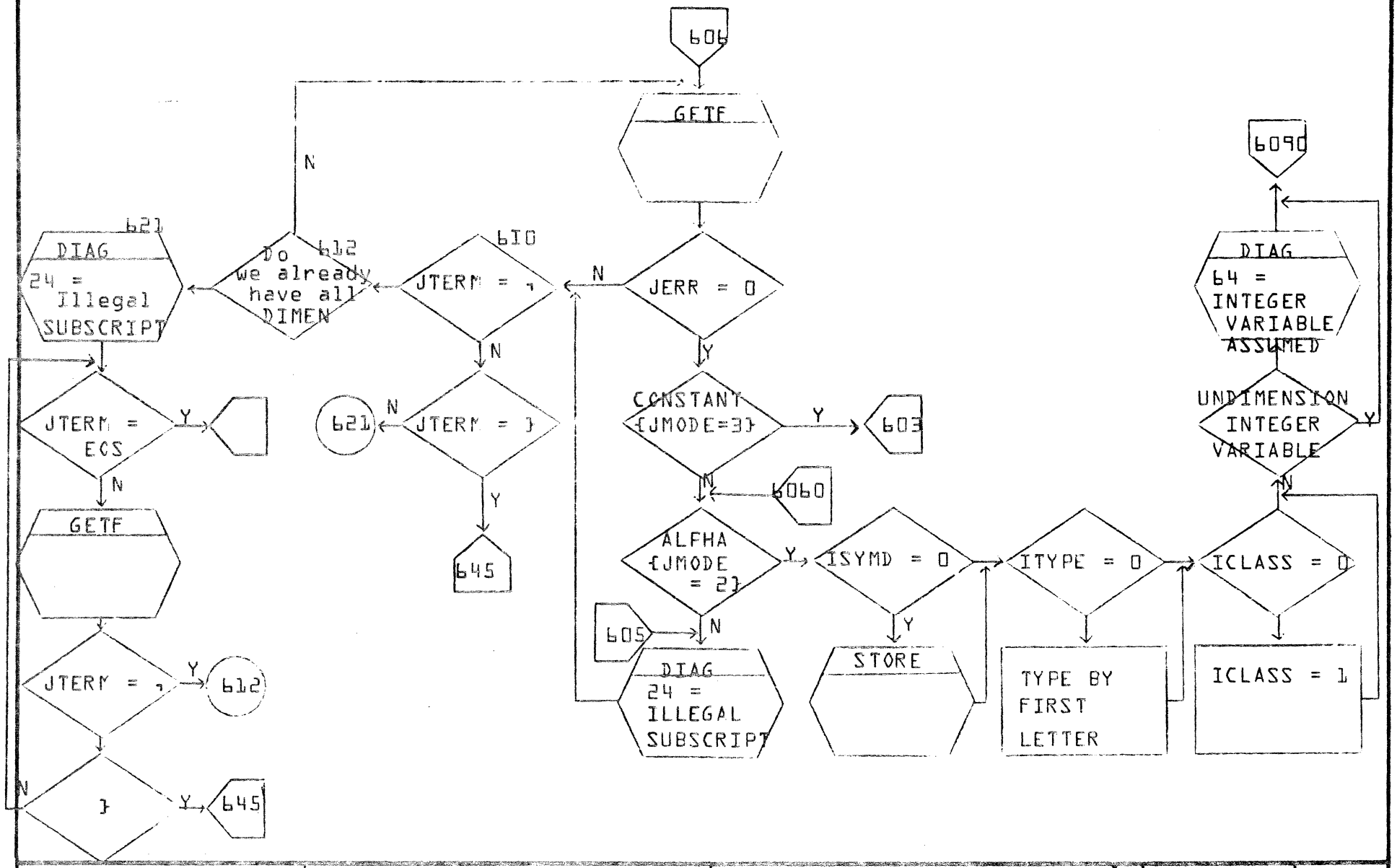
170. Run-anywhere option produces relative externals.



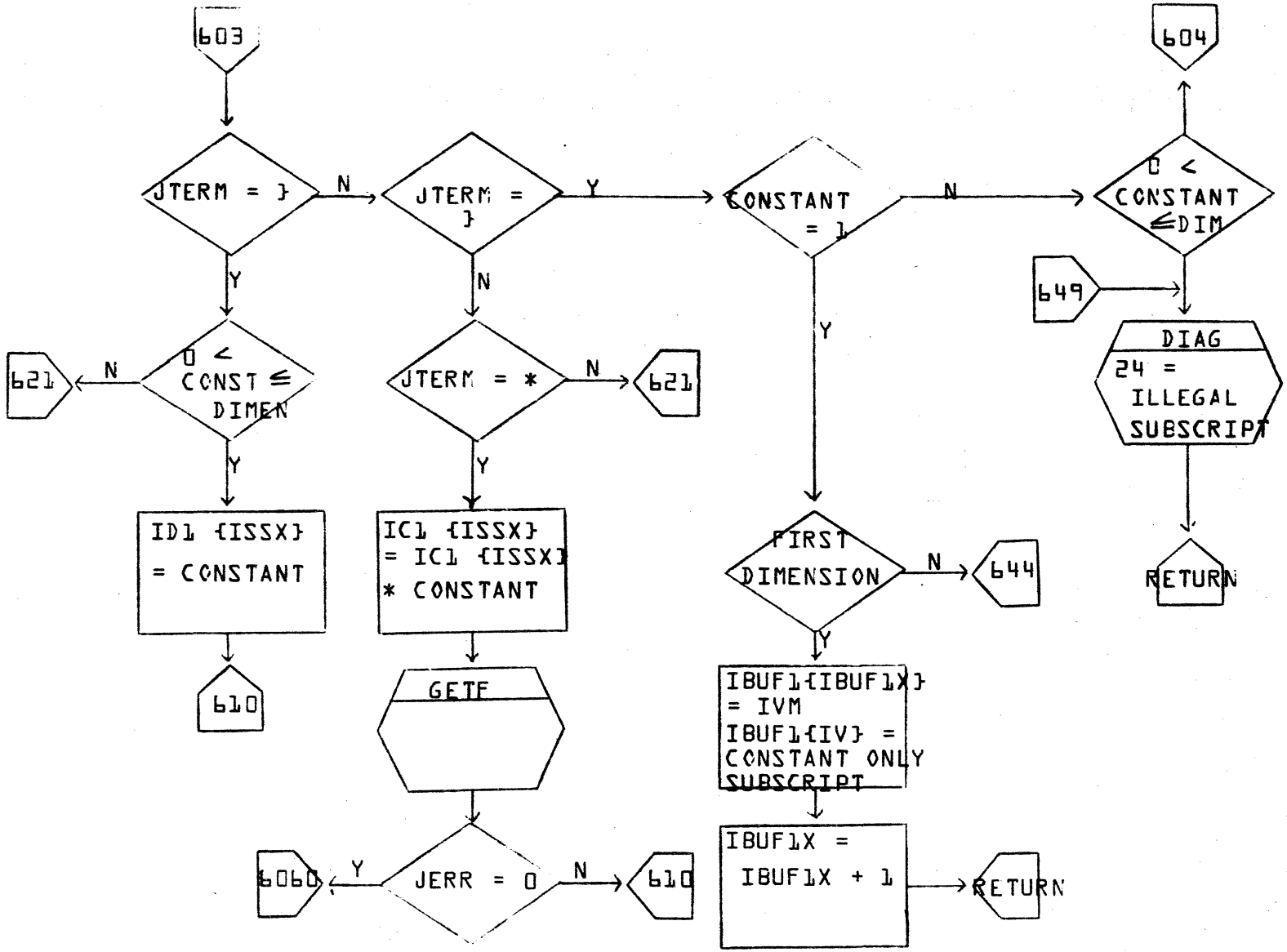


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SUBSCR		PROJECT MGR.		
		PAGE 1 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

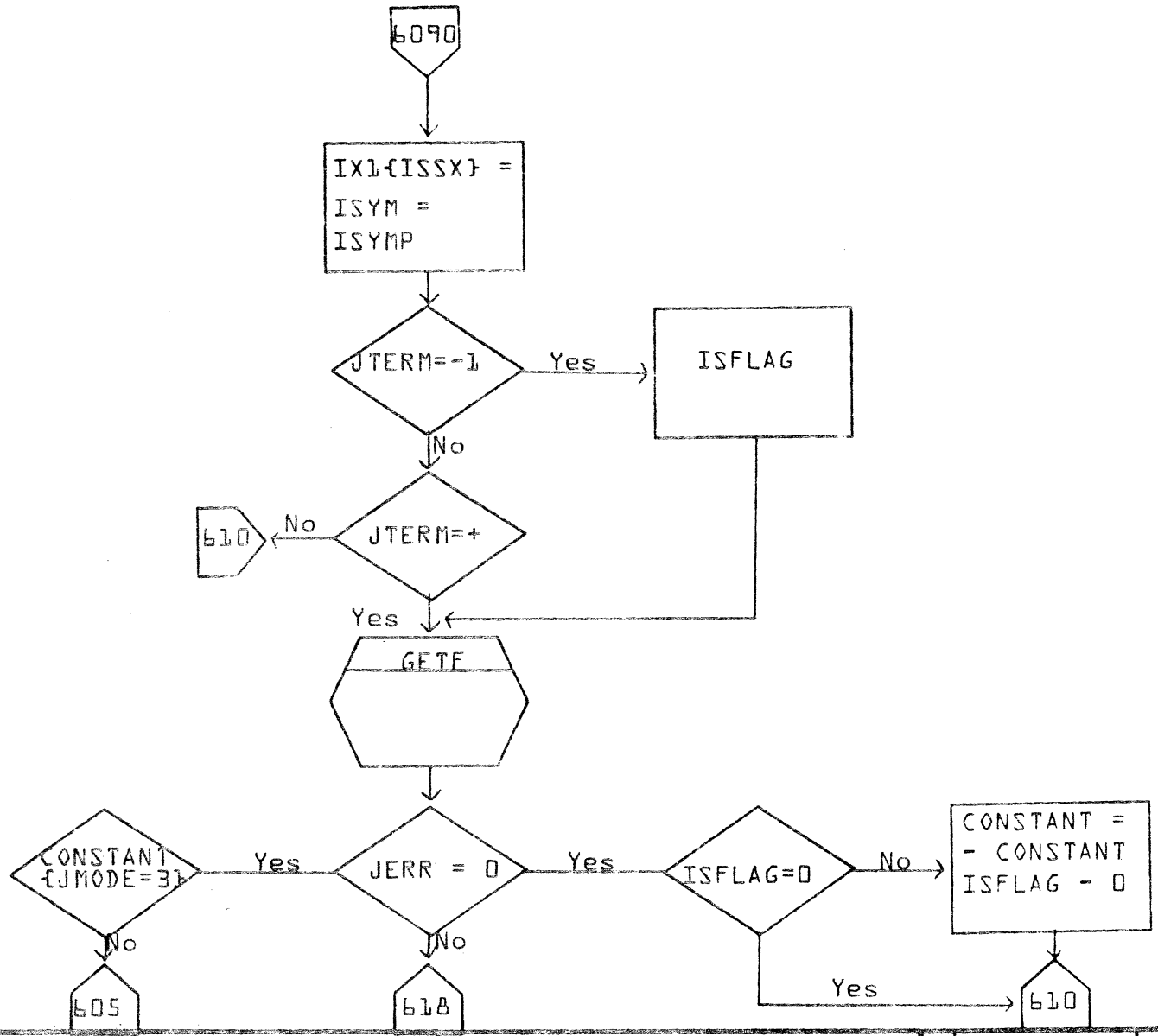


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SUBSCR		PROJECT MGR.		
		PAGE 7 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

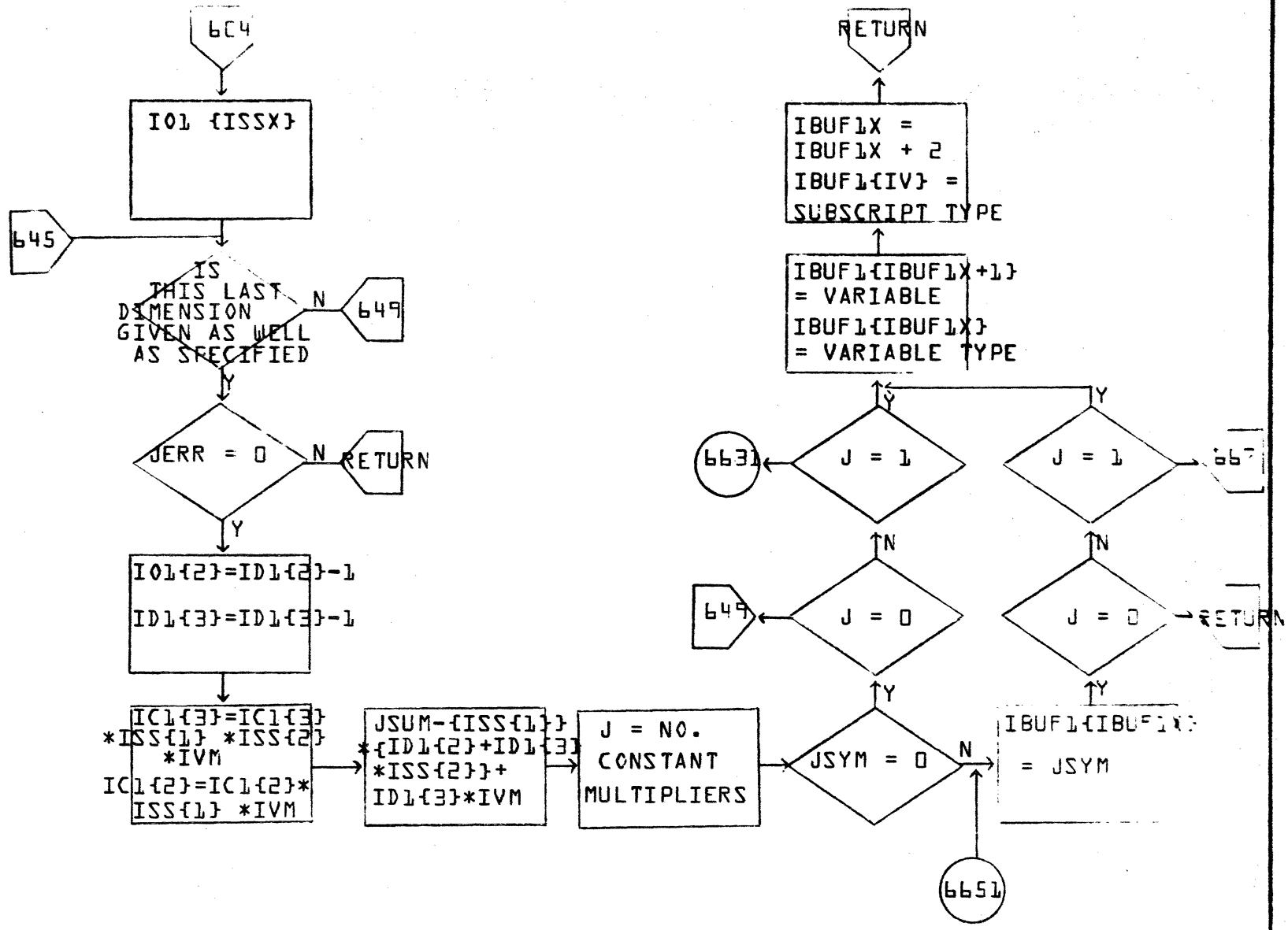


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SUBSCR		PROJECT MGR.		
		PAGE 3 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

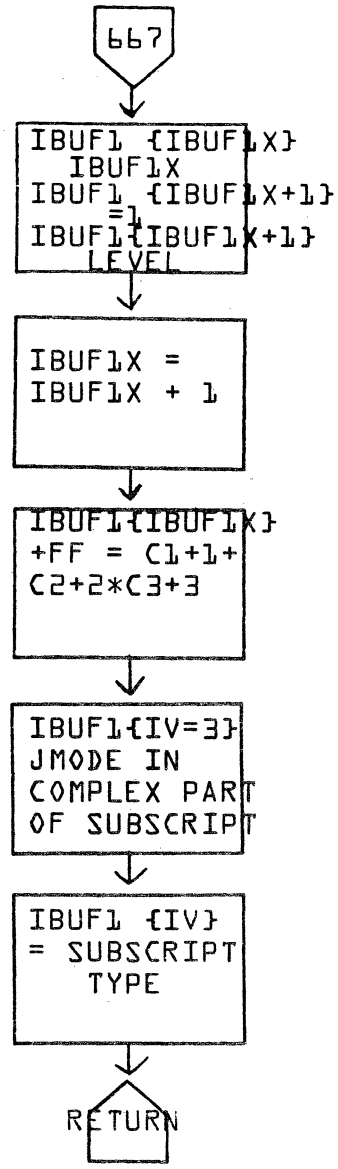


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBSCR		PROJECT MGR			
		PAGE 4 OF 6	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED DATE
	DOCUMENT TITLE SUBSCR		PROJECT MGR.		
		PAGE 5 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1365 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBSCR		PROJECT MGR.			
		PAGE 6 OF 6	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

3.4.3 Subroutine TREE

This routine picks up from IBUF1 an arithmetic expression that has been initially processed by ARITH and converts it into a tree form in IBUF2 for output. In order to make a tree, multiple passes are made over the statement. Each pass finds the primary operator and its related operands. If the operand is a subexpression, it, in turn, is broken up in a similar fashion and represents a lower level or branch of the tree. Each level of the tree is ordered by operator followed by its operands.

For example:

xa * a + b * c

is scanned and broken into the subexpressions:

x a * a

and b * c

The operator + is output

x a * a

is scanned. The operator * and the variables x, a and a are output then

b * c

is scanned. The operator * and the variables b and c are output.

The form of the Arithmetic tree set up in IBUF2 is as follows:

example = C+D

WORD 1	-1	Tree indicator
WORD 2	10	n = words in expression {from word 4 to end}
WORD 3	2	Mode of expression {1=integer, 2=real, 3=double}
WORD 4	11	Operator {see list}
WORD 5	2	n = number of operands
WORD 6	4	Pointer to 1st operand
:		Pointer to next operand, etc.
WORD {N}		N = {5 + number of operands}
:		
WORD {Z}		Operands follow operand pointers {see list}

example {cont'd}

WORD 7	7	Pointer to second operand
WORD 8	0	Normal/inverse switch
WORD 9	24	Operand type {see list}
WORD 10	→	SYMTAB Pointer to D
WORD 11	0	Normal/inverse swithc
WORD 12	24	Operand type
WORD 13	→	SYMTAB Pointer to C

Operand pointer = first word number of operand - word number of operator

that is, 4 = 8 - 4
that is, 7 = 11 - 4

Note: An operator on a lower level is treated as an operand by a higher level operator, therefore, an operator may be both an operator and operand depending on which level is being processed.

TREE is called by the subroutine ARITH. In the2.0A version, if ARITH detected mixed mode arithmetic in the expression, TREE calls MODMXR just before returning to ARITH.

Subroutines used: DIAG, GETSYM, PUNT, MODMXR.

Switches and Tables during tree building

IPERAT	Current operator
IBUF2	Table where output entry is built
IBUF2X	IBUF2 indexer
IBUF1	Table containing input expression
IBUF1X	IBUF1 indexer
LS1	.NOT. operator in effect switch
LPREN	IBUF1X value on which to stop current subexpression analysis
NOOP	Number of operators this level
MINUS	Leading variable preceded by an operator swtich
NLP	First non-left parenthesis operator has been encountered switch - used in determining existence of leading sign
ISAV	IBUF2 index of operator this level
LSWITC	Last operator was a function switch
ISSX	Level of current operator
IVM	Start of expression in IBUF2
ISFLAG	First operator switch - used in counting leading left parens
IWORK	History of expression being built in IBUF2 Four words per entry Word 1 IBUF1 index on which to stop subexpr analysis (Dummy entry indicator for .NOT if 0) Word 2 IBUF1 index for operator, this subexpression or: Assumed operator indicator (-operator in upper six bits and IBUF1 index in lower 10) Word 3 IBUF2 index of operand pointer, this operand Word 4 IBUF2 index of operator this level
ISTWK	IWORK index where entries, this level, start
LO	Class of prior operator
IMSW	Integer divide operator switch 0 = no prior operator 1 = logical 2 = relational 3 = arithmetic
LRELRQ	IWORK index
LRELEN	IBUF1 index at which to start current subexpression analysis

NOTE: Each operand (operator) except the first is preceded by a normal/inverse indicator.

Operand Types within Trees (Operators are Also Operands)

1. COMMA - does not appear in File 2
2. AND
3. OR
4. NOT - does not appear in file 2
5. LT
6. GT
7. LE
8. GE
9. EQ
10. NE
11. +
12. - does not appear in File 2 - represented by inverse +
13. * (÷ inverse *)
14. non-reorderable ÷ (integer)
15. **
16. (does not appear in File 2
17.) does not appear in File 2
18. Function
19. unused
20. subroutine
21. unused
22. function with no argument
23. subroutine with no argument
24. non-subscripted variable
25. variable only sbuscripted variable
26. increment only subscripted variable
27. variable and increment subscripted variable
28. complex subscripted variable
29. non-subscripted partial variable
30. variable only subscripted partial variable
31. increment only subscripted partial variable
32. variable and increment subscripted variable
33. complex subscripted partial variable
35. numeric constant
36. calling sequence label - pass 4 generaged
37. material constant

Types 22 - 24, 29, 34, 35 are two word entries

WORD 1 operand type
WORD 2 symbol table pointer

Types 18 - 21 take two words tp express type
WORD 1 operand type
WORD 2 symbol table pointer
WORD 3 no. operands etc.

Types 25 - 30 are of the following form
WORD 1 25,30
WORD 2 symbol table pointer
WORD 3 24,29 for subscript variable
WORD 4 symbol table pointer

Types 26 and 31 are of the following form
WORD 1 26,31
WORD 2 symbol table pointer
WORD 3 constant subscript

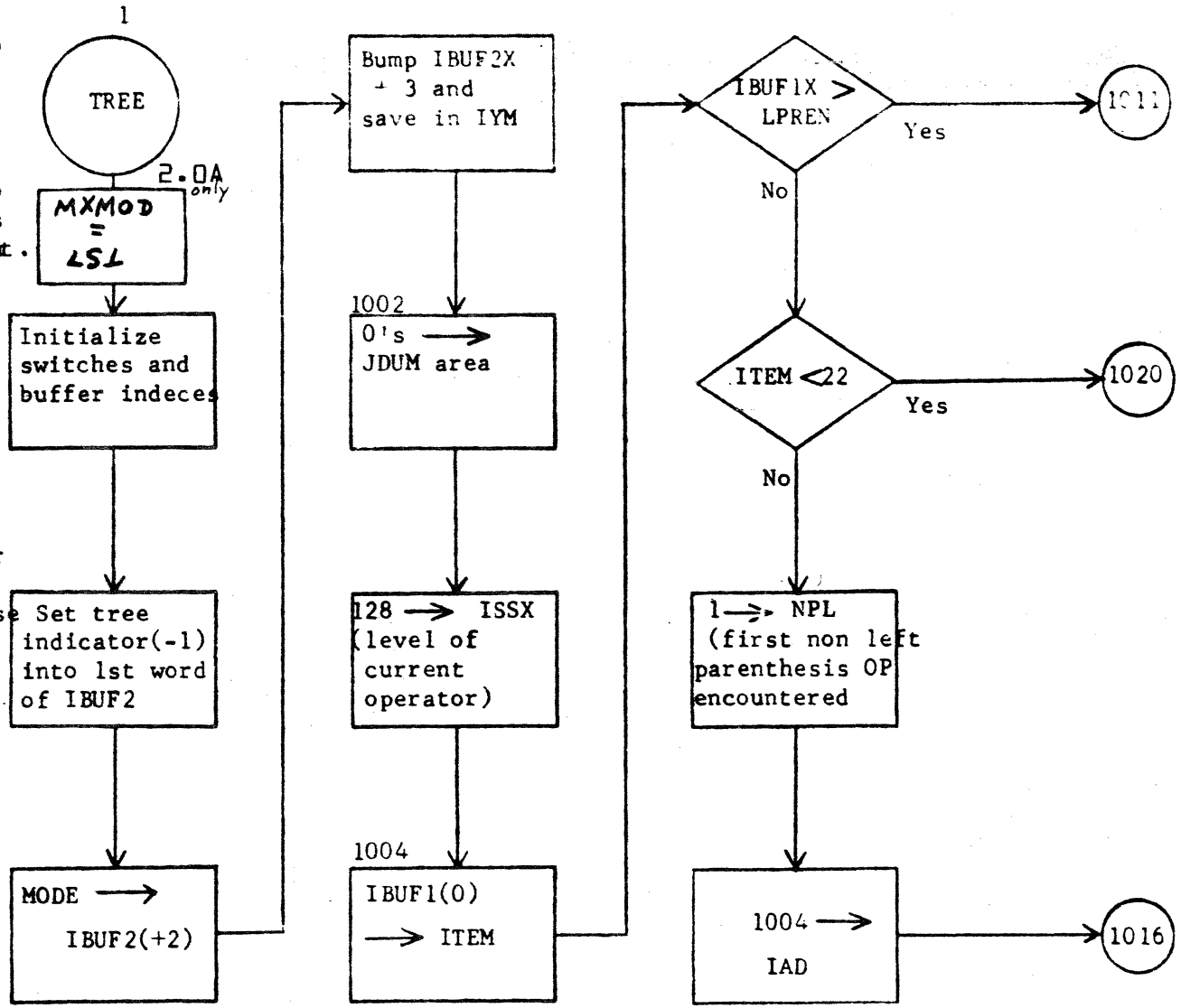
Types 27 and 32 are of the following form
WORD 1 27,32
WORD 2 symbol table pointer
WORD 3 constant portion
WORD 4 24,29 for subscript variable
WORD 5 symbol table pointer for subscript variable

Types 28 and 33 are of the following form
WORD 1 28,33
WORD 2 symbol table pointer
WORD 3 constant portion
WORD 4 operator (+ or *)
WORD 5 no. operands pointers operands

1. This routine picks up an arithmetic expression that has been initially processed by ARITH from IBUF1, converts it to tree form, and places it in IBUF2 for output.

Tree Form

Word	
1	Tree indicator(-1)
2	# of words in expression
3	Mode of expression
4	Operator
5	# operands
6	pointer base operator this level
7	Operand normal/inverse
8	Operand
	Etc.



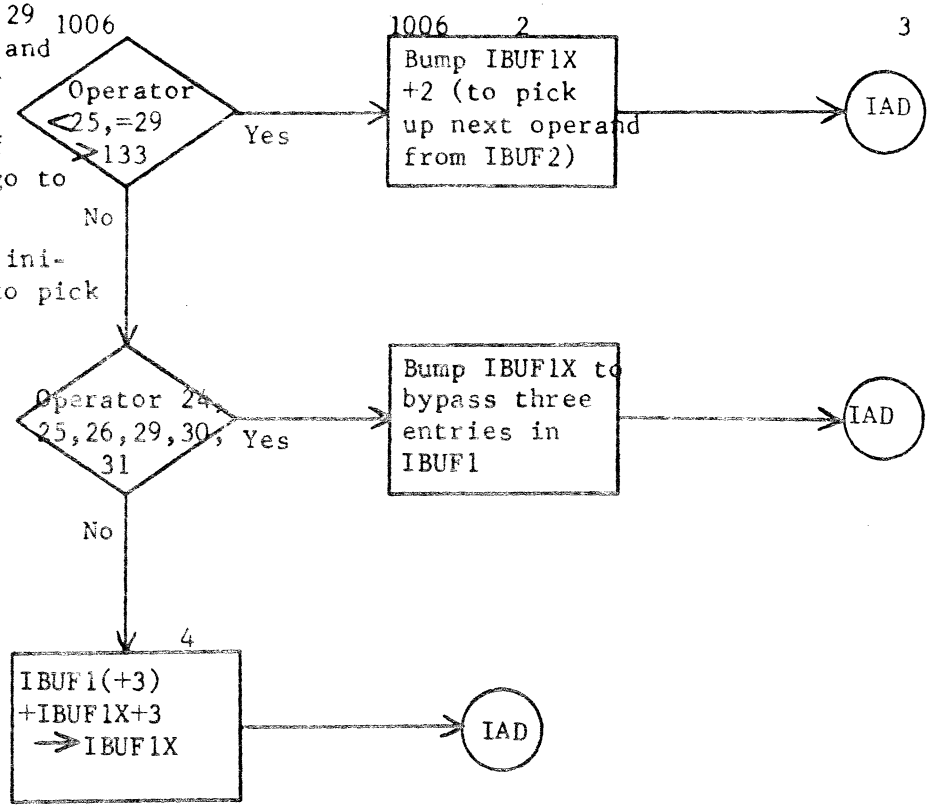
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE	DOCUMENT TITLE TREE		PROJECT MGR			
FLOWCHART		PAGE 1 OF 24	PROJECT NAME			
DECISION TABLE	NUMBER	ISSUE DATE	TASK NO.			
OTHER	DRAWN BY	DATE 3/67	TASK NAME			

AA1385 FORM

2. If operator is 24 or 29 go to statement 1007 and bump +4. If operator is 25, 30; go to 1008 and bump +3. If operator is 26, 31; go to 1009 and bump +1.

3. This return (IAD) is initially made to 1004 to pick up next operand from IBUF1.

4. Add fixed increment at IBUF1 (IBUF1X+3) to IBUF1X setting +3 and return to IBUF1X.

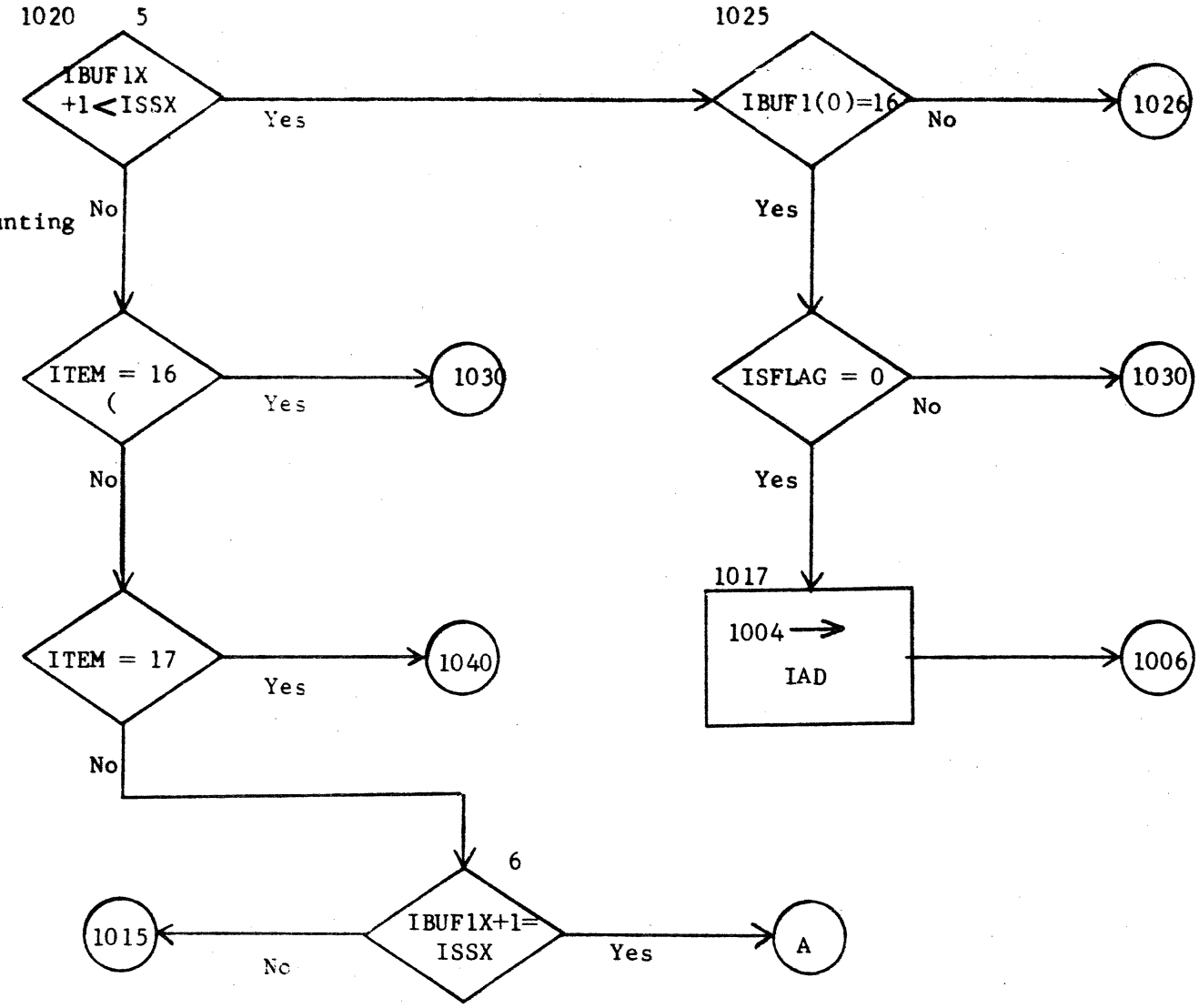


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR		
	PAGE 2 OF 24		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

5. Operator is less than 22 to come here.

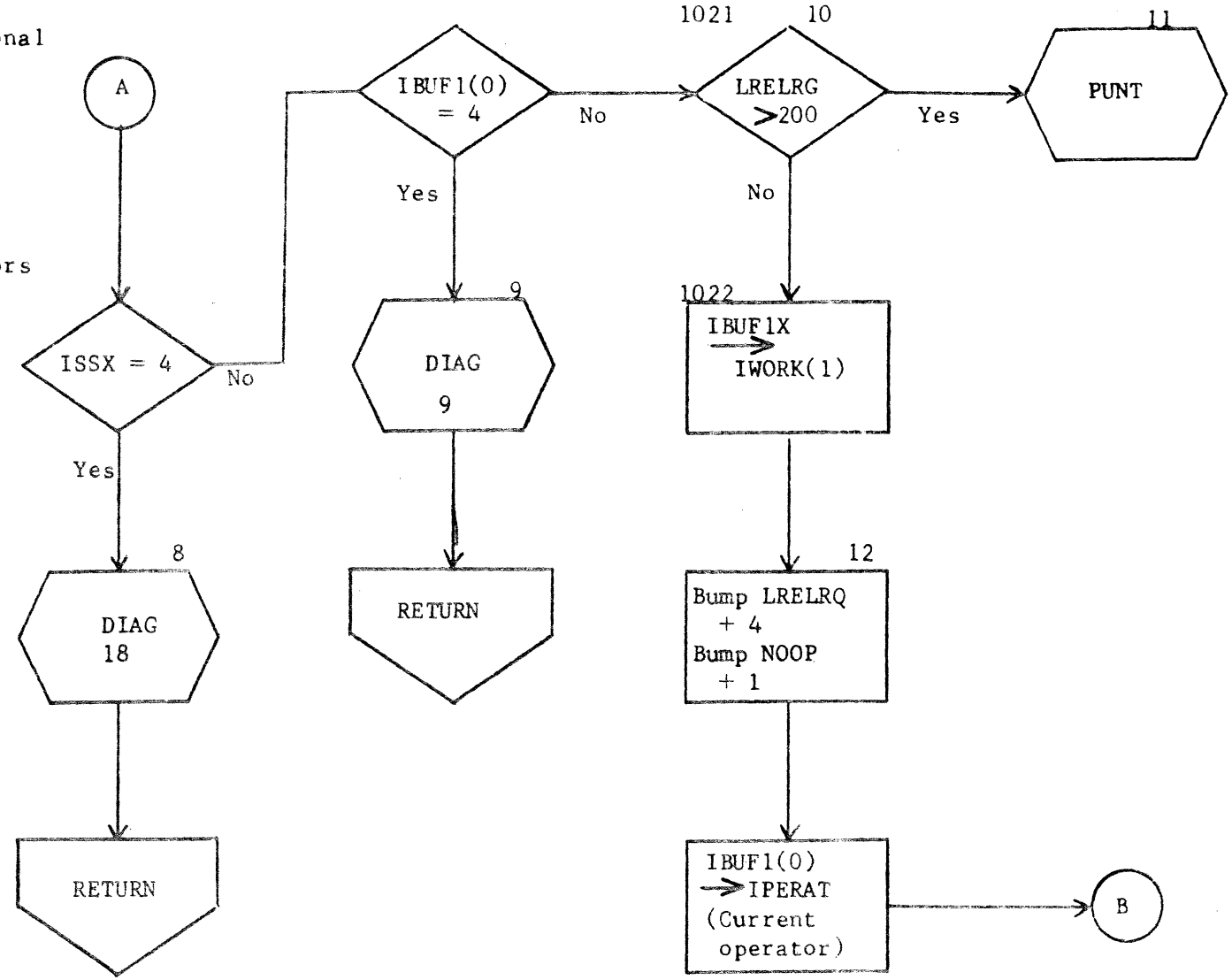
6. ISSX = level of current operator, initially 128.

7. ISFLAG = first operator switch used in counting leading left (.



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS EMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE TREE		PROJECT MGR.		
FLOWCHART ■		PAGE 3 OF 24	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE 3-67	TASK NAME		

8. Illegal use of relational operator.
9. Illegal use of NOT.
10. LRELRQ = IWORK index.
11. Error - IWORK table exceeded.
12. NOOP=number of operators this level.



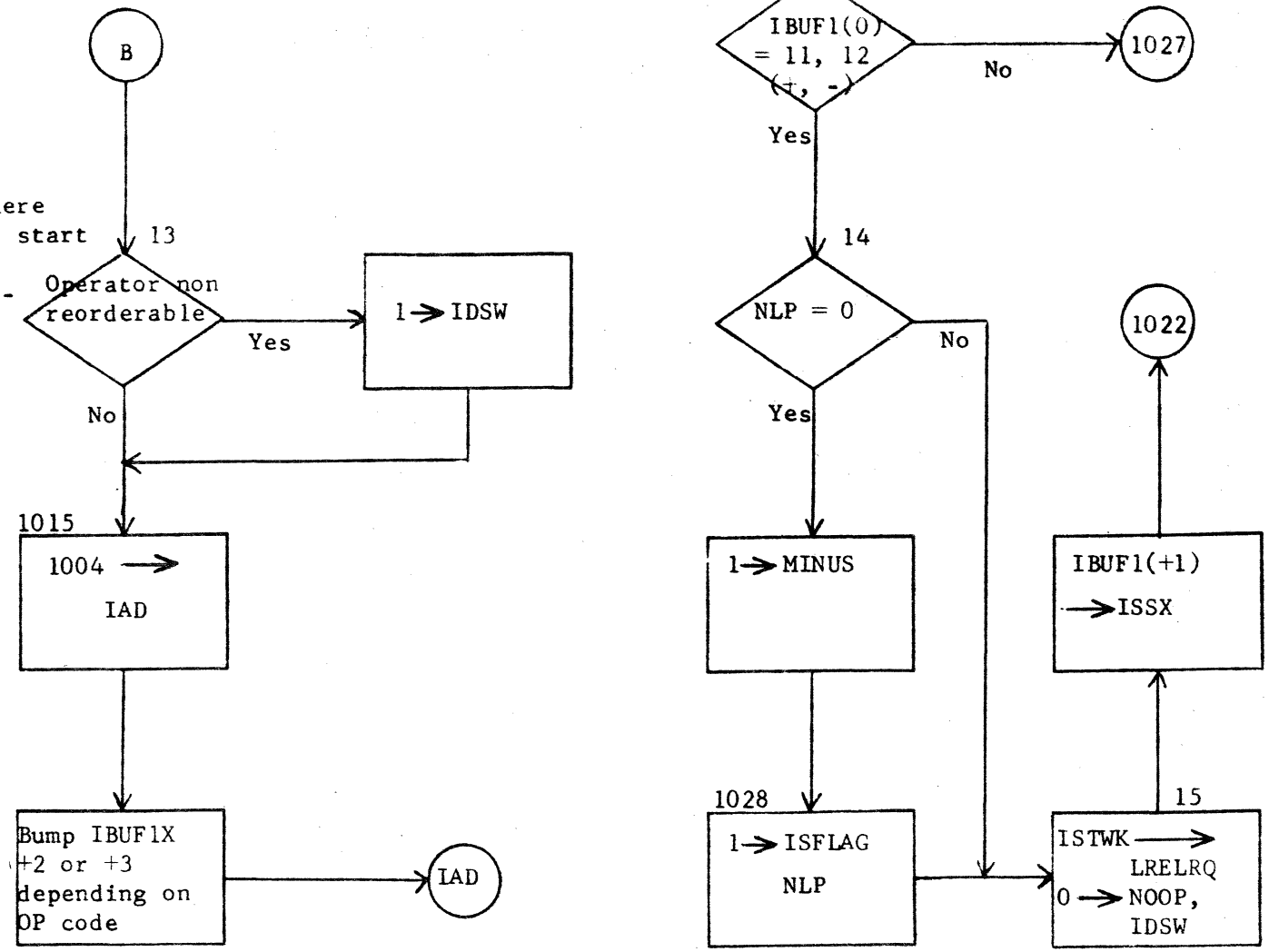
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IAS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR		
		PAGE 4 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE 3-67	TASK NAME		

13. IPERAT = 14
.AND.
IBUF1(+2) ≠ 0.

14. NPL = non left parenthesis operator for determining leading sign.

15. ISTWK = IWORK index where entries for this level start

NOOP = number of operators this level.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ■ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TREE		PROJECT MGR.			
		PAGE 5 OF 24	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 3-67	TASK NAME			

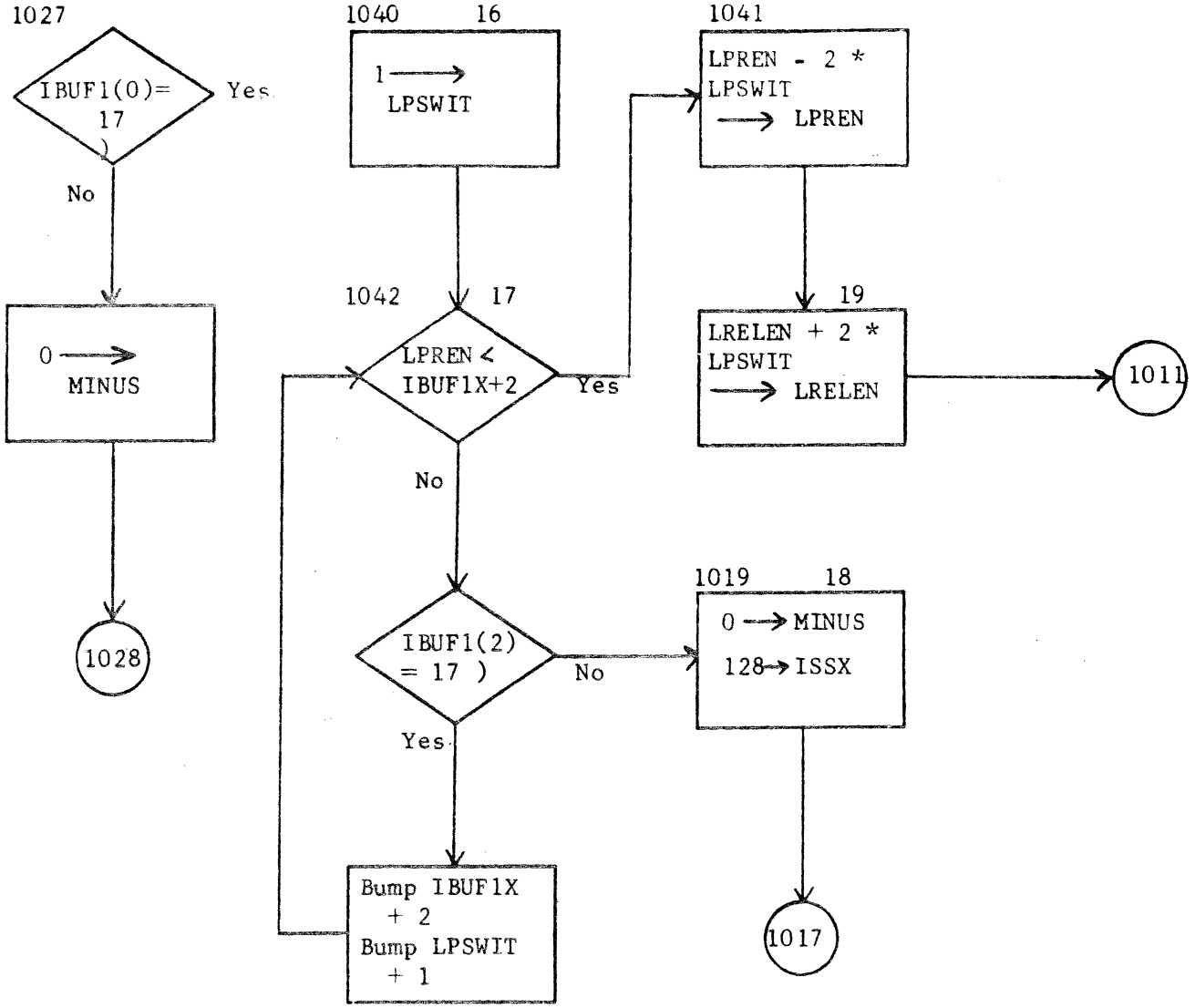
16. LPSWIT is count of non-imbedded).

17. Is this end of expression?

18. Reset level back to maximum.

19. LRELEN = IBUF1 index at which to start current subexpression analysis.

    LPREN = IBUF1X value at which to stop current subexpression analysis.

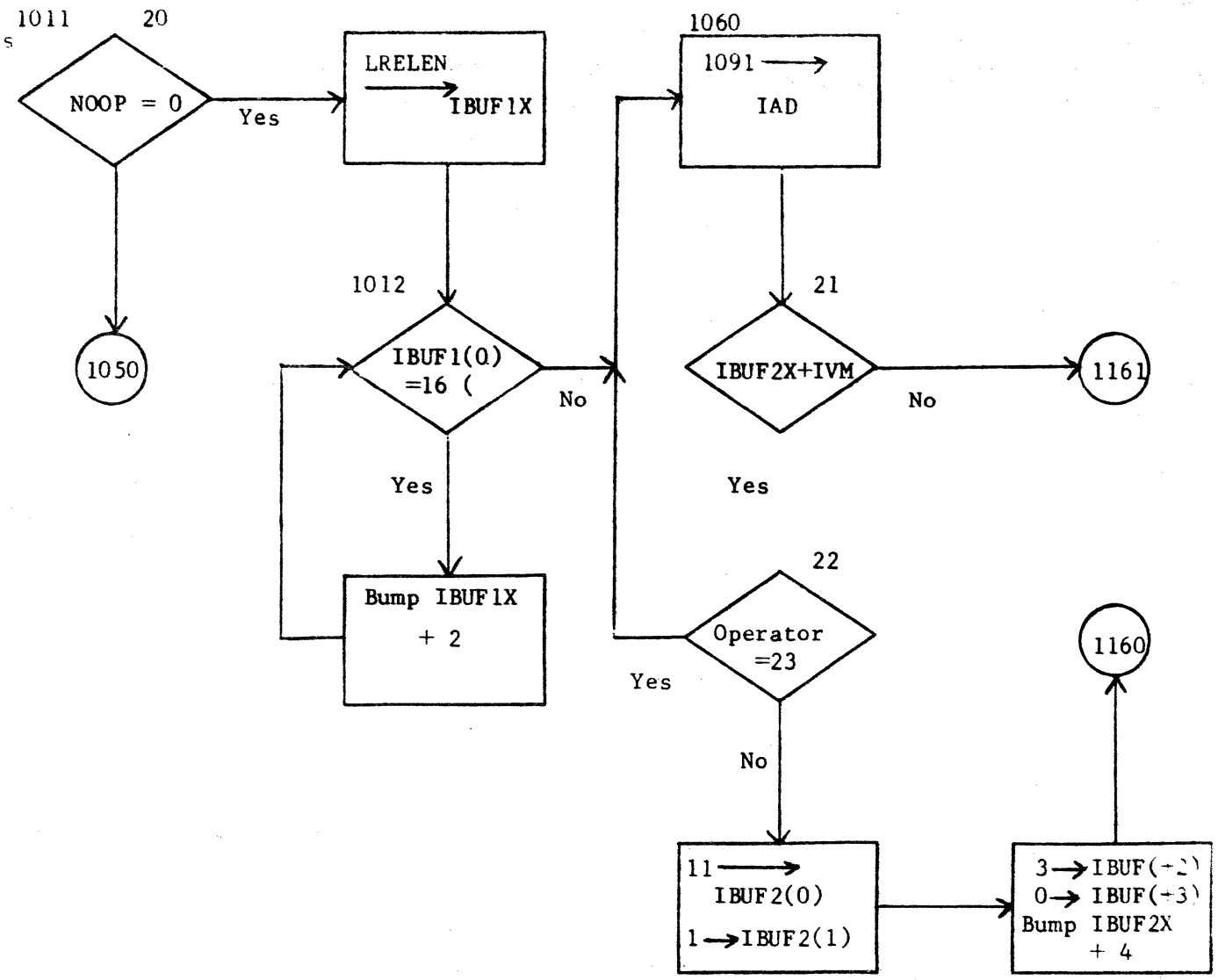


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ■ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR.		
		PAGE 6 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

20. End of expression comes here.

21. IVM = start of expression in IBUF2.

22. Subroutine with no argument.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ■ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR.		
		PAGE 9 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

AA1385 (FORMERLY CA127-1)

23. Put variable type and symbol table pointer into IBUF2.

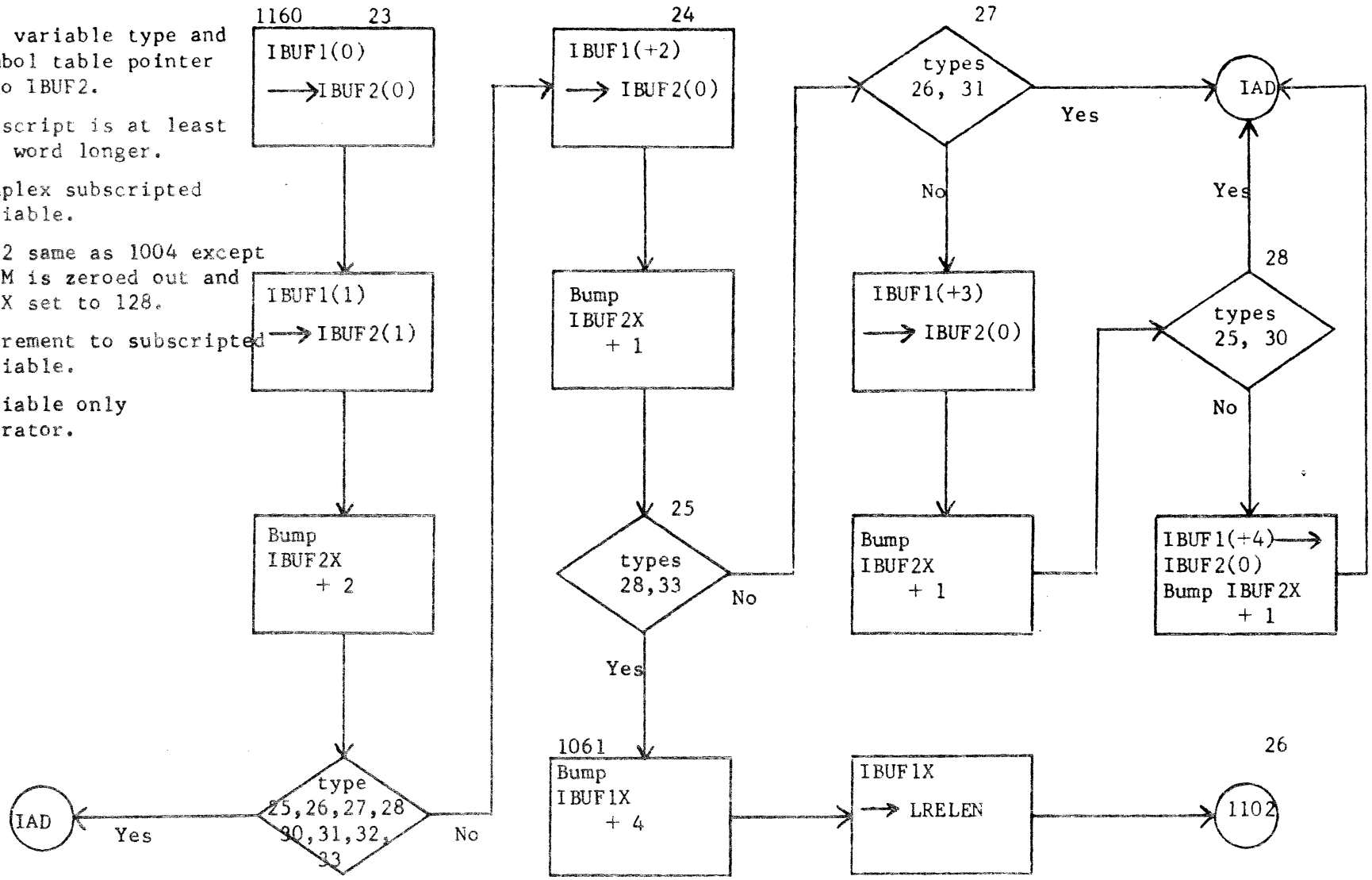
24. Subscript is at least one word longer.

25. Complex subscripted variable.

26. 1002 same as 1004 except JDUM is zeroed out and ISSX set to 128.

27. Increment to subscripted variable.

28. Variable only operator.



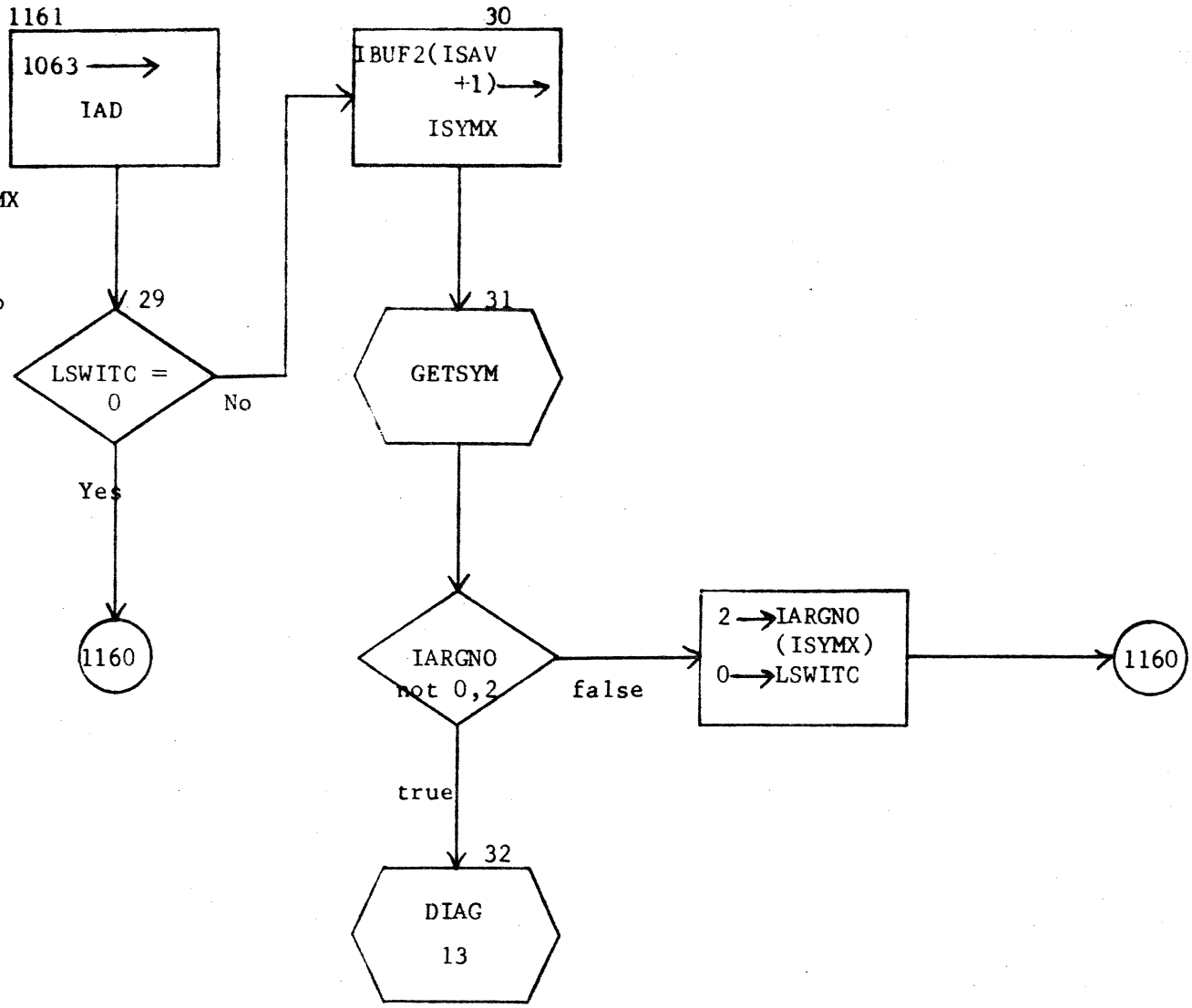
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ■ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TREE		PROJECT MGR.			
		PAGE 8 OF 24	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 3-67	TASK NAME			

29. Last operator was a function switch.

30. ISQV = index to operator this level.

31. Get page and symbol table pointer that ISYMX points to.

32. Number of arguments differ in references to same subprogram.

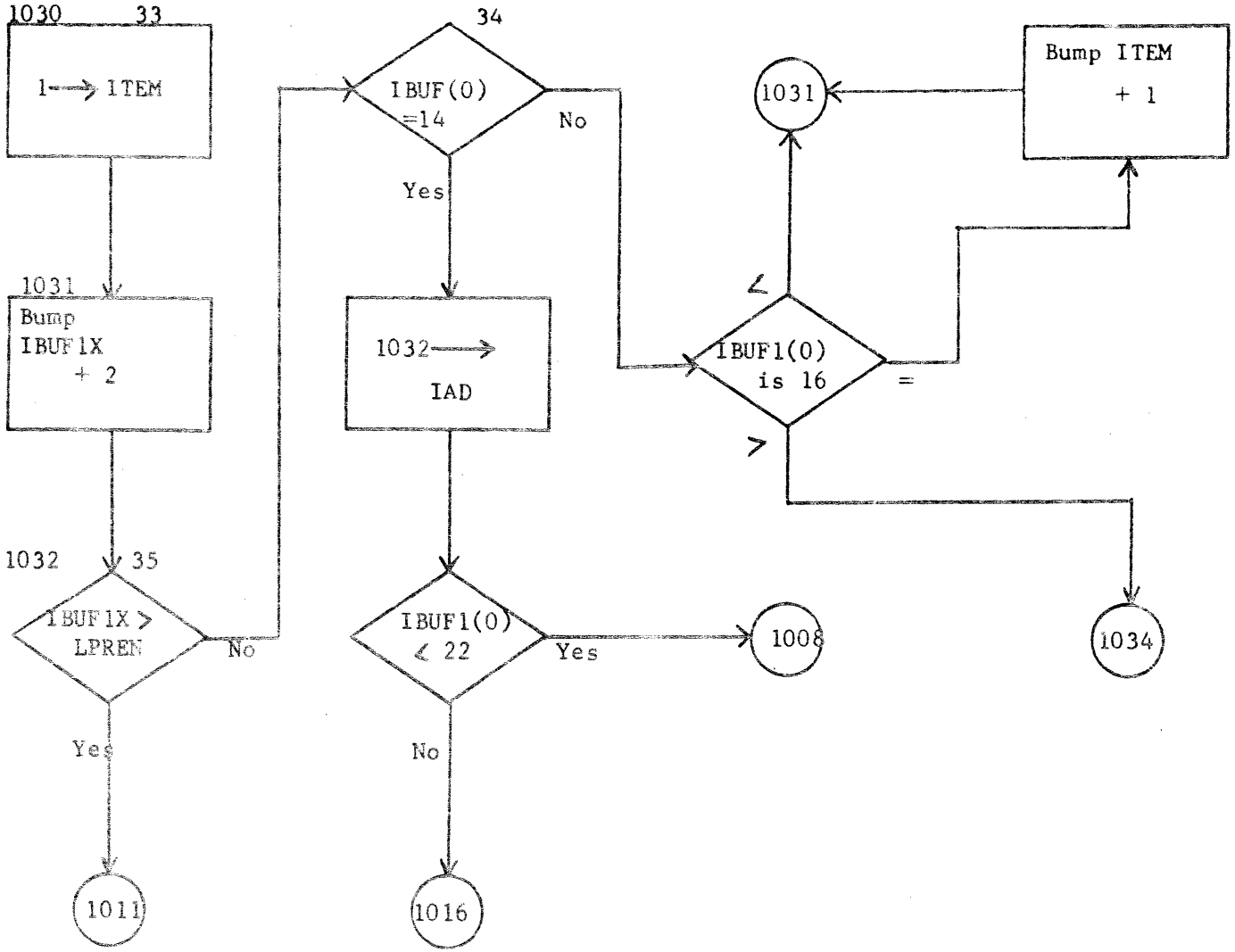


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE TREE		PROJECT MGR.			
FLOWCHART ■		PAGE 9 OF 24	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 3-67	TASK NAME			

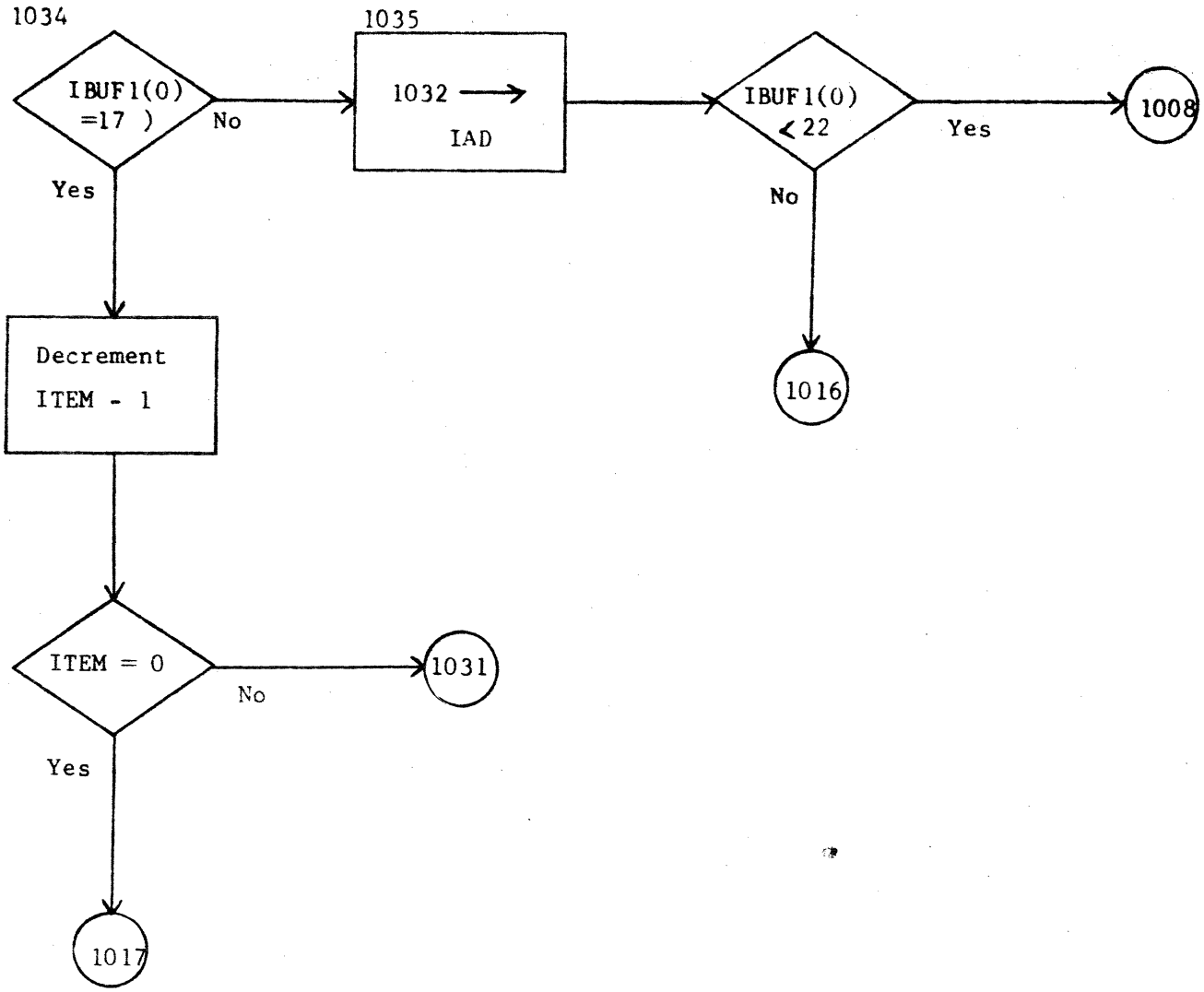
33. We found an imbedded (, slew to).

34. Non-reorderable.

35. End of expression.



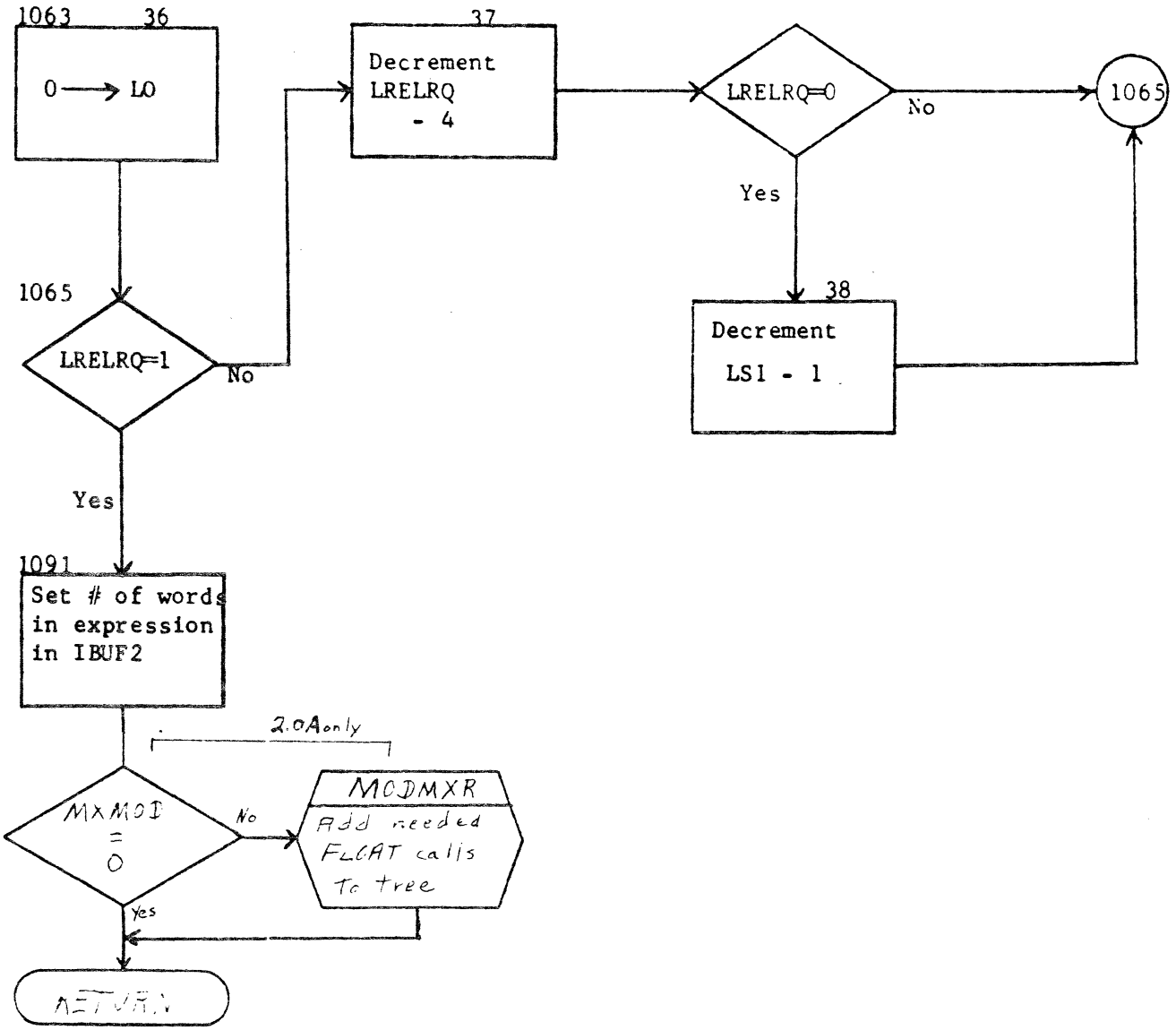
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TREE		PROJECT MGR			
		PAGE 11 OF 24	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 3-67	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE TREE		PROJECT MGR.			
FLOWCHART ☒		PAGE 11 OF 24	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 3-67	TASK NAME			

AA1385 (FORMERLY CA127-1)

36. Gotten to by assign IAD.
37. IWORK index.
38. NOT operator in effect switch.

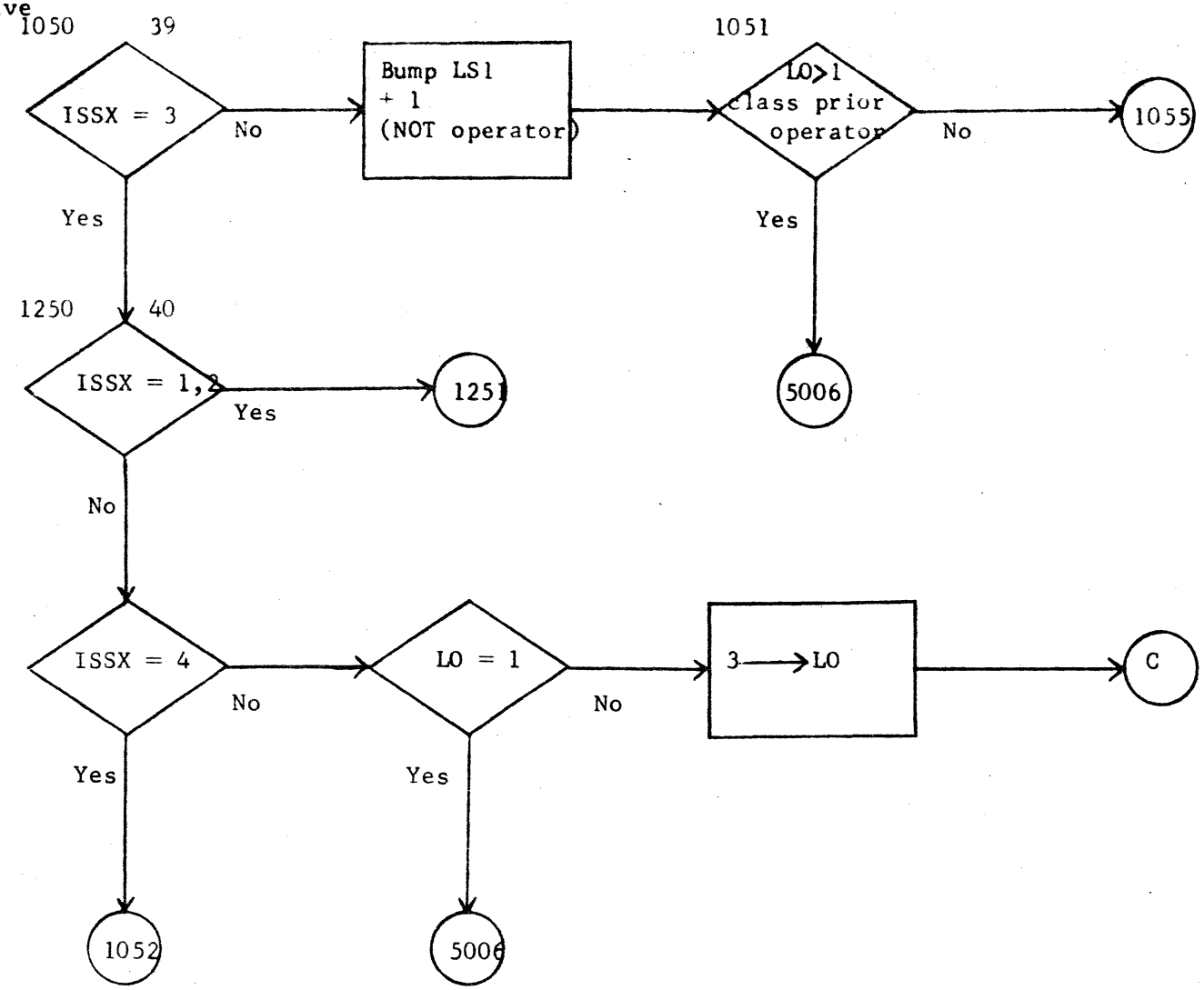


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR		
		PAGE 12 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-69	TASK NAME		

39. Number of operators have just been found to be NZ (NOOP).

40. ISSX = level of current operator.

Level	Operator
0	,
1	OR
2	AND
3	NOT
4	relational
5	+ -
6	* /
7	**
8	( ) function

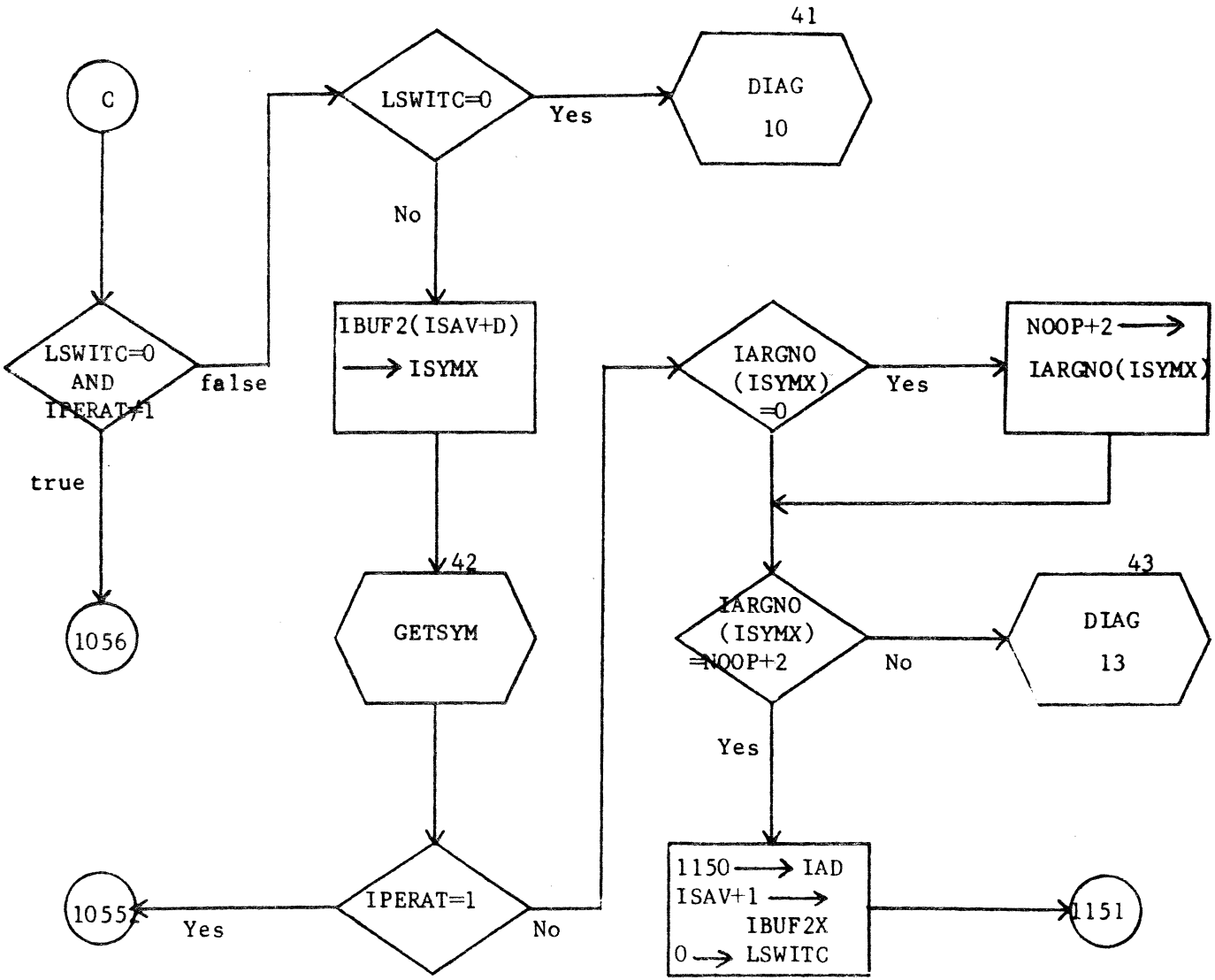


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS EMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR.		
		PAGE 13 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

41. Illegal operator or operand.

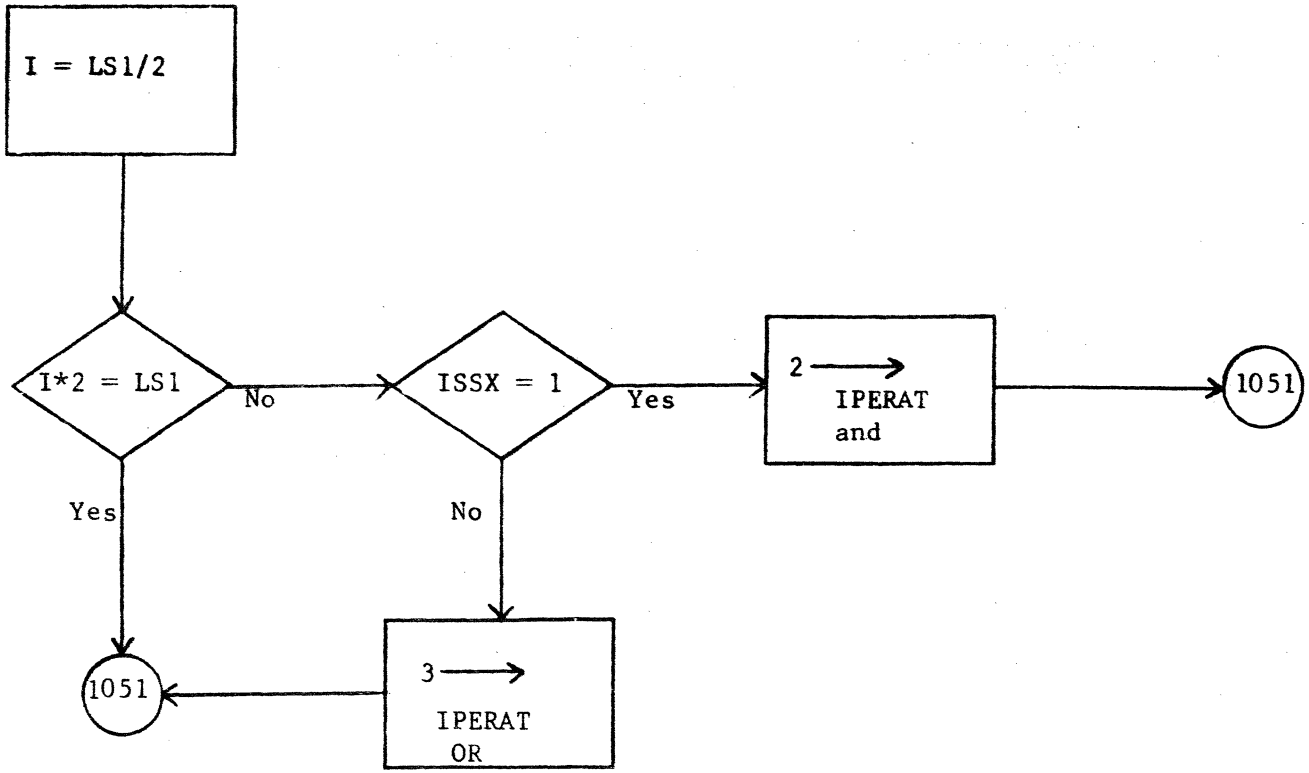
42. Get page # and symbol position.

43. # of arguments differ in references to same subprogram.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE TREE		PROJECT MGR.			
FLOWCHART ☒		PAGE 14 OF 24	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐						

44. Look for odd numbers (nested nots). 1251 44

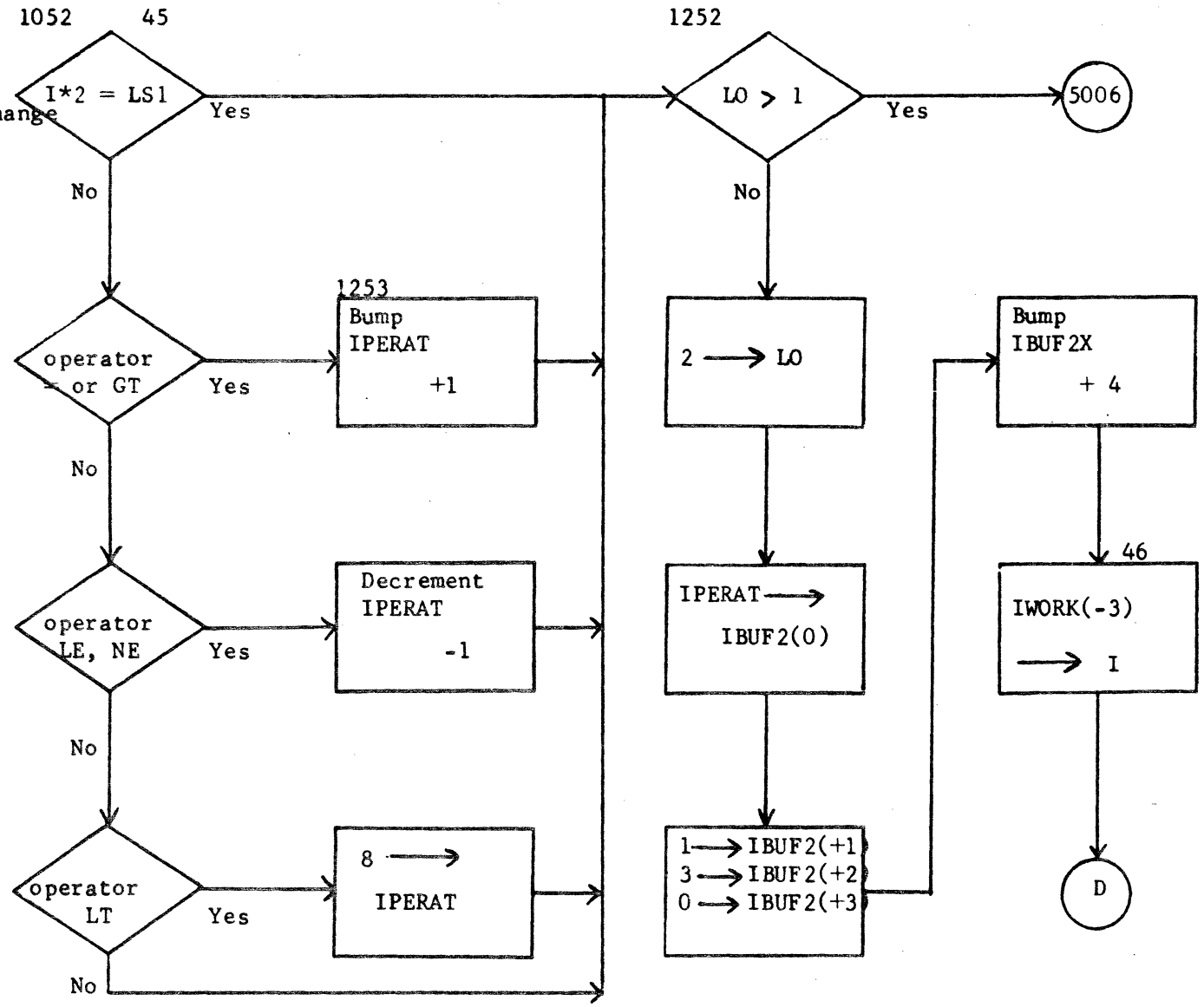


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE TREE		PROJECT MGR.		
FLOWCHART ☒		PAGE 15 OF 24	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE 3-67	TASK NAME		

AA1385 (FORMERLY CA127-1)

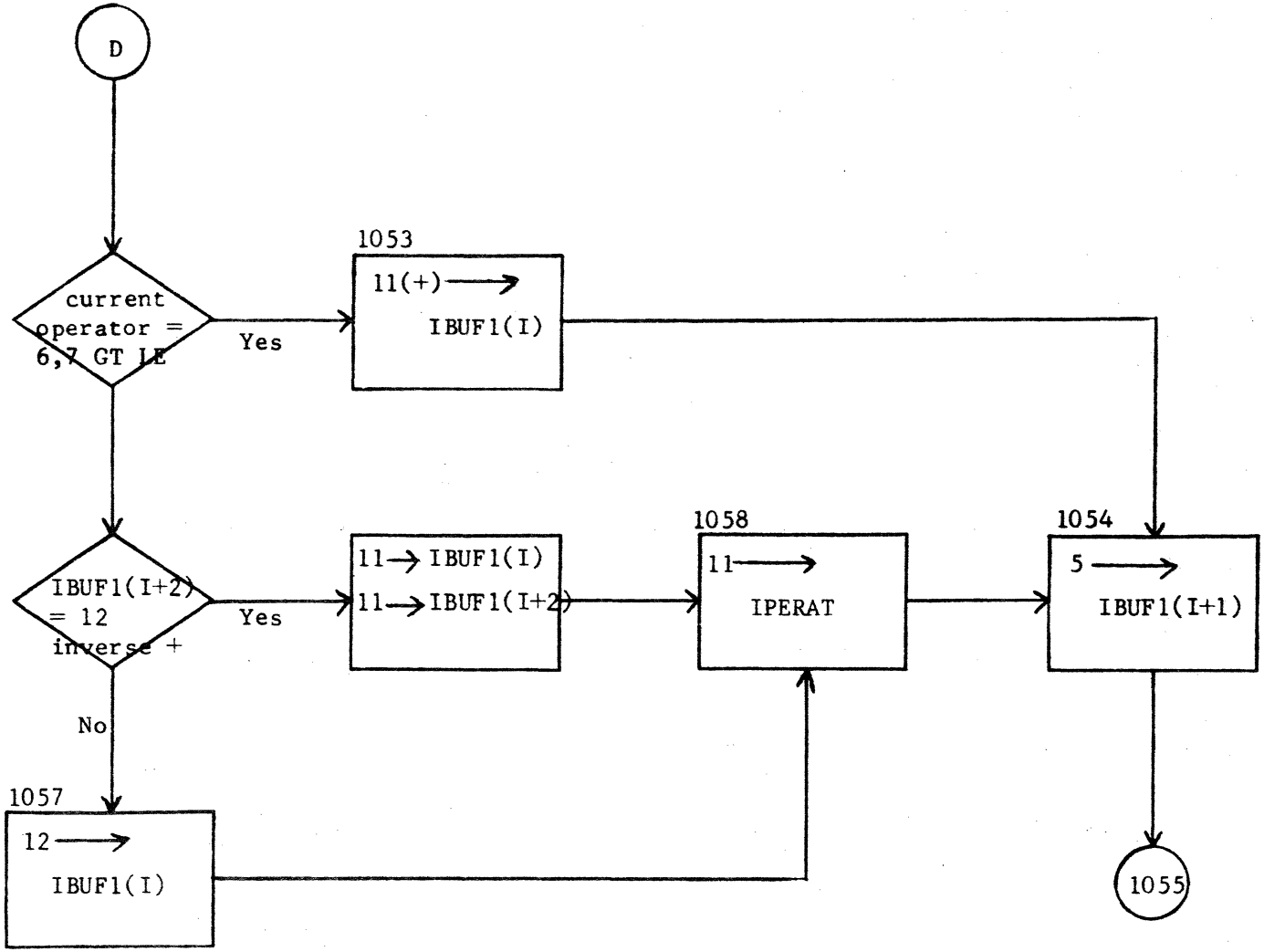
45. ISSX contains a relational operator (page 13).

46. If written, AiRL, B, change to .RL. A + B, else make it .RL. A - B.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TREE	PROJECT MGR.			
	PAGE 16 OF 24	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			

E2E 1-3

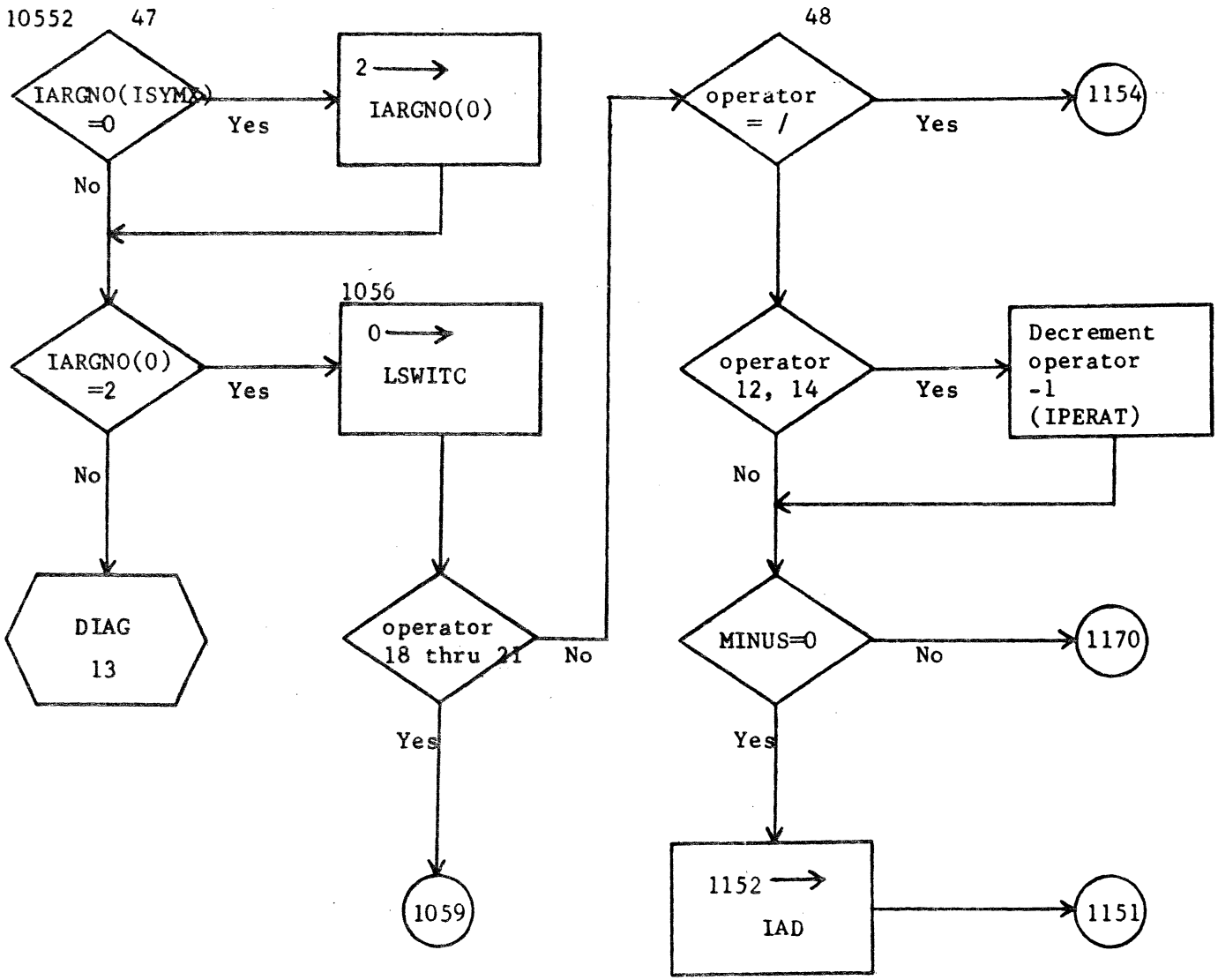


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR.		
	PAGE 17 OF 24		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

AA1385 (FORMER CA127-1)

47. Check number of parameters this call vs. number of parameters for any prior call.

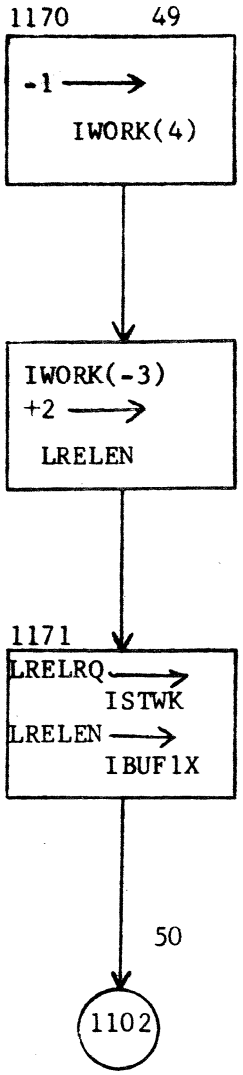
48. Operator = 1, MINUS set NZ, NOOP = 1.

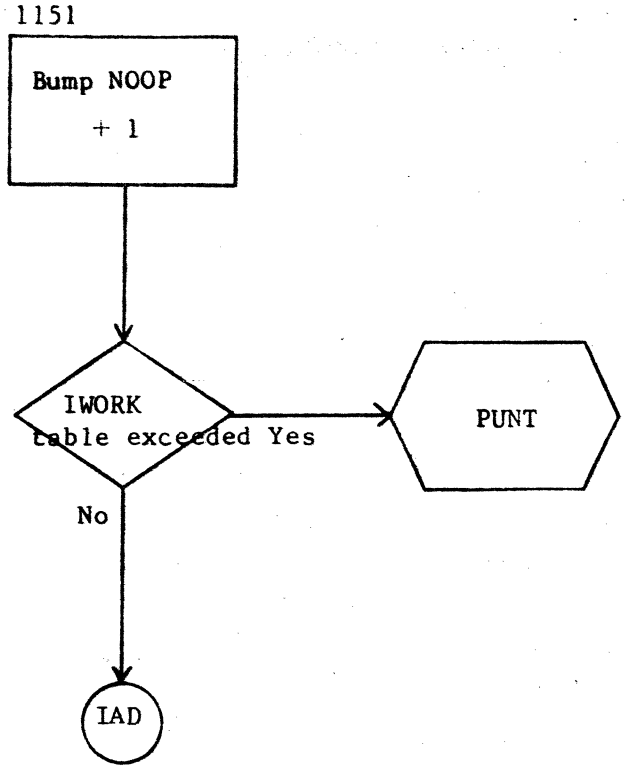


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐	DOCUMENT CLASS INS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
FLOWCHART ☒	DOCUMENT TITLE TREE		PROJECT MGR.		
DECISION TABLE ☐		PAGE 18 OF 24	PROJECT NAME		
OTHER ☐	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

49. NOT operator set up in IWORK but not in IBUF2.

50. Back to start.





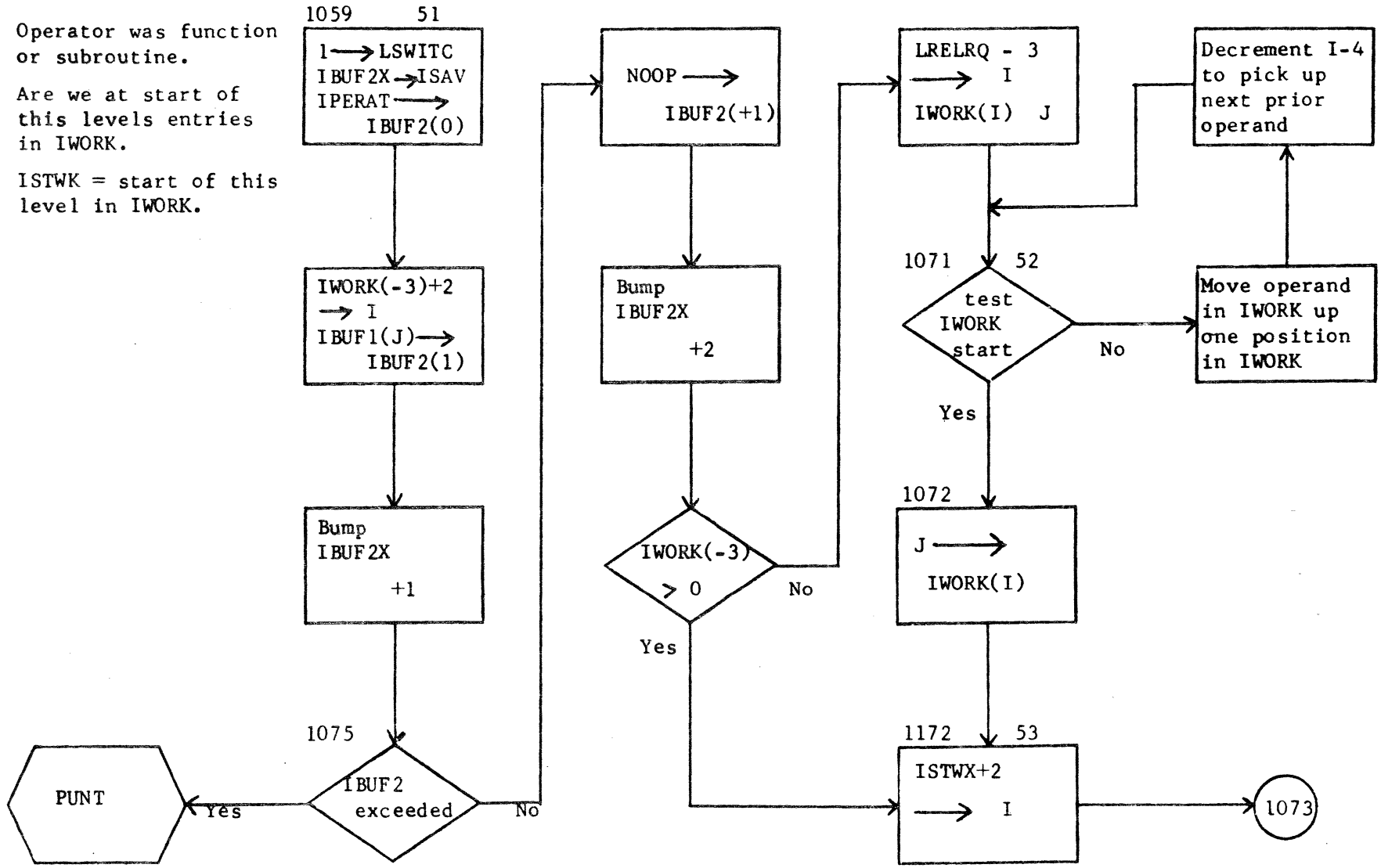
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TAEE		PROJECT MGR.		
		PAGE 19 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

1 2 3 4 5

51. Operator was function or subroutine.

52. Are we at start of this levels entries in IWORK.

53. ISTWK = start of this level in IWORK.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TRACE		PROJECT MGR.		
		PAGE 20 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

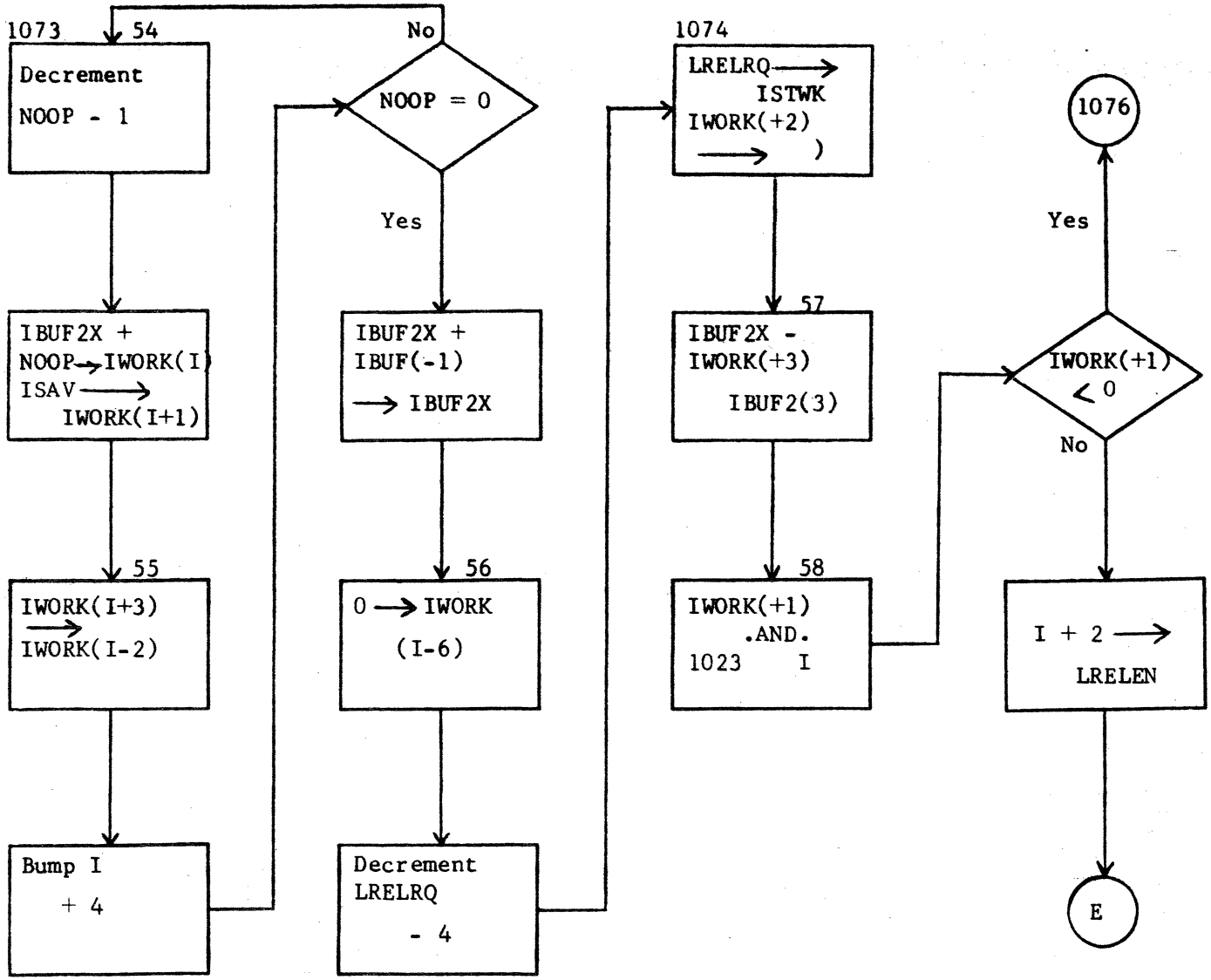
54. Put in IBUF2, settings for operand + operator in IWORK.

55. Put in stop pointer.

56. Clear out last stop pointer.

57. J = location of PTR in IBUF2.

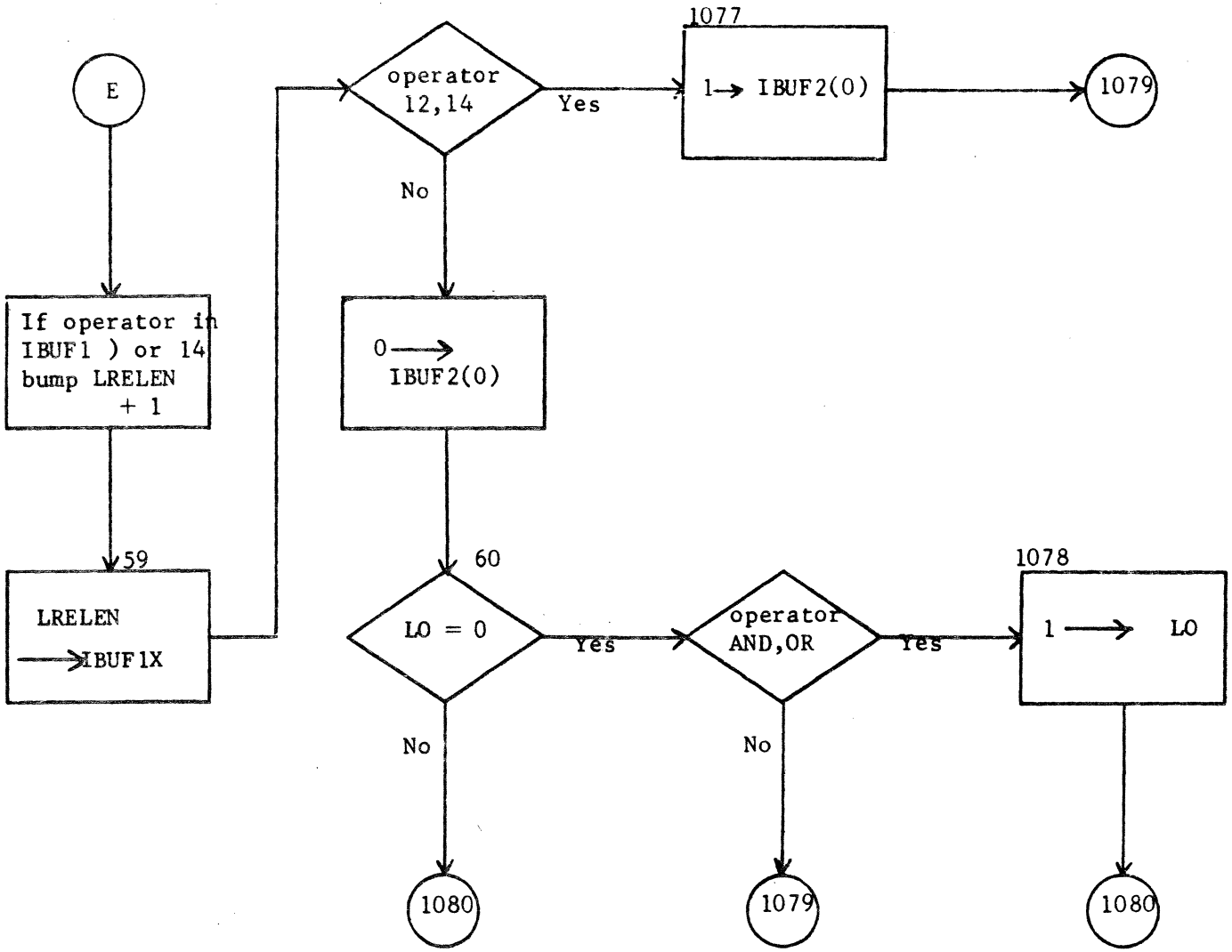
58. Set new starting item.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TREE		PROJECT MGR.			
		PAGE 21 OF 24	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 3-67	TASK NAME			

59. Set new IBUF1 index.

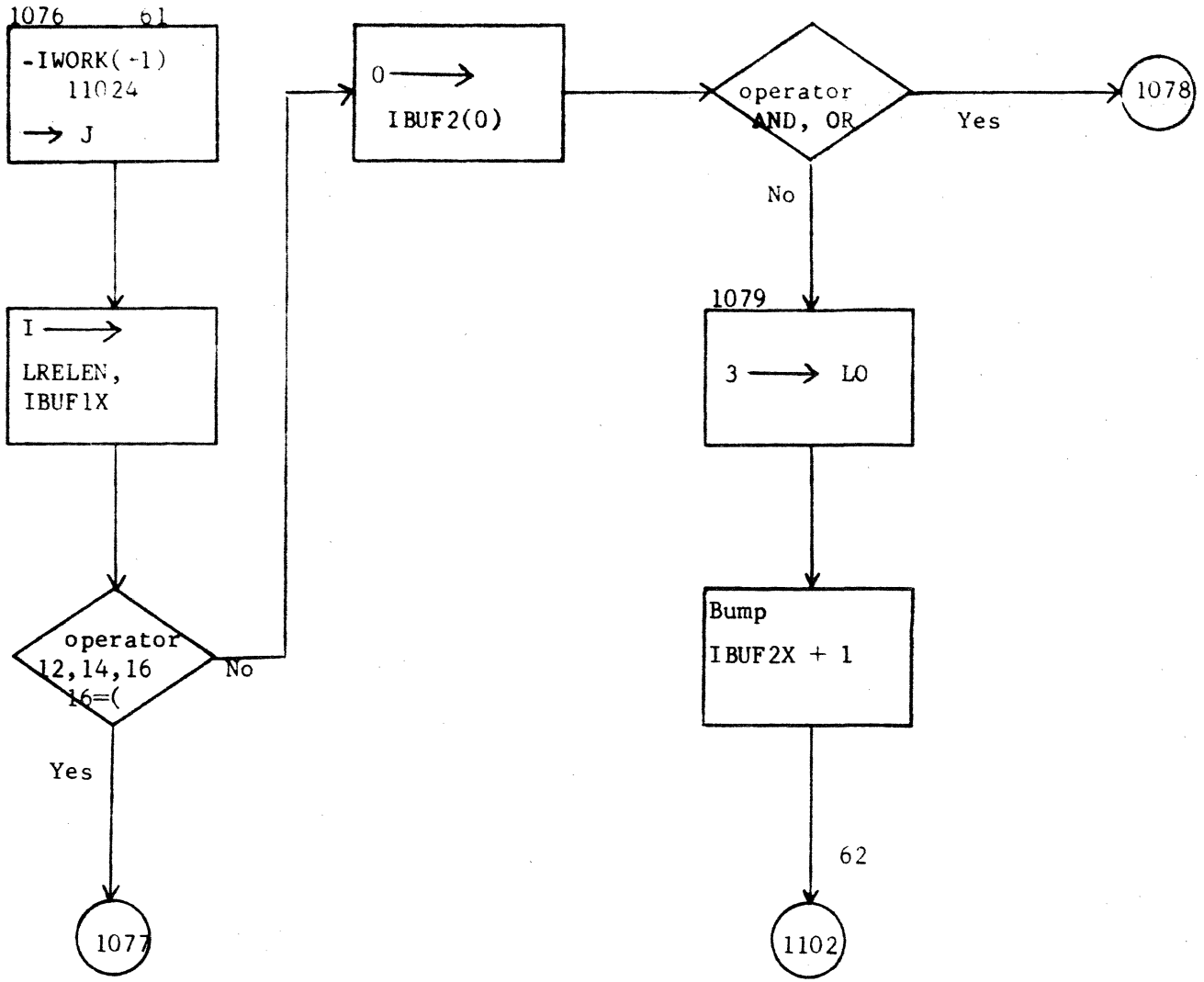
60. LO = class of prior operand.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TREE	PROJECT MGR.			
	PAGE 22 OF 24	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			

61. Shift right 10 positions.

62. Back to beginning.



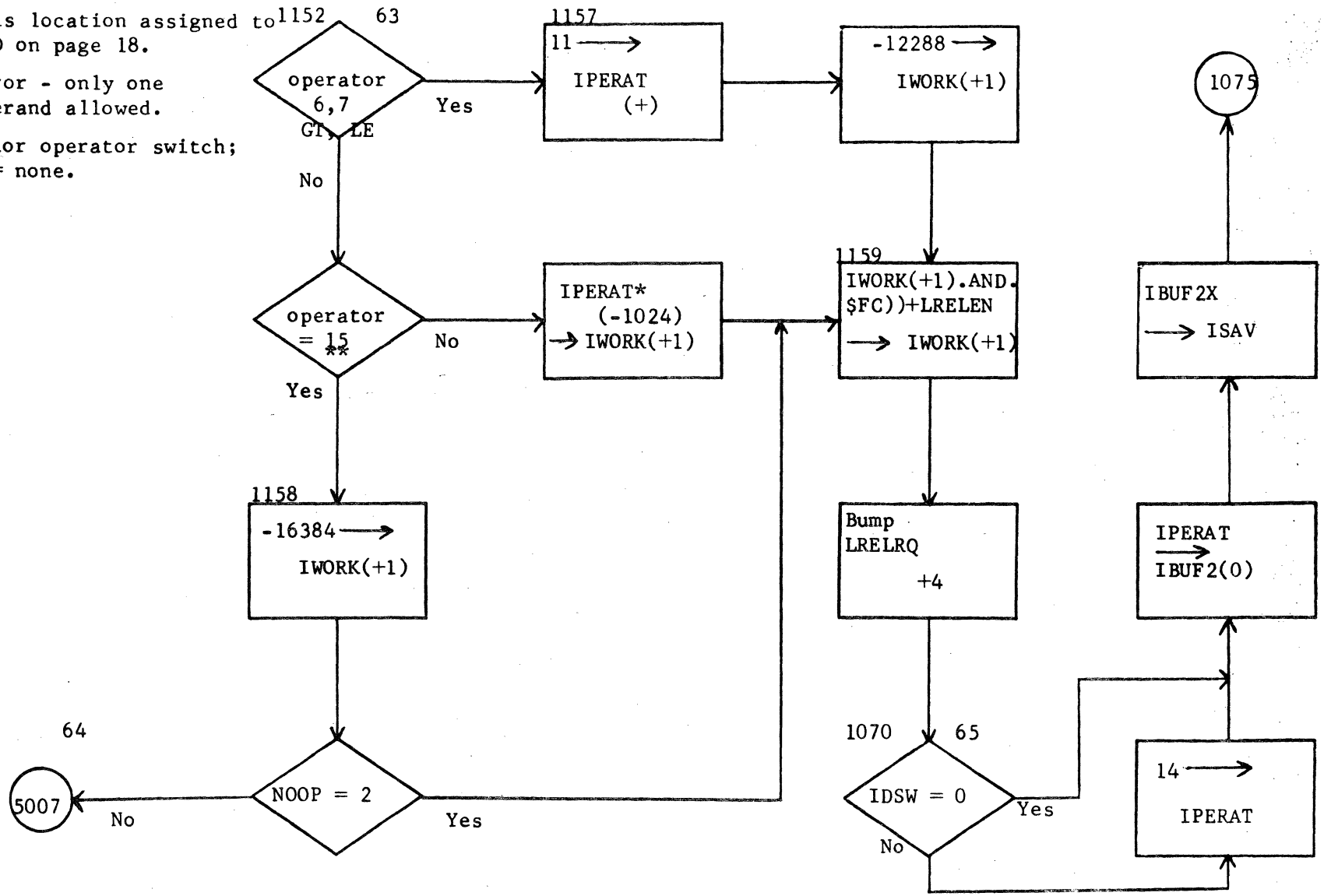
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR.		
		PAGE 23 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

AA1385 (FORMERLY CA127-1)

63. This location assigned to IAD on page 18.

64. Error - only one operand allowed.

65. Prior operator switch; 0 = none.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TREE		PROJECT MGR.		
	PAGE 24 OF 24		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-69	TASK NAME		

3.4.4 Subroutine MODMXR

This routine is called from TREE only if a mixed mode expression is being processed. It transforms the tree form generated in IBUF2 to indicate calls to function operators as necessary. Mixed mode expressions and their related calls are as follows:

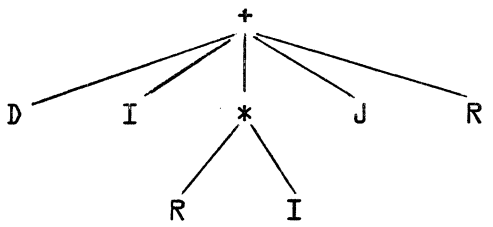
INTEGER - REAL	call	FLOAT
INTEGER - DOUBLE PRECISION	call	DFLT
REAL - DOUBLE PRECISION	call	DBLE

Please refer to section 3.4.3 for a description of the tree form generated by TREE.

One or two lower levels may be inserted in the tree at each mixed mode level. If the mixed mode expression is REAL and there is more than one integer branch on the same level, these branches are grouped so that the integer operation is performed before the call to FLOAT. If the mixed mode expression is DOUBLE PRECISION and there is more than one integer branch or more than one real branch on the same level, these branches are grouped respectively so that the integer operation is performed before the call to DFLT and the real operation is performed before the call to DBLE. Integer divides are not reorderable. The call to a function becomes the operator of a single branch on the previously mixed mode level.

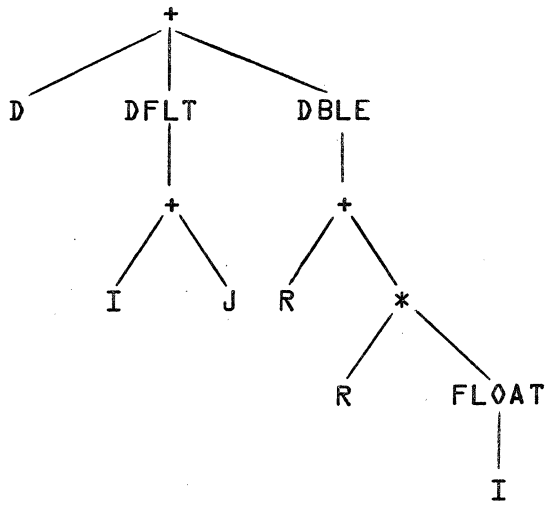
For example: D is a double precision variable
R is a real variable
I is an integer variable

The expression: D + I + R*I + J + R
generates the mixed mode tree.

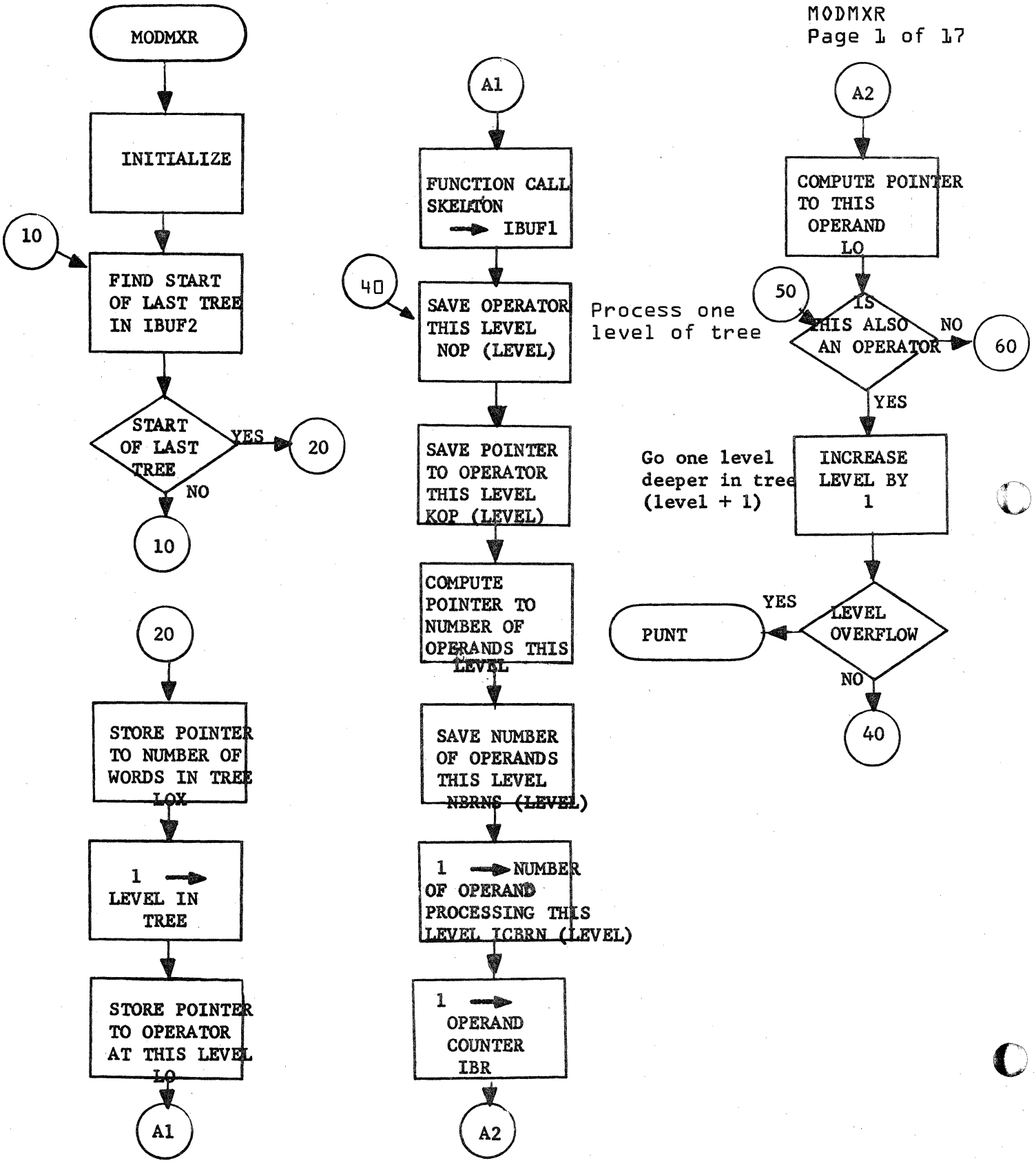


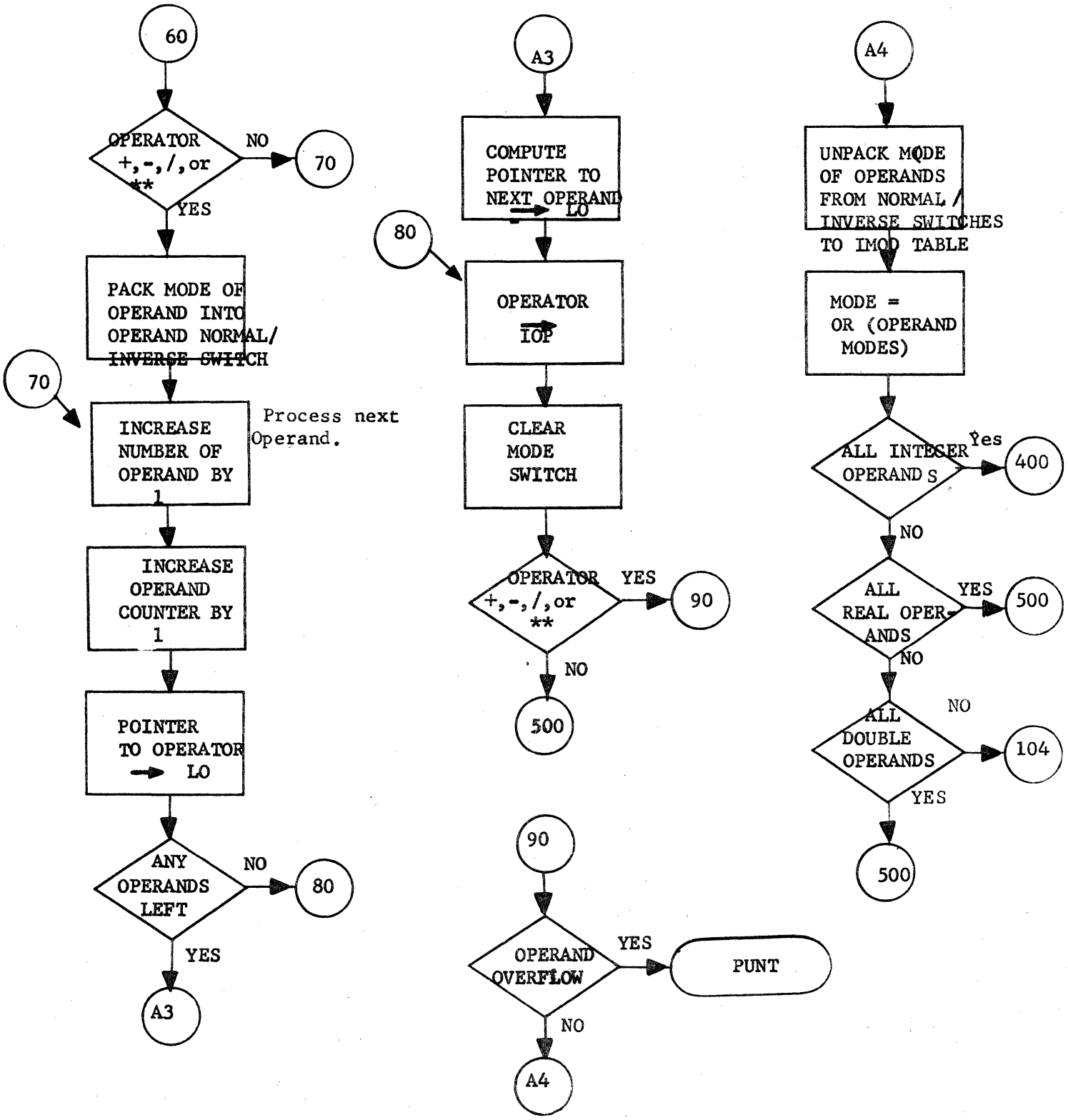
Level 1
Level 2

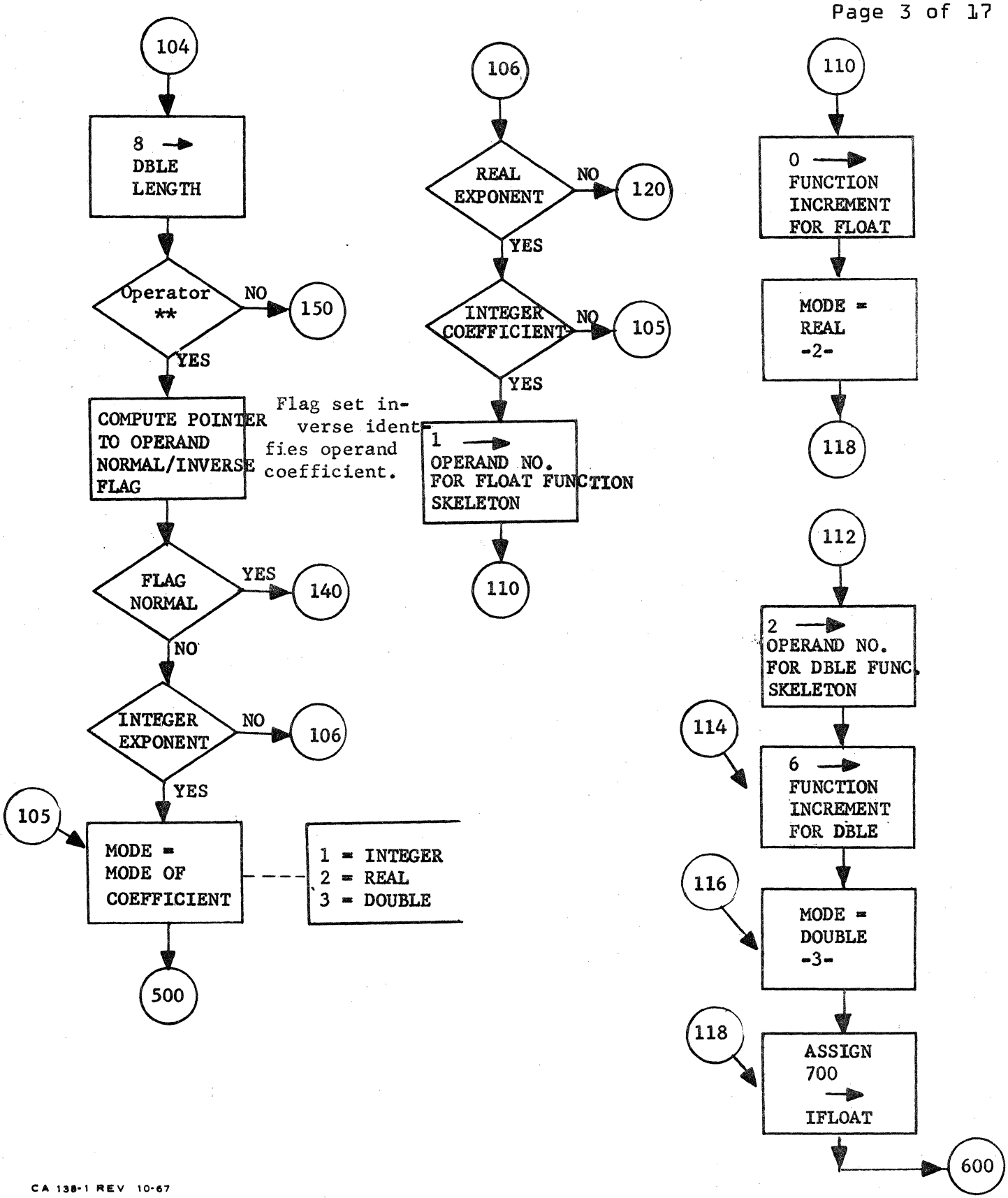
MODMXR transforms this tree into the following form:

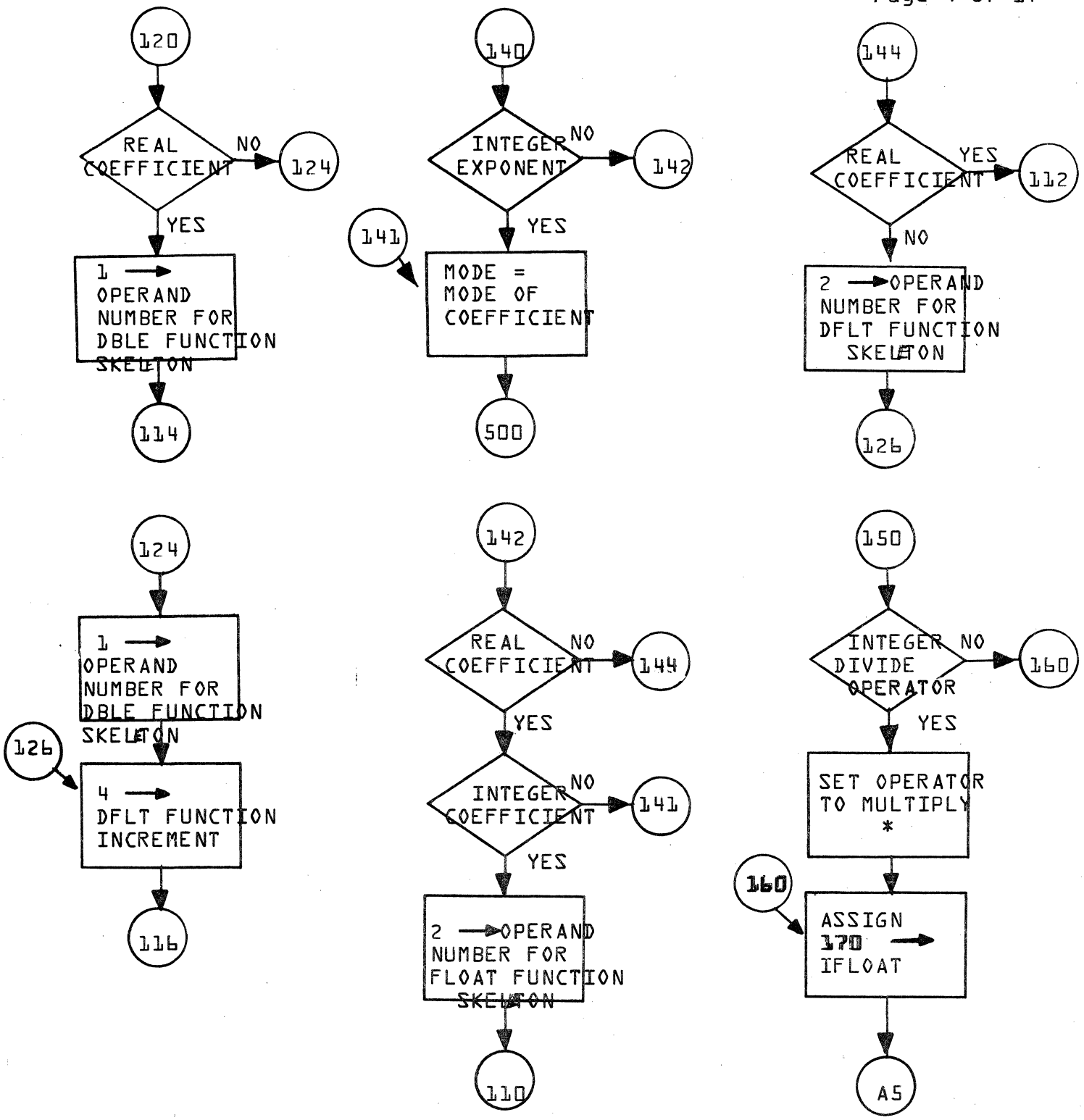


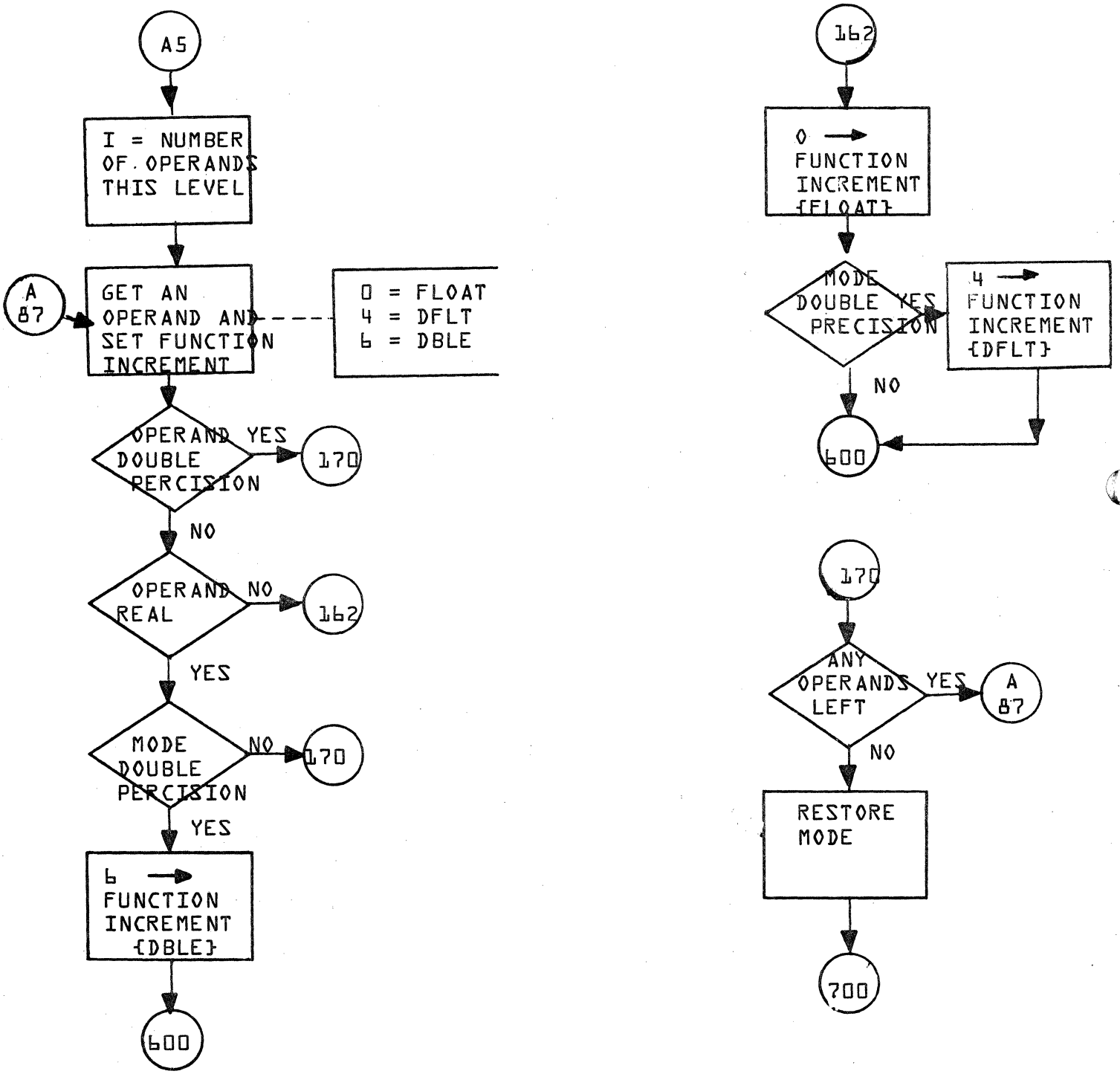
Level 1
Level 2
Level 3
Level 4
Level 5

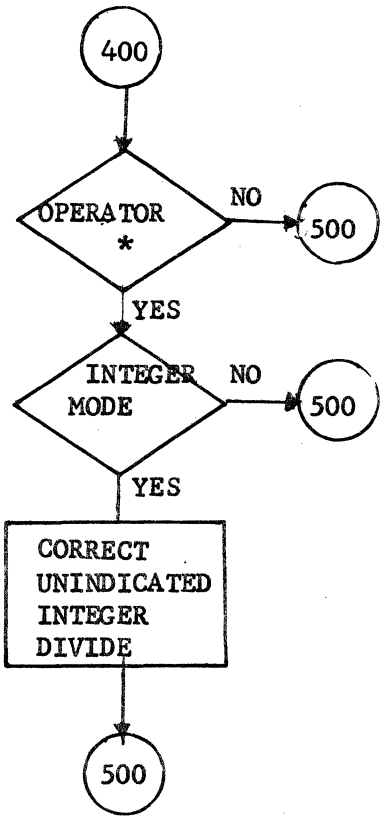

MODMXR
Page 1 of 17
MODMXR
INITIALIZE
10
FIND START OF LAST TREE IN IBUF2
START OF LAST TREE
YES
20
NO
10
20
STORE POINTER TO NUMBER OF WORDS IN TREE LOX
1 → LEVEL IN TREE
STORE POINTER TO OPERATOR AT THIS LEVEL LO
A1
A1
FUNCTION CALL SKELETON → IBUF1
40
SAVE OPERATOR THIS LEVEL NOP (LEVEL)
SAVE POINTER TO OPERATOR THIS LEVEL KOP (LEVEL)
COMPUTE POINTER TO NUMBER OF OPERANDS THIS LEVEL
SAVE NUMBER OF OPERANDS THIS LEVEL NBRNS (LEVEL)
1 → NUMBER OF OPERAND PROCESSING THIS LEVEL ICBRN (LEVEL)
1 → OPERAND COUNTER IBR
A2
Process one level of tree
A2
COMPUTE POINTER TO THIS OPERAND LO
50
IS THIS ALSO AN OPERATOR
NO
60
YES
Go one level deeper in tree (level + 1)
INCREASE LEVEL BY 1
YES
PUNT
LEVEL OVERFLOW
NO
40

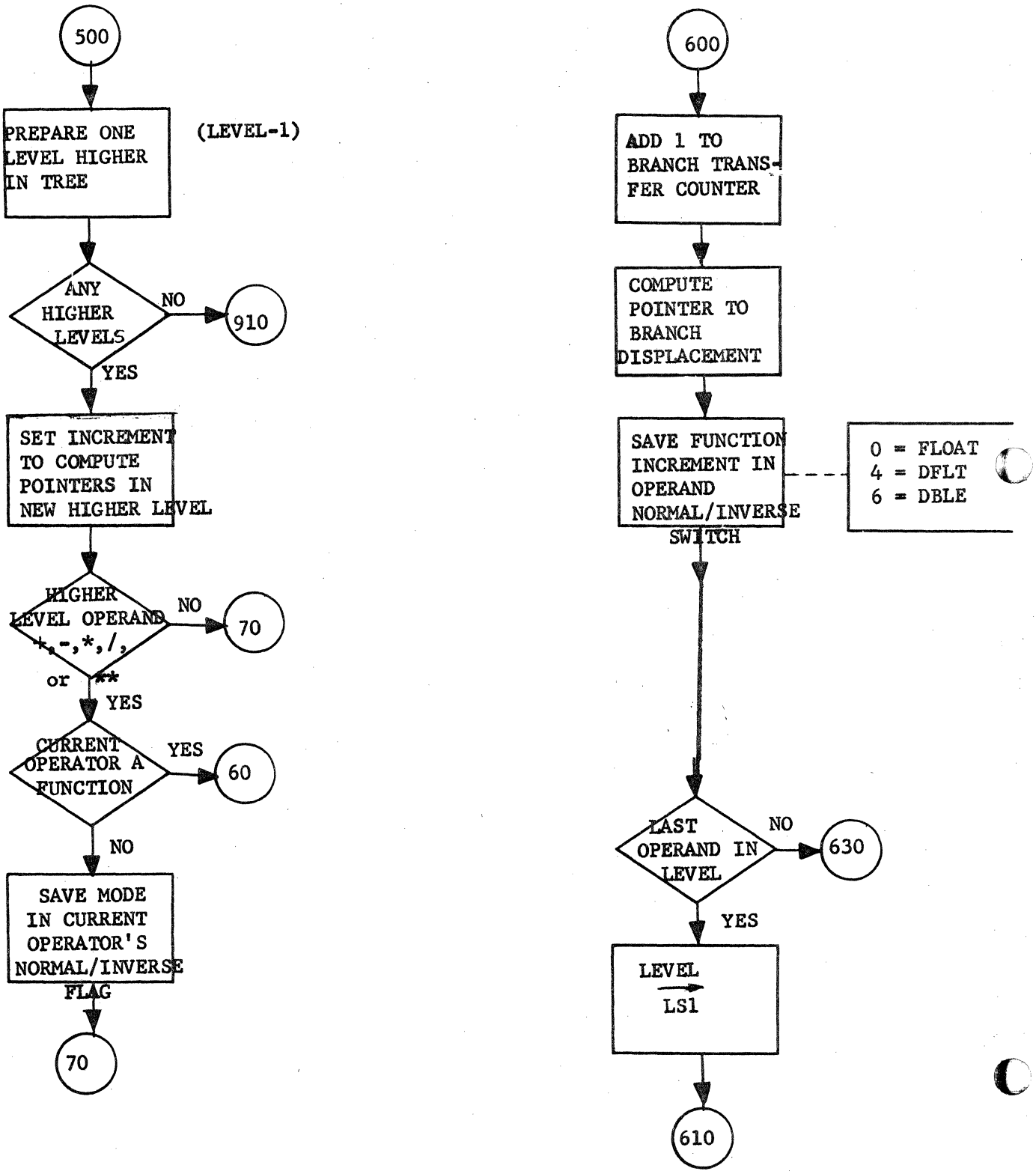

60
OPERATOR +,-,/,or **
NO
70
YES
PACK MODE OF OPERAND INTO OPERAND NORMAL/ INVERSE SWITCH
70
INCREASE NUMBER OF OPERAND BY 1
Process next Operand.
INCREASE OPERAND COUNTER BY 1
POINTER TO OPERATOR → LO
ANY OPERANDS LEFT
NO
80
YES
A3
A3
COMPUTE POINTER TO NEXT OPERAND → LO
80
OPERATOR → IOP
CLEAR MODE SWITCH
OPERATOR +,-,/,or **
YES
90
NO
500
90
OPERAND OVERFLOW
YES
PUNT
NO
A4
A4
UNPACK MODE OF OPERANDS FROM NORMAL / INVERSE SWITCHES TO IMOD TABLE
MODE = OR (OPERAND MODES)
ALL INTEGER OPERANDS
Yes
400
NO
ALL REAL OPERANDS
YES
500
NO
ALL DOUBLE OPERANDS
NO
104
YES
500

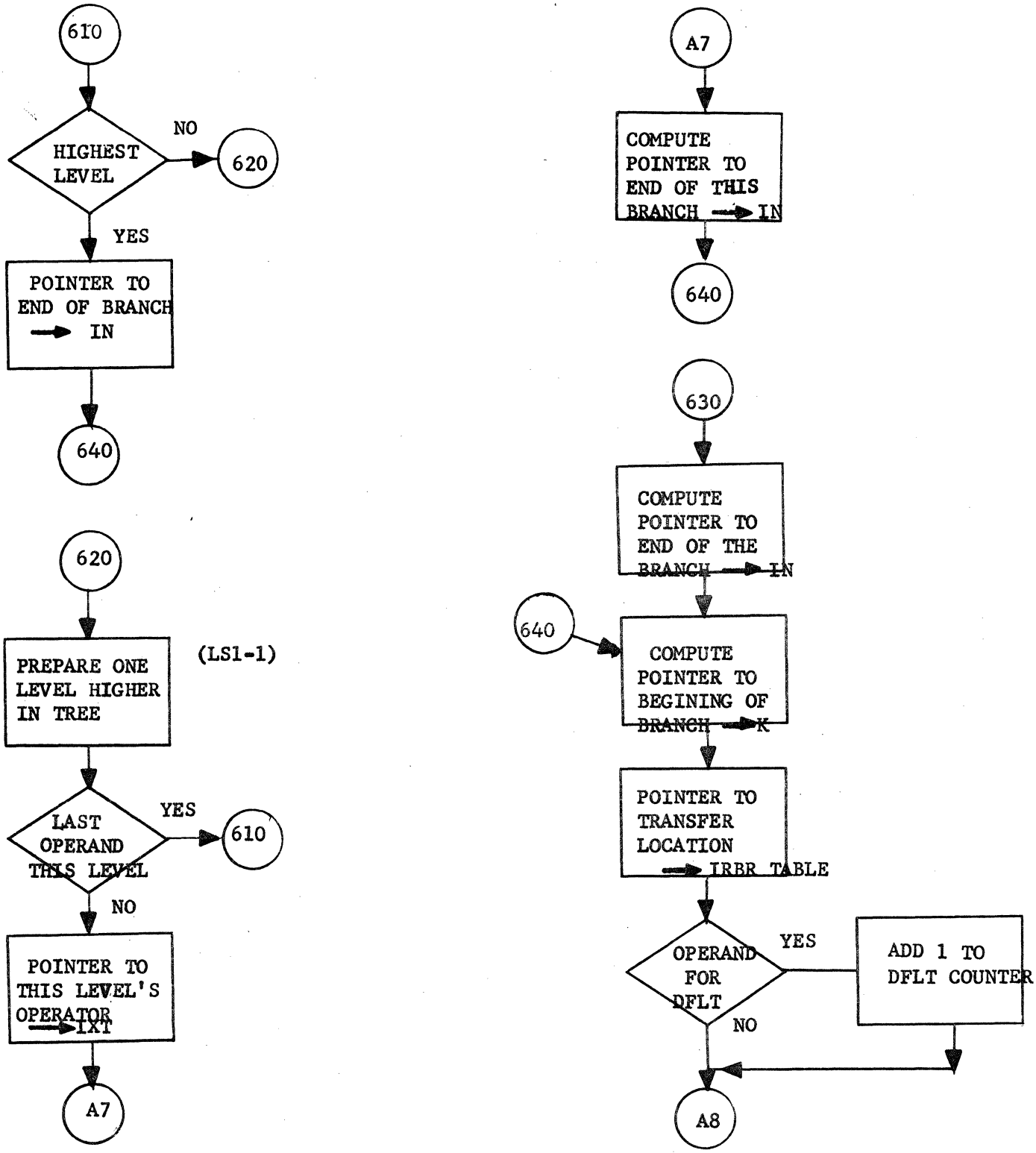

104
8 → DBLE LENGTH
Operator **
NO
150
YES
COMPUTE POINTER TO OPERAND NORMAL/INVERSE FLAG
Flag set inverse identifies operand coefficient.
FLAG NORMAL
YES
140
NO
INTEGER EXPONENT
NO
106
YES
105
MODE = MODE OF COEFFICIENT
1 = INTEGER
2 = REAL
3 = DOUBLE
500
106
REAL EXPONENT
NO
120
YES
INTEGER COEFFICIENT
NO
105
YES
1 → OPERAND NO. FOR FLOAT FUNCTION SKELETON
110
110
0 → FUNCTION INCREMENT FOR FLOAT
MODE = REAL -2-
118
112
2 → OPERAND NO. FOR DBLE FUNC. SKELETON
114
6 → FUNCTION INCREMENT FOR DBLE
116
MODE = DOUBLE -3-
118
ASSIGN 700 → IFLOAT
600

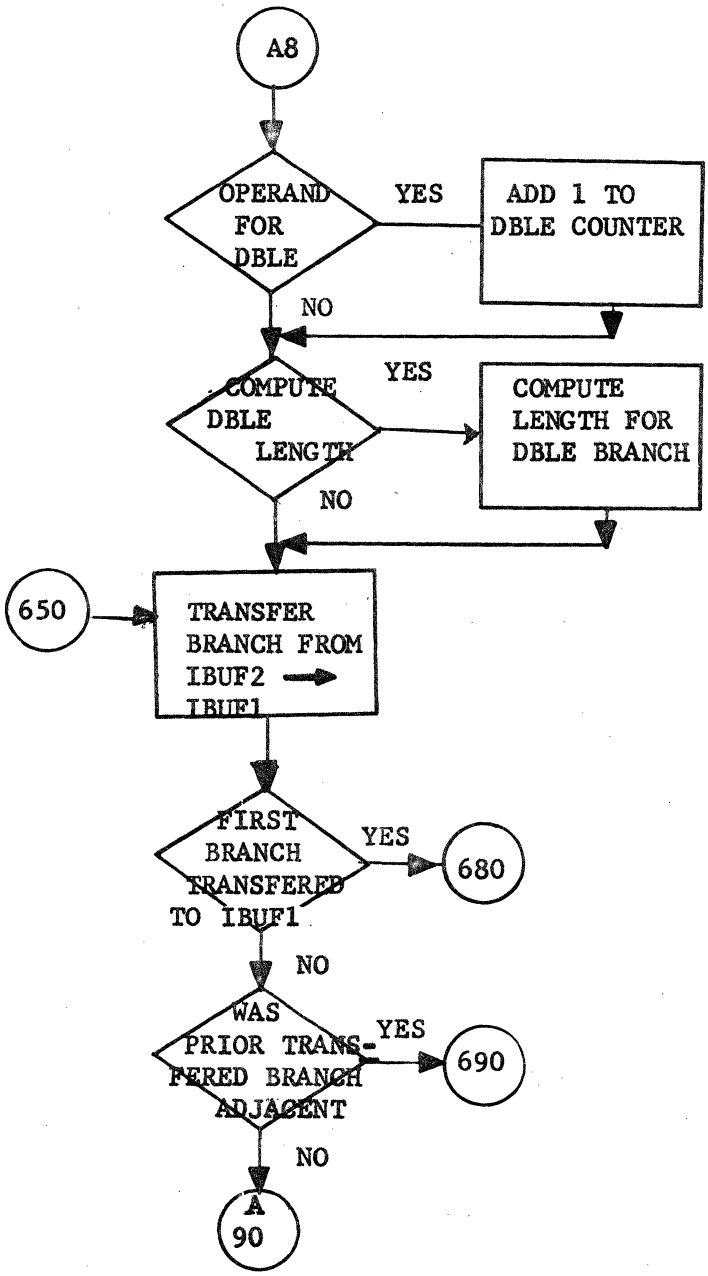

120
REAL COEFFICIENT
NO
124
YES
1 → OPERAND NUMBER FOR DBLE FUNCTION SKELETON
114
140
INTEGER EXPONENT
NO
142
YES
141
MODE = MODE OF COEFFICIENT
500
144
REAL COEFFICIENT
YES
112
NO
2 → OPERAND NUMBER FOR DFLT FUNCTION SKELETON
126
124
1 → OPERAND NUMBER FOR DBLE FUNCTION SKELETON
126
4 → DFLT FUNCTION INCREMENT
116
142
REAL COEFFICIENT
NO
144
YES
INTEGER COEFFICIENT
NO
141
YES
2 → OPERAND NUMBER FOR FLOAT FUNCTION SKELETON
110
150
INTEGER DIVIDE OPERATOR
NO
160
YES
SET OPERATOR TO MULTIPLY *
160
ASSIGN 170 → IFLOAT
A5

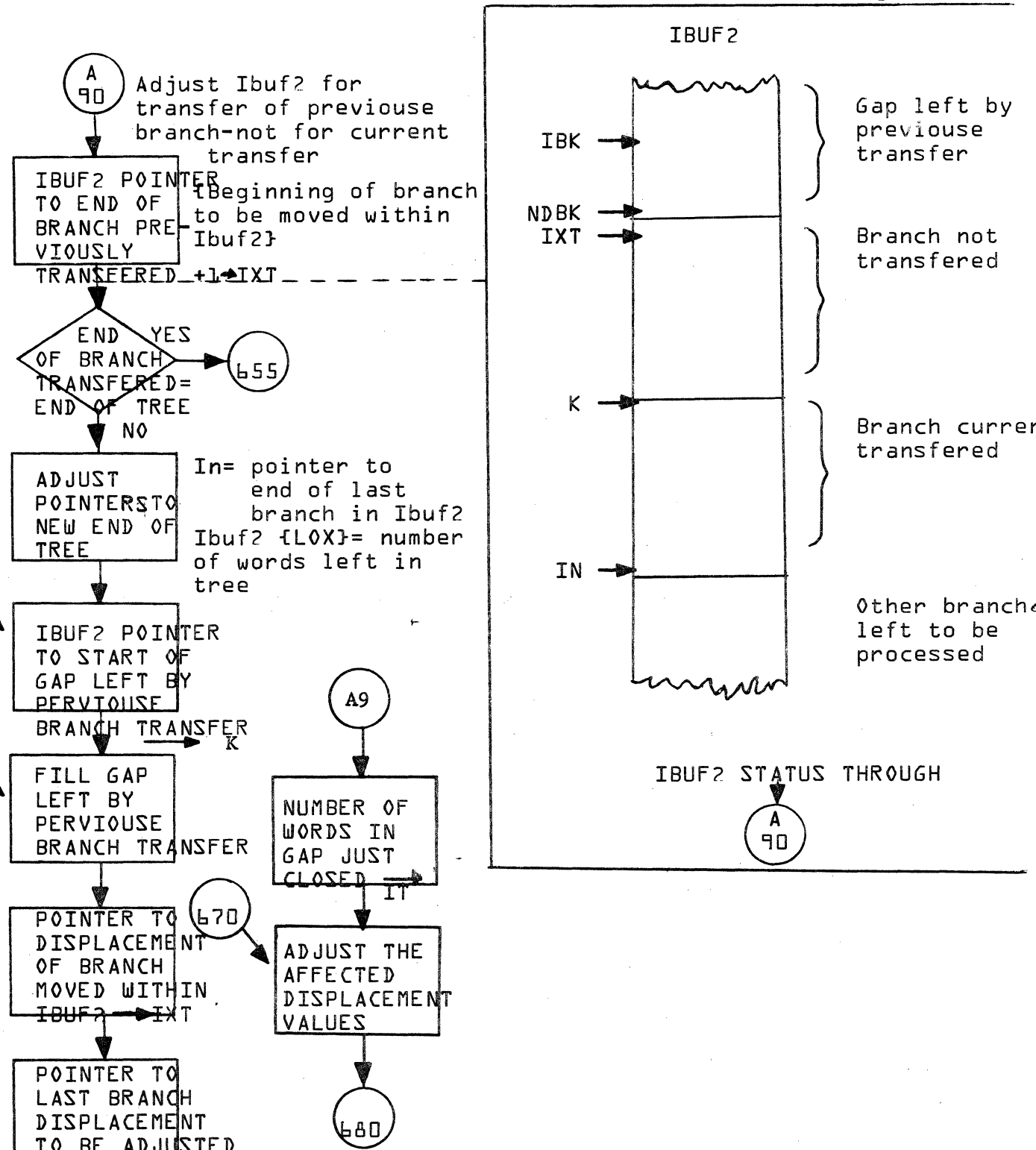

A5
I = NUMBER OF OPERANDS THIS LEVEL
A 87
GET AN OPERAND AND SET FUNCTION INCREMENT
0 = FLOAT
4 = DFLT
6 = DBLE
OPERAND DOUBLE PERCISION
YES
170
NO
OPERAND REAL
NO
162
YES
MODE DOUBLE PERCISION
NO
170
YES
6 → FUNCTION INCREMENT {DBLE}
600
162
0 → FUNCTION INCREMENT {FLOAT}
MODE DOUBLE PRECISION
YES
4 → FUNCTION INCREMENT {DFLT}
NO
600
170
ANY OPERANDS LEFT
YES
A 87
NO
RESTORE MODE
700

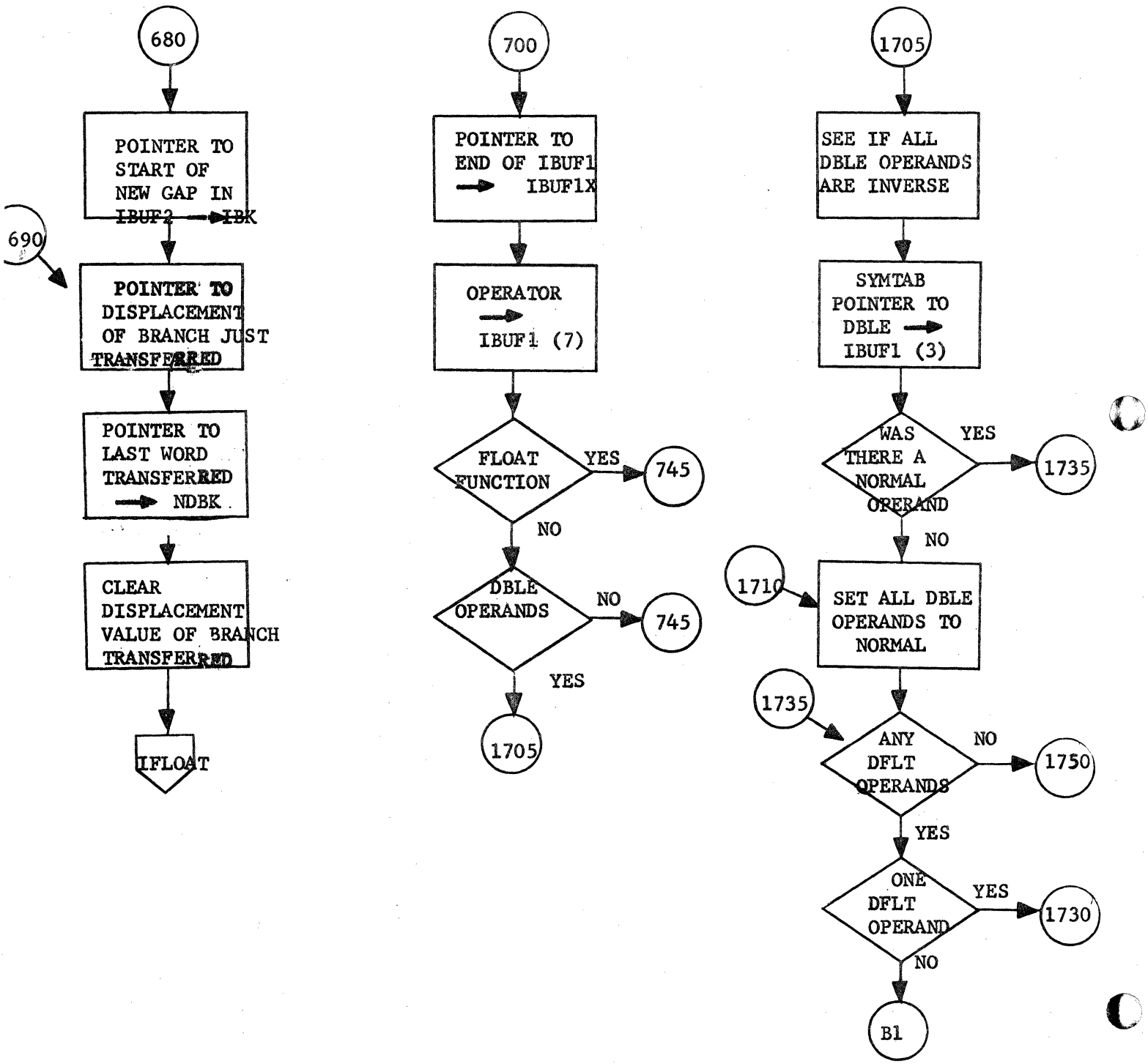

400
OPERATOR *
NO
500
YES
INTEGER MODE
NO
500
YES
CORRECT UNINDICATED INTEGER DIVIDE
500

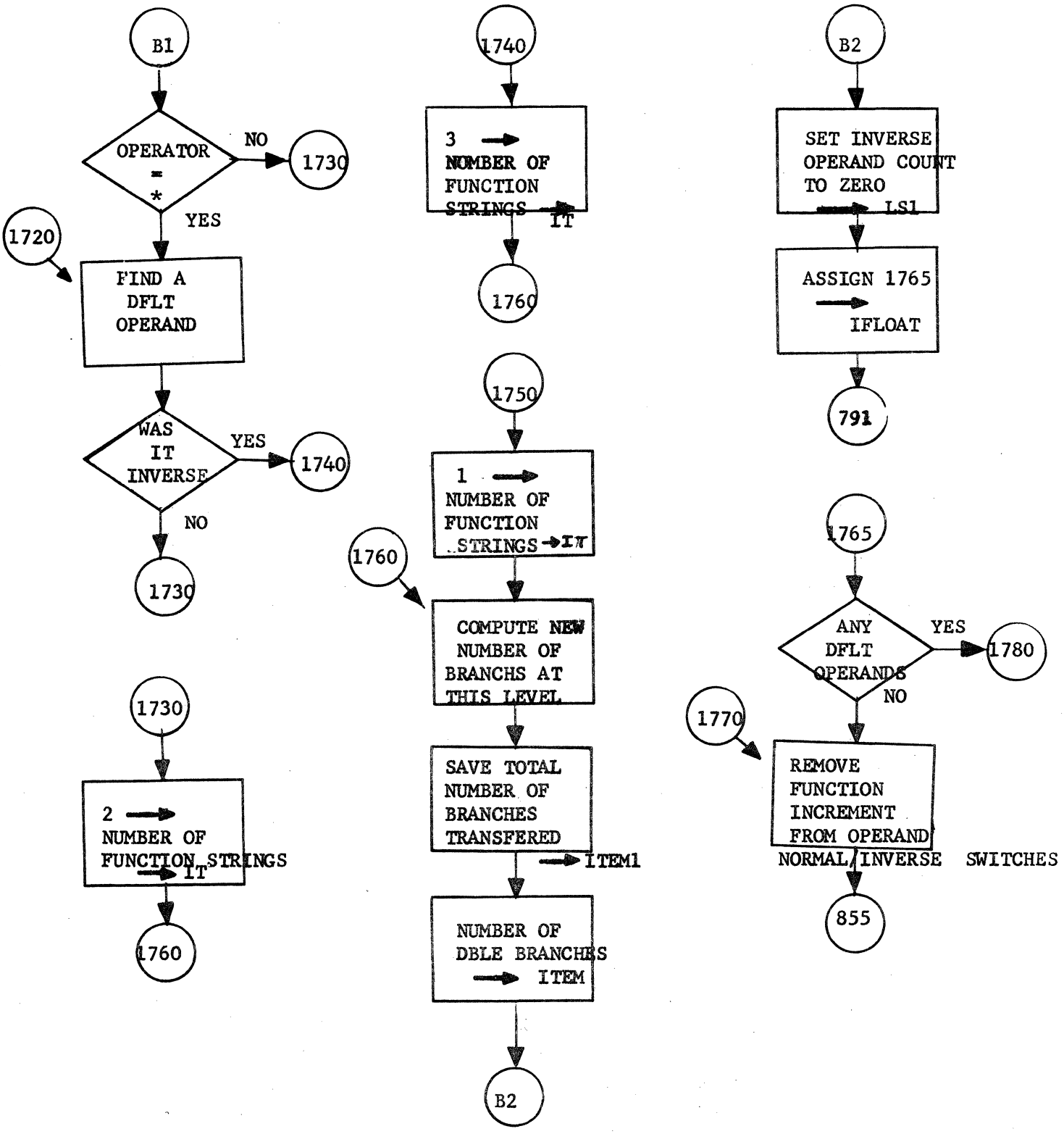

500
PREPARE ONE LEVEL HIGHER IN TREE
(LEVEL-1)
ANY HIGHER LEVELS
NO
910
YES
SET INCREMENT TO COMPUTE POINTERS IN NEW HIGHER LEVEL
HIGHER LEVEL OPERAND +,-,*,/, or **
NO
70
YES
CURRENT OPERATOR A FUNCTION
YES
60
NO
SAVE MODE IN CURRENT OPERATOR'S NORMAL/INVERSE FLAG
70
600
ADD 1 TO BRANCH TRANSFER COUNTER
COMPUTE POINTER TO BRANCH DISPLACEMENT
SAVE FUNCTION INCREMENT IN OPERAND NORMAL/INVERSE SWITCH
0 = FLOAT
4 = DFLT
6 = DBLE
LAST OPERAND IN LEVEL
NO
630
YES
LEVEL → LS1
610

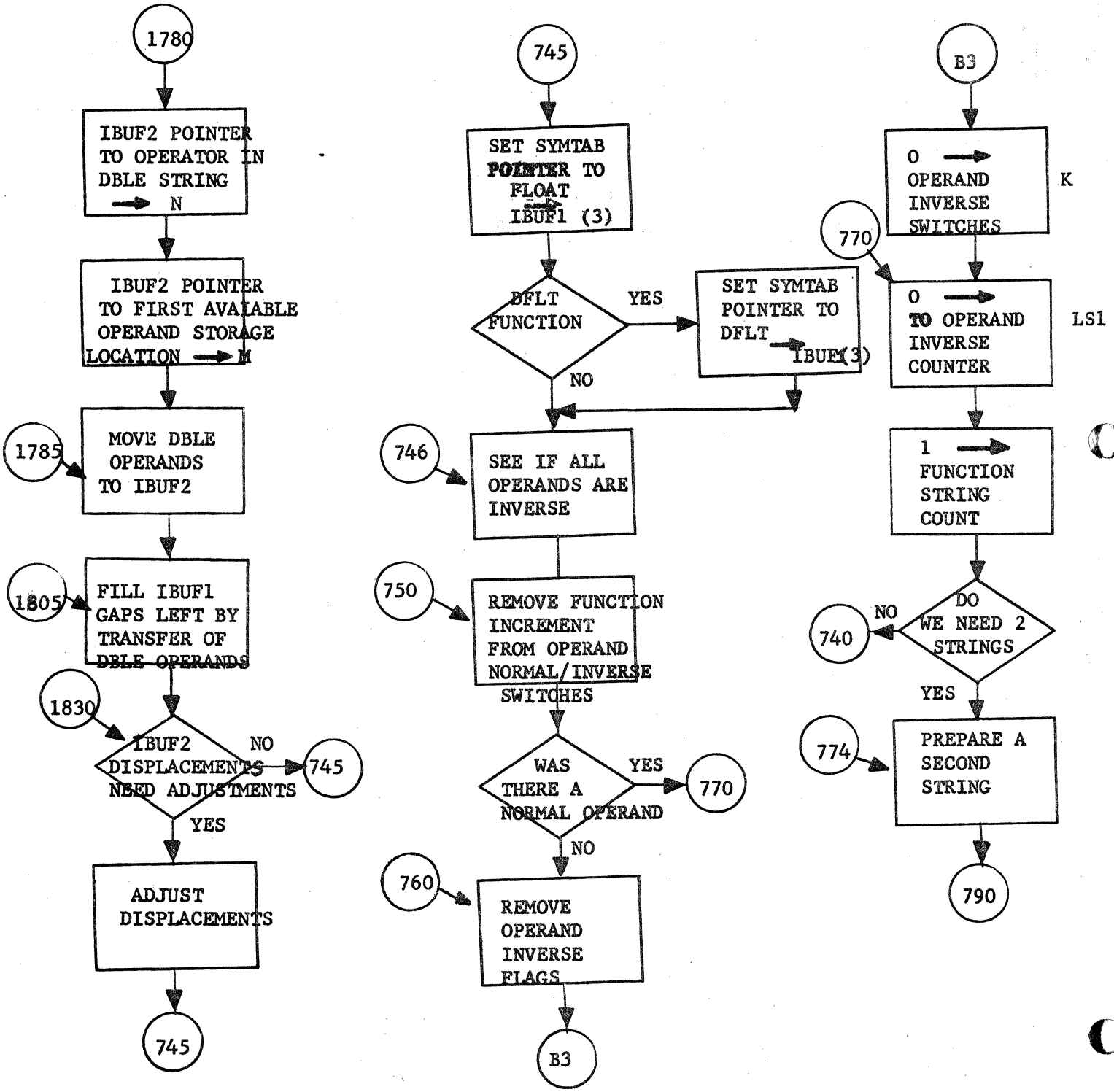

610
HIGHEST LEVEL
NO
620
YES
POINTER TO END OF BRANCH → IN
640
620
PREPARE ONE LEVEL HIGHER IN TREE
(LS1-1)
LAST OPERAND THIS LEVEL
YES
610
NO
POINTER TO THIS LEVEL'S OPERATOR → IXT
A7
A7
COMPUTE POINTER TO END OF THIS BRANCH → IN
640
630
COMPUTE POINTER TO END OF THE BRANCH → IN
640
COMPUTE POINTER TO BEGINING OF BRANCH → K
POINTER TO TRANSFER LOCATION → IRBR TABLE
OPERAND FOR DFLT
YES
ADD 1 TO DFLT COUNTER
NO
A8

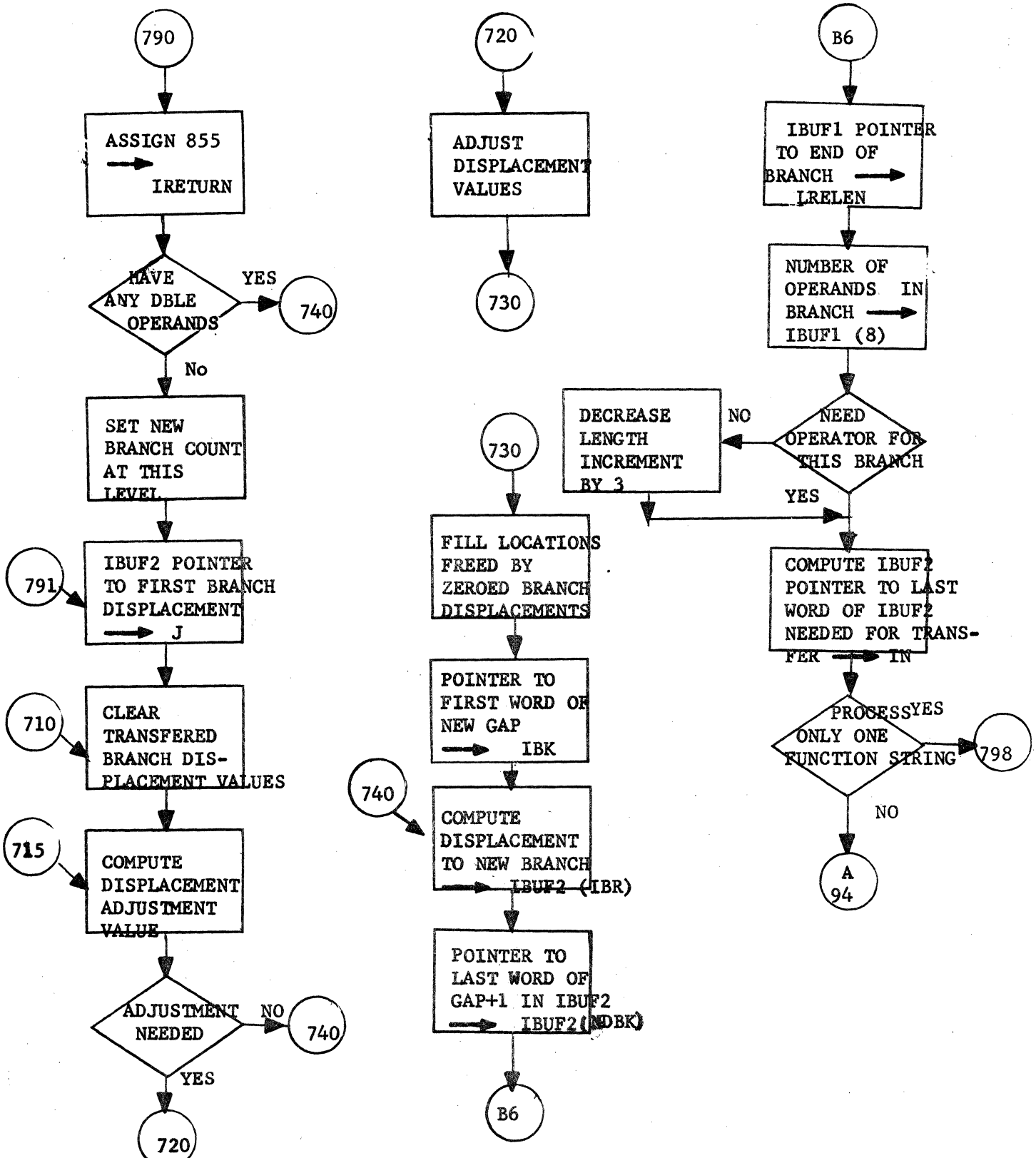

A8
OPERAND FOR DBLE
YES
ADD 1 TO DBLE COUNTER
NO
COMPUTE DBLE LENGTH
YES
COMPUTE LENGTH FOR DBLE BRANCH
NO
650
TRANSFER BRANCH FROM IBUF2 → IBUF1
FIRST BRANCH TRANSFERED TO IBUF1
YES
680
NO
WAS PRIOR TRANS-FERED BRANCH ADJACENT
YES
690
NO
A
90

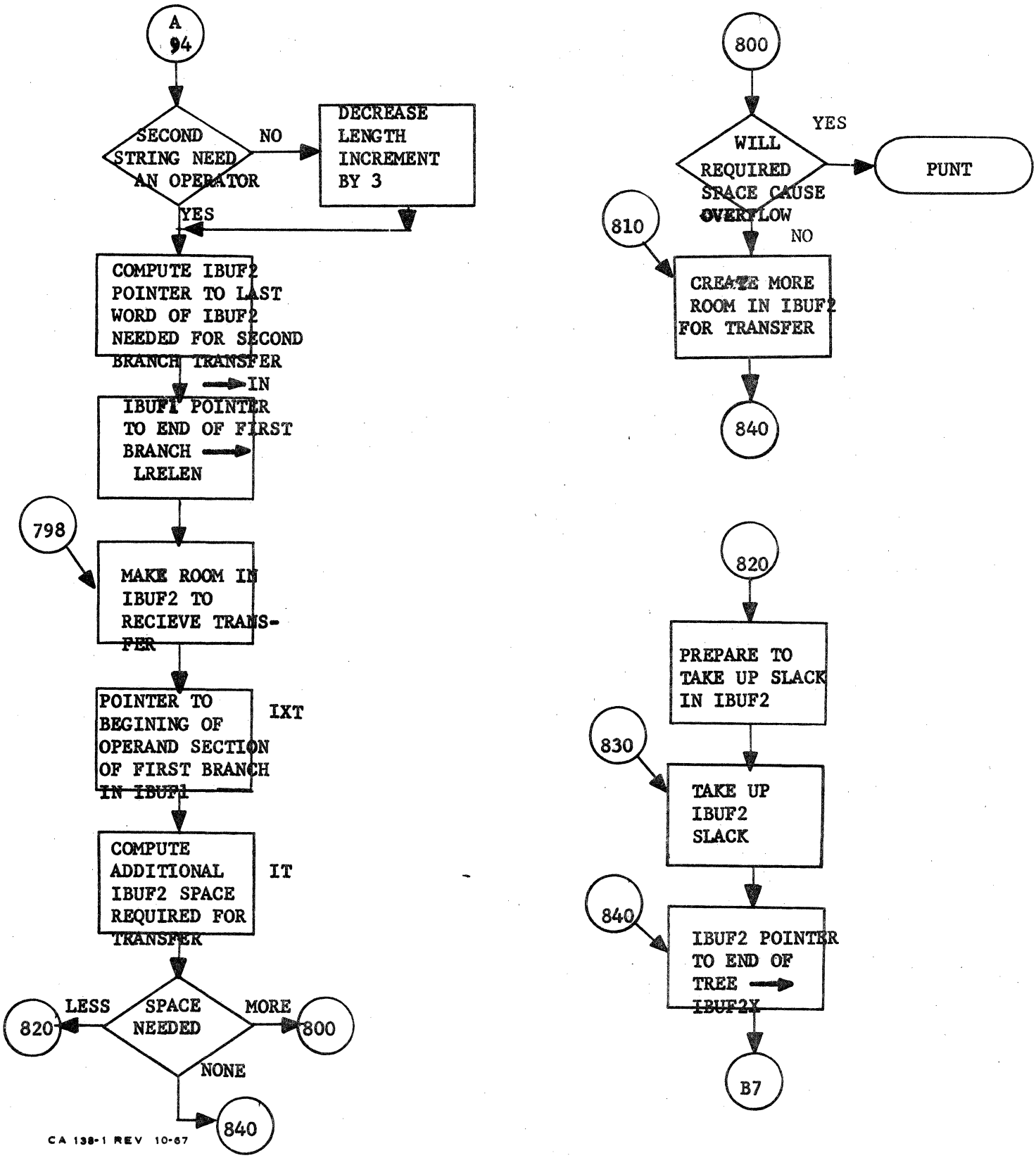

A 90
Adjust Ibuf2 for transfer of previouse branch-not for current transfer
IBUF2 POINTER TO END OF BRANCH PREVIOUSLY TRANSFERED +1→IXT
{Beginning of branch to be moved within Ibuf2}
END OF BRANCH TRANSFERED= END OF TREE
YES
655
NO
ADJUST POINTERS TO NEW END OF TREE
In= pointer to end of last branch in Ibuf2
Ibuf2 {LOX}= number of words left in tree
655
IBUF2 POINTER TO START OF GAP LEFT BY PERVIOUSE BRANCH TRANSFER →K
660
FILL GAP LEFT BY PERVIOUSE BRANCH TRANSFER
A9
NUMBER OF WORDS IN GAP JUST CLOSED →IT
670
POINTER TO DISPLACEMENT OF BRANCH MOVED WITHIN IBUF2 →IXT
ADJUST THE AFFECTED DISPLACEMENT VALUES
680
POINTER TO LAST BRANCH DISPLACEMENT TO BE ADJUSTED →S1
A9
IBUF2
IBK
NDBK
IXT
K
IN
Gap left by previouse transfer
Branch not transfered
Branch currently transfered
Other branches left to be processed
IBUF2 STATUS THROUGH
A 90

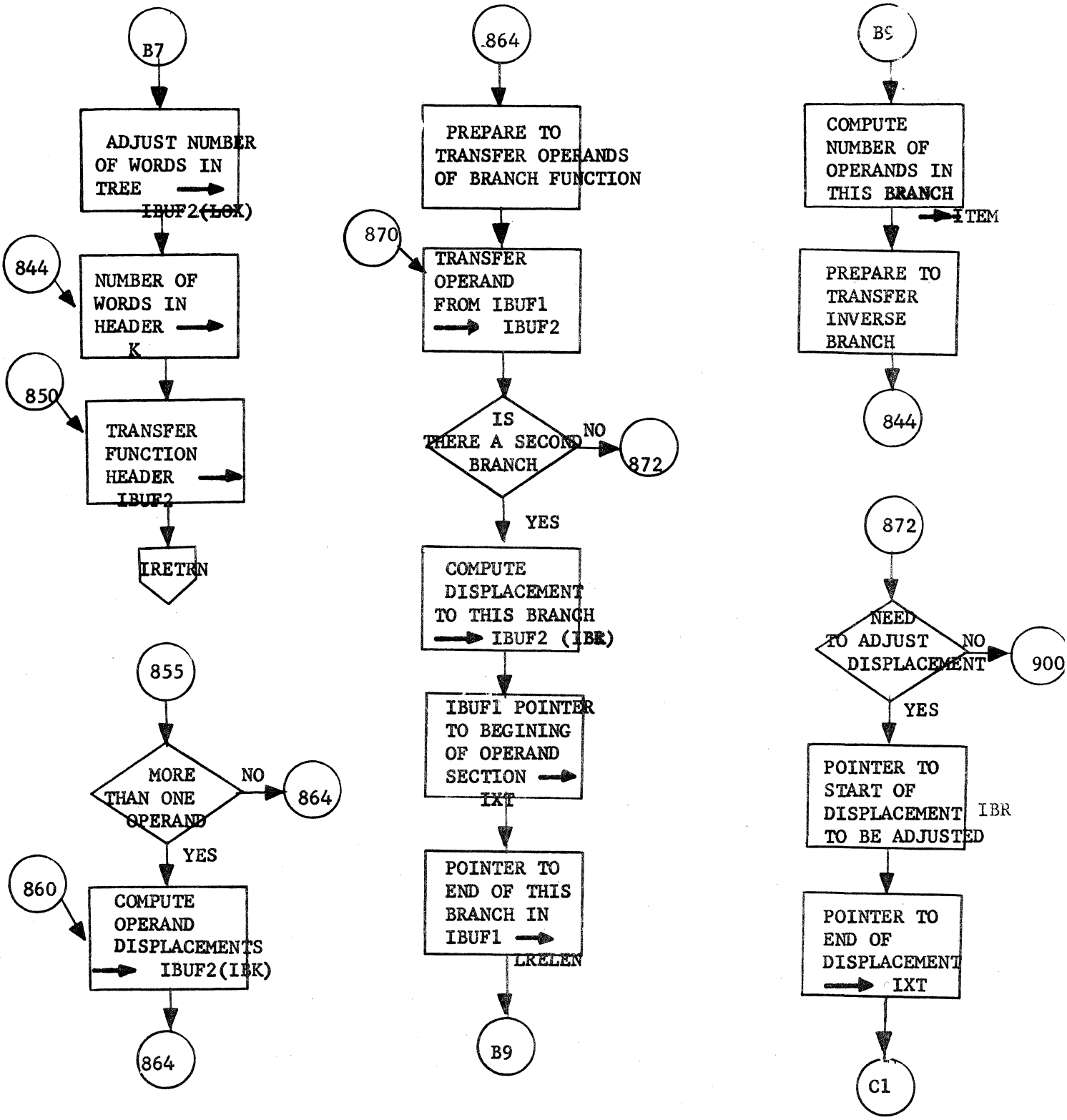

680
POINTER TO START OF NEW GAP IN IBUF2 → IBK
690
POINTER TO DISPLACEMENT OF BRANCH JUST TRANSFERRED
POINTER TO LAST WORD TRANSFERRED → NDBK
CLEAR DISPLACEMENT VALUE OF BRANCH TRANSFERRED
IFLOAT
700
POINTER TO END OF IBUF1 → IBUF1X
OPERATOR → IBUF1 (7)
FLOAT FUNCTION
YES
745
NO
DBLE OPERANDS
NO
745
YES
1705
1705
SEE IF ALL DBLE OPERANDS ARE INVERSE
SYMTAB POINTER TO DBLE → IBUF1 (3)
WAS THERE A NORMAL OPERAND
YES
1735
NO
1710
SET ALL DBLE OPERANDS TO NORMAL
1735
ANY DFLT OPERANDS
NO
1750
YES
ONE DFLT OPERAND
YES
1730
NO
B1

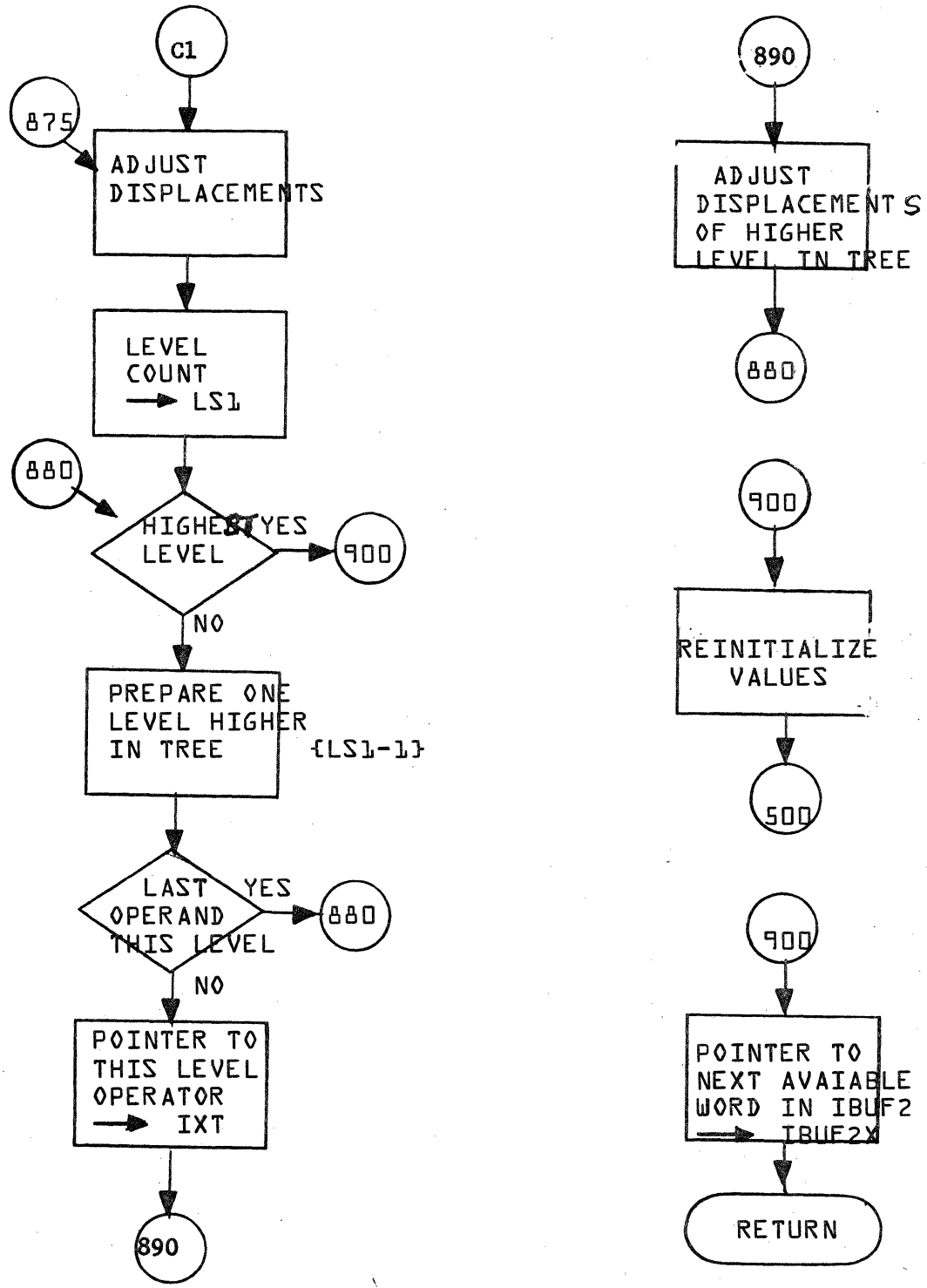

B1
OPERATOR = *
NO
1730
YES
1720
FIND A DFLT OPERAND
WAS IT INVERSE
YES
1740
NO
1730
1730
2 → NUMBER OF FUNCTION STRINGS → IT
1760
1740
3 → NUMBER OF FUNCTION STRINGS → IT
1760
1750
1 → NUMBER OF FUNCTION STRINGS → IT
1760
COMPUTE NEW NUMBER OF BRANCHS AT THIS LEVEL
SAVE TOTAL NUMBER OF BRANCHES TRANSFERED → ITEM1
NUMBER OF DBLE BRANCHES → ITEM
B2
B2
SET INVERSE OPERAND COUNT TO ZERO → LS1
ASSIGN 1765 → IFLOAT
791
1765
ANY DFLT OPERANDS
YES
1780
NO
1770
REMOVE FUNCTION INCREMENT FROM OPERAND
NORMAL/INVERSE SWITCHES
855


1780
IBUF2 POINTER TO OPERATOR IN DBLE STRING → N
IBUF2 POINTER TO FIRST AVAILABLE OPERAND STORAGE LOCATION → M
1785
MOVE DBLE OPERANDS TO IBUF2
1805
FILL IBUF1 GAPS LEFT BY TRANSFER OF DBLE OPERANDS
1830
IBUF2 DISPLACEMENTS NEED ADJUSTMENTS
NO
745
YES
ADJUST DISPLACEMENTS
745
745
SET SYMTAB POINTER TO FLOAT IBUF1 (3)
DFLT FUNCTION
YES
SET SYMTAB POINTER TO DFLT IBUF1(3)
NO
746
SEE IF ALL OPERANDS ARE INVERSE
750
REMOVE FUNCTION INCREMENT FROM OPERAND NORMAL/INVERSE SWITCHES
WAS THERE A NORMAL OPERAND
YES
770
NO
760
REMOVE OPERAND INVERSE FLAGS
B3
B3
0 → OPERAND INVERSE SWITCHES
K
770
0 → TO OPERAND INVERSE COUNTER
LS1
1 → FUNCTION STRING COUNT
DO WE NEED 2 STRINGS
NO
740
YES
774
PREPARE A SECOND STRING
790


790
ASSIGN 855 → IRETURN
HAVE ANY DBLE OPERANDS
YES
740
No
SET NEW BRANCH COUNT AT THIS LEVEL
791
IBUF2 POINTER TO FIRST BRANCH DISPLACEMENT → J
710
CLEAR TRANSFERED BRANCH DISPLACEMENT VALUES
715
COMPUTE DISPLACEMENT ADJUSTMENT VALUE
ADJUSTMENT NEEDED
NO
740
YES
720
720
ADJUST DISPLACEMENT VALUES
730
730
FILL LOCATIONS FREED BY ZEROED BRANCH DISPLACEMENTS
POINTER TO FIRST WORD OF NEW GAP → IBK
740
COMPUTE DISPLACEMENT TO NEW BRANCH → IBUF2 (IBR)
POINTER TO LAST WORD OF GAP+1 IN IBUF2 → IBUF2(NDBK)
B6
B6
IBUF1 POINTER TO END OF BRANCH → LRELEN
NUMBER OF OPERANDS IN BRANCH → IBUF1 (8)
NEED OPERATOR FOR THIS BRANCH
NO
DECREASE LENGTH INCREMENT BY 3
YES
COMPUTE IBUF2 POINTER TO LAST WORD OF IBUF2 NEEDED FOR TRANSFER → IN
PROCESS ONLY ONE FUNCTION STRING
YES
798
NO
A
94


A 94
SECOND STRING NEED AN OPERATOR
NO
DECREASE LENGTH INCREMENT BY 3
YES
COMPUTE IBUF2 POINTER TO LAST WORD OF IBUF2 NEEDED FOR SECOND BRANCH TRANSFER
IN
IBUF1 POINTER TO END OF FIRST BRANCH
LRELEN
798
MAKE ROOM IN IBUF2 TO RECIEVE TRANS-FER
POINTER TO BEGINING OF OPERAND SECTION OF FIRST BRANCH IN IBUF1
IXT
COMPUTE ADDITIONAL IBUF2 SPACE REQUIRED FOR TRANSFER
IT
LESS
SPACE NEEDED
MORE
820
800
NONE
840
800
YES
WILL REQUIRED SPACE CAUSE OVERFLOW
PUNT
810
NO
CREATE MORE ROOM IN IBUF2 FOR TRANSFER
840
820
PREPARE TO TAKE UP SLACK IN IBUF2
830
TAKE UP IBUF2 SLACK
840
IBUF2 POINTER TO END OF TREE
IBUF2X
B7


B7
ADJUST NUMBER OF WORDS IN TREE
IBUF2(LOX)
844
NUMBER OF WORDS IN HEADER
K
850
TRANSFER FUNCTION HEADER
IBUF2
IRETRN
855
MORE THAN ONE OPERAND
NO
864
YES
860
COMPUTE OPERAND DISPLACEMENTS
IBUF2(IBK)
864
864
PREPARE TO TRANSFER OPERANDS OF BRANCH FUNCTION
870
TRANSFER OPERAND FROM IBUF1
IBUF2
IS THERE A SECOND BRANCH
NO
872
YES
COMPUTE DISPLACEMENT TO THIS BRANCH
IBUF2 (IBR)
IBUF1 POINTER TO BEGINING OF OPERAND SECTION
IXT
POINTER TO END OF THIS BRANCH IN IBUF1
LRELEN
B9
B8
COMPUTE NUMBER OF OPERANDS IN THIS BRANCH
ITEM
PREPARE TO TRANSFER INVERSE BRANCH
844
872
NEED TO ADJUST DISPLACEMENT
NO
900
YES
POINTER TO START OF DISPLACEMENT TO BE ADJUSTED
IBR
POINTER TO END OF DISPLACEMENT
IXT
C1


C1
875
ADJUST DISPLACEMENTS
LEVEL COUNT → LS1
880
HIGHER LEVEL
YES
900
NO
PREPARE ONE LEVEL HIGHER IN TREE
{LS1-1}
LAST OPERAND THIS LEVEL
YES
880
NO
POINTER TO THIS LEVEL OPERATOR → IXT
890
890
ADJUST DISPLACEMENTS OF HIGHER LEVEL IN TREE
880
900
REINITIALIZE VALUES
500
900
POINTER TO NEXT AVAIABLE WORD IN IBUF2 → IBUF2X
RETURN

4.0 PHASEB

4.1 Phase B General Description

4.2 OPERATION OF THE PHASE B DRIVER (PHASE B)

4.3 PHASE B STATEMENT PROCESSORS

4.4 PHASE B SUBROUTINES

TABLE OF CONTENTS

4.0 PHASE B

4.1 GENERAL DESCRIPTION

4.1.1 Phase B Function Within Compiler

Phase B is the instruction generation pass of the 1700 Tape FORTRAN compiler, or, what is termed the pre-assembly phase. Its responsibility is translating FORTRAN statements into instructions. Other peripheral tasks relevant to the pre-assembly and assembly phase (Phase C) are carried out by Phase B.

4.1.2 Input Description

Input to Phase B is of two types:

1. Pertinent information residing in that portion of the compiler's common blocks which remains intact from one pass to the next pass. (LABELLED COMMON) The information contained therein is discussed in detail in 4.5.

2. An input file on disk corresponding to the output file of Phase A. It consists of a consecutive string of logical records the last of which represents the END line. The logical records in the Phase B input file correspond one to one, and in the same physical order as the 1700 Tape FORTRAN source statements, with the following set of exceptions.

   a. Only executable statements reach the Phase B input file with the following additions and exceptions: Statements which are not classified as executable but which are passed:

      (1) SUBROUTINE
      (2) FUNCTION
      (3) DATA
      (4) FORMAT
      (5) END
      (6) Statement Function

      Statements which are classified as executable but which are not passed comprise the set of Input/Output and Auxiliary Input/Output statements:

      (1) READ (formatted)
      (2) WRITE (formatted)
      (3) READ (unformatted)
      (4) WRITE (unformatted), and ...
      (5) REWIND

(6) BACKSPACE
(7) ENDFILE

b. Certain conditions in compilation cause the creation of additional executable statements in conjunction with those which appeared in the original source input.

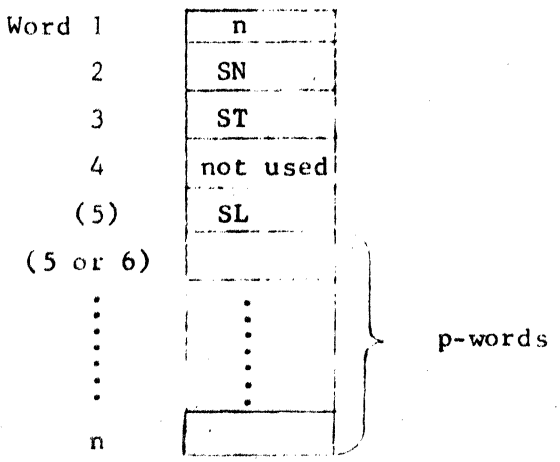
1. Input/Output and Auxiliary Input/Output statements as noted in paragraph a, are replaced in the same relative position within the source input, with an expanded set of statements consisting of one or more statements of type CALL, BEGIN DO, END DO. (See discussion of END DO below.)

2. When two conditions are satisfied, a STOP statement is created and inserted immediately prior to the END line. First, the program being compiled is a main program, and second, the last executable statement of the program is neither a STOP, STOP n, or any other unconditional transfer statement. (Unconditional transfer statements which comprise the latter (truth) portion of a logical IF do not qualify.)

3. For each DO statement in the source input the Phase β input file contains a BEGIN DO statement record. But additionally, a special statement, END DO, internal to the compiler, is generated uniquely for each BEGIN DO and located logically in the input file so as to aid Phase β in DO-loop instruction generation.

c. The Logical IF statement appears in the Phase β input file as two separate statements.

The figure below illustrates the format of a record of the Phase β input file.



Where:

n = the number of words in the record ($n \geq 4$)
SN = s.n. or -s.n., where s.n. is the statement number
ST = the statement type
SL = the symbol table pointer of the statement label $p \geq 0$

The statement label is optional and its existence is indicated by SN being negative. When SN is negative, the remaining p words of the record begin at word 6 and end at word n. When SN is positive, the remaining p words of the record begin at word 5 and end at word n.

The interpretation of the last p words of any record by the Phase β processors is primarily determined by ST, the statement type. Each statement type designates what information, if any, follows in the remaining p words.

Table 4-1 lists all statements processed in the 1700 Tape FORTRAN compiler. Each statement is cross-referenced with its statement type code and its disposition by Phase β. Refer to the following key for further explanation:

(n.u.) - this ST code is <u>not used</u> in 1700 Tape FORTRAN

(n.r.) - this statement type is not received by Phase β in its input file.

"<u>processed by XXXXXX</u>" - Phase β driver calls upon subprogram (named XXXXXX) which is designed to perform the appropriate processing (usually generation of instructions) for that particular type of statement.

"<u>processed by Phase β driver</u>" - indicates that the appropriate processing for that particular type of statement is performed within the Phase driver and no subprogram is called which is specifically designed to process that statement type.

TABLE 4 - 1

Statement Types and Phase B Disposition

Type Code	Statement Name	Disposition by Phase B
0	DIMENSION	(n.r.)
1	COMMON	(n.r.)
2	INTEGER	(n.r.)
3	REAL	(n.r.)
4	(n.u.)	
5	(n.u.)	
6	(n.u.)	
7	SINGLE	(n.r.)
8	BYTE, SIGNED BYTE	(n.r.)
9	(n.u.)	
10	EXTERNAL	(n.r.)
11	RELATIVE	(n.r.)
12	EQUIVALENCE	(n.r.)
13	BLOCK DATA	(n.r.)
14	SUBROUTINE	processed by SUBFUN
15	FUNCTION	processed by SUBFUN
16	DATA	processed by NOPROC
17	FORMAT	processed by NOPROC
18	Replacement Statement	processed by ARITHR
19	Statement Function	processed by PHASE B DRIVER
20	ASSIGN	processed by Phase B driver
21	CALL	processed by Phase B driver
22	RETURN	processed by Phase B driver
23	(n.u.)	
24	GO TO (unconditional)	processed by Phase B driver
25	GO TO (computed)	processed by CGOTO
26	GO TO (assigned)	processed by Phase B driver
27	CONTINUE	processed by Phase B driver
28	STOP	processed by Phase B driver
29	STOP n	processed by Phase B driver

Type Code	Statement Name	Disposition by Phase B
30	PAUSE	processed by Phase B driver
31	PAUSE n	processed by Phase B driver
32	END	processed by END
33	ENDFILE	(n.r.)
34	REWIND	(n.r.)
35	BACKSPACE	(n.r.)
36	READ (unformatted)	(n.r.)
37	READ (formatted)	(n.r.)
38	WRITE (unformatted)	(n.r.)
39	WRITE (formatted)	(n.r.)
40	BEGIN DO	processed by BGINDO
41	END DO	processed by BANANA
42	Arithmetic IF	processed by PHASE B DRIVER
43	Logical IF	processed by PHASE B DRIVER
44	ASSEM	processed by ASSEM

Phase B specifications will not give a general summary of the detailed formats of the input records for each statement type. Refer to 7.5 for a detailed description. References will be made to such formats as may be appropriate in clarifying explanations.

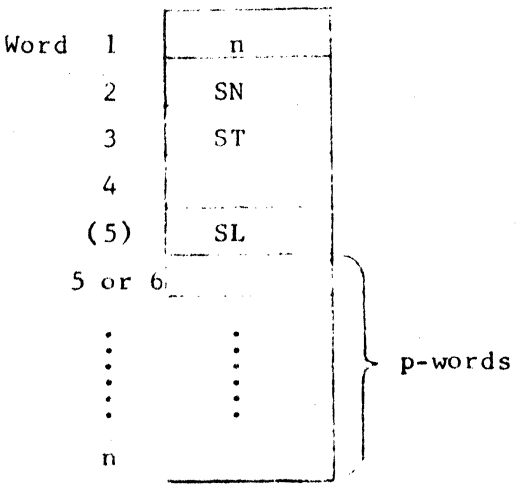
4.1.3 Output Description

Output from Phase B is of two types

1. Pertinent information (either retained from previous passes or set by Phase B) residing in that portion of the compiler's common blocks which remains intact from Phase B to Phase C. (The information contained therein is discussed in detail in 4.5.) Use and definition of the information is discussed throughout Phase B description.

2. An output file on disk corresponding to what will become the Phase C input file. It consists of a consecutive string of logical records, the last of which is indicative of the end of the file.

   The figure below illustrates the general format of a record of the Phase B output file:



Three cases of the general format occur:

Case 1:

Where:

n = the number of words in the record ($6 \leq n \leq 9$)

SN = (not used)

ST = 49

SL = (does not exist)

the next p words begin with word 5

$2 \leq p \leq 5$

Case 1 is the instruction record. The p words describe a single instruction generated by Phase β.

Case 2:

Where:

All definitions are the same for the Phase β input records (see 4.1.2 P 2c). Case 2 occurs for the case in which the processing of an input record by Phase β primarily is the transfer of that record to the output file exactly as it was read without modification. Records which fall into this category are DATA and FORMAT (ST = 16 or ST = 17)

Case 3:

Where:

n = 1

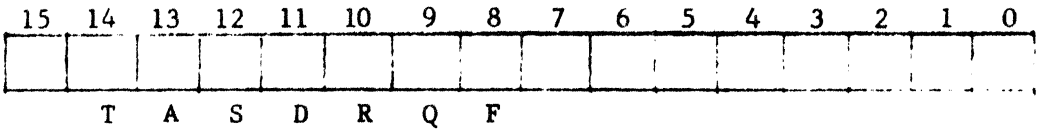
Case 3 is a flag to Phase C to indicate that the next record is a case 2 type record.

4.1.3.1 Instructions

The 2 to 5 words in the case 1 record format describe a single instruction generated by Phase β. This format is as follows:

Word	1		Indicator Word
	2	I	
	(3)	O	
	(4)	A	
	(5)	S	

Word 1 is always the INDICATOR word. It consists of a series of flags and indicators which describe the pseudo instruction and imply the existence and meaning of the words following the INDICATOR. The format of this word follows:



1. T is a special flag to indicate whether or not this pseudo instruction is a label item.

   T = 0, this any type of instruction except a label.

   T = 1, this instruction is a label.

   A single label item labels the instruction immediately following. A group of two or more label items which occur consecutively are considered to be equivalent and label the instruction immediately following the group of labels.

2. A specifies whether or not this instruction has an additive. An additive is an adjustment of the operand address. For example:

   I = J(7)
   would result in pseudo code:

   LDA J +6

   STA I, where the value +6 is the additive.

   Phase B must pass this additive on to the next phase which after assigning an address to J must then increase the reference to it by 6.

   A = 0, no additive

   A = 1, additive.

3. S specifies whether or not the operand address is subscripted.

   S = 0 not subscripted

   S = 1 subscripted

4. D tells whether instruction is to be made indirect or not.

   D = 0 not indirect

   D = indirect

5. R Decision whether instruction is to be made 1 or 2 words.

   R = 0 decision is left to next phase which will make any such command 1 word whenever possible, dependent upon the size of constant operands and distances of relative addresses, etc.

   R = 1 force instruction two words, regardless of operand size. This is usually done where operand has to be plugged.

6. Q&F These two bits are set aside for the next phase as working space in its index register assignment, one for index Q and one for FF. In special instances, Phase 4 may set Q and generate its own instructions to load Q.

7. ∅ operand type

   A. For instruction code ≠ 2

      ∅ = 0 normal case, operand is a symbol table pointer. Addressing mode is assigned by assembly pass (Phase 5)

      ∅ = 1 Operand is an absolute value. For the instruction code being a storage reference instruction, the operand is less than 256_{10}

Ø = 2 Operand is an absolute value. For the instruction code being a storage reference instruction, Phase 5 assigns two-word constant address mode.

Ø = 3 Not used

Ø = 4 Operand is relocatable address constant (biased)

Ø = 5 Operand is absolute address constant (unbiased)

B. For instruction code = 2

Ø = 0 not used

Ø = 1 This address constant is in a table of addresses used in a Computed GO TO statement. The address is self relative and is calculated using 16-bit arithmetic.

Ø = 2 This address constant is a parameter in a calling sequence to the floating point subroutine.

Ø = 3, 4, 5 refers to parameters in a calling sequence to a subroutine other than the floating point subroutine.

Ø = 3 Address self relative computed in 15-bit arithmetic with 15 bit set to 1.

Ø = 4 Relocatable, address. Address is location of referenced operand.

Ø = 5 Same as type 4 except address is unbiased.

Word 2 is <u>always</u> the instruction code <u>except</u> when the instruction is a label (bit T=1). Instruction codes are shown in table below:

OCTAL CODE	DECIMAL CODE	INSTRUCTION	OCTAL CODE	DECIMAL CODE	INSTRUCTION
1	1	BSS	41	33	INQ
2	2	ADC	42	34	LRS
3	3	CON	43	35	LLS
4	4	END	44	36	QRS
5	5	PST	45	37	QLS
6	6	STN	46	38	ARS
7	7		47	39	ALS
10	8		50	40	AJLGZ
11	9		51	41	AJEZ
12	10	JMP	52	42	AJLZ
13	11	RTJ	53	43	AJGEZ
14	12	LDA	54	44	AJLEZ
15	13	AND	55	45	AJGZ
16	14	STA	56	46	
17	15	STQ	57	47	QJEZ
20	16	RAO	60	48	
21	17	ADD	61	49	QJGEZ
22	18	SUB	62	50	
23	19	LDQ	63	51	
24	20	DVI	64	52	FCM
25	21	ADQ	65	53	FSB
26	22	MUI	66	54	FMU

OCTAL CODE	DECIMAL CODE	INSTRUCTION
27	23	EOR
30	24	
31	25	
32	26	
33	27	
34	28	KLDQ
35	29	KSTQ
36	30	ENA
37	31	INA
30	32	ENQ

OCTAL CODE	DECIMAL CODE	INSTRUCTION
67	55	FDV
70	56	FLD
71	57	
72	58	FST
73	59	FAD
74	60	CAQA
75	61	TCQA
76	62	TRAQ
77	63	TCAA
100	64	TCQQ
110	72	DCM
111	73	DSB
112	74	DMN
113	75	DDV
114	76	DLD
115	77	
116	78	DST
117	79	DAD

Each instruction corresponds uniquely to one actual 1700 instruction with the noted exceptions:

1. END, PST, STN are discussed where appropriate in other sections.

2. FCM, FSB, FMU, FDV, FLD, FST, FAD and DCM, DSB, DMU, DDV, DLD, DST, DAD are single precision and double precision floating point instuctions used only within Phase B. Phase C doesn't encounter floating commands as such, but rather the floating calling sequences:

```
RTJ  FLOT
CON  ...
ADC
ADC
 :
etc.
```

or

```
RTJ  DFLOT
CON  ...
ADC
ADC
 :
etc.
```

3. "Place holding"--pseudo instructions BSS, ADC and CON.

4. Conditional Jumps

   Phase B has occasion to generate code to test the accumulator or Q-register for 6 possible conditions:

   Acc or Q $<$ 0 {less than zero}
   " $=$ 0 {equal to zero}
   " $>$ 0 {greater than zero}
   " $\leq$ 0 {less than or equal to zero}
   " $\geq$ 0 {greater than or equal to zero}
   " $\neq$ 0 {less than or greater than zero}

   The 1700 Instructions Test--The condition in such a manner that if the condition is true, control is transferred to the operand address and if the condition is false, the program continues in its normal sequence.

   The conditional jump instructions {for the A-register} in the 1700 operate in the following manner:

   A. SAZ, condition is satisfied if all 16 bits of A are 0.

   B. SAN, condition is satisfied if any or all of the 16 bits of A are $\neq$ 0.

C. SAP, condition is satisfied if bit 15 (sign bit) of A is 0.
D. SAM, condition is satisfied if bit 15 of A is $\neq$ 0.

Problems in testing arise from the manner in which these instructions operate:

1. -0, will not be detected by SAZ.
2. -0, will falsely satisfy SAN.
3. SAP, does not exactly test for greater than 0 but for A = +0 or A $\geq$ 0.
4. SAM, does not exactly test for less than 0 but for A = -0 or A $<$ 0.

(The same holds true for the analogous Q-register tests, SQZ, SQN, ...etc.)

To aid in effective code generation, eight special conditional jumps have been created. Each corresponds to a specific test needed by Phase B to test the accumulator. Each results in the generation of some group of real instructions. The resulting group of real instructions is left to Phase C due to information which is unavailable during Phase B operations. (Specifically, the distance of the operand address from the test instructions affects the type of code generation.)

The table below cross-references the special codes with the code that will be generated by Phase C depending upon the noted conditions:

if:			Phase C will generate the following code:	
PHASE B pseudo code		Meaning	If operand address ADD is not more than 15 cells distant in a forward direction from its references:	If address Add is more than 15 cells distant in a forward direction, or is in a backward direction:
AJD,LZ	ADD	A $<$ $\pm$0	SAM ADD-*-1	SAP (1 or 2) JMP* ADD
AJP,EZ	ADD	A = $\pm$0	SAZ ADD-*-1	SAN (1 or 2) JMP* ADD
AJP,GZ	ADD	A $>$ $\pm$0	SAZ 1 SAP ADD-*-1	SAZ (2 or 3) SAM (1 or 2) JMP ADD
AJP,LEZ	ADD	A $\leq$ $\pm$0	SAM ADD-*-1 SAZ ADD-*-1	SAZ 1 SAP (1 or 2) JMP* ADD
AJP,GEZ	ADD	A $\geq$ $\pm$0	SAP ADD-*-1	SAM (1 or 2) JMP* ADD
AJP,LGZ	ADD	A $\neq$ $\pm$0	SAN ADD-*-1	SAZ (1 or 2) JMP* ADD

(NOTE: Operands in parenthesis are two alternatives depending upon whether JMP* ADD is a one or two-word command, respectively.)

There is limited use of Q-register testing. Only the two instructions QJP,EZ and QJP,GEZ (analogous to AJP, EZ and AJP,GEZ) are needed, thus the total number of conditional jumps is eight.

Word 3:

1. Whenever the instruction code is an instruction which has an operand, the operand is in word 3. The meaning of the operand, whether its a constant or a symbol table pointer is dependent upon field Ø in the indicator word.

2. Whenever the instruction code is an instruction which has no operand, word 3 will contain no significant information.

Word 4:

Whenever field A in the indicator word is set ≠0 word 4 will contain the additive value to the operand contained in Word 3.

Word 5:

Whenever field S in the indicator is set ≠0 word 5 will contain the symtab pointer to the variable which subscripts the operand.

4.2 OPERATION OF THE PHASE 8 DRIVER (PHASE 8)

PHASE 8 is the driver which controls the overall operation of Phase 8. Its functions are:

1. Initialize all flags and switches necessary to begin the phase.
2. Generate special program-wide labels.
3. Generate all internal arrays, constants and variables by calling HELEN.
4. Where program is run-anywhere, generate special instructions which compute relocation address at object-time.
5. Call READIR to read input records. Make decision on which processor to call based upon the statement type. Call that processor. Statement functions, arithmetic IF statements, and logical IF statements are processed in PHASEB as described in 4.2.2-4.
6. In case of logical IF reset input buffer pointers and output special labels as is appropriate.
7. Monitor the processing of statement functions and reinitialize flags and generate certain labels as appropriate.

4.2.1 Program Structure

Program structure is a function of the program type and the run-anywhere switch. The illustration below will clarify how these conditions interrelate to produce the program structure. The ordering of the various elements of the program is equivalent to the ordering in the table.

run-anywhere switch	main-program	block data subprogram	subroutine	function subprogram	code generated
on					LABEL TOP (label of top of program. Relo address 0000)
	x				LABEL Q8QNAM (generated name of main program) JMP* MBEGIN (jump from first cell of program to first executable instruction)
	x		x	x	generation of all programmer-defined arrays, variables, and constants in this FORTRAN program. (structure of this code is elucidated in section on program HELEN)
	x		x	x	code generation for 1st statement function, code generation for 2nd statement function, . . ., code generation for n-th statement function (structure of this code is elucidated in section dealing with statement function processing, PHASE B)

run-anywhere switch	main-program	block data subprogram	subroutine	function subprogram	code generated
	x		x	x	LABEL MBEGIN (label of first executable instruction in program)
on					RTJ* BIAS ADC TØP BIAS BSS 1 LDA BIAS ADD BIAS -1 STA BIAS (this code computes a value representing the value of the relocation address of the program and places that value in BIAS. Run-anywhere considerations require that the bias must be computed at run time.)
	x		x	x	main body of program includes code generation of all executable statments, and generation of jumps around format statements.
			x	x	LABEL RETURN (label referenced by jumps which are the result of RETURN statements)
			x	x	code to restore indexes (structure of this code is elucidated in section on program INXRST)

run-anywhere switch	main-program	block data subprogram	subroutine	function subprogram	code generated
				x	LDA or FLD of value of function into accumulator or floating point accumulator respectively.
			x	x	JMP* (KPRNAM) (indirect jump to entry point) KABEL KPRNAM
			x	x	entry code. code which is responsible for picking up parameters and plugging them into the program with the proper increments. The code also includes the entry point BSS 1 and the jump to the first executable statement (structure of this code is elucidated in section on program ENTCOD)
	x	x	x	x	NEND (terminating pseudo instruction)

4.2.2 STATEMENT FUNCTION PROCESSOR

PHASEB processes the statement function in the following steps:

1. Generates and outputs a label to tag the first executable instruction of the statement function.

2. Places the symtab pointer of the statement function name into holder.

3. Initializes table KSFAT with the statement function arguments. Upon completion of this operation the table appears in its initialized state as shown

KSFAT (12,2)

KSFAT (1,1)	ptr 1	0	KSFAT (1,2)
KSFAT (2,1)	ptr 2	0	
	•		
	•		
	•		
KSFAT (n,1)	ptr n	0	
	0		
	•		
	•		
	•		
	•		
	•		
	•		
KSFAT (12,1)			

where ptr 1-n are the symtab pointers of the statement function arguments. The 0 at KSFAT (M+1,1) signifies the logical end of the parameter list. The second word is used later to contain pointers to assigned temporary storage (See KPCSTK)

4. Update NDPRS (number of argument-increment pairs) by the number of arguments.

5. Determine if the mode of the right side of the statement function expression is different from the mode of the function and generate instructions to call the appropriate conversion routines {integer to real, real to integer, integer to double precision, etc.} where necessary.

6. Generate instructions to restore indexes, the exit command {indirect jump through the entry point}, the entry point, and sets up and calls ENTCOD to generate instructions for parameter pickup.

7. Decrement KSFCNT {# of statement functions} by 1.

4.2.3 ARITHMETIC IF STATEMENT PROCESSOR

Processes the Arithmetic IF statement. For purposes of discussion, the Arithmetic If can be described as follows:

```
      IF (E) K1, K2, K3
   L  (statement)
```

where, E is the arithmetic expression
K1 is the address for E < 0
K2 is the address for E = 0
K3 is the address for E > 0
and L is the label of the next statement following the Arithmetic IF.

Processing takes place in two basic steps:

1. The expression (E) is given to subroutine ASUPER for processing. PHASEB does not directly call ASUPER however, but goes through routine AFIDL which in turn calls ASUPER and is responsible for insuring that after the expression has been evaluated that -0 is eliminated thus simplifying the conditional testing.

2. One or more conditional and/or unconditional jumps are generated such that testing is performed in the most efficient manner. Two indexes are used IDTI and IDTJ whose values are dependent upon various considerations about the values of K1, K2, etc. The values of these indexes then determine the most efficient code to be generated. The settings and their meanings are shown:

```
IDTI = 1,  K1=K2
IDTI = 2,  K1=K3
IDTI = 3,  K2=K3
IDTI = 4,  K1≠K2≠K3≠K1

IDTJ = 1,  K1=L
IDTJ = 2,  K2=L  (K1≠L)
IDTJ = 3,  K3=K  (K1≠L, K2≠L)
IDTJ = 4,  K1≠L, K2≠L, K3≠L
```

After values for IDTI and IDTJ have been computed, then instructions are generated according to the following table:

IDTI=

IDTJ=	1	2	3	4
1	AJP,GZ K3	AJP,EZ K2	AJP,GEZ K3	AJP,LZ K1 AJP,ZR K2 JMP K3
2		AJP,LGZ K1	AJP,LZ K1	AJP,LZ K1 AJP,ZR K2 JMP K3
3	AJP,LEZ K1			AJP,LZ K1 AJP,ZR K2
4	AJP,LEZ K1 JMP K3	AJP,LGZ K1 JMP K2	AJP,LZ K1 JMP K2	AJP,LZ K1 AJP,ZR K2 JMP K3

4.2.4 LOGICAL IF STATEMENT PROCESSOR

PHASEB is the processor of the Logical IF statement. For reference, the Logical IF is defined:

IF (e) S

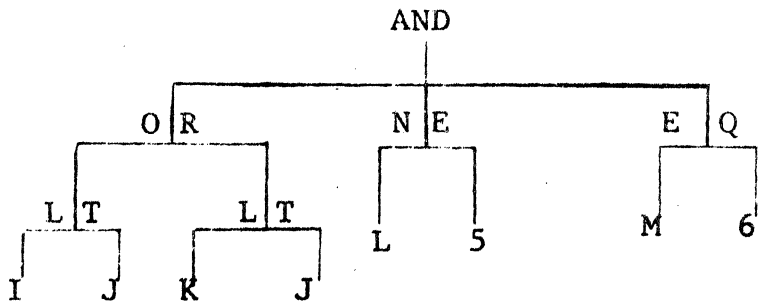
where:

e is a logical expression
S is any executable statement except a DØ statement or another Logical If.

The logical expression becomes a "tree structure" in Phase A and is processed as such in Phase B. "Level" tables are employed in the processing of the tree similar to those used in the processing of arithmetic expressions. For illustration the following logical expression will be used:

((I.LT.J.OR.K.LT.J).AND.(L.NE.5).AND.(M.EQ.6))

The tree structure becomes:



Relational expressions (I.LT.J, L.NE.5...etc.) are treated simply as arithmetic expressions where the operator is + and one of the two variables is made an inverse. The table below indicates which variable is made the inverse:

relational expression is interpreted:

I.LT.J	I-J
I.GT.J	J-I
I.LE.J	J-I
I.GE.J	I-J
I.EQ.J	I-J
I.NE.J	I-J

The order is immaterial in the EQ and NE cases. But in the other 4 cases the order of subtraction is determined such that the fewest number of test instructions are required to determine whether or not the condition has been satisfied.

Every logical and relational operator is treated as a binary value; that is either true or false. According to the true or false condition of the operator there is a corresponding branch to either of two possible points.

All logical and relational operations are assigned "truth" and "false" addresses which are based upon:

1. The type of logical operator of which they are a branch.

2. Whether or not the branch is the last branch of the logical operator.

The following table shows how assignments are made:

The type of logical operator of which this operator is a branch	This operator is not the last branch		This operator is the last branch	
	truth address	false address	truth address	false address
AND	the next branch of the AND	the same as the false address of the AND operator	the same as the truth address of the AND operator	the same as the false address of the AND operator
ØR	the same as the truth address of the OR operator	the next branch of the OR	the same as the truth address of the OR operator	the same as the false address of the OR operator

Processing proceeds as follows.

1. The 2nd statement of the Logical IF (S) is also read into the input buffer and it is determined whether or not that statement is an unconditional GØ TØ or RETURN statement.

2. If the 2nd statement was a GØ TØ or RETURN the truth address of the highest level operator is set to that address. (LOGIF is set to 2)

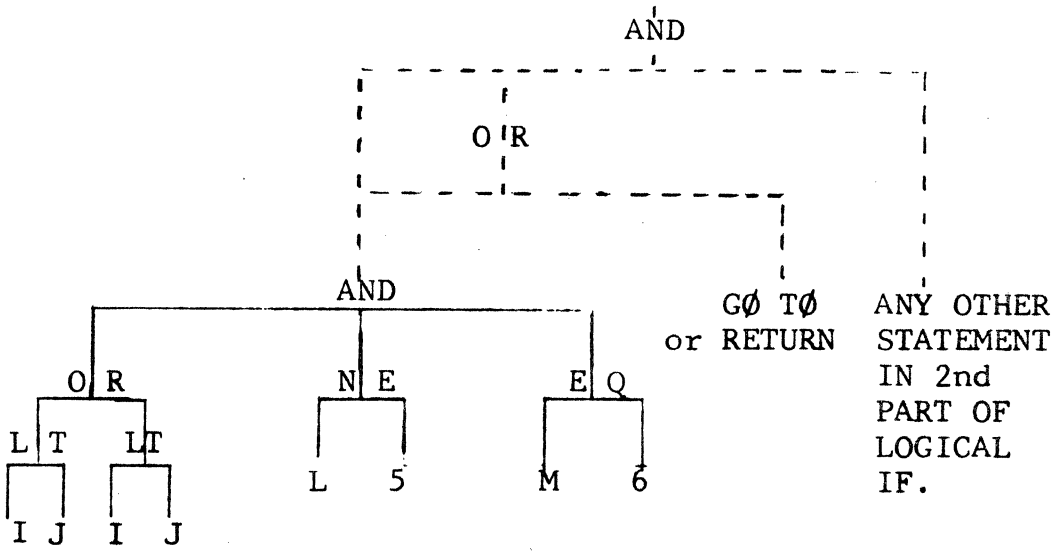
3. If the 2nd statement was not a GO TO or RETURN, the truth address is a generated label. (LOGIF is set to 1)

4. The false address is a generated label.

5. In the actual level tables the highest (1st) level operator is actually an artificial one based upon the following criterion:

   a. If the 2nd statement was a GO TO or RETURN, the operator is set to ØR.
   b. If not, the operator is set to AND.

The actual logical expression then becomes the first branch of an artifically created AND or ØR operator:



6. The processing then proceeds to cycle through all the branches of the logical operator (from left to right) when a branch is itself a logical operator, the level tables are updated to the next level down and the processing proceeds at that level with the first branch. When a branch is encountered which is a relational operator (and therefore an arithmetic expression) the code is generated via subroutine ASUPER to compute the value of the expression.

   Next, a conditional jump command is generated. Each relational expression has associated with it a "truth" condition and a "false" condition and likewise a corresponding conditional jump command:

relational	arithmetic	conditional jump	
		true	false
I.LT.J	I-J	AJP,LZ	AJP,GEZ
I.GT.J	J-I	AJP,LZ	AJP,GEZ
I.LE.J	J-I	AJP,GEZ	AJP,LZ
I.GE.J	I-J	AJP,GEZ	AJP,LZ
I.EQ.J	I-J	AJP,EZ	AJP,LGZ
I.NE.J	I-J	AJP,LGZ	AJP,EZ

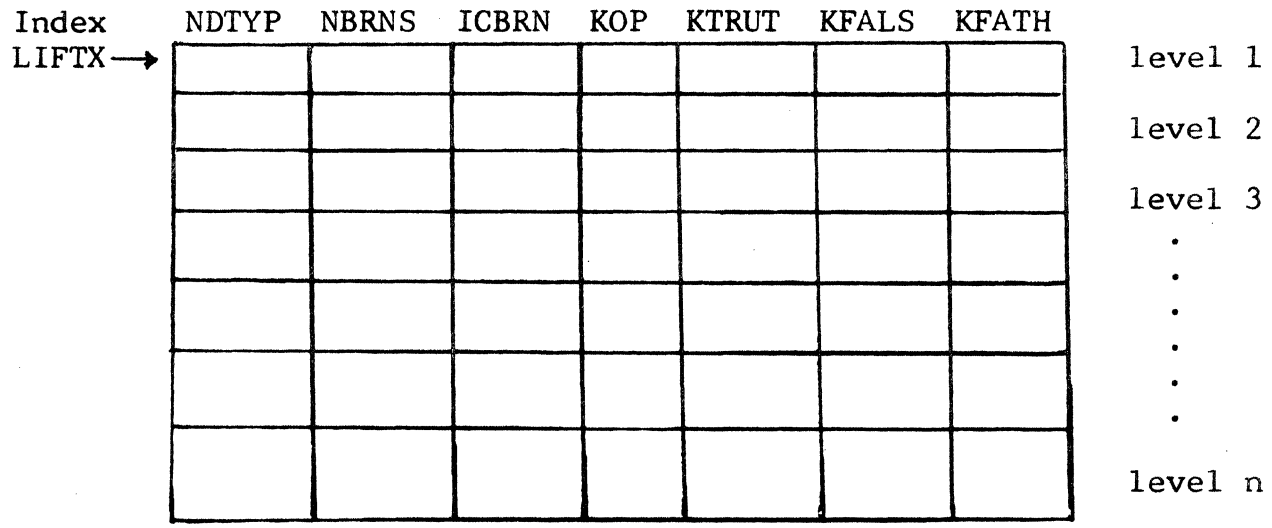
Only one of the two conditional jumps is actually generated. It is always true that either the truth address or the false address is the next expression to be computed (graphically, left to right across the tree, the "next" expression is the one to the immediate right; henceforth called the "fall-through address.") Therefore, if the truth case is actually the "fall-through," the conditional jump is the jump to the "false" address. If the false case is actually the "fall-through" the conditional jump is the jump to the "truth" address.

Returning to the example code is generated in the following order:

1. Test if I is less than J.
   compute I-J
2. If true, go to the next branch of the AND
   AJP,LZ LABEL 1
3. If false, fall through.
   test if K is less than J
   compute K-J
4. If false, go to the address of the statement after the Logical If.
   AJP,GEZ LABEL 2
5. If true, fall through. Output label LABEL 1
   test if L is not equal to 5
   compute L-5
6. If false, go to the address of the statement after the Logical IF
   AJP,EZ LABEL 2

7. If true, fall through.
   Test if M is equal to 6.
   compute M-6
8. (If 2nd half of Logical If is neither a GØ TØ or RETURN)
   If false, go to the address of the statement after the Logical IF.
   AJP,EZ LABEL 2
   If true, fall through to the statement which is the 2nd half of the Logical IF
9. (If 2nd half of Logical If is a GØ TØ or RETURN)
   If true, go to the address specified in the GØ TØ or implicit in the RETURN.
   AJP,NEZ X
   If false, fall through to the statement after the Logical If. (The GØ TØ or RETURN is then ignored.)

Level Tables



where:

NDTYP = type of operator (AND,ØR) (node type).
NBRNS = number of branches
ICBRN = current branch
KØP = (working pointer)
KTRUT = holder of "truth" address symtab ptr.
KFALS = holder of "false" address symtab ptr.
KFATH = holder of "fall-through" address symtab ptr.

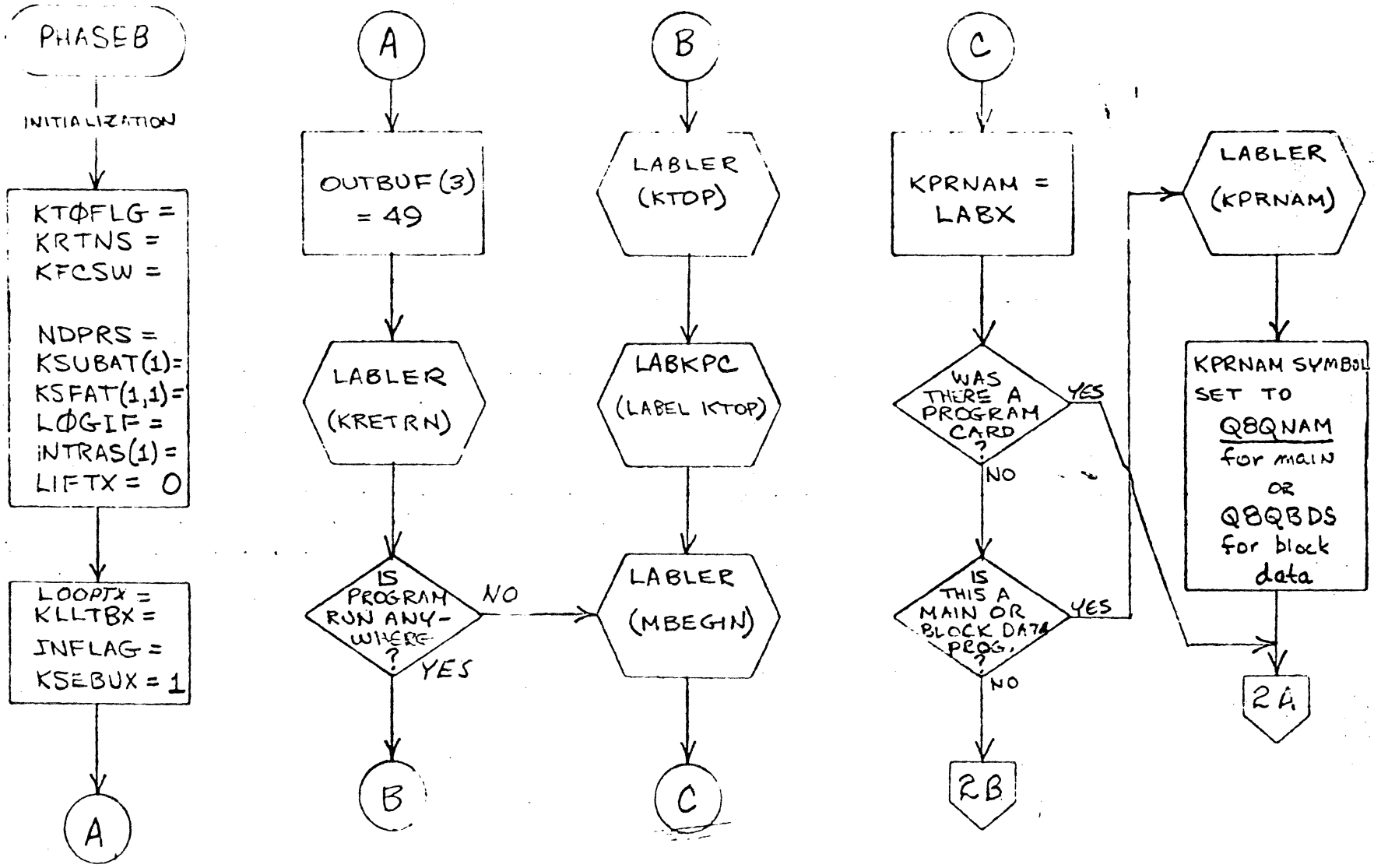
PHASEB checks the LOGIF switch whose settings are:

LOGIF = 0, no Logical IF statement has occurred
= 1, 2nd statement of Logical IF was not a GØ TØ or RETURN
= 2, 2nd statement of Logical IF was a GØ TØ or RETURN

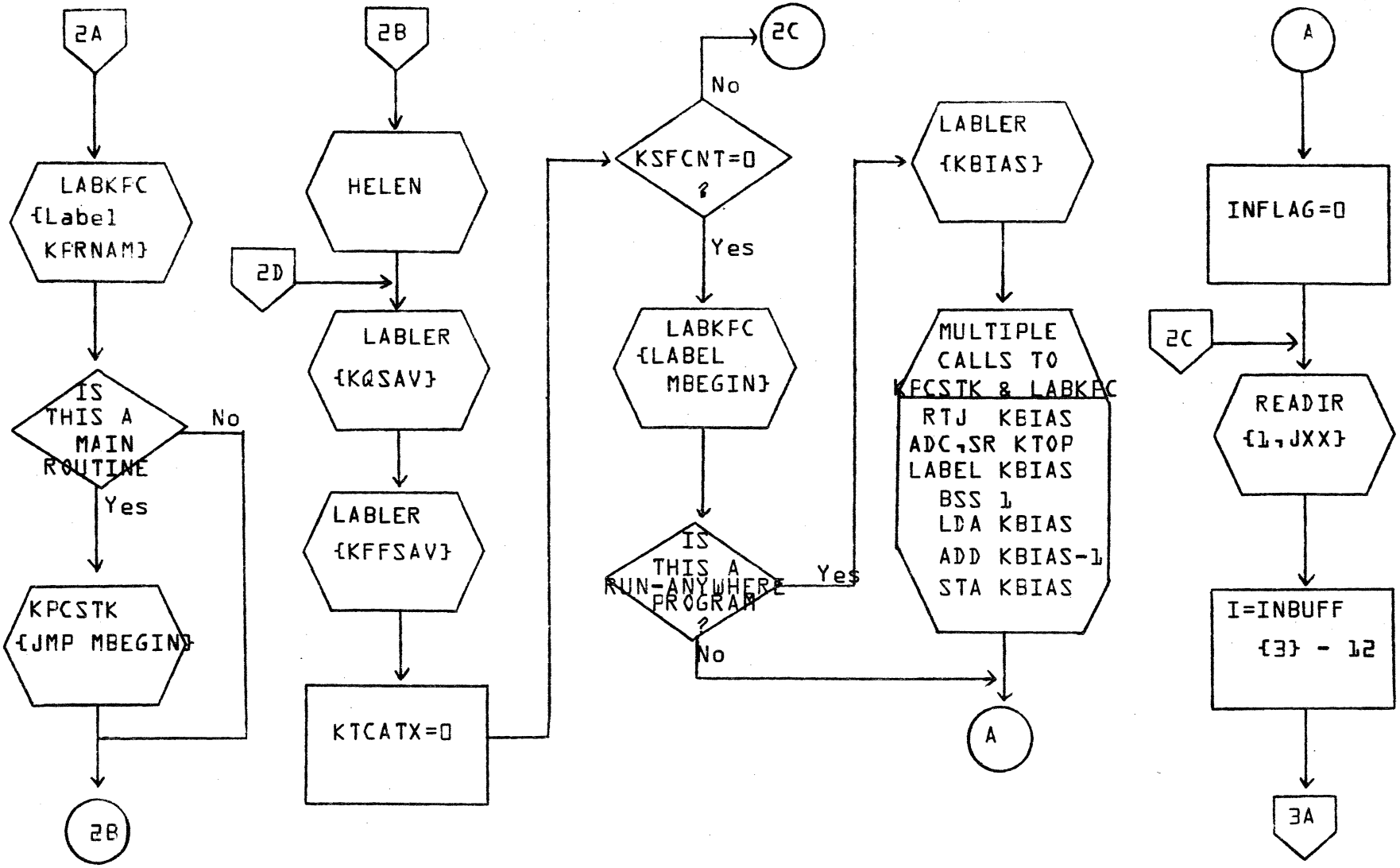
IF LØGIF = 1 PHASEB sets up its buffer and buffer index for processing of the 2nd half statement of the

Logical IF and outputs the level 1 "fall-through" label If LOGIF = 2 PHASEB ignores the 2nd half statement and outputs the lead 1 "false" label. It then returns to proceed with normal processing and read in the next statement. LOGIF is set to 0.

After the completion of any other statement processing, LOGIF is tested for =1. If so PHASE outputs the level 1 "false" label, then sets LØGIF = 0.



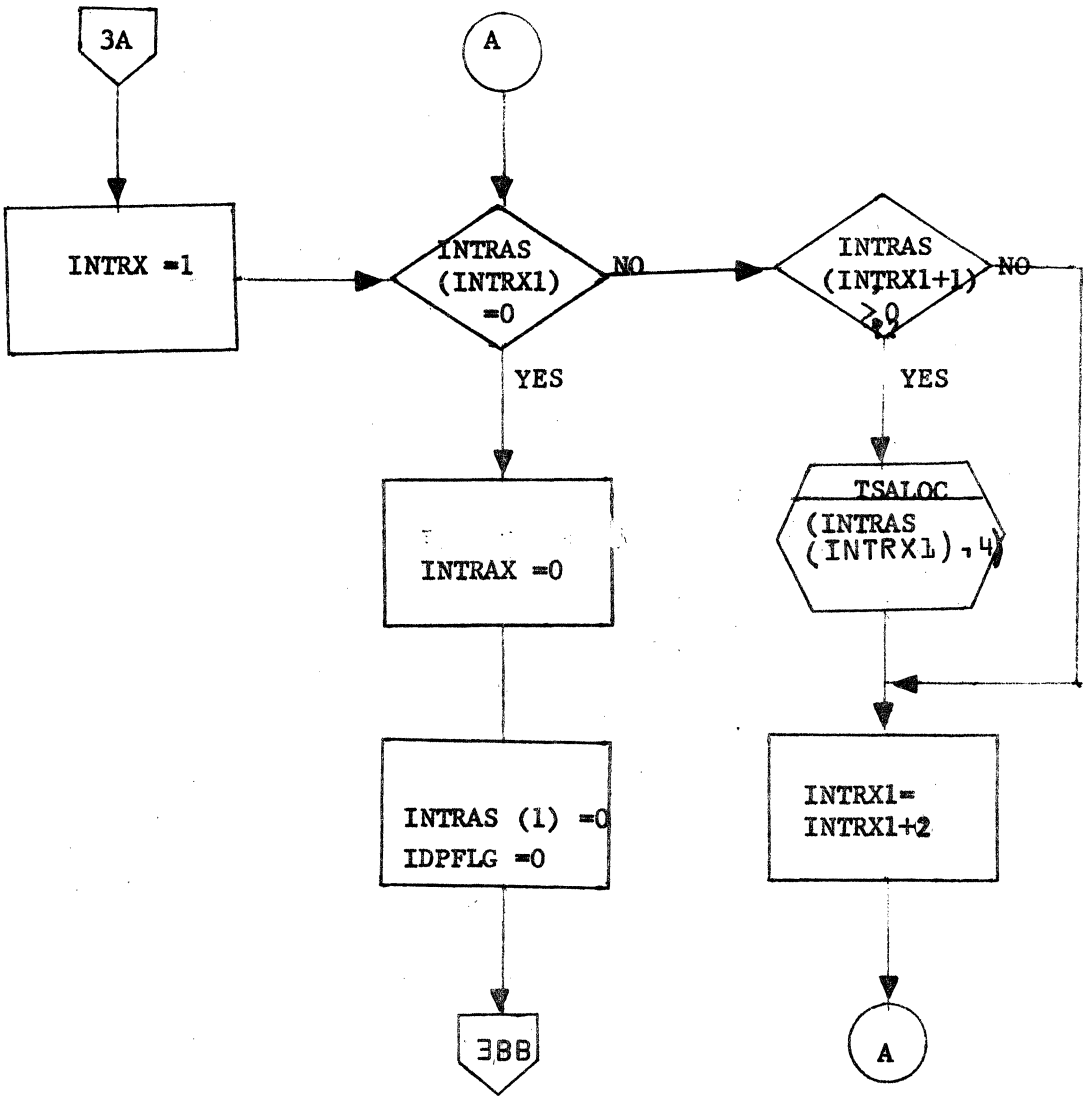
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INIS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASE B		PROJECT MGR.			
		PAGE 1 OF 20	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 7/1	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASE 8	PROJECT MGR.			
	PAGE 2 OF 208	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

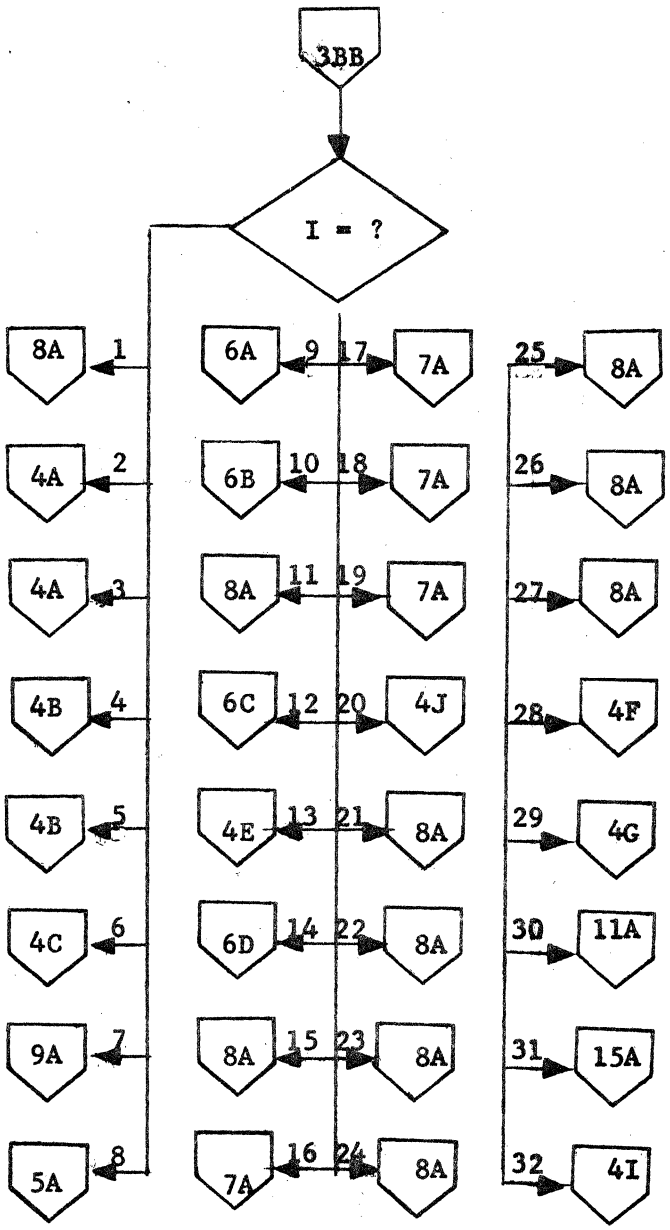
PRINTED IN U.S.A.



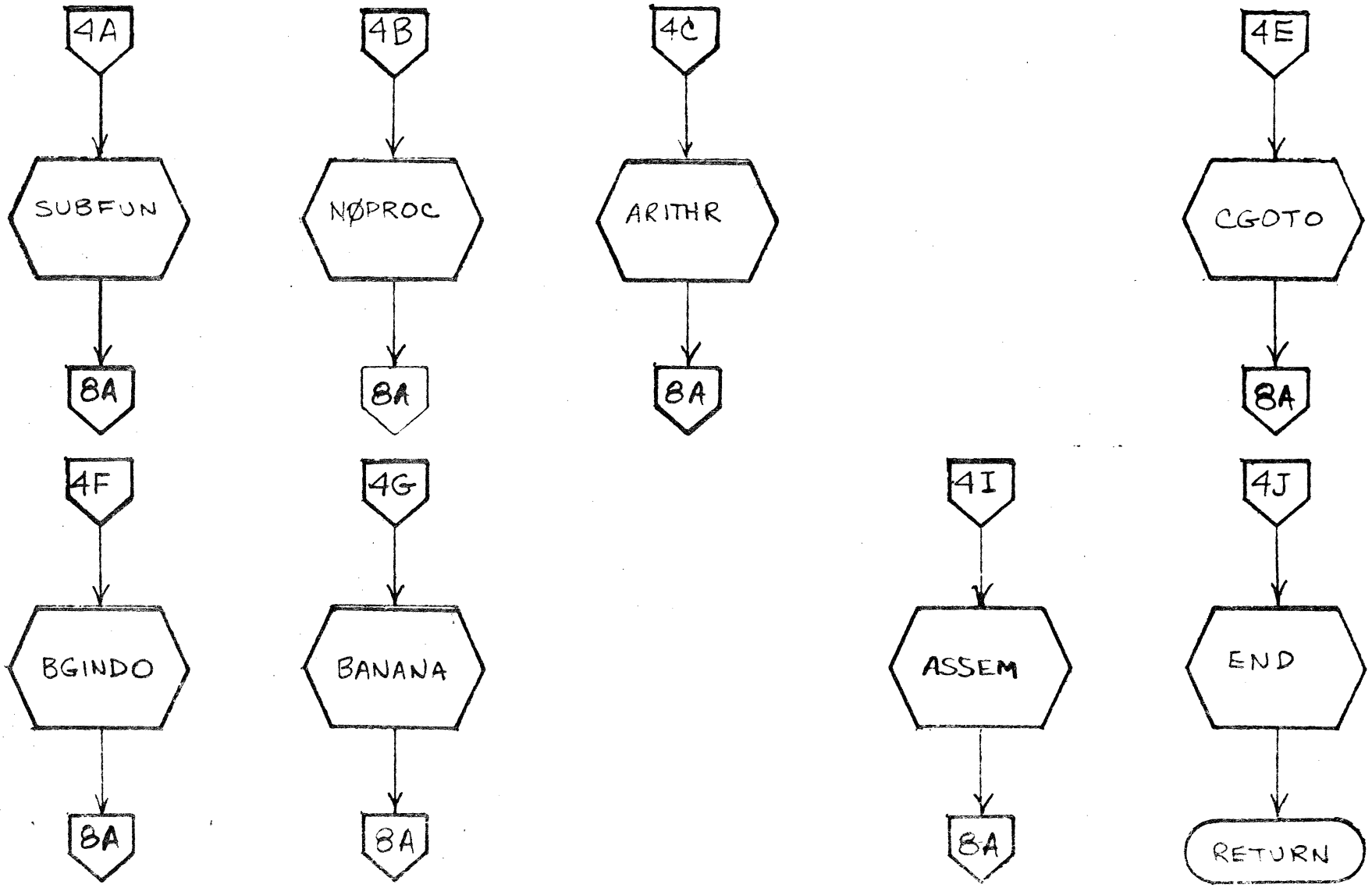
TITLE	PHASEB		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF 20

CONTROL DATA CORPORATION

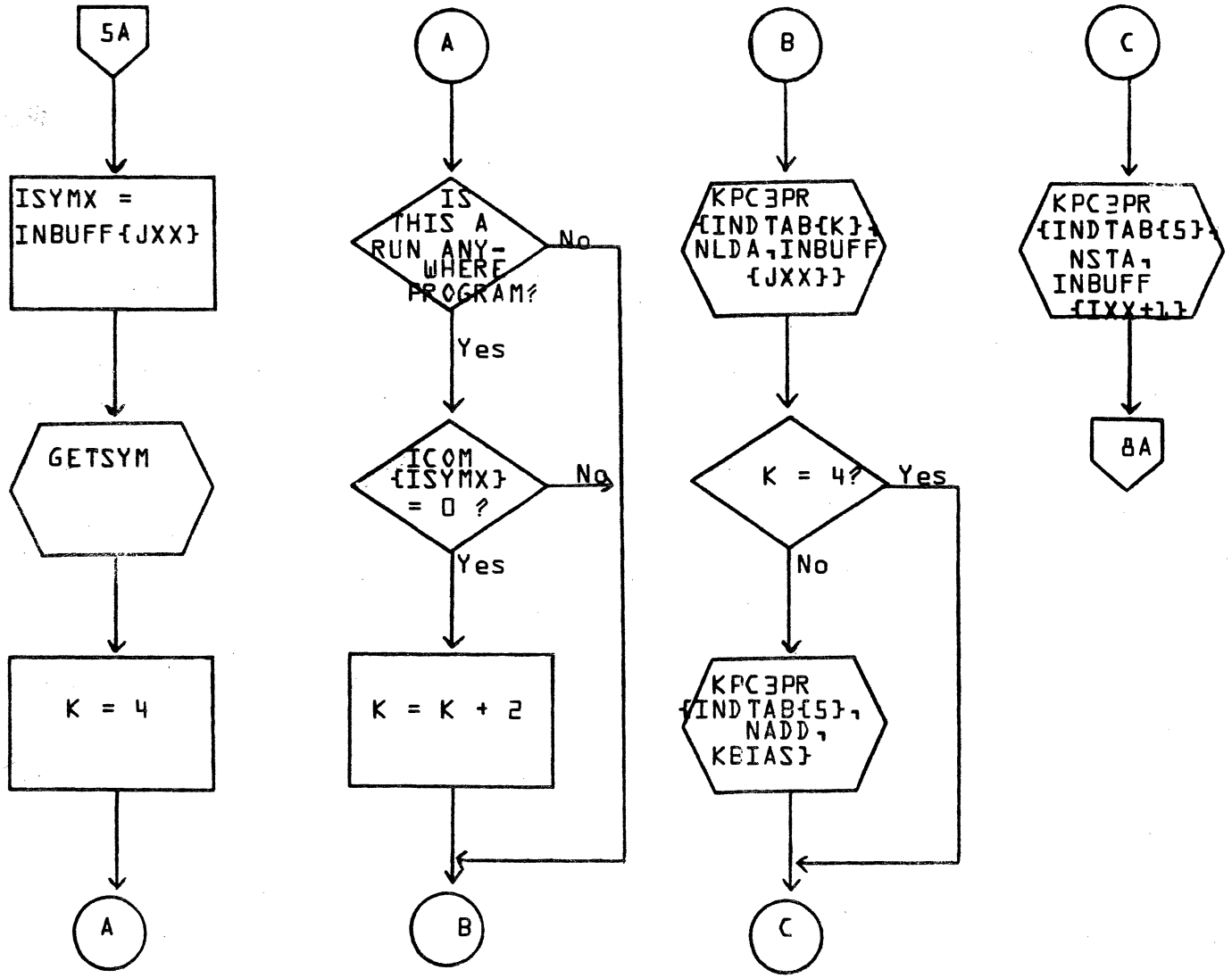
LA JOLLA RESOURCE CENTER



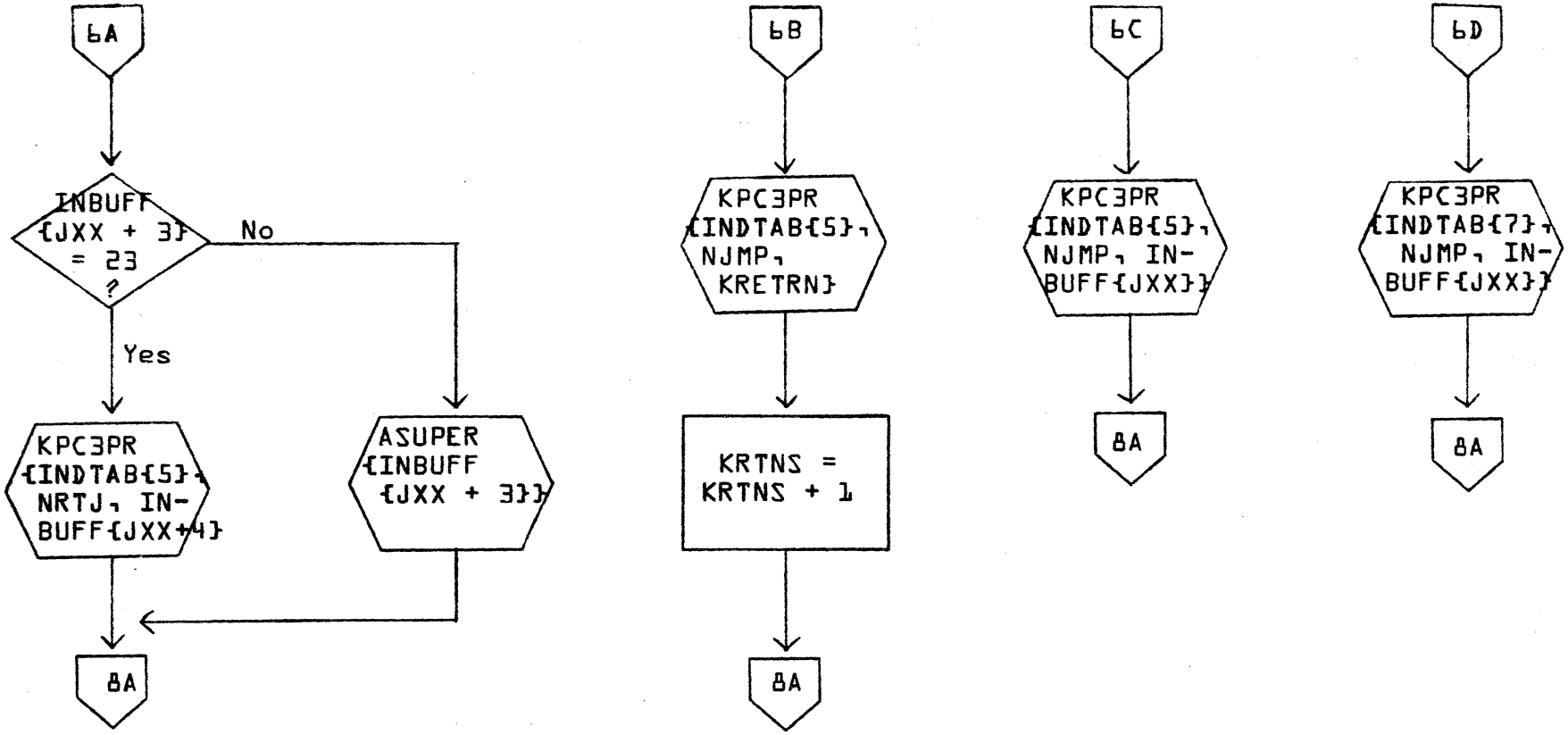
TITLE PHASEB			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3A OF 20



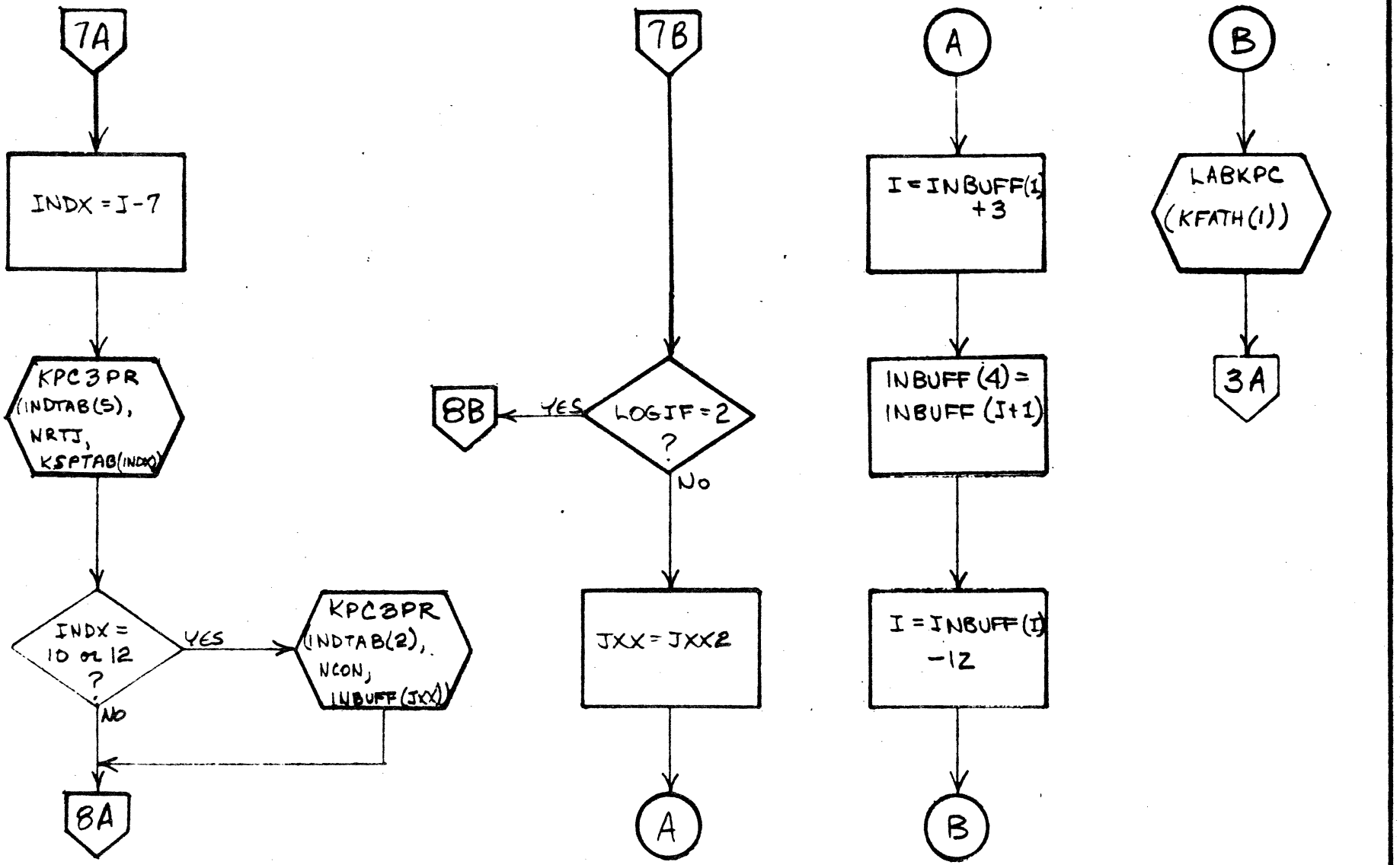
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASE B	PROJECT MGR.			
	PAGE 4 OF 20	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3 6	TASK NAME			



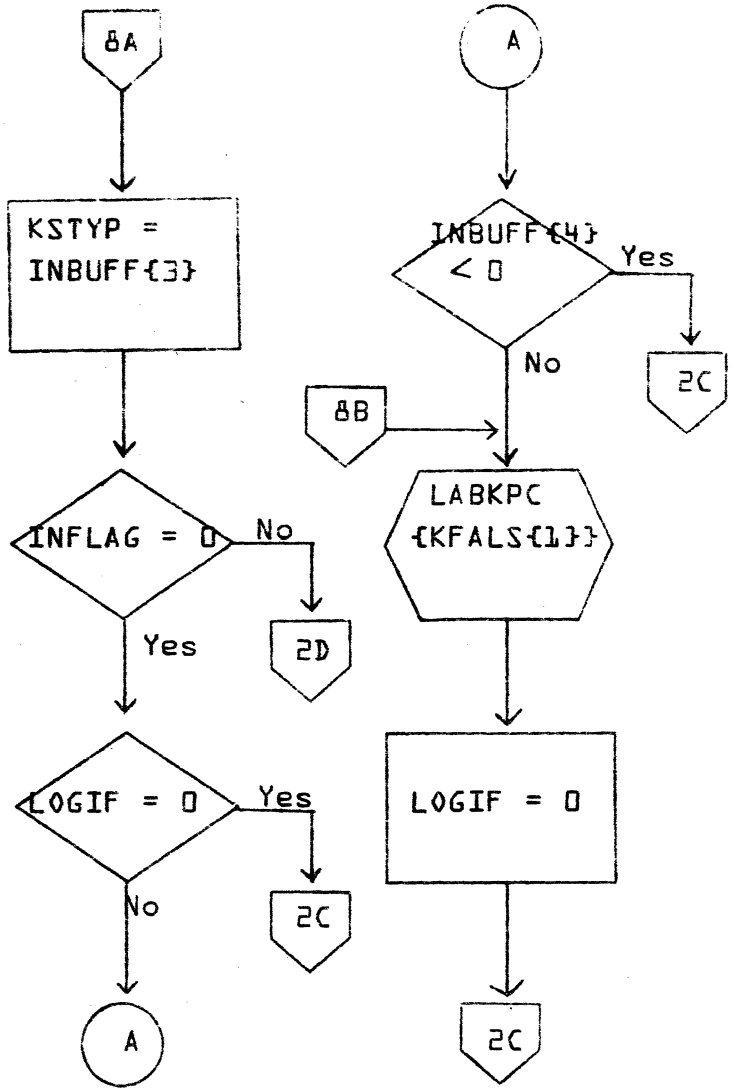
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PHASE B		PROJECT MGR.		
		PAGE 5 OF 20	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASEB		PROJECT MGR.			
		PAGE 6 OF 20	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

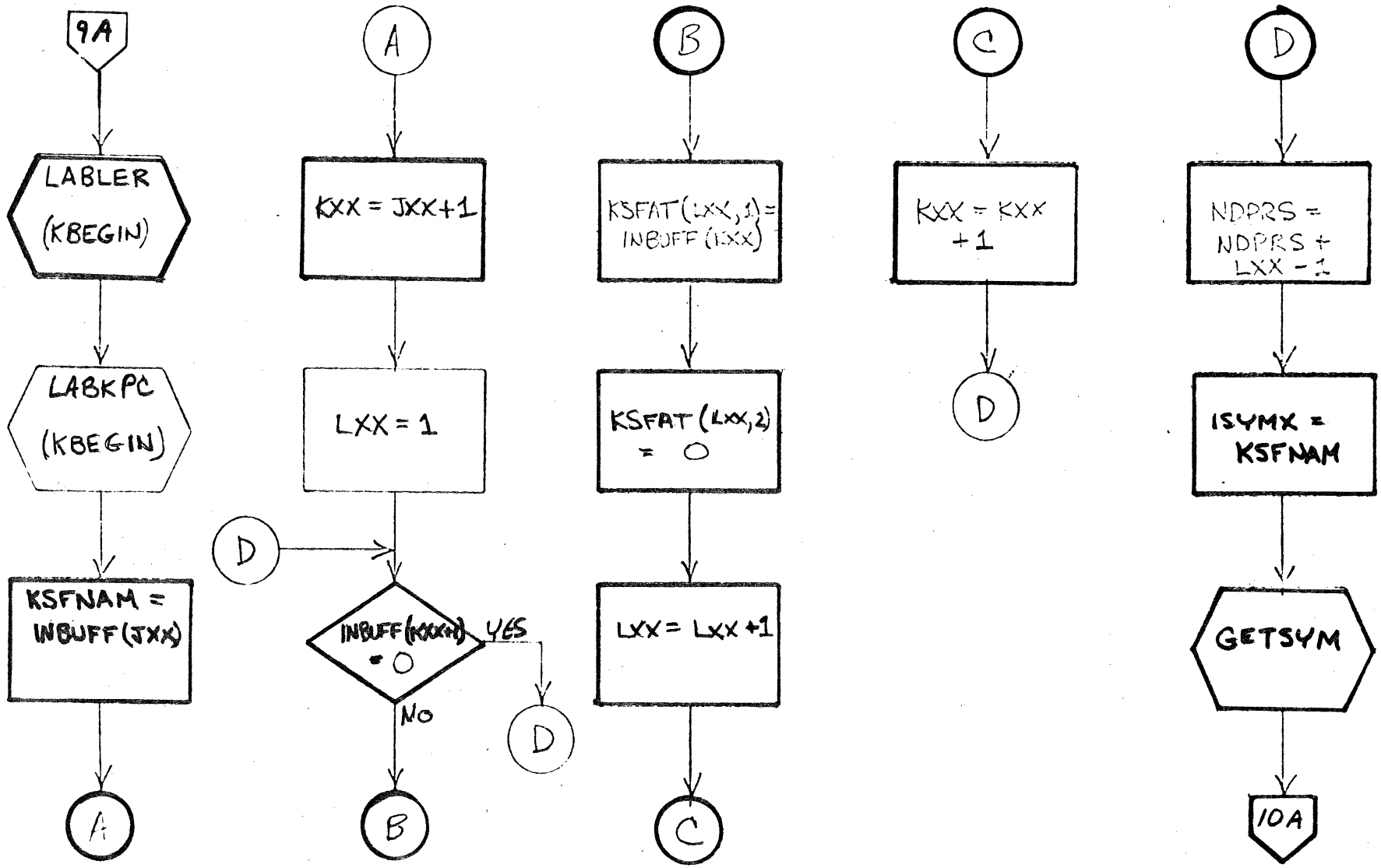


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PHASE B		PROJECT MGR.			
FLOWCHART ☒		PAGE 7 OF 20	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY	DATE 3-6	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PHASE 4		PROJECT MGR.			
FLOWCHART ☐		PAGE 8 OF 8	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

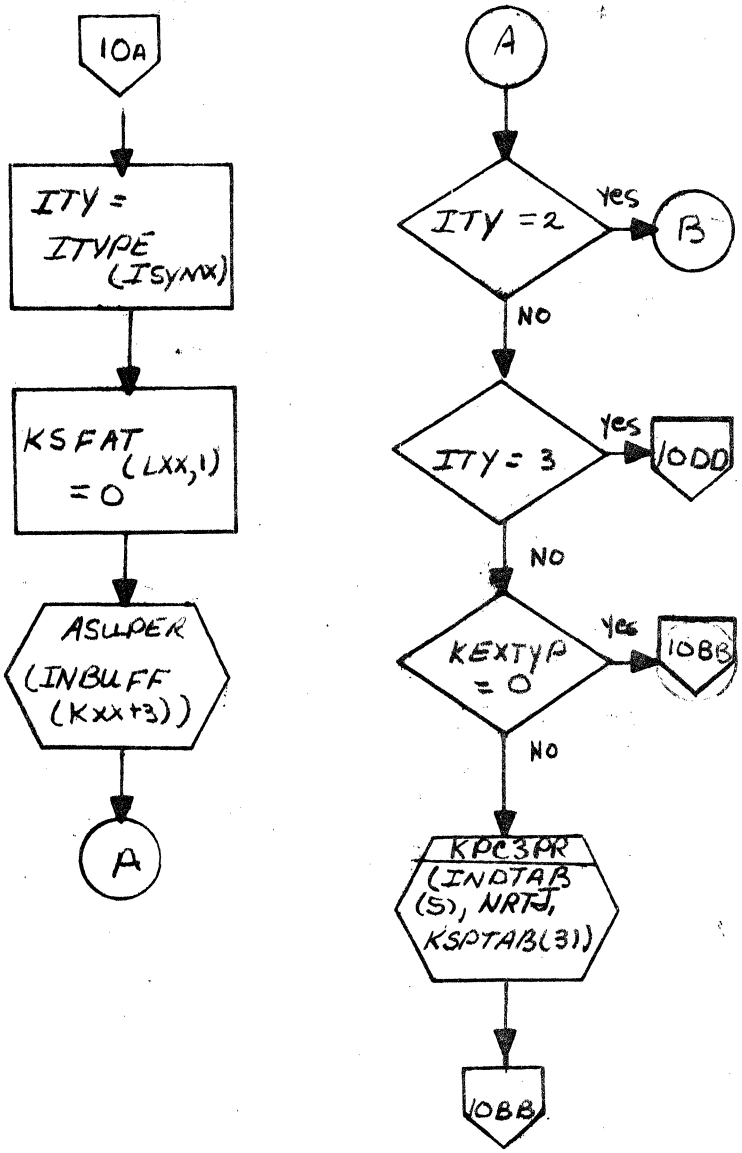
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV. / APPROVED / DATE
DOCUMENT TITLE: PHASE B		PROJECT MGR.	
	PAGE 9 OF 20	PROJECT NAME	
REV. LTR.	ISSUE DATE	TASK NO.	
DRAWN BY	DATE	TASK NAME	

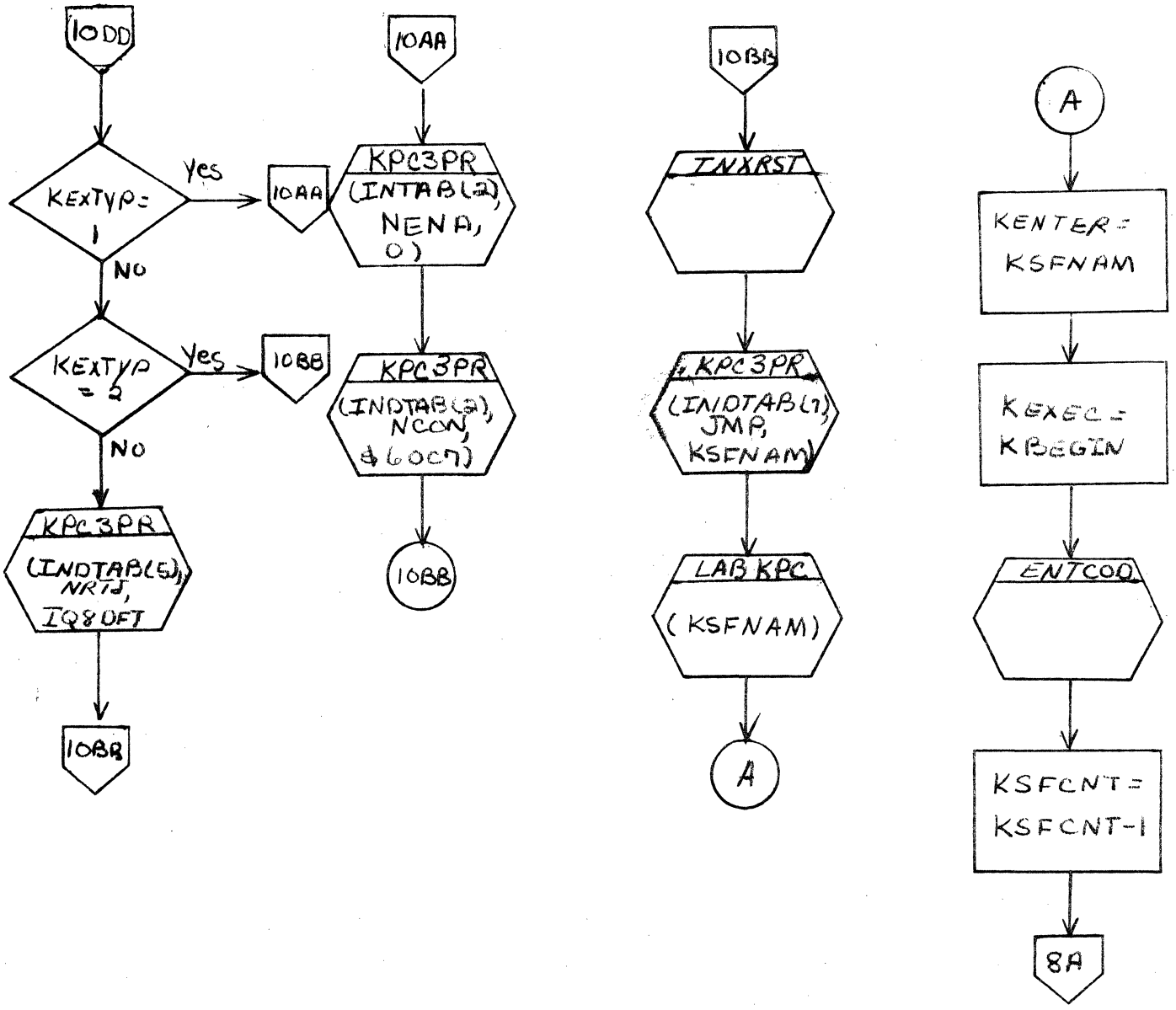


TITLE PHASEB			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 10 OF 20

CSD 203

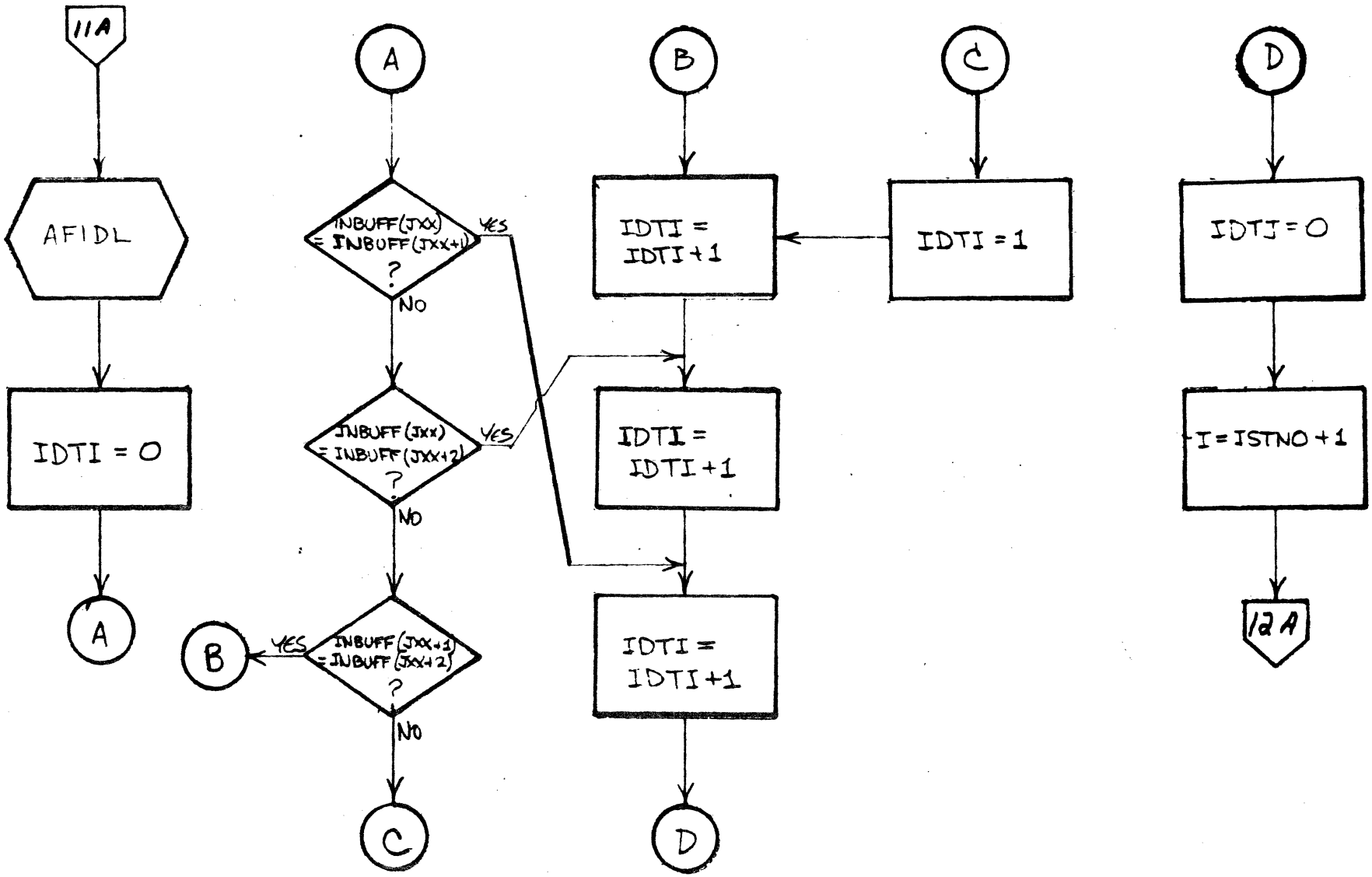
CONTROL DATA CORPORATION

LA JOLLA FACILITY



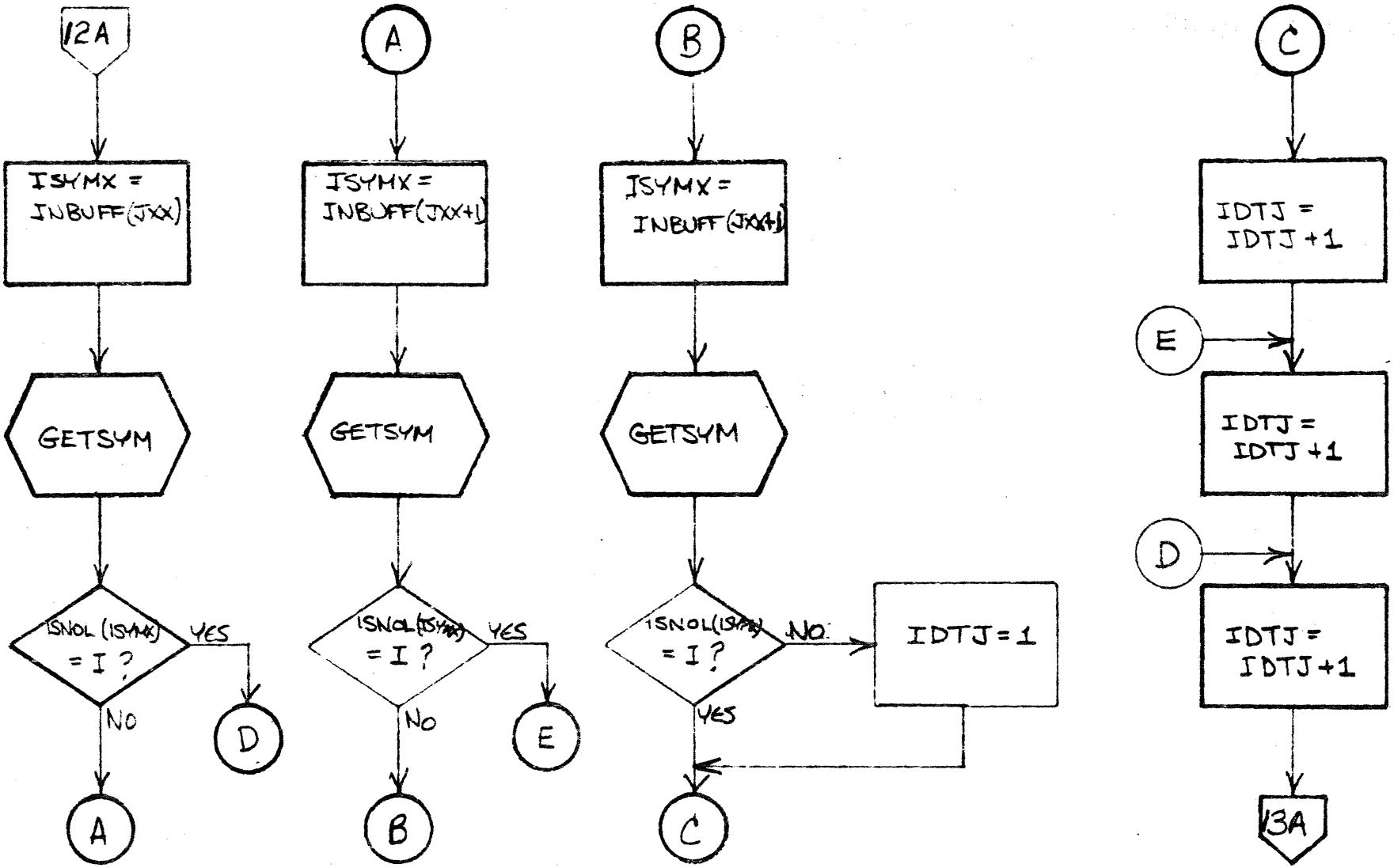
TITLE	PHASEB		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 10A OF 20

CSD 203

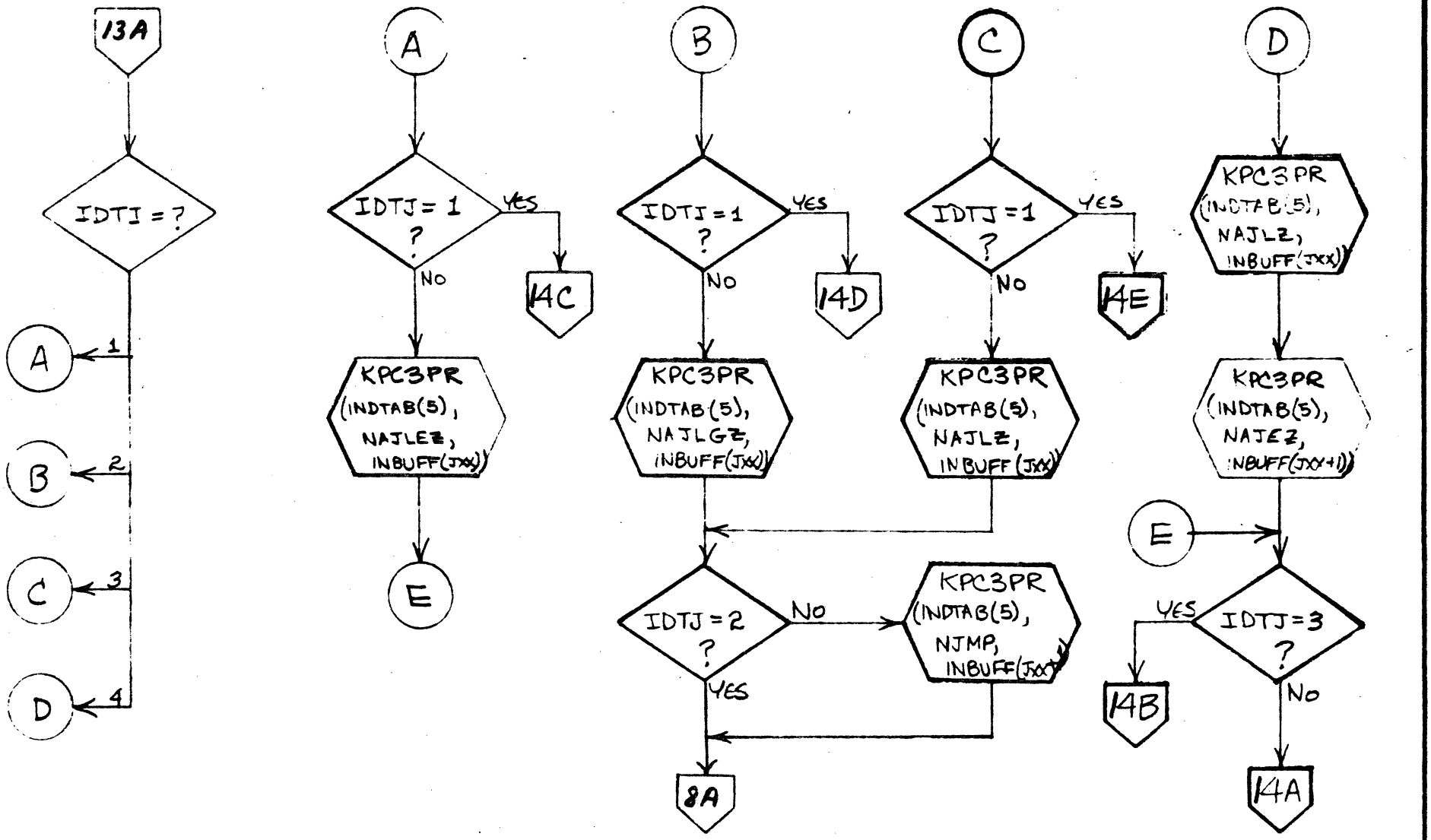


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700 DOCUMENT TITLE PHASE B PAGE 11 OF 20 NUMBER ISSUE DATE DRAWN BY DATE 5/6/7	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

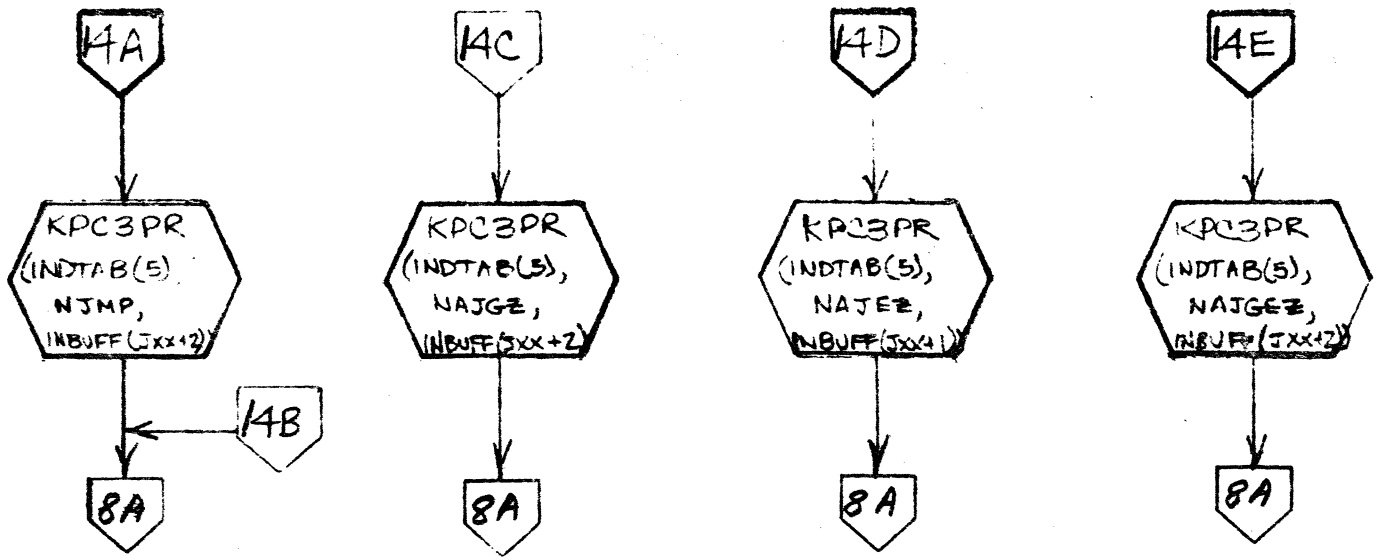
Eh-h



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASE B	PROJECT MGR.			
	PAGE 12 OF 20	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3 67	TASK NAME			

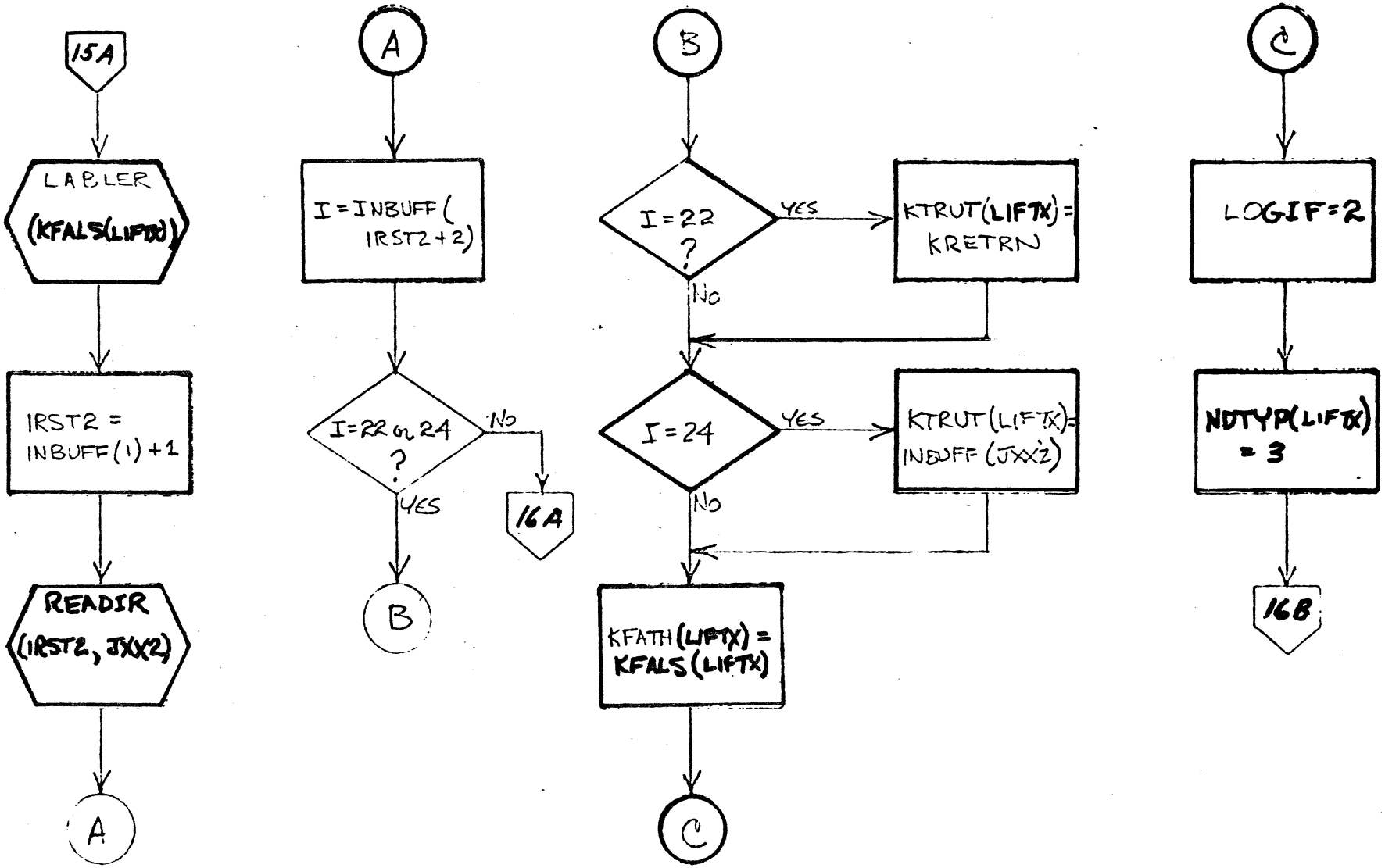


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASEB	PROJECT MGR.			
	PAGE 13 OF 20	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			

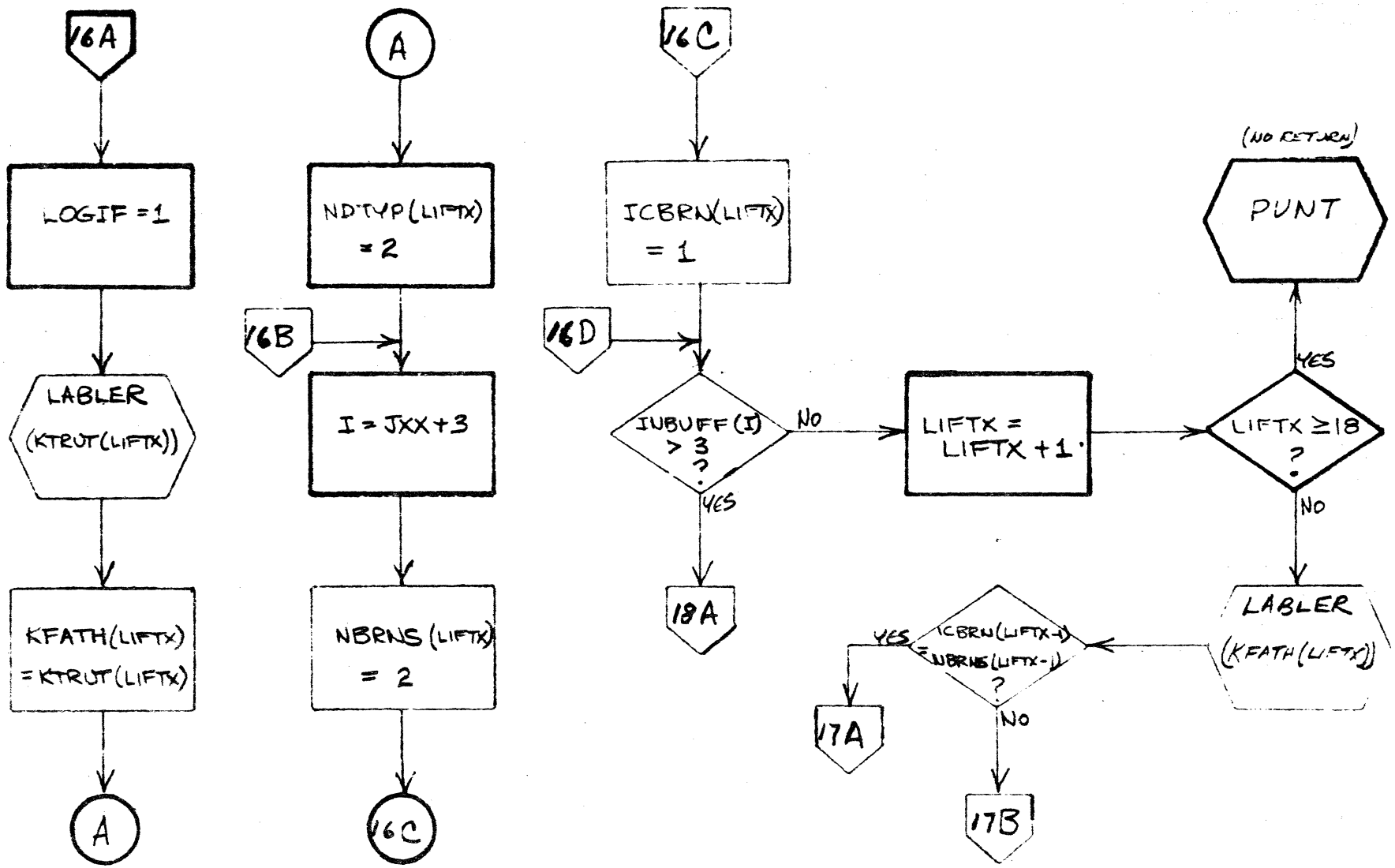


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASE B	PROJECT MGR.			
	PAGE 14 OF 20	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 7-67	TASK NAME			

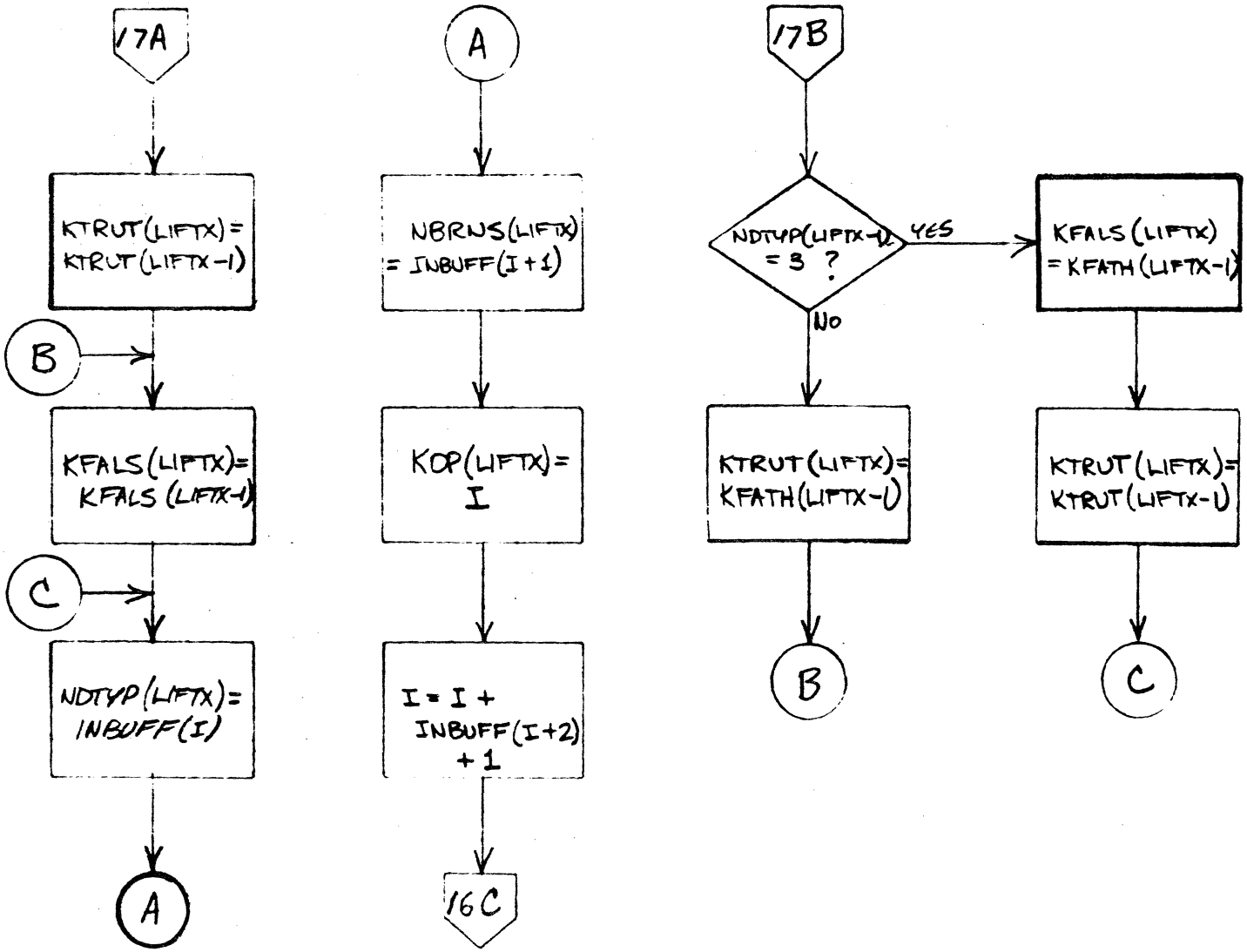
AA1388 (FORMERLY CA127-1)



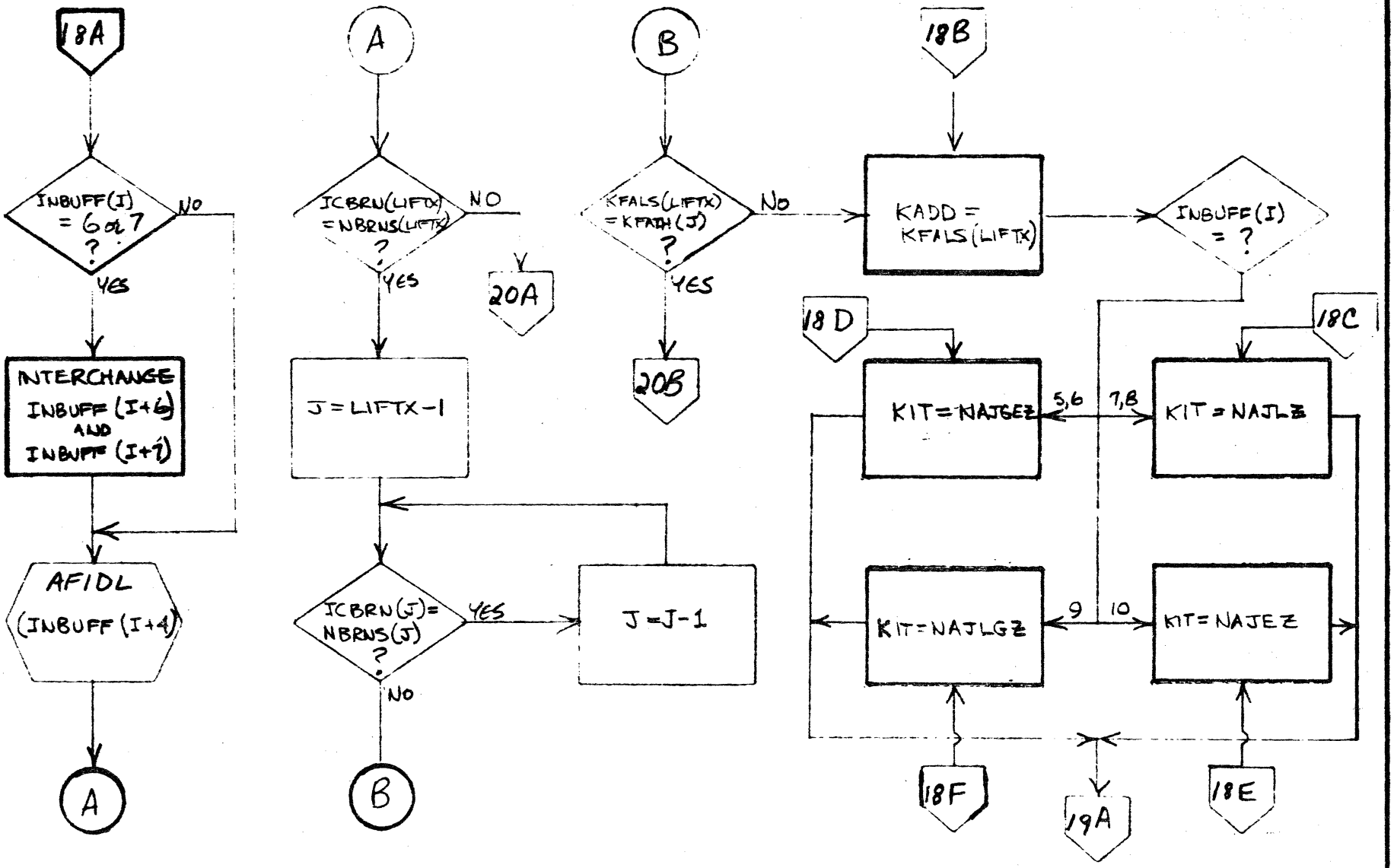
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PHASE B		PROJECT MGR.		
		PAGE 15 OF 20	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 6/1	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE PHASE B PAGE 16 OF 20 NUMBER ISSUE DATE DRAWN BY DATE 3-67	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

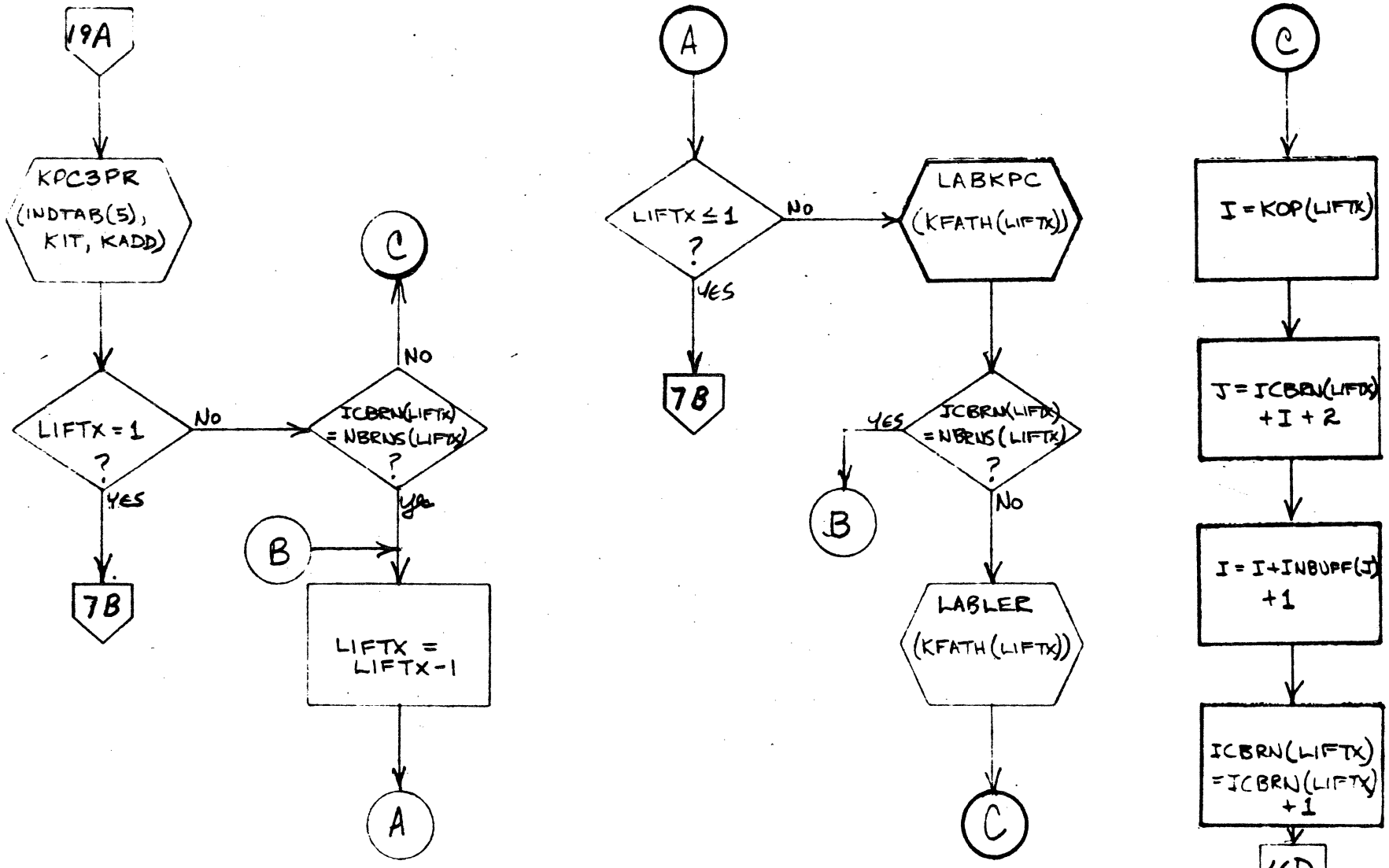


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASE B		PROJECT MGR.			
		PAGE 17 OF 20	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 3-67	TASK NAME			

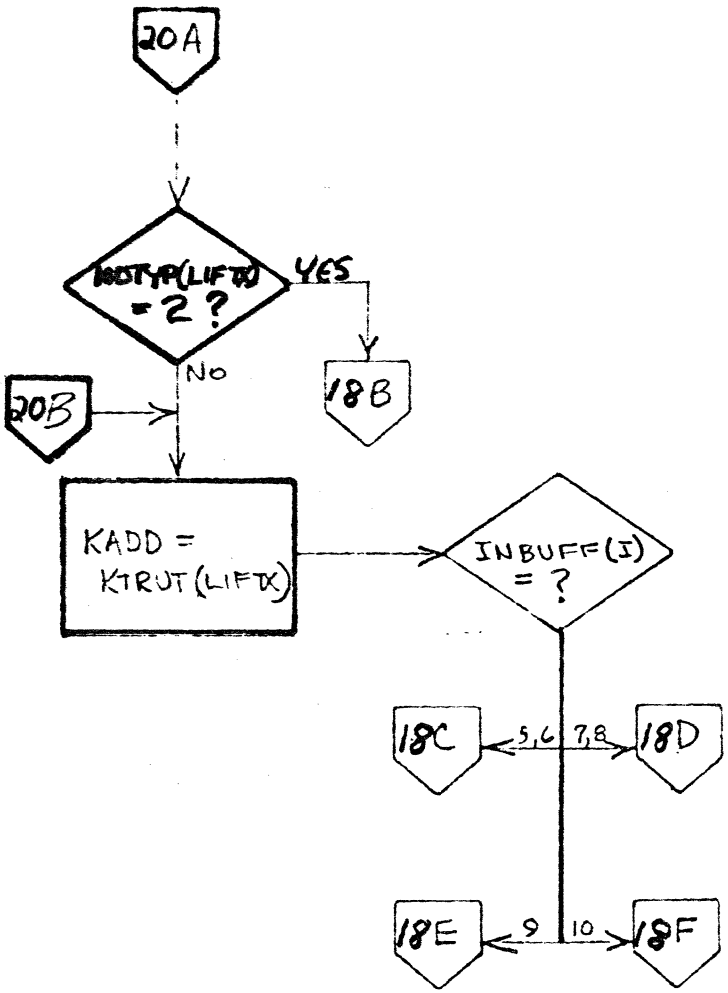


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE PHASEB PAGE 18 OF 20 NUMBER ISSUE DATE DRAWN BY DATE 3-67	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

4-50



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PHASE B	PROJECT MGR.			
	PAGE 19 OF 20	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

- SAMPLE CODE
- FLOWCHART [X]
- DECISION TABLE
- OTHER

DOCUMENT CLASS: IMS MACH TYPE: 1700
DOCUMENT TITLE: PHASEB
PAGE 20 OF 20
NUMBER ISSUE DATE
DRAWN BY DATE 3-1-71

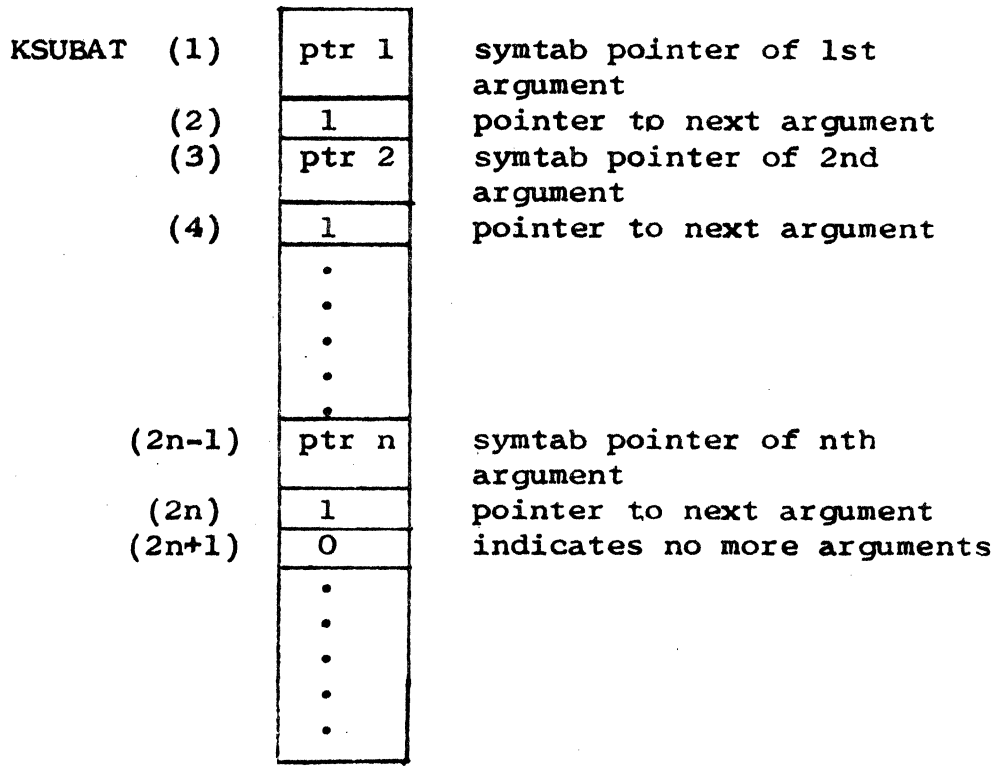
PROJECT NO.
PROJECT NO.
PROJECT NAME
TASK NO.
TASK NAME

REV.	APPROVED	DATE

4.3 PHASE B STATEMENT PROCESSORS

4.3.1 SUBFUN

This routine processes both the SUBROUTINE and FUNCTION statements. Its primary task is to transfer the arguments of the subroutine or function subprogram (in the form of symbol table pointers) to KSUBAT, the Subroutine Argument Table. (A detailed outline of the statement formats will be found in 4.4.20) Completion of this process leaves KSUBAT in its initialized form as shown:



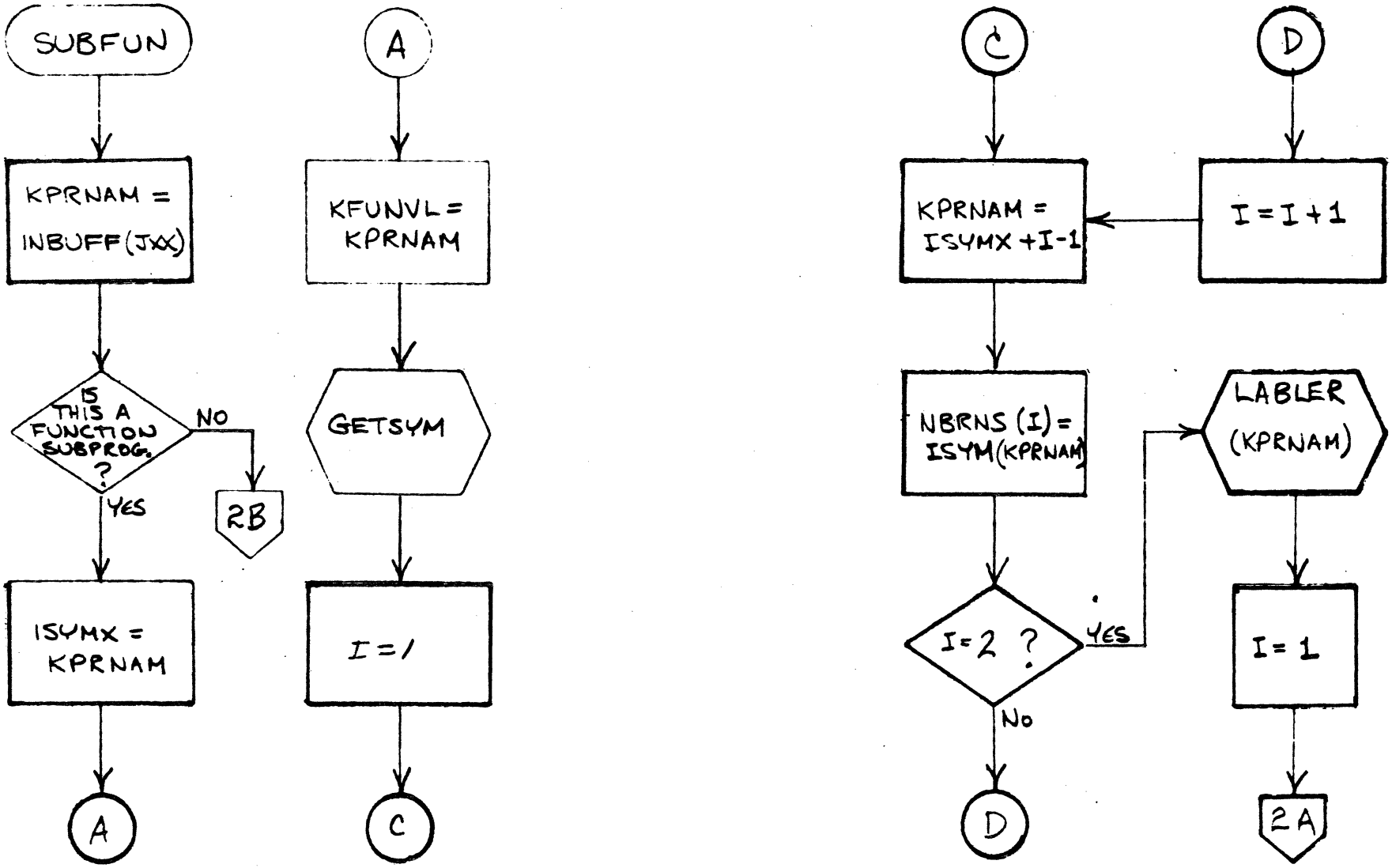
The generalized form of this table is found under the discussion of subroutine interface.

In addition to transfer of arguments SUBFUN also performs the following additional operations:

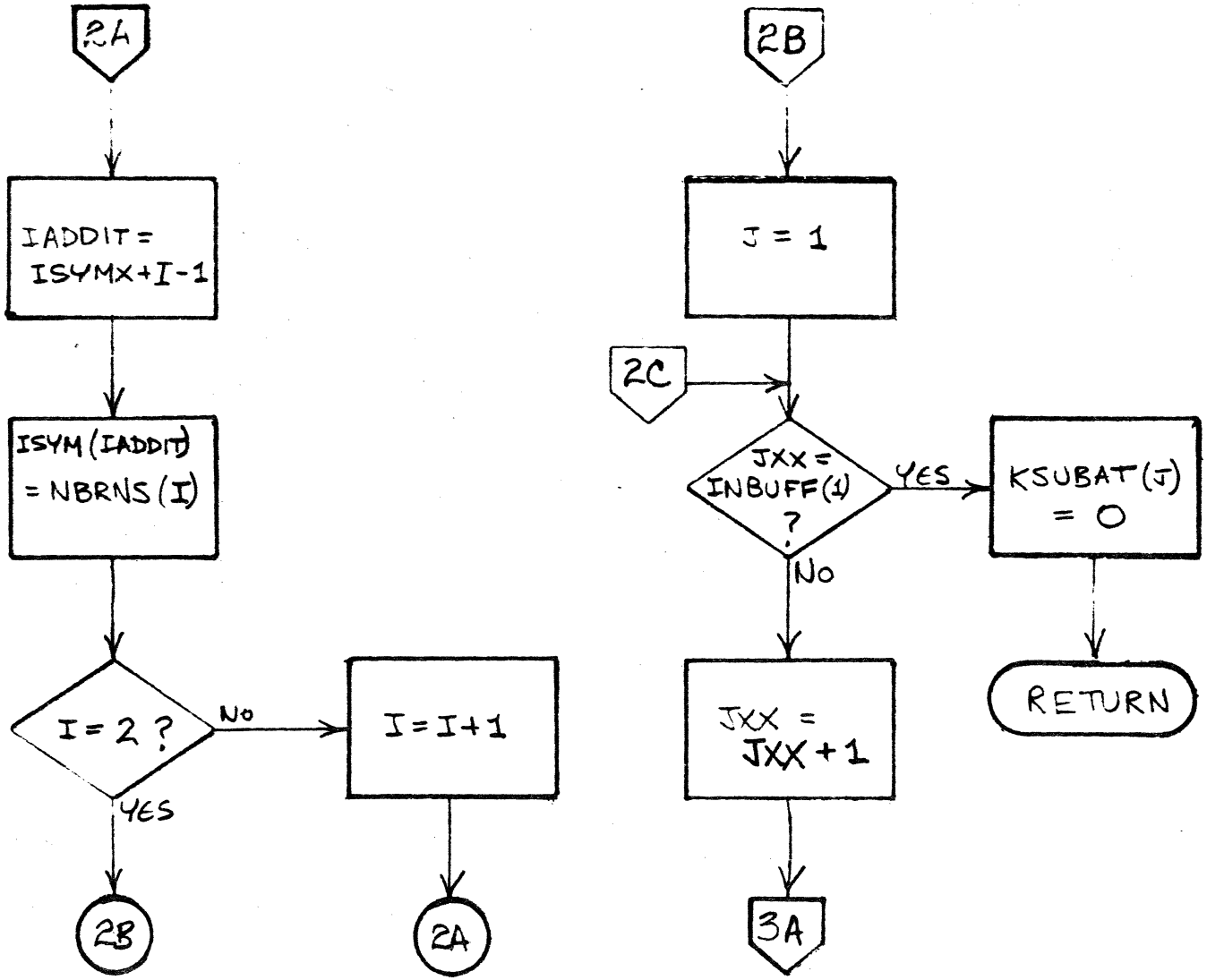
1. transfers the subroutine or function name into the standard holder KPRNAM.

2. In the case of a function subprogram, creates a new holder to contain the results of the function.

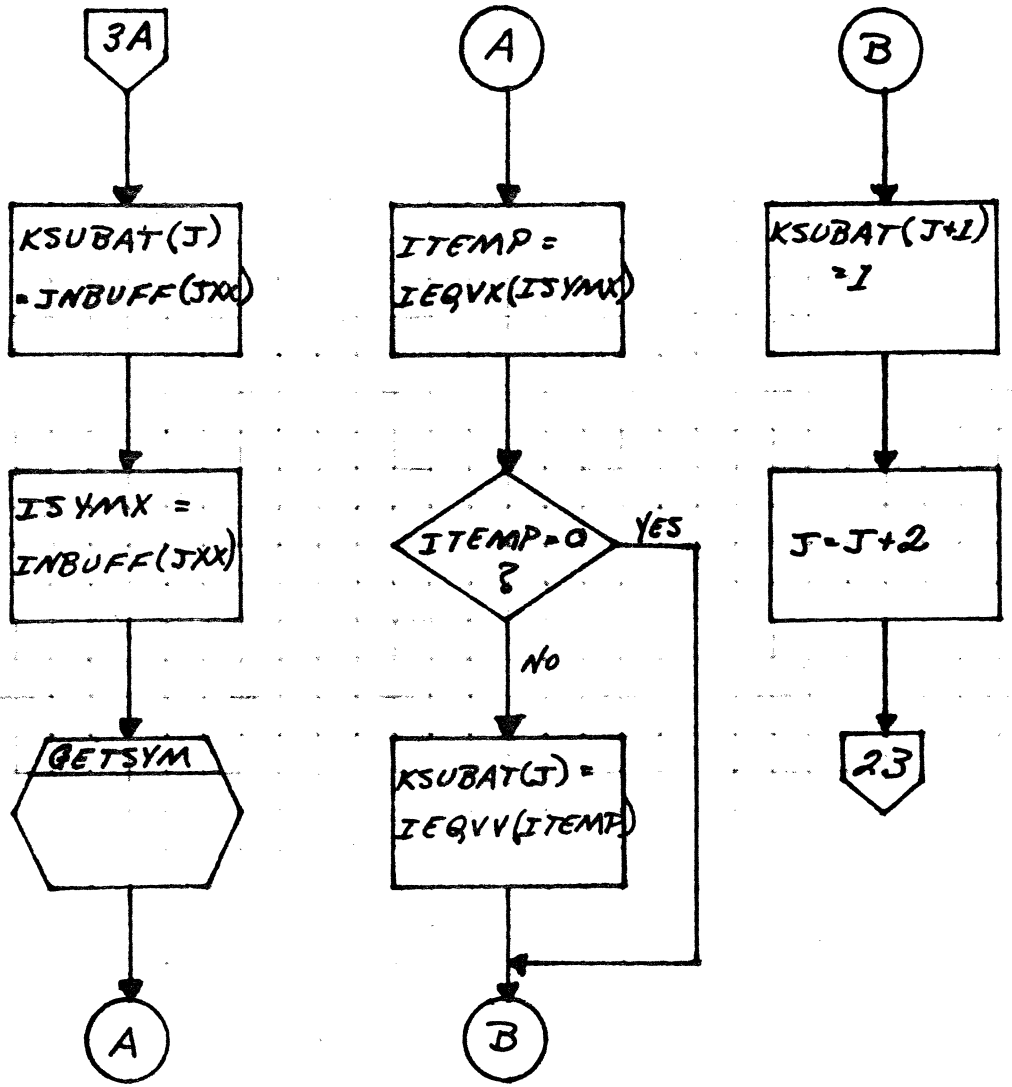
Flowcharts:



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE SUBFUN PAGE 1 OF 3 NUMBER ISSUE DATE DRAWN BY DATE 3-67	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBFUN	PROJECT MGR.			
	PAGE 2 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBFUN	PROJECT MGR.			
	PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

4.3.2 NOPROC

NOPROC's basic function is to transfer from the input buffer to the output file of PHASE B records of those statements which require no direct processing.

Operating is as follows:

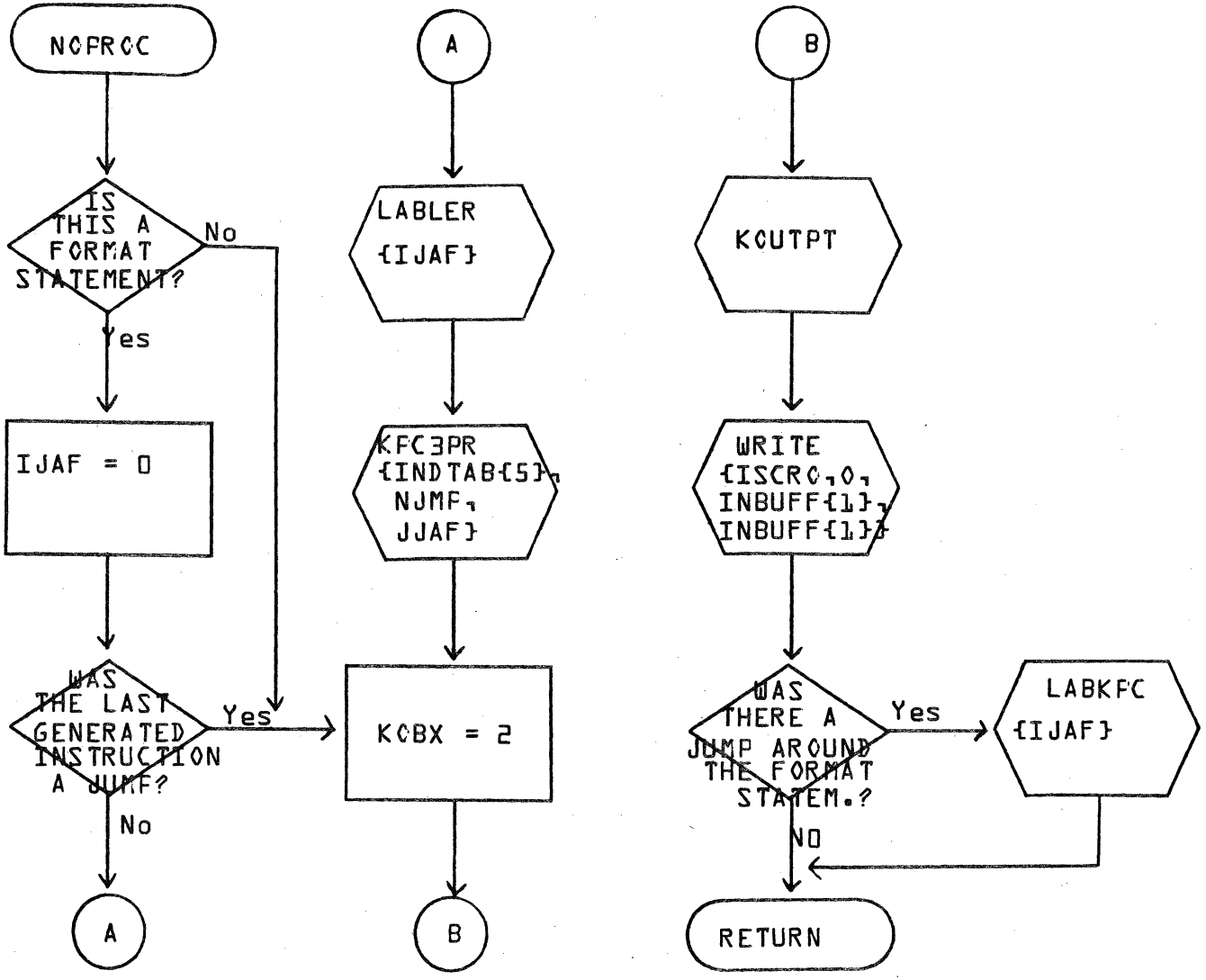
1. If statement is a format and if the last instruction generated by PHASE B was not an unconditional jump: generate a label and output a jump to that label.

2. Output special flag record which indicates that the record following is a DATA or FORMAT (see output description).

3. Output the DATA or FORMAT record exactly as it was input.

4. If a jump around a FORMAT was generated, then output the label that was generated to jump to.

Flowcharts:



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE NCPROC		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

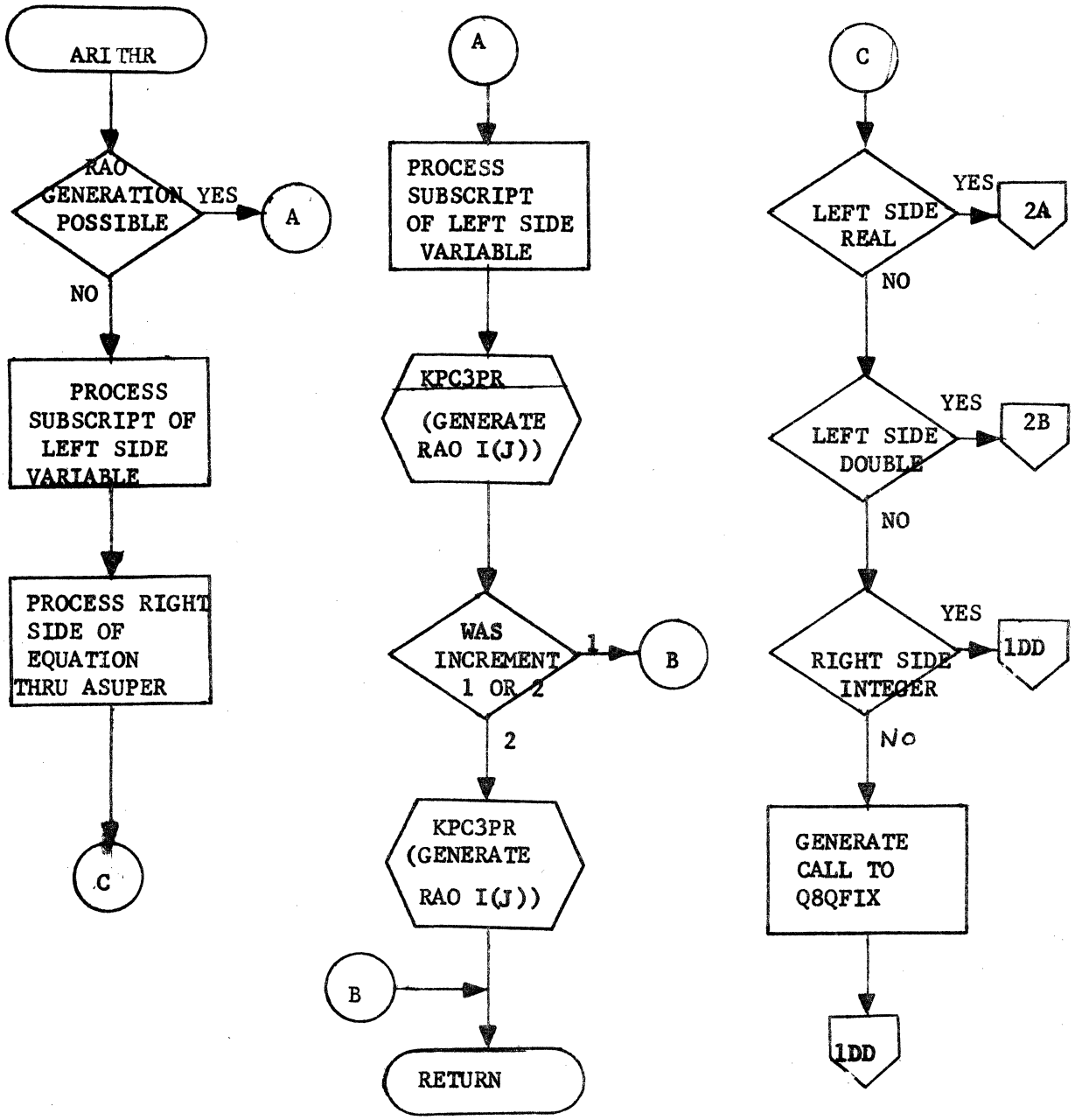
4.3.3 ARITHR

ARITHR is the routine to process the arithmetic replacement statement. It first checks for the special case I{J}=I{J}+1 or 2, meaning that a non-partial variable is increased by either one or two. This allows for the use of one or two RAO commands in place of the normal sequence of commands. Otherwise the processing proceeds as follows: The subscript of the variable on the left of the equal sign is processed through the subscript expression processor SUBPR1. The variable or expression on the right side of the equal sign {=} is processed through the arithmetic expression processor, ASUPER. Finally, the STA, FST, or DST is generated into the left side variable preceded, if necessary, by a necessary call to a conversion routine {real to integer, integer to real, integer to double precision, etc.} whenever there is a difference between the mode of the right side expression and the left side variable. When the left side variable is a partial, code is generated to store the value into that partial.

Flowcharts:

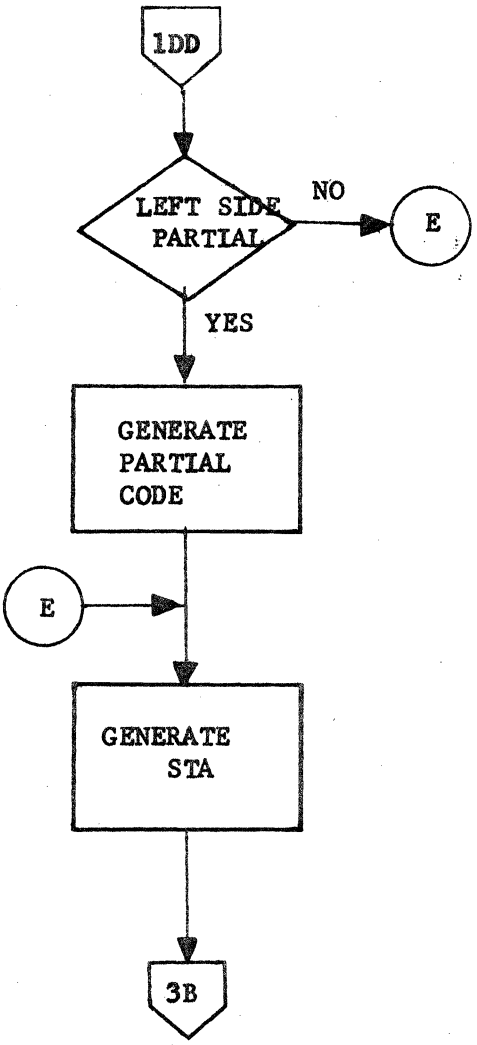
CONTROL DATA

LA JOLLA RESOURCE CENTER



TITLE ARITHR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 1 OF 3

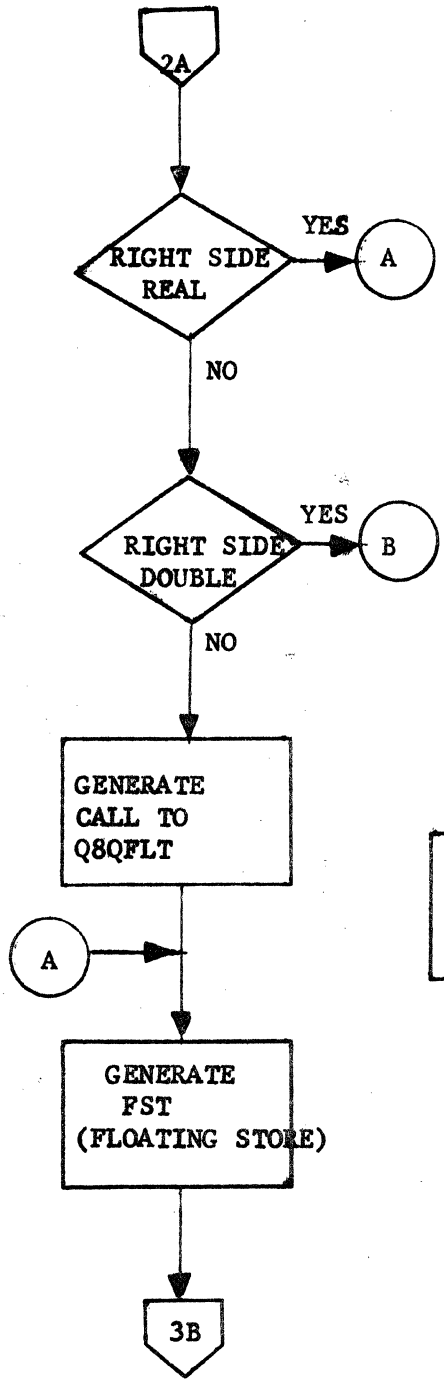
CSD 203 ANALYSIS

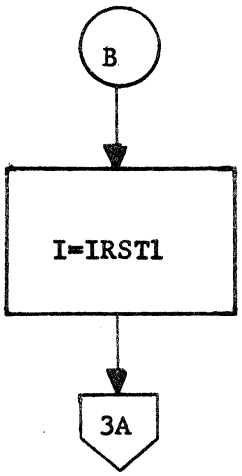

1DD
LEFT SIDE PARTIAL
NO
E
YES
GENERATE PARTIAL CODE
E
GENERATE STA
3B

CONTROL DATA CORPORATION

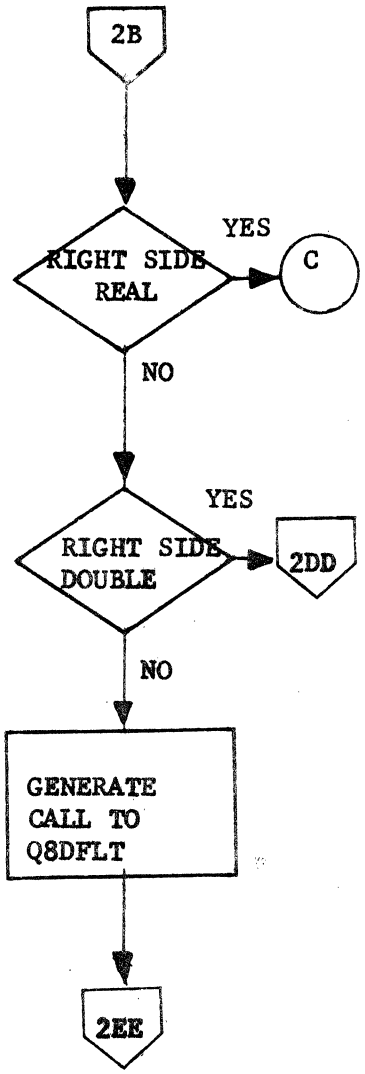
LA JOLLA RESOURCE CENTER

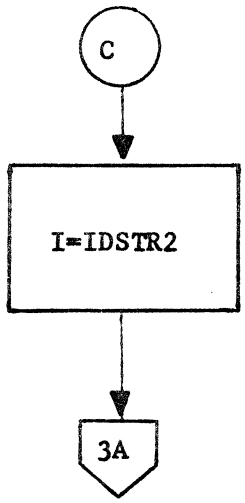
LEFT SIDE IS REAL





LEFT SIDE IS DOUBLE



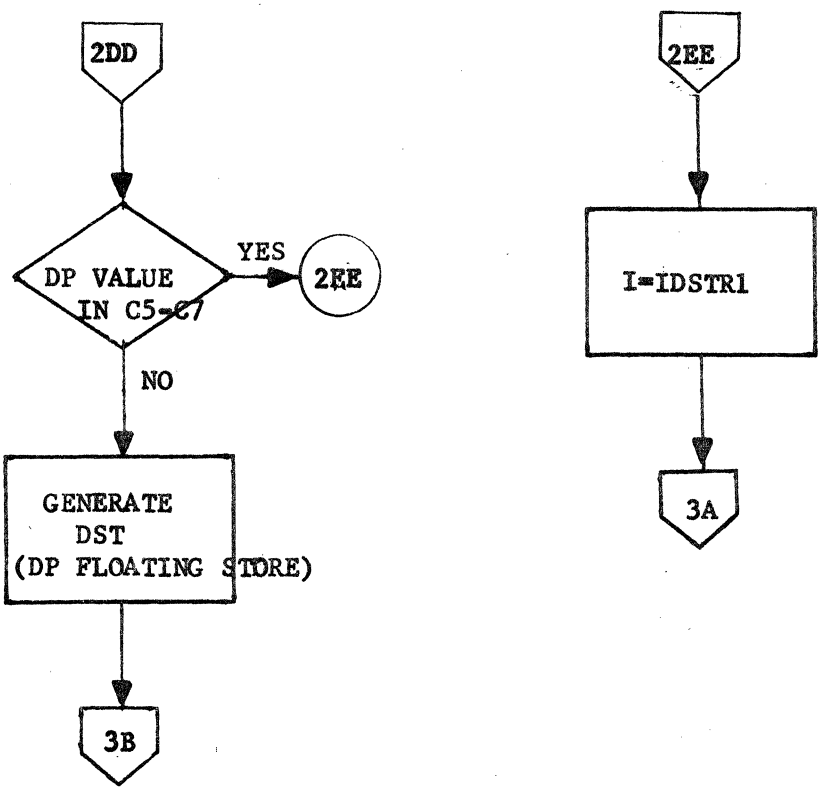


TITLE ARITHR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2 OF 3

CSD 203 ANALYSIS

CONTROL DATA CORPORATION

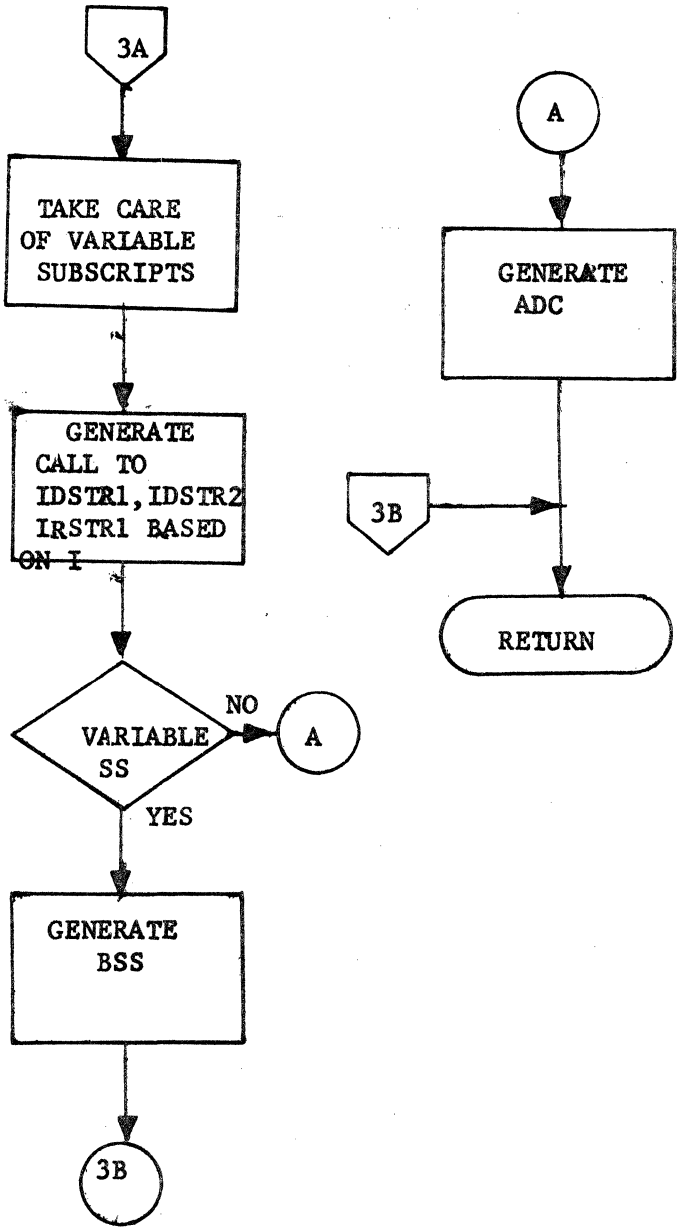
LA JOLLA RESOURCE CENTER



TITLE	ARITHR		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2A OF 3

CONTROL DATA CORPORATION

LA JOLLA RESOURCE CENTER



TITLE ARITHR			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF 3

CSD 202 ANALYSIS

4.3.5 CGØTØ

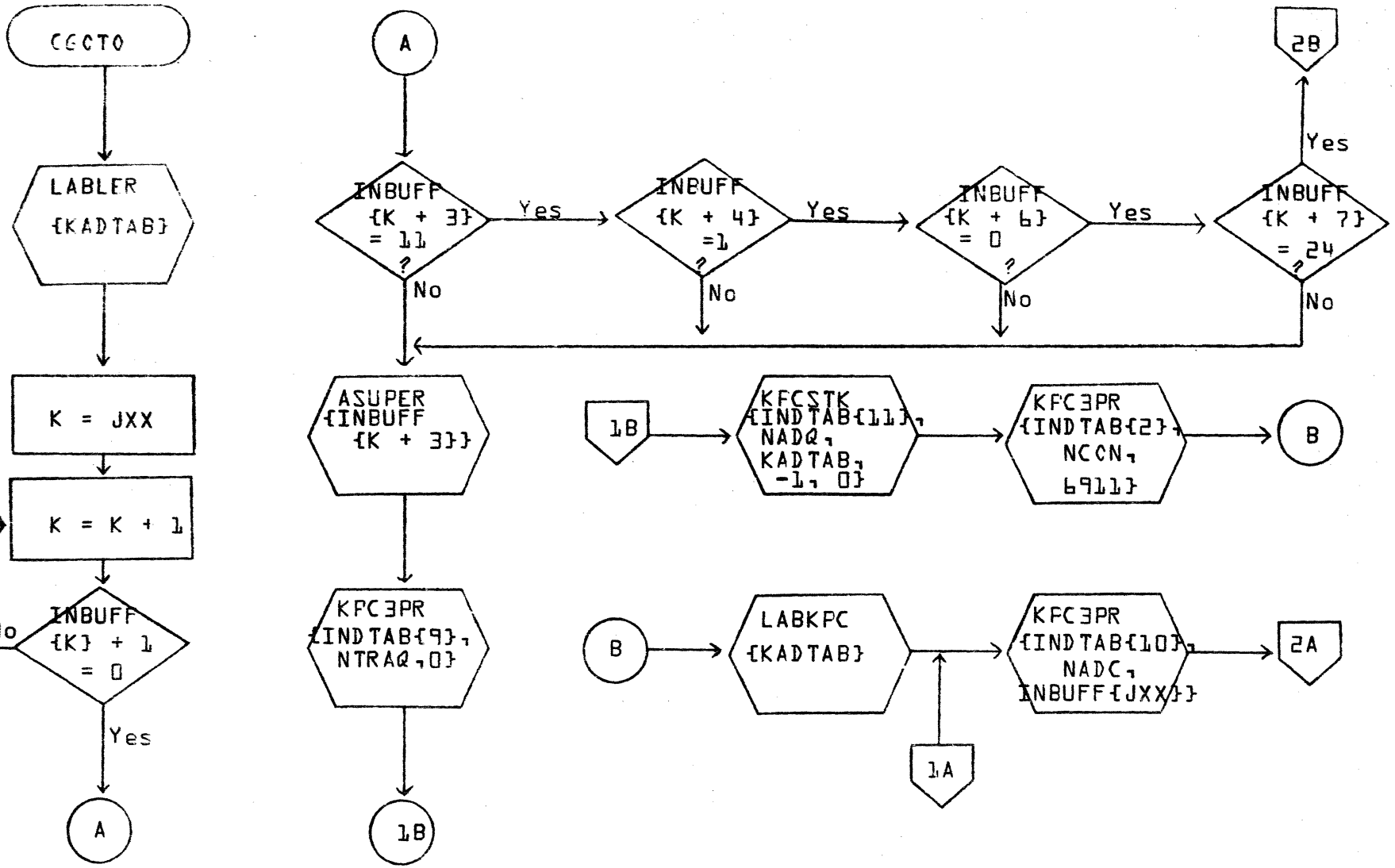
CGØTØ processes the computed GO TO statement in the following manner:

1. Creates a label to label the table of addresses in the instruction generation.

2. Tests if the GO TO expression is only a variable. If so a LDQ command is generated. If not, the instructions are generated to compute the value of the expression followed by a TRAQ command to place the value in Q.

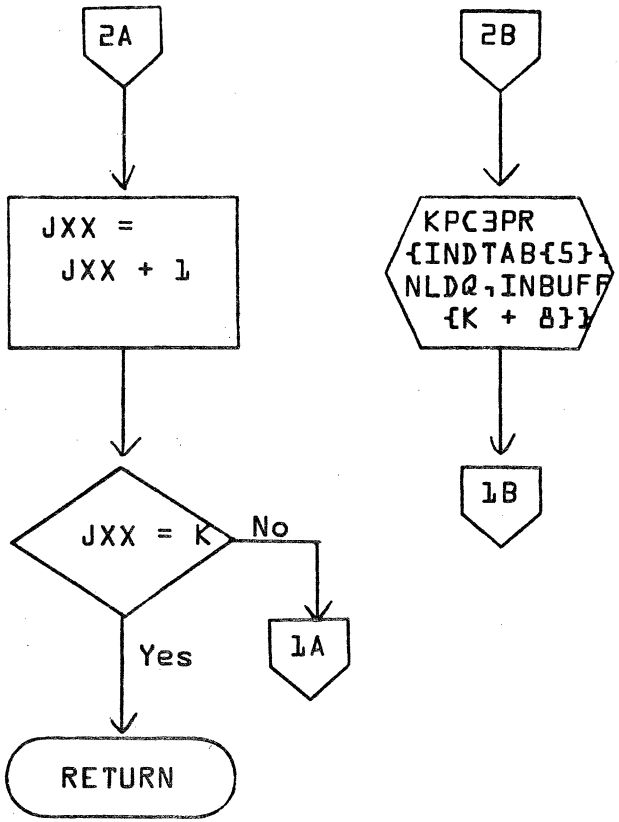
3. Instructions are generated to index into and jump through the proper table entry.

4. The label for the address table is generated followed by the table of addresses corresponding to the label list in the computed GO TO.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE CGCTO	PROJECT MGR.			
	PAGE 1 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

4-14



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE CGOTO	PROJECT MGR.			
	PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

4.3.6 END

END processes the END statement in the following manner:

1. For block data and main programs, the "end" pseudo-instruction is generated signifying the end of the Phase B output and Phase C input.

2. For subroutines and function subprograms, the label of reference for all return statements is output followed by instruction generation of instructions to restore index registers.

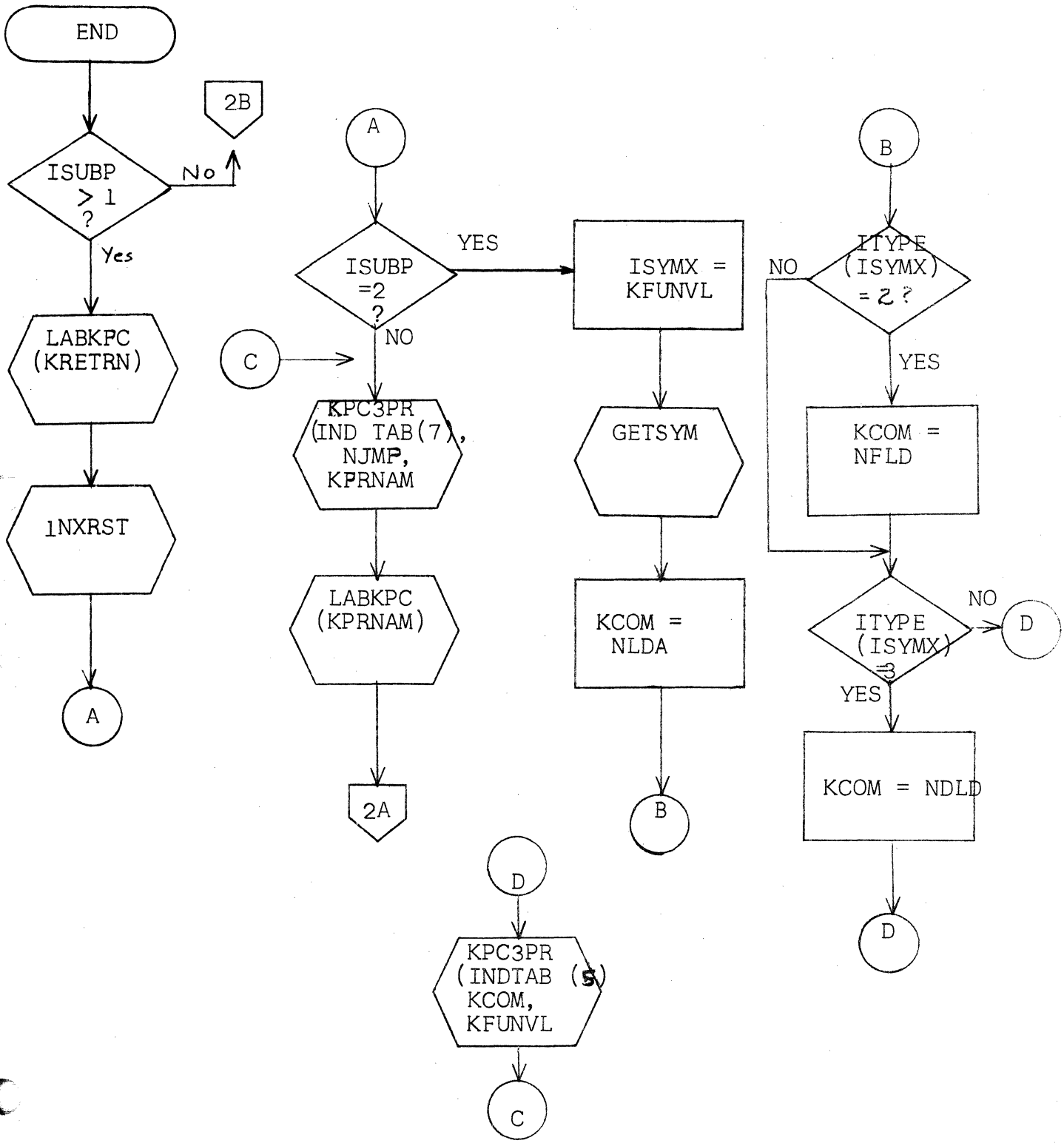
3. For function subprograms, another instruction is then generated to load the value of the function (either LDA, FLD, or DLD).

4. For subroutines and function subprograms, then a jump command through the entry point is generated followed by output of the entry point label. Then the parameters are set up and ENTCOD is executed to generate the parameter pickup instruction sequence. Then the "end" is generated as in step 1.

5. Finally the SYMTAB points to the program name is transferred to holder for use by Phase C.

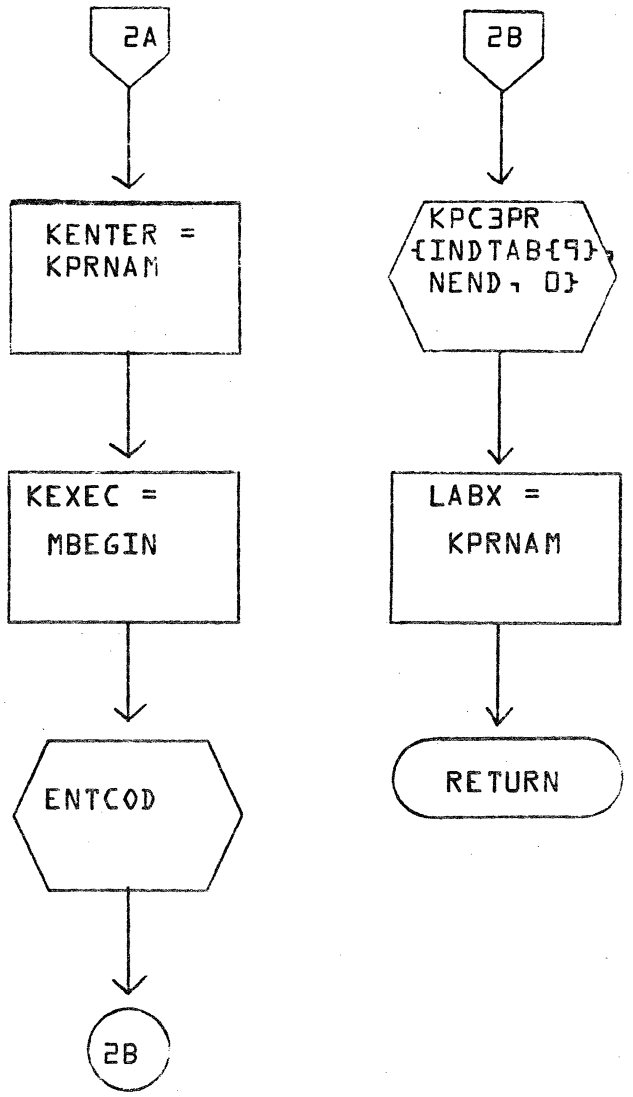
CONTROL DATA CORPORATION

LA JOLLA RESOURCE CENTER



TITLE	END		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 1 OF 2

CSD 203 ANALYSIS



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE END		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 2	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

4.3.7 BGINDØ and BANANA

DØ statements are reorganized by Phase 1 into two separate statements Begin Do and End Do processed in Phase 4 by BGINDO and BANANA respectively. Detailed descriptions of the philosophy and processing of the DO statement can be found in Phase 1.

The Begin Do occurs physically in the same position in the Phase 4 input file as the DO statement does in the FORTRAN source input. The END DØ corresponding to a BEGIN DØ follows the terminal statement of the associated DØ and precedes the next statement. When more than one DO statement has the same terminal statement the End Do's will occur consecutively in the output file, the first one corresponding to the innermost Begin Do, etc.

Both BGINDO and BANANA generate executable instructions which work together to perform the loop. In generating executable instructions both core space and execution time was taken into consideration. Several factors enter into the analysis:

1. Class of the initial parameter (i.e. constant or variable).
2. Class of terminal parameter.
3. Class of the incrementation parameter.
4. Whether the loop is an incrementing or decrementing loop.
5. In certain cases, whether or not the initial parameter is a constant of 1.
6. In certain cases, whether or not the terminal parameter is a constant of 1.
7. In certain cases, whether or not the incrementation parameter is a constant of 1.

In the vast majority of cases the number of times a loop is executed is a function of the loop parameters (initial, terminal, and incrementation values). If n represents this number then the number of times the terminal value must be compared with the control variable is:

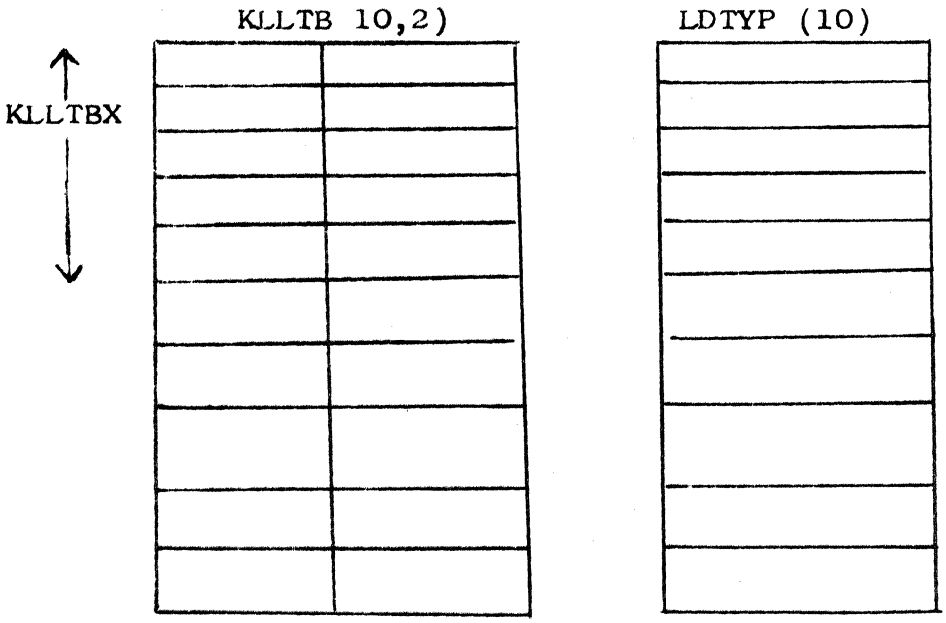
n, if the comparison takes place at the end of the range.
n+1, if the comparison takes place at the beginning of the range.

Thus, whenever it is possible, the instructions which perform this comparison are placed at the end of the loop. The following cases are those where the comparison is made after the first execution of the DO range:

1. The beginning and terminal values of the loop are constants.
2. Decrementing loops where the terminal value is the constant 1.
3. Incrementing loops where the initial value is the constant 1.

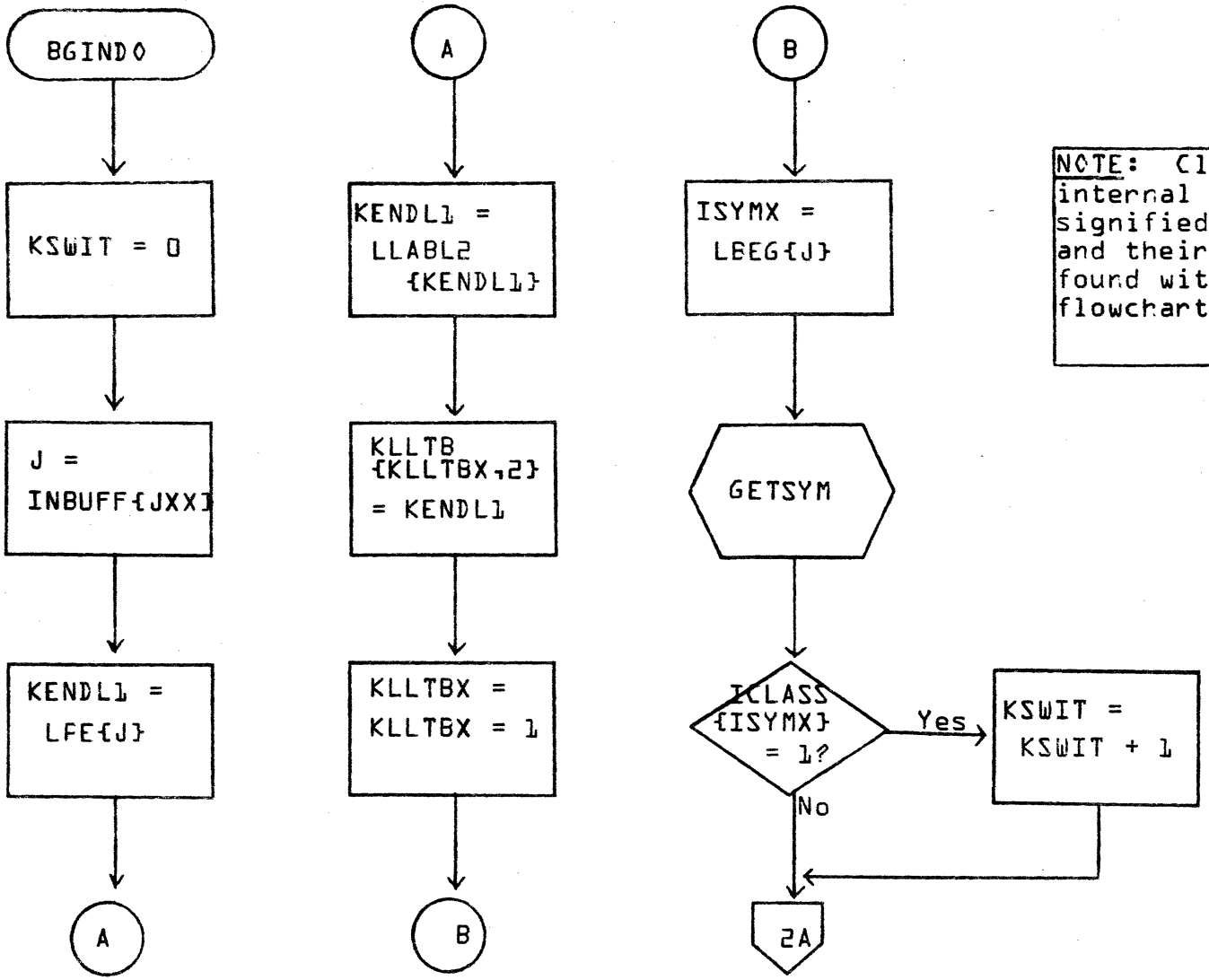
Clearly, these are cases which, by their nature, require that the loop be executed at least once and it is therefore not necessary to make a check before beginning the loop.

BGINDO communicates with BANANA through two tables KLLTB and LPTYP. KLLTB is a ten entry table with two words per entry and LPTYP is a ten entry table with one word per entry as shown:



Both tables are indexed by KLLTBX. BGINDO adds an entry and updates index KLLTBX for each Begin Do encountered. BANANA uses the information in an entry and decrements KLLTBX for each End Do encountered. The first word of the KLLTB entry is used to contain the symtab pointer to a generated label. This label is created by BGINDO to reference the end of a loop (i.e. the instruction to be executed when the loop has been satisfied). This label is only needed (and thus only created) when testing is done at the beginning of the loop. The second word of the KLLTB entry is used to contain the symtab pointed to the top-of-loop label, i.e. the label of the first instruction to be operated whenever the loop range is executed again. LPTYP contains a code that BGINDO uses to communicate with BANANA and designate what instructions should be generated at the end of the loop.

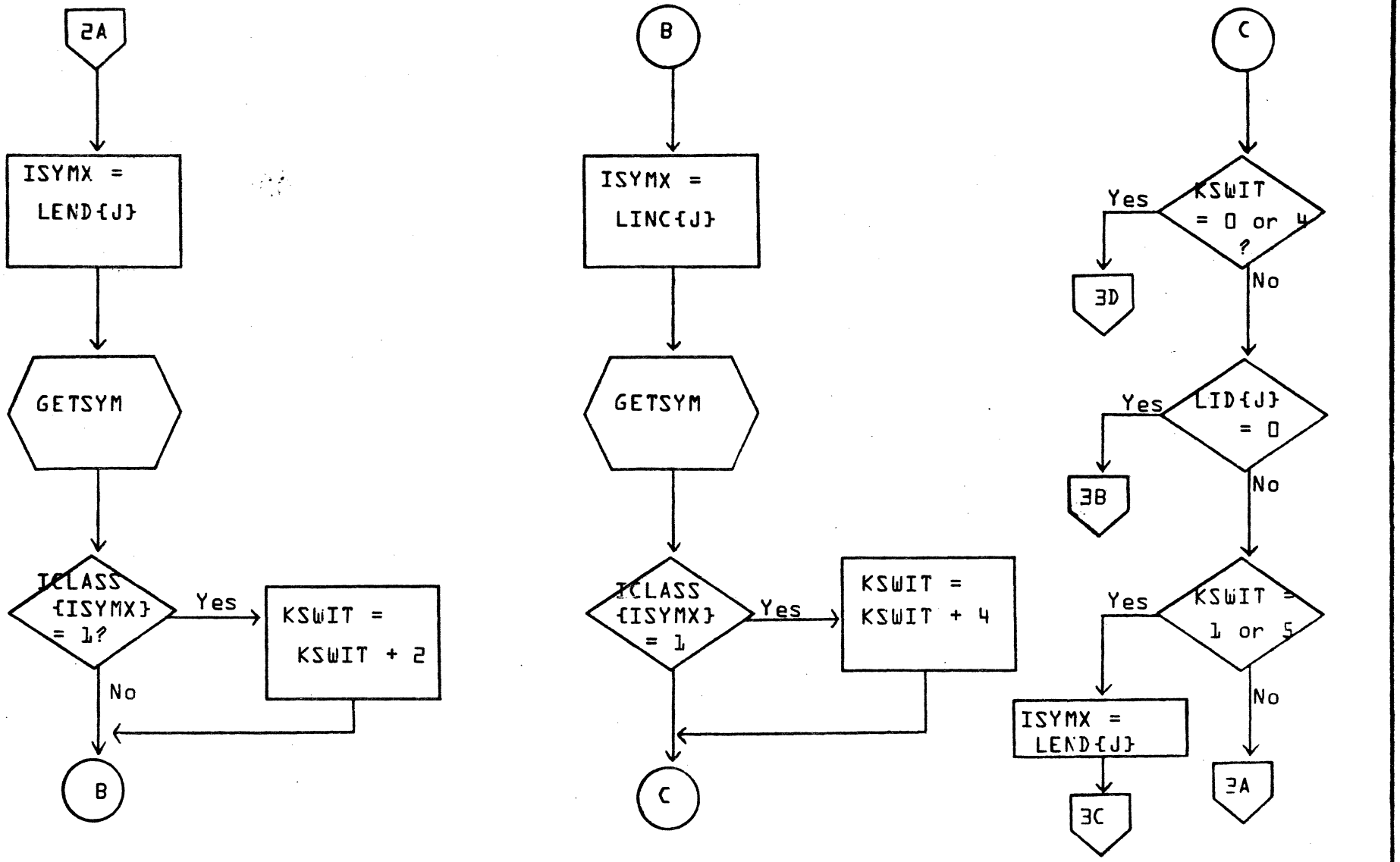
Flowcharts:



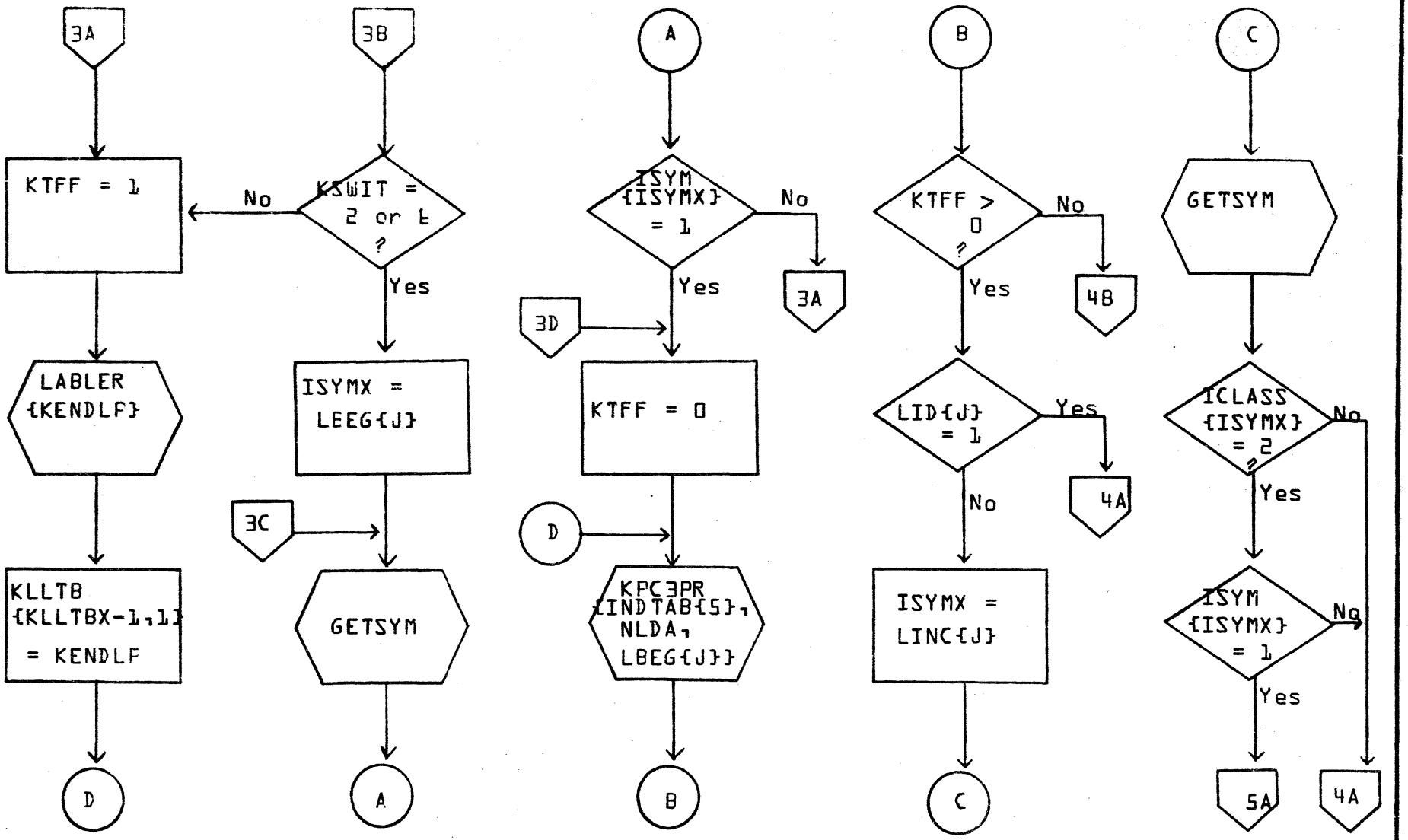
NOTE: Closed subroutines internal to the program are signified by numeric names and their flowcharts are found within the program flowcharts

CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BGINDO		PROJECT MGR.		
		PAGE 1 OF 5	PROJEGT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

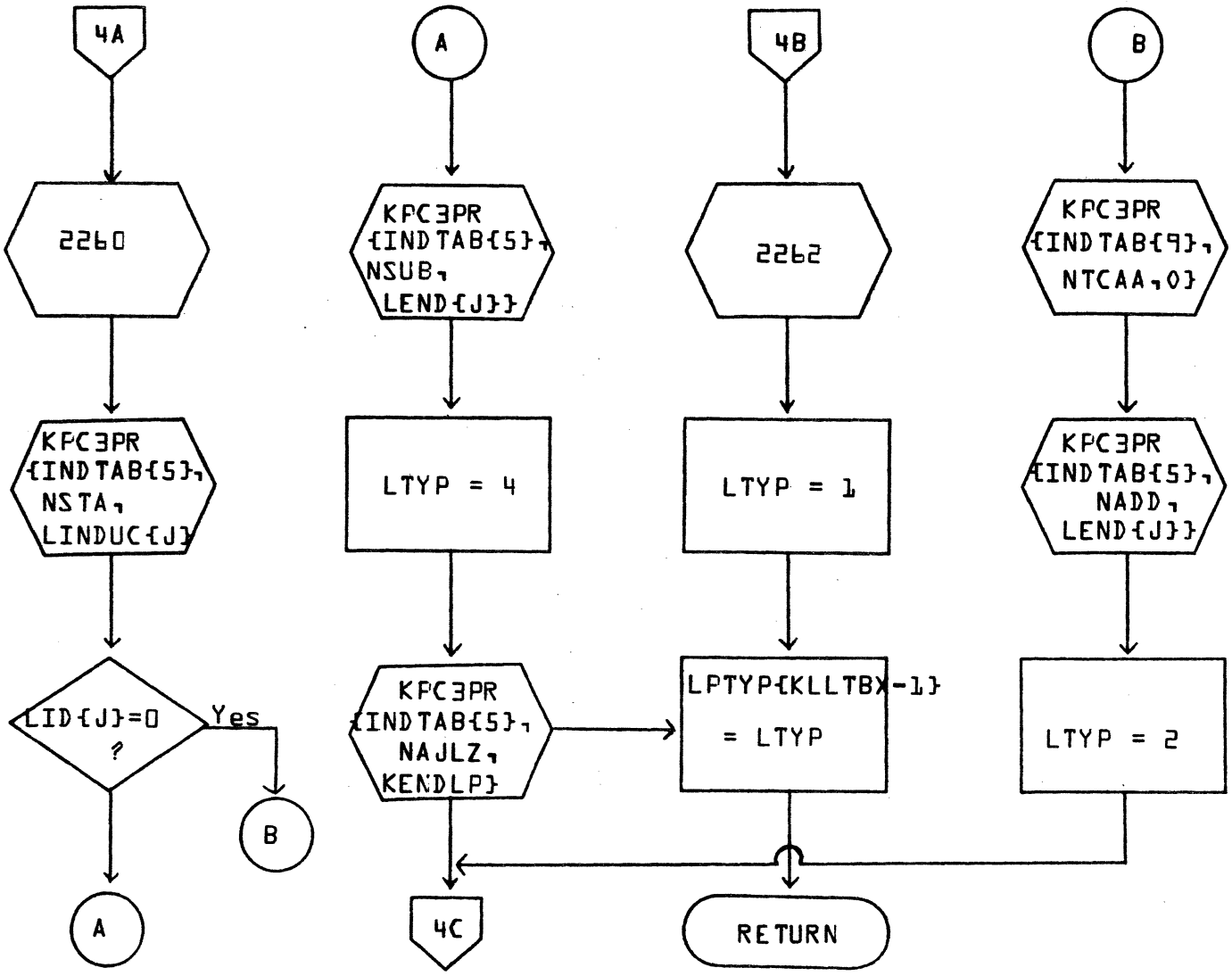


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BGINDO		PROJECT MGR.		
		PAGE 2 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

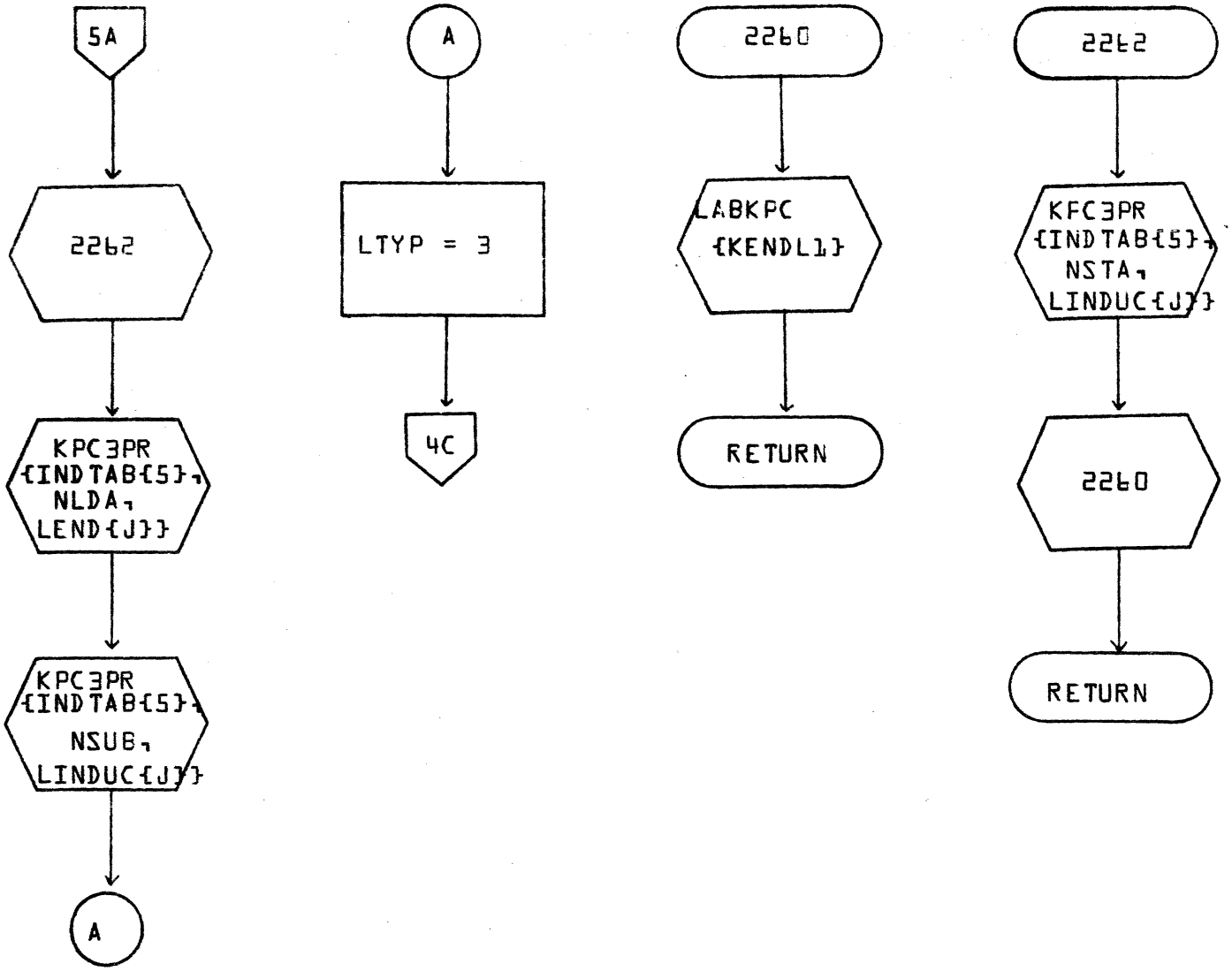


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BGINDO		PROJECT MGR.		
		PAGE 3 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

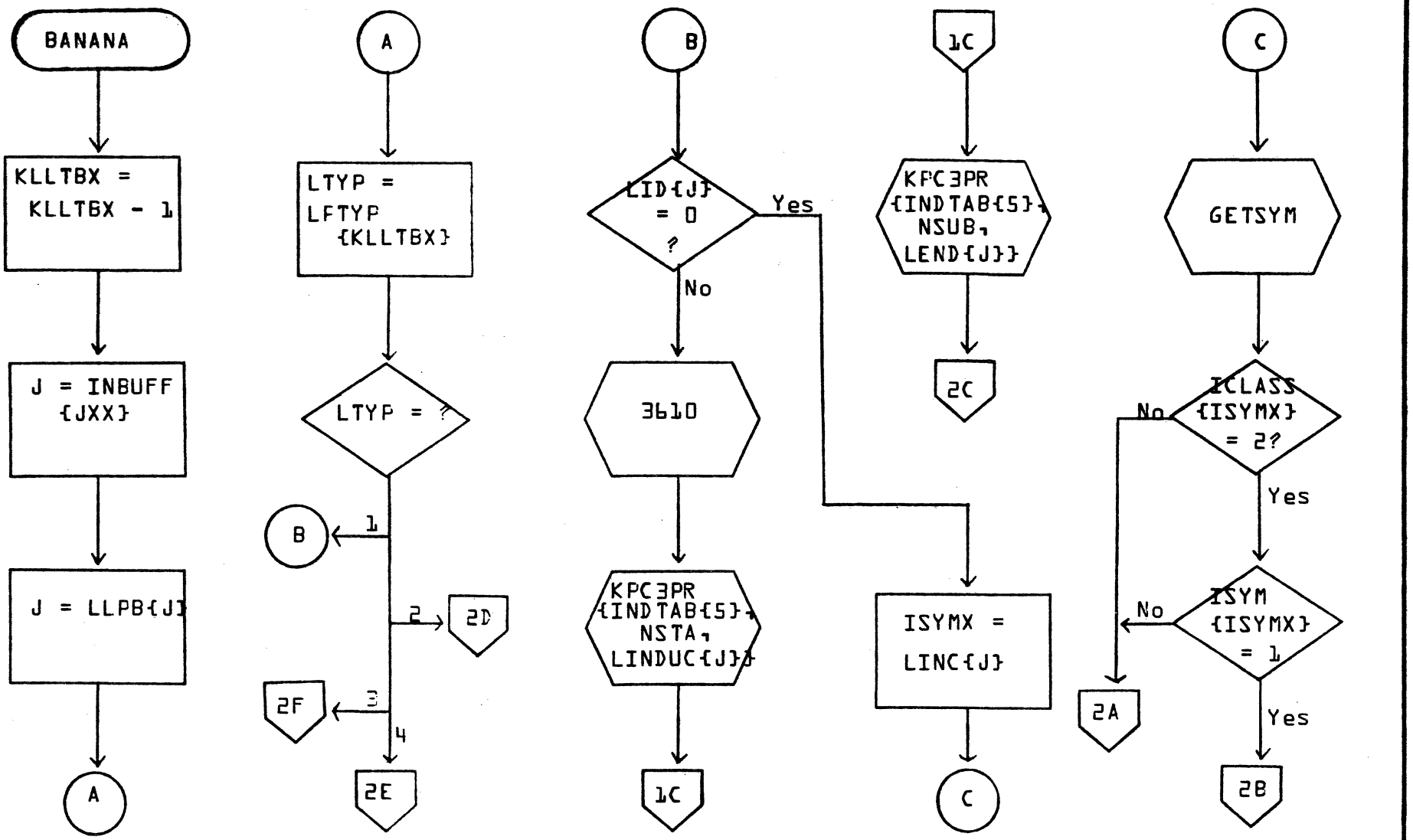
AA1385 (FORMERLY CA127-1)



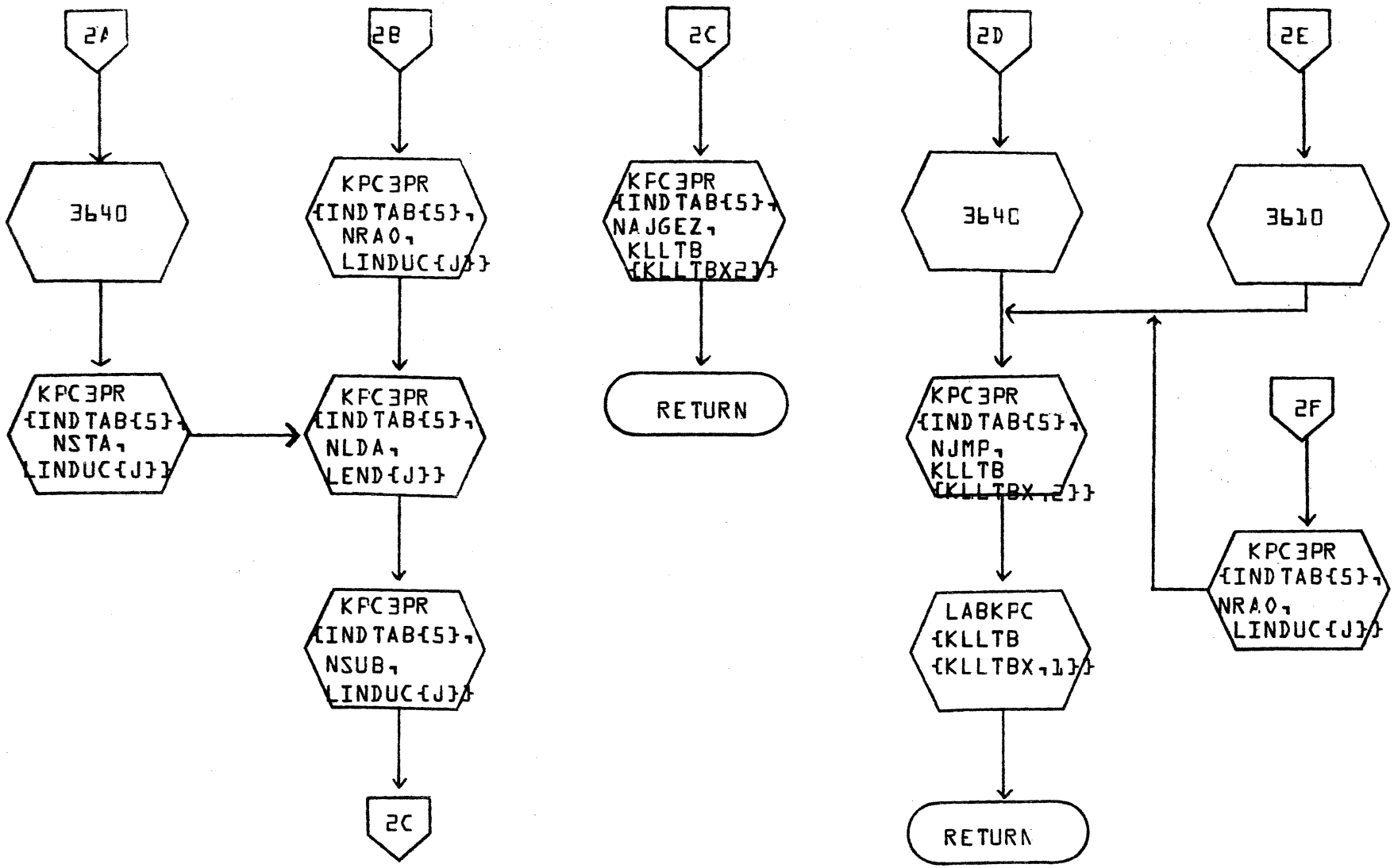
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BGINDO	PROJECT MGR.			
	PAGE 4 OF 5	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



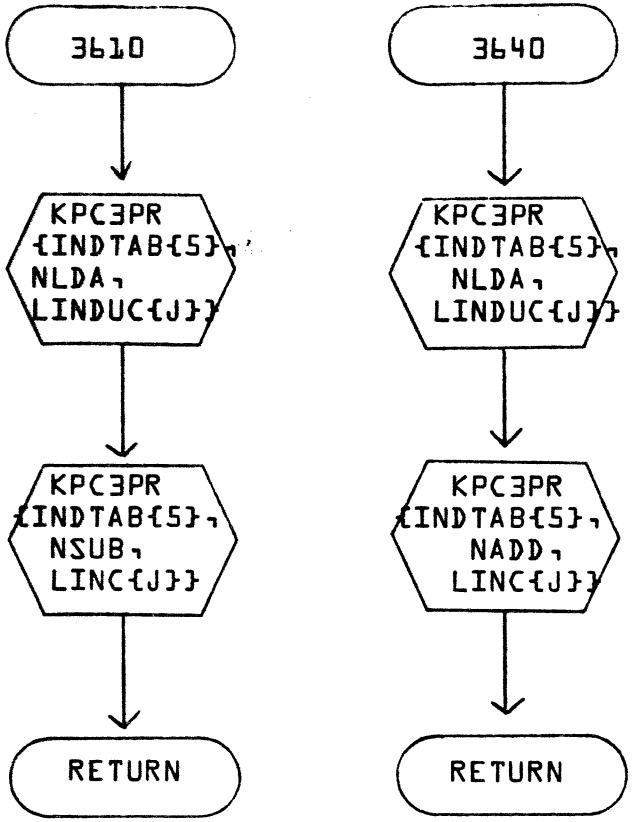
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE BGINDO		PROJECT MGR.		
FLOWCHART ☐		PAGE 5 OF 5	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BANANA	PROJECT MGR.			
	PAGE 1 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



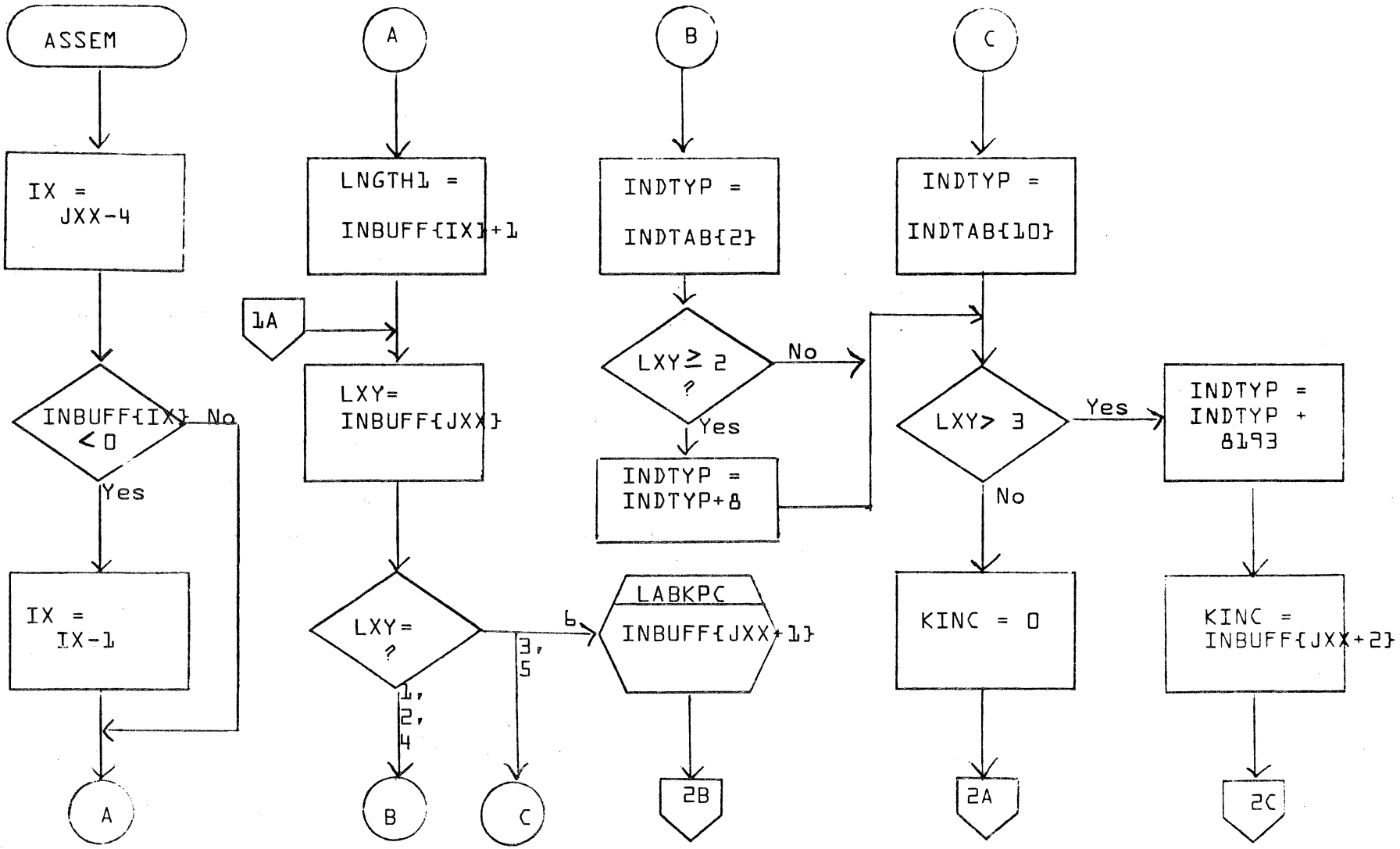
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BANANA		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BANANA		PROJECT MGR.		
		PAGE 3 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

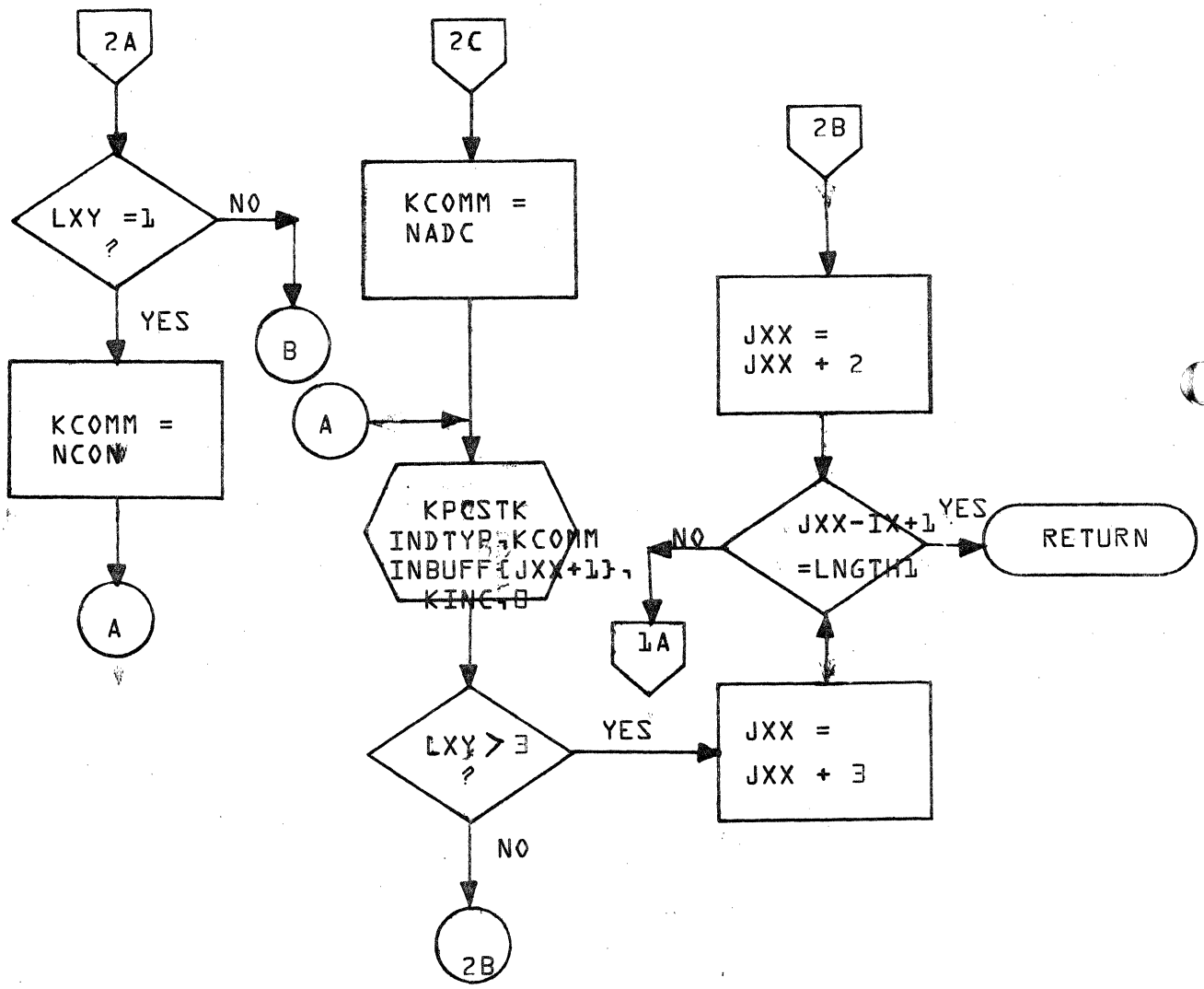
4.3.10 ASSEM

ASSEM is the processor of the ASSEM statement. The ASSEM statement provides the facility for the FORTRAN programmer to generate hexadecimal constants, address constants, and statement labels. These are received by Phase B as a consecutive list and processing is simply a matter of generating CON and ADC commands as appropriate.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ASSEM PROCESSOR	PROJECT MGR.			
	PAGE 1 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

FIG



TITLE	ASSEM PROCESSOR	DRG. NO.
		REVISION
DRAWN BY	PROJ. / DATE	SHEET 2 OF 2

CSD 203

4.4 PHASEB Subroutines

4.4.1 AFIDL

Subroutine AFIDL is an intermediate link between the PHASEB in-line processor of arithmetic IF statements and logical IF statements.

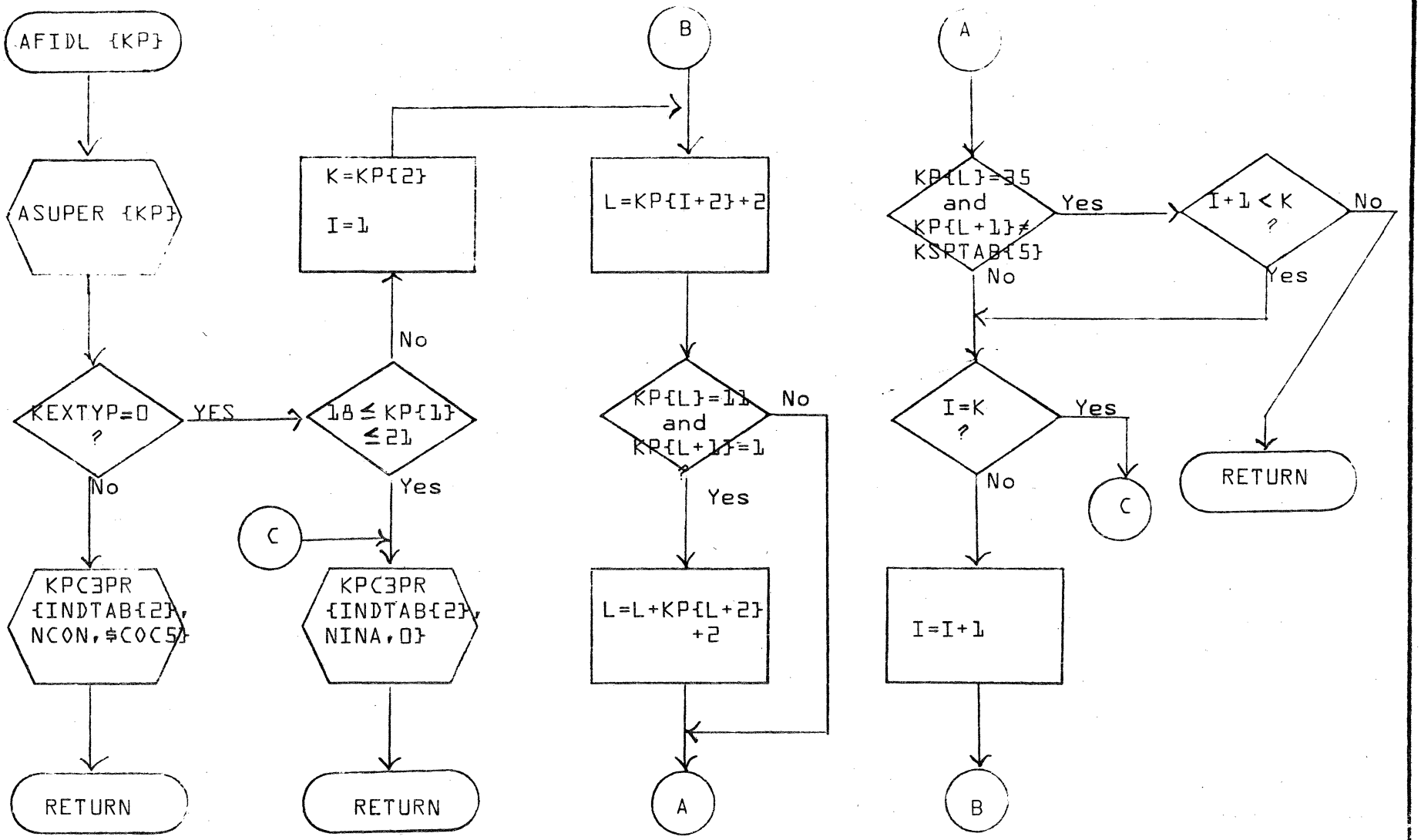
AFIDL calls ASUPER and in addition performs certain checking and code generation. {ASUPER is responsible for generating code to evaluate arithmetic expressions.}

1. AFIDL looks at the mode of the expression which was processed by ASUPER. If it is either a single precision or double precision floating point expression AFIDL generates an instruction to load the upper portion of the pseudo-accumulator {address C5} into the hardware accumulator so that test instructions may operate upon that value.

2. If the expression type was integer, AFIDL determines whether or not it is necessary to generate an

   INA 0

   command to insure that the value -0 {$FFFF_{16}$} is not left in the accumulator. {Test instructions do not allow for -0.} If necessary, AFIDL generates such an instruction.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE AFIDL		PROJECT MGR.			
		PAGE 1 OF 1	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

4.4.2 ASUPER

ASUPER is the arithmetic "tree" processor. It is called once by each statement processor where the statement being processed contains an arithmetic expression. The following statements contain arithmetic expressions.

1. Arithmetic Replacement Statement

   Processed by ARITHR which calls ASUPER for the expression on the right of the equal (=) sign.

2. Statement Function

   Processed by PHASEB which calls ASUPER for the expression on the right of the equal (=) sign.

3. CALL Statement

   Processed in-line by Phase B which calls ASUPER for the CALL expression. (A CALL statement is put into the "tree" format where the CALL is the highest level operator and its parameters are the operands.

4. GO TO (Computed)

   Processed by CGOTO which calls ASUPER for the expression which follows the GO TO parenthesized label list.

5. Arithmetic IF

   Processed by PHASEB which calls AFIDL and AFIDL calls ASUPER in turn for the expression within the parenthesis of the IF.

6. Logical IF

   Processed by PHASEB which calls AFIDL and AFIDL in turn calls ASUPER for each arithmetic expression within the parenthesized logical expression.

The job of ASUPER is to generate executable instructions which perform the evaluation of an expression. This is performed in two steps.

I. The arithmetic tree is stepped through, operand by operand, in a search for subscripts. The disposition of them is based upon their type and whether or not their operator is a CALL, non-inlin function call, or exponentiation (**). If the operand is a non subscripted variable or non-subscripted partial variable, it is passed over.

   A. Variable only subscripted variable (partial or non-partial).

B. Increment only subscripted variable (partial or non-partial).
C. Variable and increment only subscripted variable (partial or non-partial).
D. Complex subscripted variable (partial or non-partial).

Then one of two subscript processors (SUBPR1 or SUBPR2) is called upon to further process the subscripted variable.

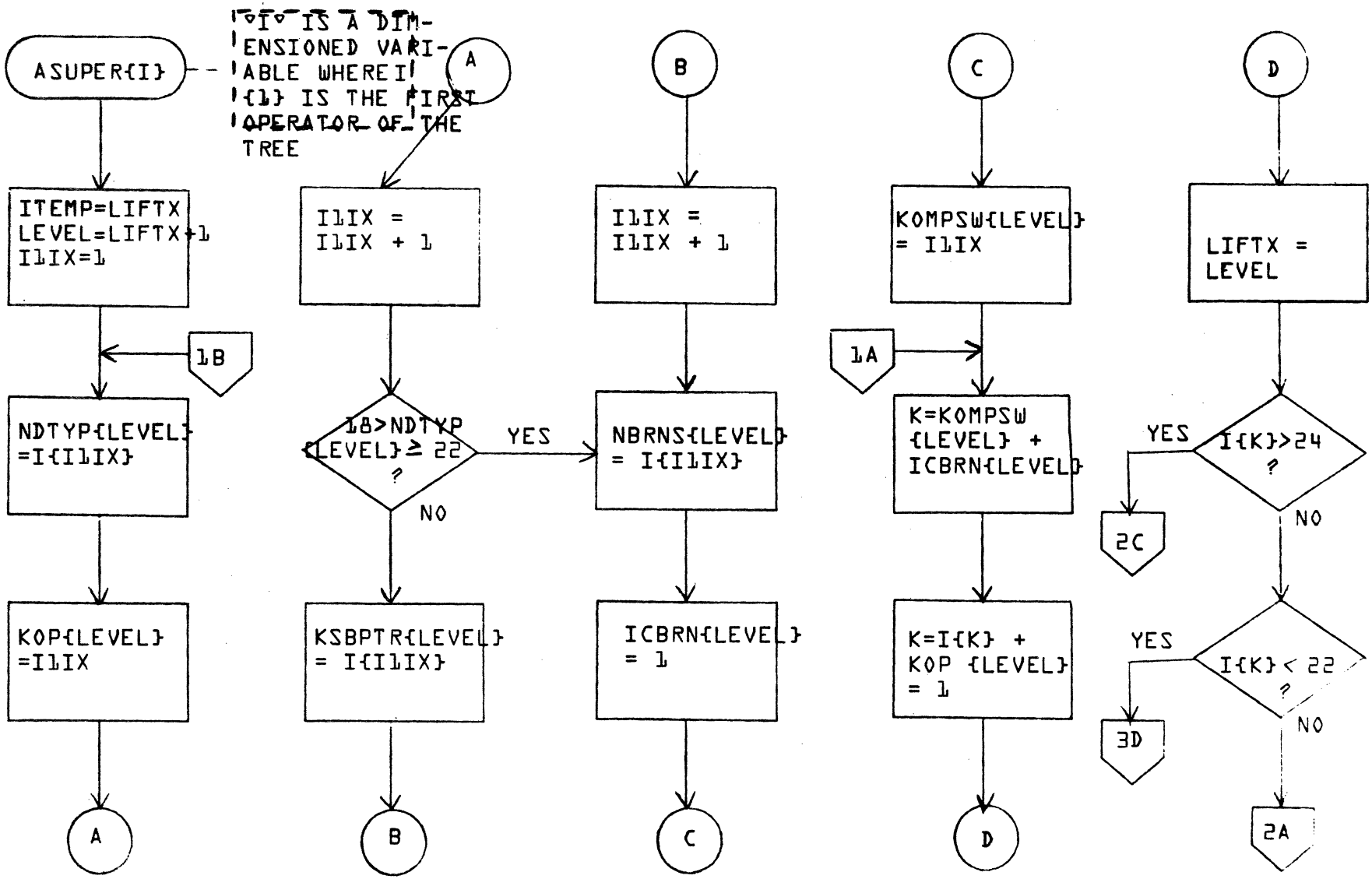
A. If the variable is an operand of an operator which is:

1. a CALL
2. a non-inline function, or
3. an exponentiation (**)

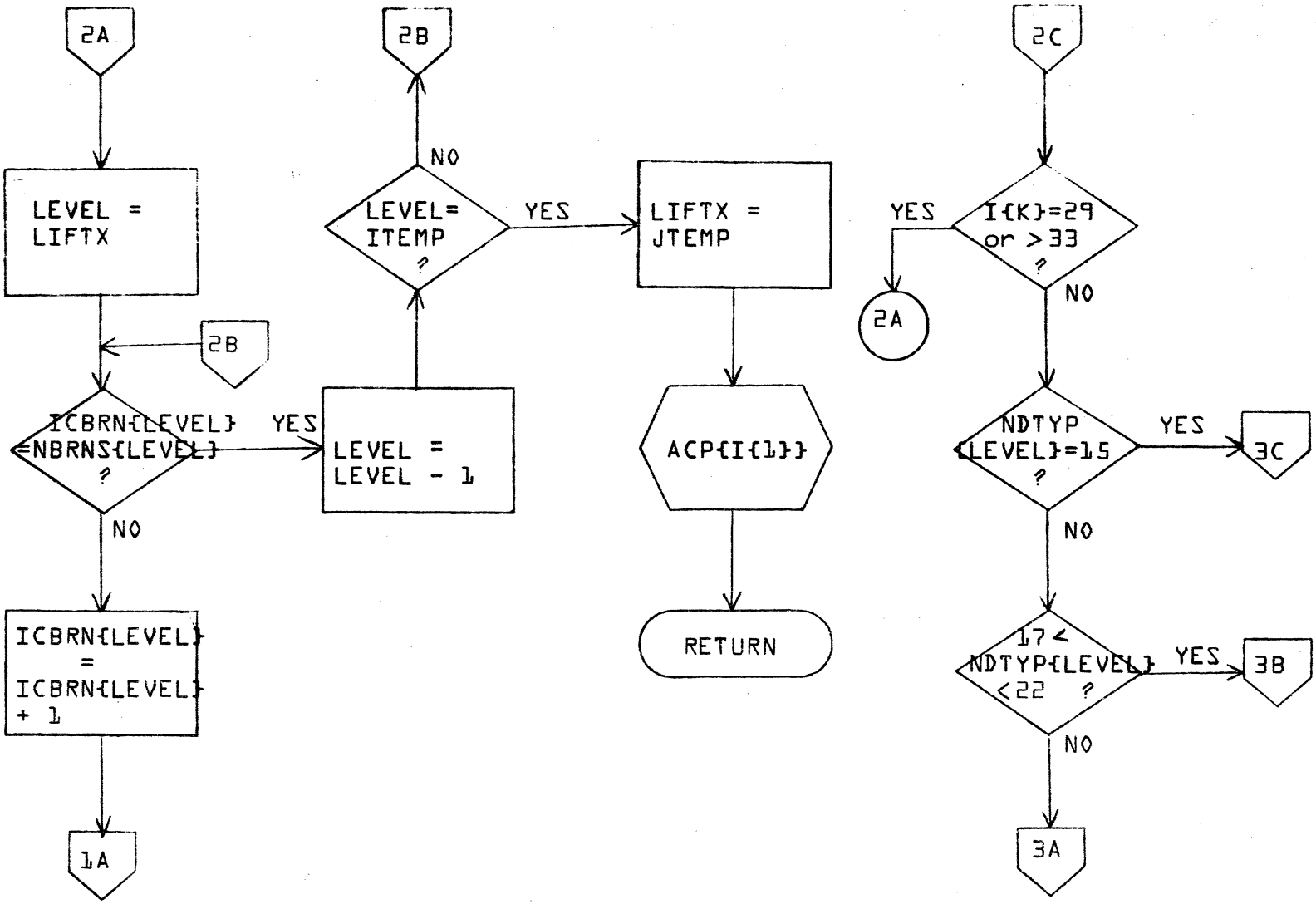
then SUBPR2 is called.

B. If the variable is an operand of any other operator then SUBPR1 is called.

II. Having completed the processing of all subscripts then ASUPER calls ACP which generates the instructions which evaluate the expression.



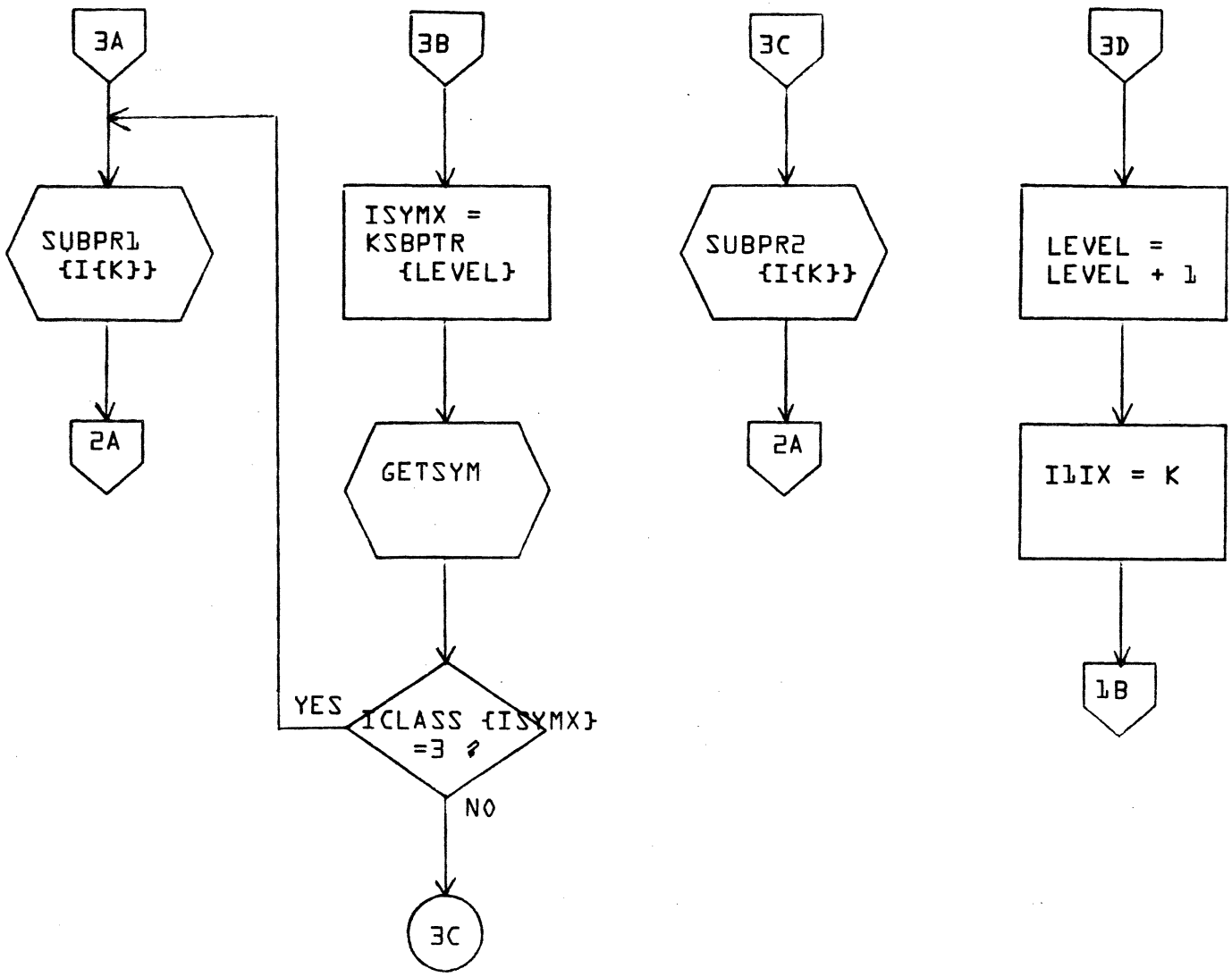
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ASUPER	PROJECT MGR.			
	PAGE 1 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ASUPER		PROJECT MGR.			
FLOWCHART ☒		PAGE 2 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ASUPER	PROJECT MGR.			
	PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

4.4.4 ENTCOD

Refer to section 4.2.1 (Program Structure)

ENTCOD is responsible:

1. For Subroutines and Function Subprograms, generation of all instructions from the entry point to the transfer to the start of the routine (label MBEGIN).
2. For Statement Functions (with Main Subroutine, or Function Subprograms) generation of all instructions from the statement function entry point to the transfer to the start of the Statement Function.

The Structure of the entry code (thus the instruction generated by ENTCOD) is as shown:

CON 0	constant of 0 marks the entry point to the subroutine or function subprogram or statement function.
STQ QSAV LDQ FF STQ FFSAV	instructions which save the value of Q and I indexes by storing in created storage locations QSAV and FFSAV.
RTJ Q8PREP ADC name	return jump to the parameter pickup routine (PARAMS) at the initialization entry Q8PREP with an address constant of the entry point as a parameter. This code is generated only if there is at least one parameter to the subroutine or function that the entry code is being generated for.
RTJ Q8PKUP (series of INA and PST commands to plug this parameter)	return jump to the parameter pickup routine (PARAMS) at the pickup entry Q8PKUP which returns the address of the first parameter in the A register.
RTJ Q8PKUP (series of INA and PST commands to plug this parameter)	pickup of address of second parameter.
	⋮
RTJ Q8PKUP (series of INA and PST commands to plug this parameter)	pickup of address of nth parameter.

JMP	KEXEC	jump to beginning of program. This jump actually initiates the execution of the subroutine or function, having completed the necessary index saving and parameter pickup and plugging
FFSAV	BSS 1	storage cells to save Q and I index registers
QSAV	BSS 1	

The INA and PST commands which follow each RTJ Q8PKUP are based upon the final settings of KSUBAT, the subroutine argument table (for subroutines and function subprograms) or the settings of KSFAT, the statement function argument table (for entry code of statement functions).

1. Subroutines and Function Subprograms. At the point of operation of program ENCODE, KSUBAT has been completely built and it reflects each argument and each increment that has occurred with that argument throughout the compilation. The increments are sorted from lowest to highest and to each argument/increment pair is assigned a unique temporary storage cell.

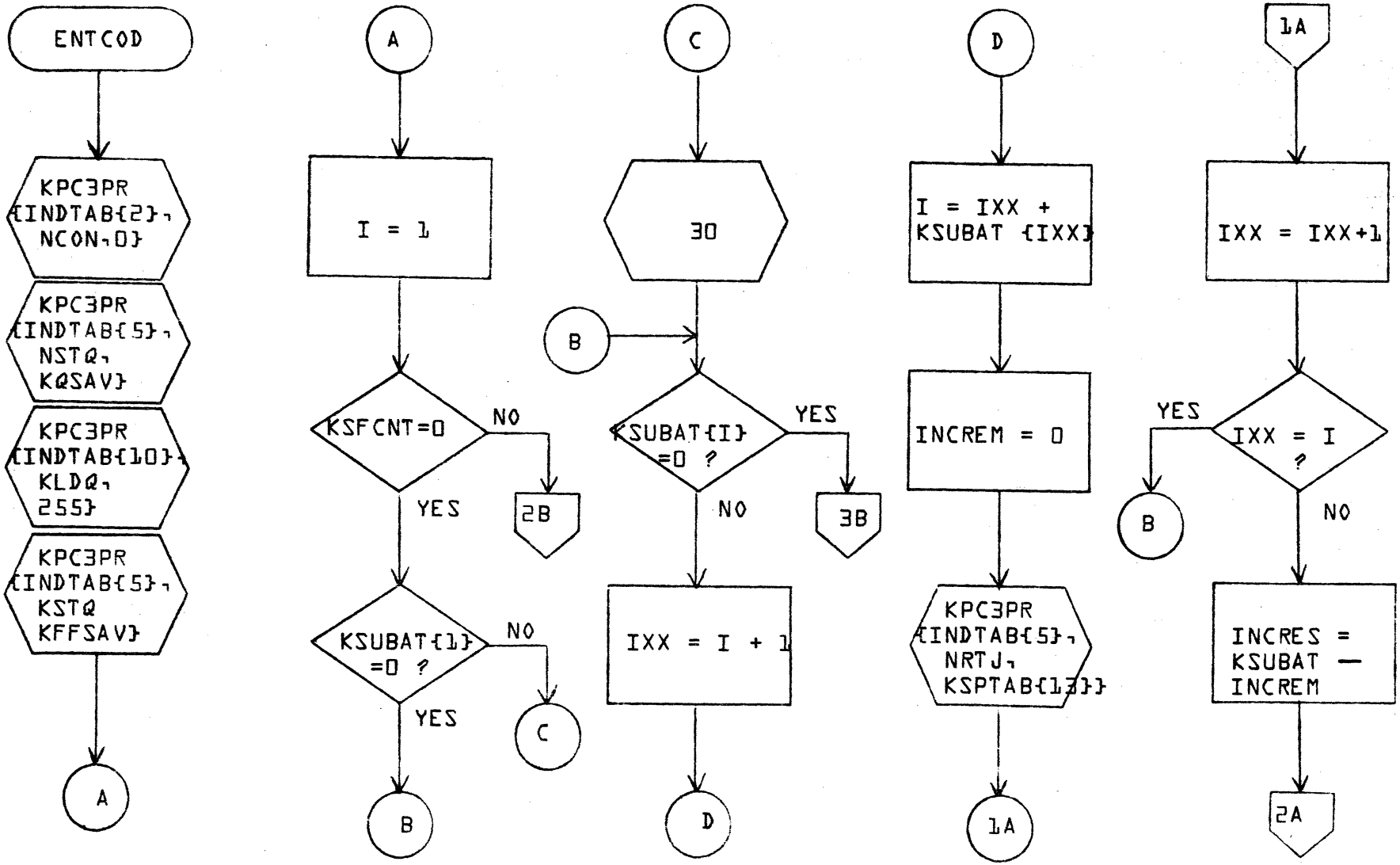
   Having called Q8PKUP which returns with the address of the argument in the accumulator ENTCOD then generates instructions which compute each resulting address (address of argument and increment) and a PST (Parameter store) where the operand of the PST is the temporary storage assignment.

2. Statement Functions. At the point of operation of program ENTCOD, KSFAT has been completely built and it reflects each argument and a unique temporary storage cell assigned to that argument. (Statement function argument can occur with only the nominal increment of 0 since they may never be subscripted).

   Having called Q8PKUP which returns with the address of the argument in the accumulator ENTCOD then generates a PST where the operand of the PST is the temporary storage assignment.

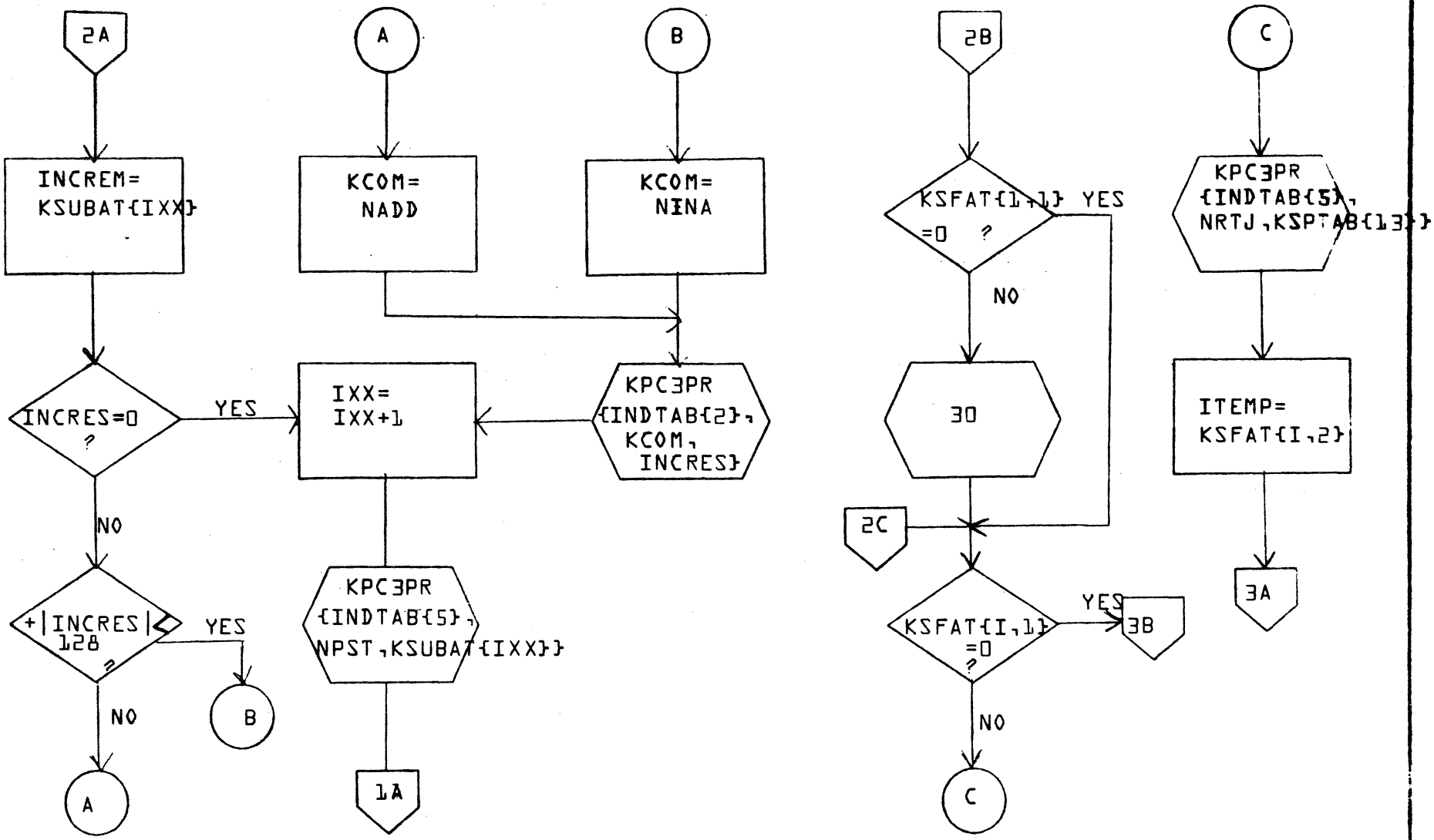
Section 4.4.20 on DUMMY contains a more detailed description of the structure and dynamics of KSUBAT and KSFAT and a more detailed description of subprogram interface design.

Flowcharts:

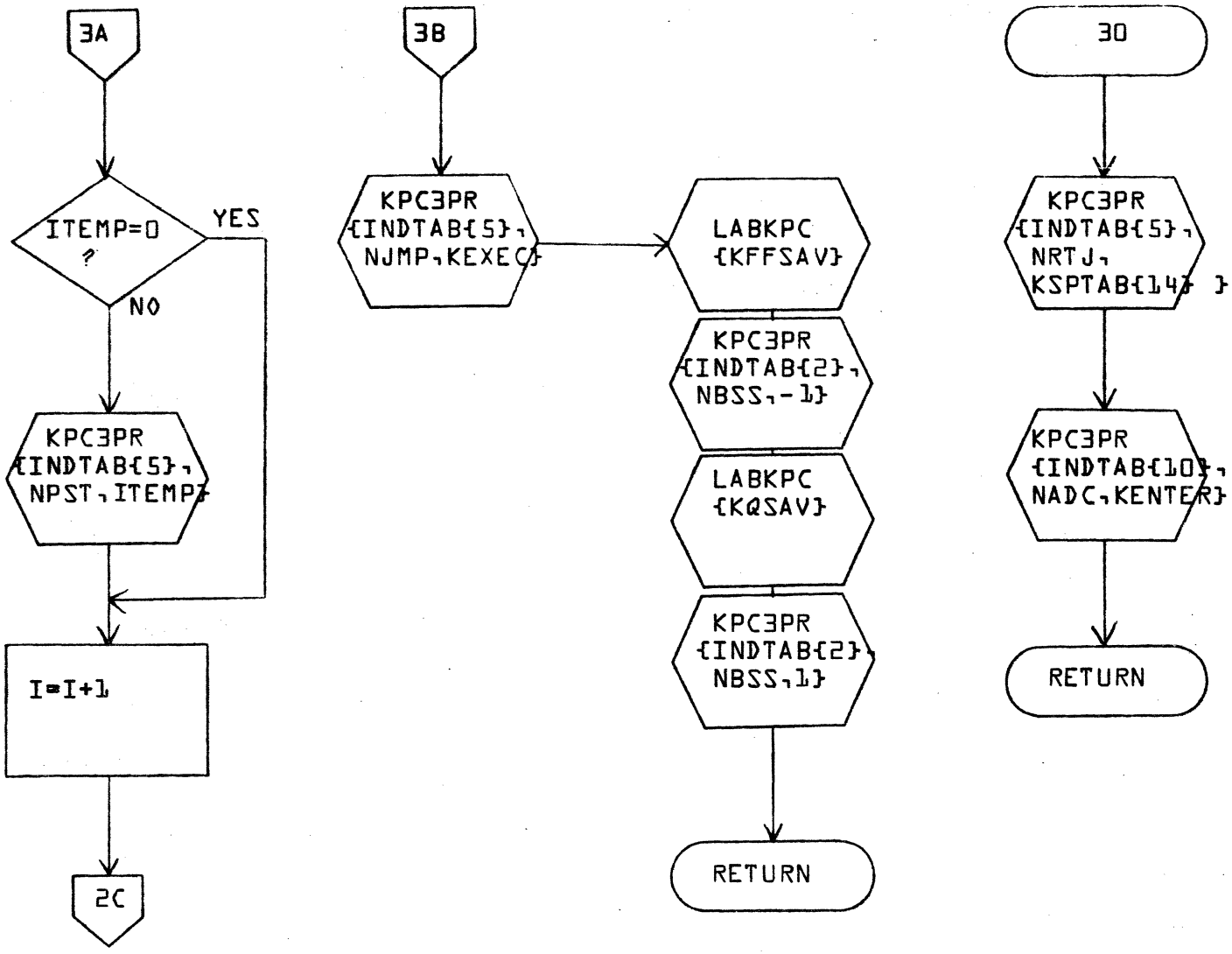


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENTCOD	PROJECT MGR.			
	PAGE 1 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENTCOD	PROJECT MGR.			
	PAGE 2 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENTCOD		PROJECT MGR.			
		PAGE 3 OF 3	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

4-94

4.4.5 FCMSTK

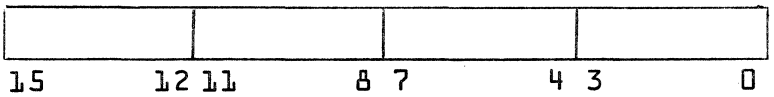
FCMSTK is a subroutine used exclusively by KPCSTK for building floating-point calling sequences. The type of the floating-point calling sequence (single precision or double precision) is transparent to FCMSTK. If the symbol table pointer is minus one, it indicates the special case of a double precision load from C5. FCMSTK specifically is responsible for monitoring the KFCET table. The structure of KFCET is shown:

KFCET

	n,1	n,2	n,3
1,m	BYTE WORD	NOT USED	
2,m	INDICATOR WORD	OPERAND	ADDITIVE
3,m	⋮	⋮	⋮
4,m	⋮	⋮	⋮
5,m	⋮	⋮	⋮

The byte word (KFCET(1,1)) is divided into four 4-bit bytes in accordance with the floating point calling sequence format. KBYTX indexes the byte position as shown:

byte word



and:

KBYTX = 1, bits 15-12
= 2, bits 11-8
= 3, bits 7-4
= 4, bits 3-0

KFCETX indexes the rows of KFCET. There is space for from 0 to 4 entries in KFCET where each entry is 3 words and comprises a row of KFCET (Rows 2 through 5).

Parameters: There are two parameters to FCMSTK:

KF1 - floating instruction {see floating point package for a detailed list of floating commands}. The floating instruction is a 4 - bit value.

KF2 - operand for those floating commands such as FLD, DLD, FST, and DST where there is an operand; KF2 = 0 for those commands which have no operands {FCOM, etc.}.

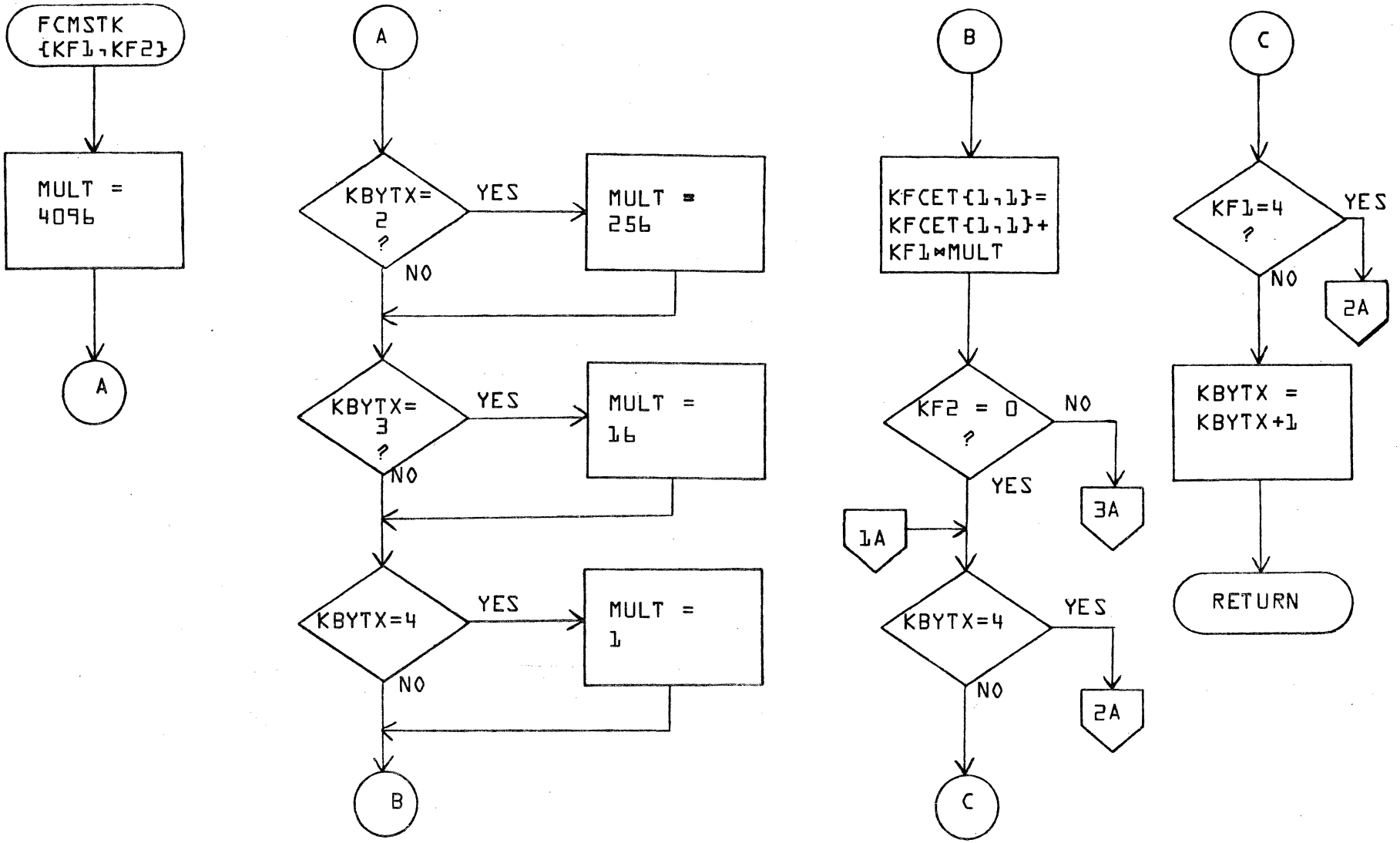
FCMSTK receives a call from KFCSTK with the parameter set for a floating command.(Hence, a request to add an entry to KFCET). FCMSTK adds the instruction byte to the byte word and updates index KBYTX. If there is an operand, FCMSTK updates KFCETX to the next entry and adds the operand to the table. (The indicator word and the additive are picked up from the original calling sequence to KPCSTK, KQ(1) and KQ(4) respectively. The exception is an indexing command, KF1 = F_{16}, where the indicator word is created as a function of the current addressing mode and there is not an additive.)

If:

1. All four bytes of the byte word have been filled with floating commands, or,
2. The last floating command entered was a stop (4)

then FCMSTK "dumps" the KFCET table in the following manner:

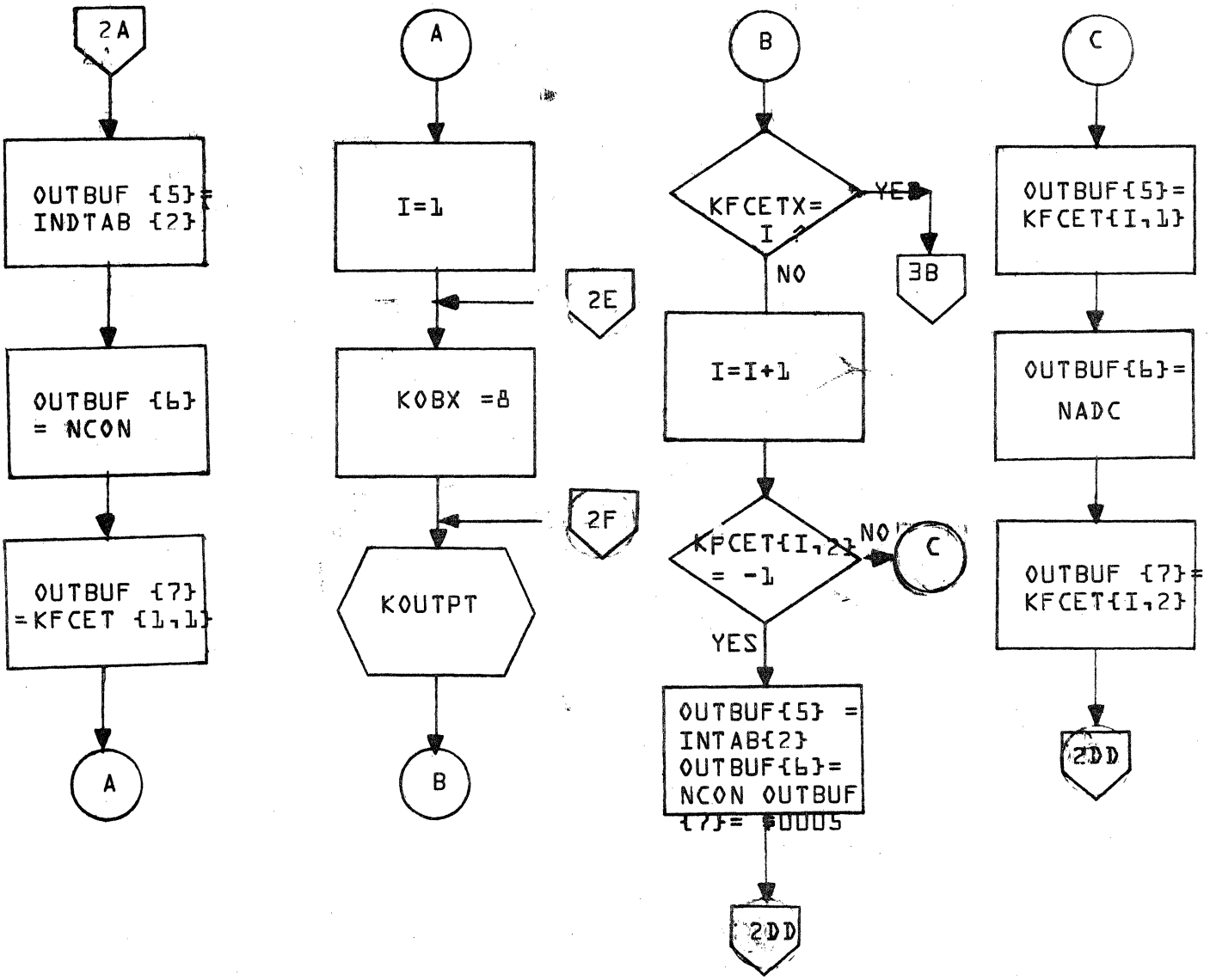
1. Generates the byte word as an integer constant.
2. Generates each entry in the table as an address constant.
3. Reinitializes all indexes to the table.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: FCMSTK		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

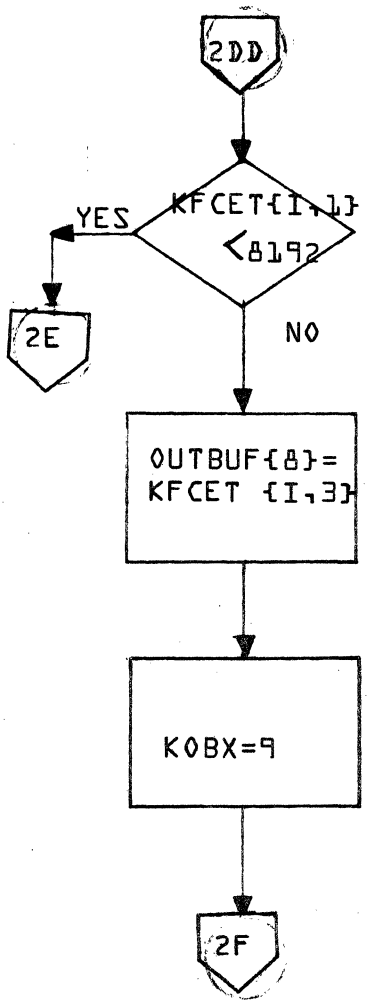
CONTROL DATA CORPORATION

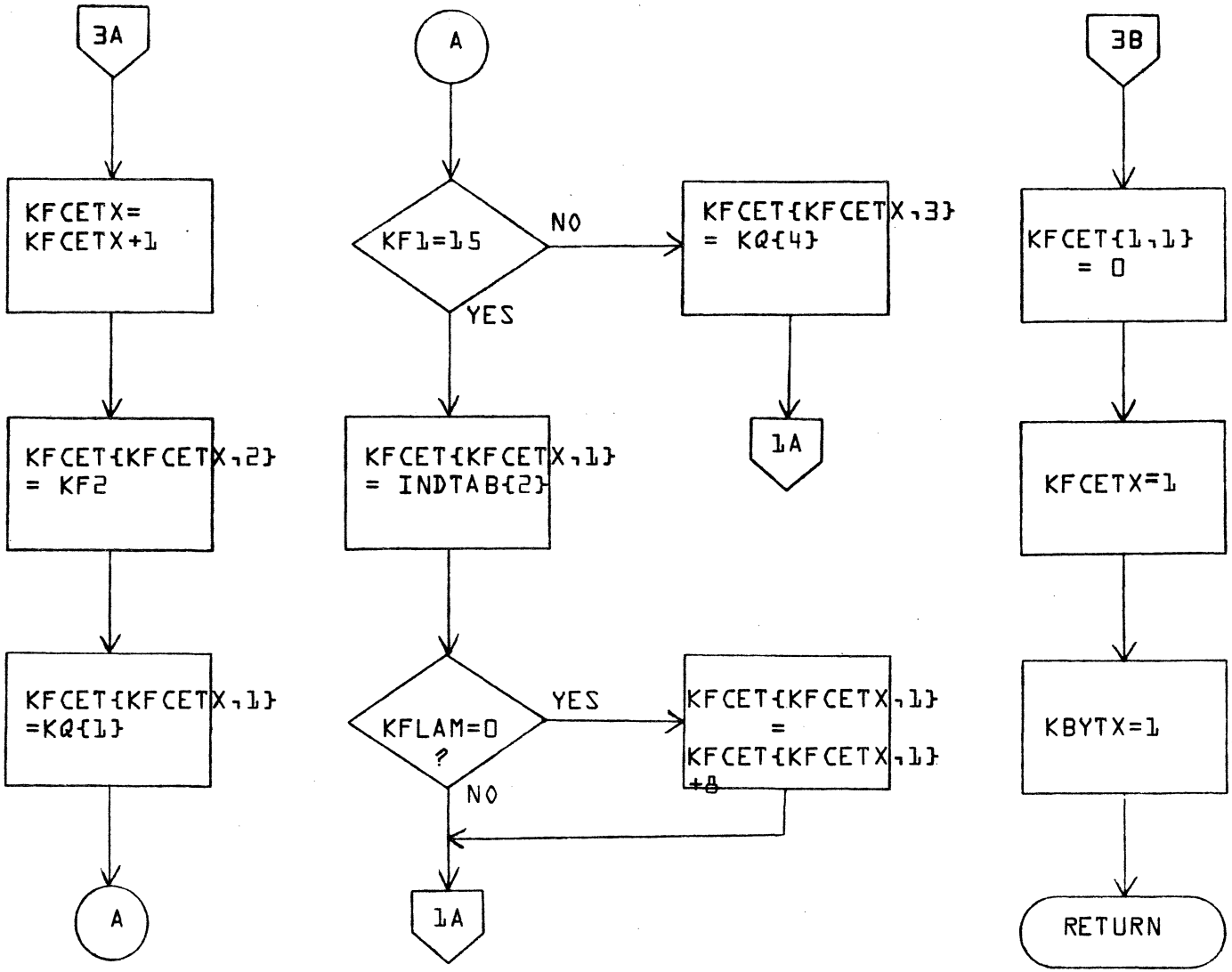
LA JOLLA FACILITY



TITLE	FCMSTK		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2 OF 3

CSD 203


2DD
KFCET(I,1)
<8192
YES
2E
NO
OUTBUF(8)=
KFCET (I,3)
KOBX=9
2F



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FCMSTK	PROJECT MGR.			
	PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

4.4.6 FINK

FINK is a subroutine responsible for generating instructions of in-line functions. The choice to make a function in line or not in-line was made on an individual basis for each function based upon:

1. The number of locations which would be required to perform the function in line. And in contrast,
2. The number of locations which would be required to execute a calling sequence.
3. The overhead that must be paid to make a function a separate subroutine.

The following functions were made in-line functions. They are listed with the instructions which are generated by FINK to execute them.

1. IABS(I)

```
     LDA        I
     AJP,GEZ    SKIP
     TCA        A
SKIP  :
      :
```

2. AND (I,J)

```
     LDA        I
     AND        J
```

3. OR (I,J)

```
     LDA        I
     LDQ        J
     TCA        A
     TCQ        Q
     CAQ        A
```

4. EOR (I,J)

```
     LDA        I
     EOR        J
```

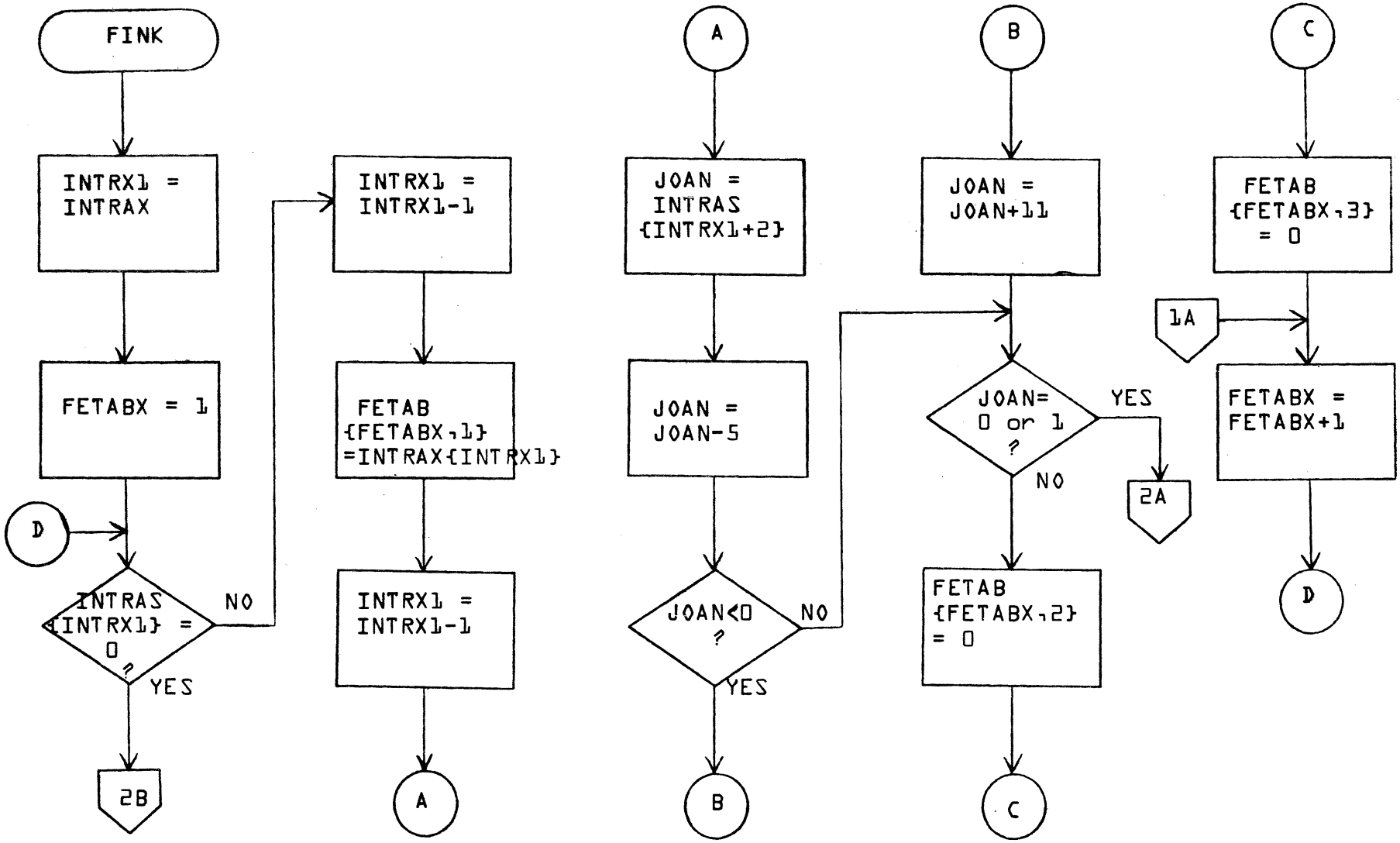
5. NOT (I)

```
     LDA        I
     TCA        A
```

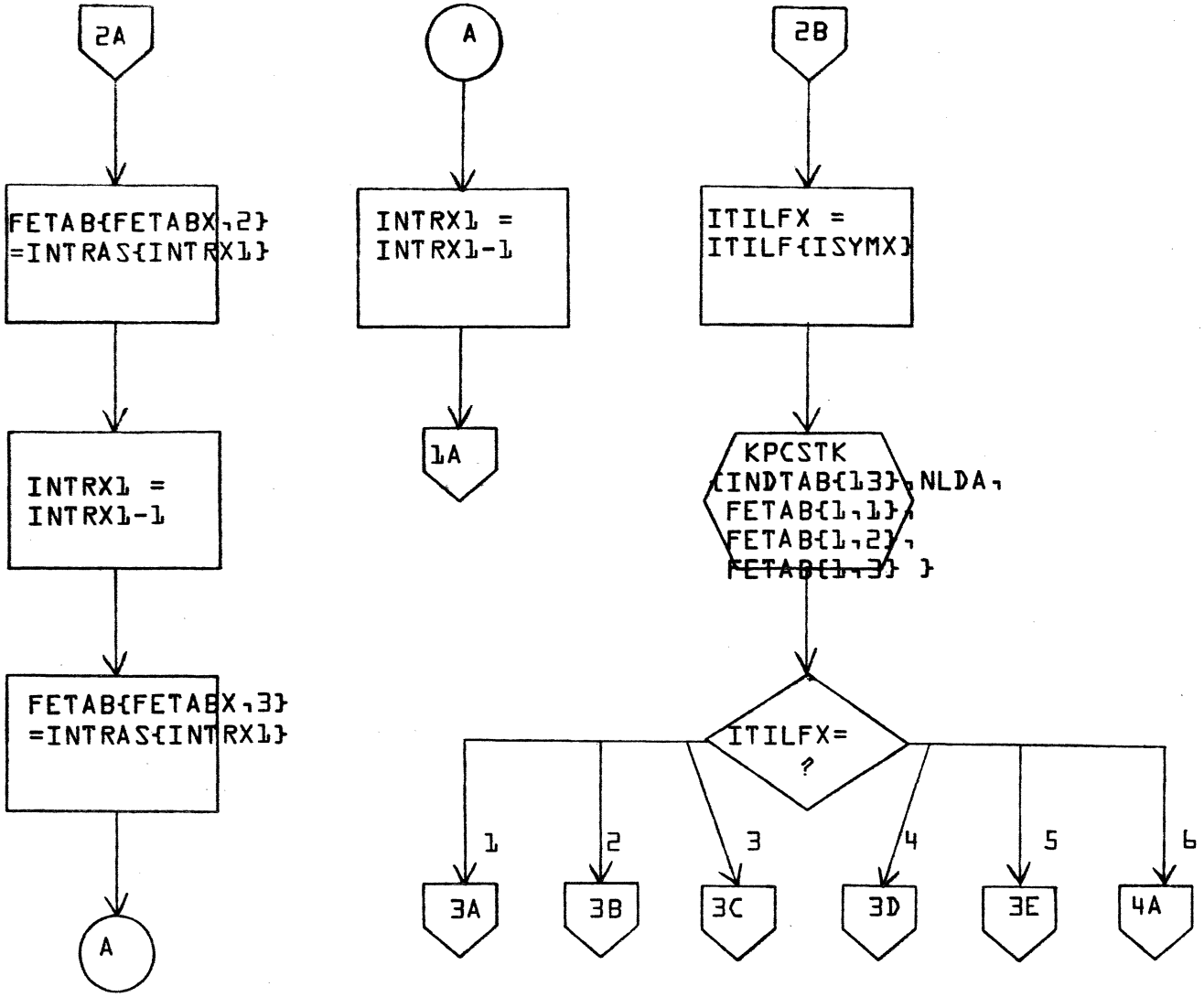
6. ISIGN (I,J)

```
     LDA        I
     MUI        J
     LRS        31
     EOR        I
```

All other functions are library subroutines.



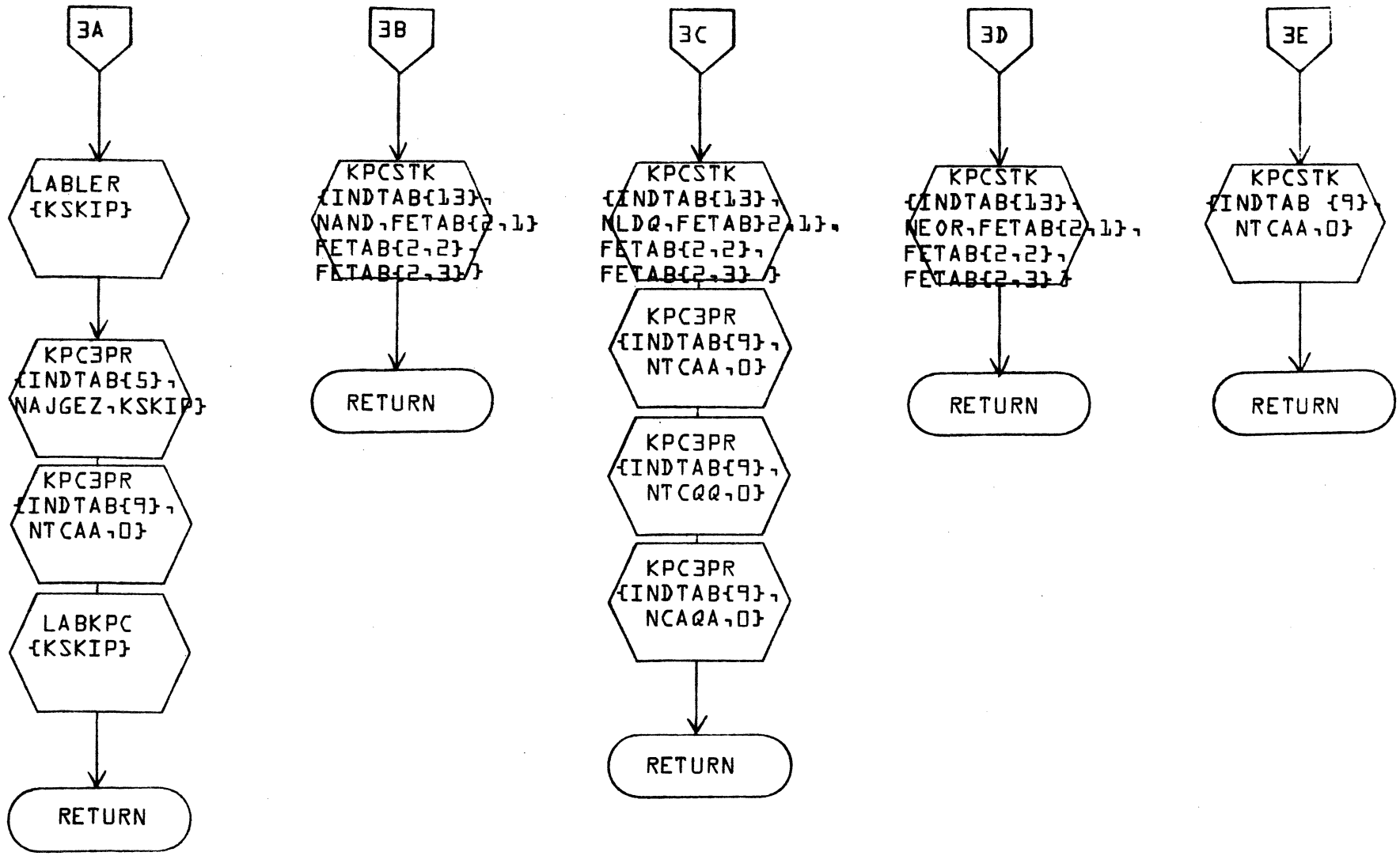
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: FINK		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



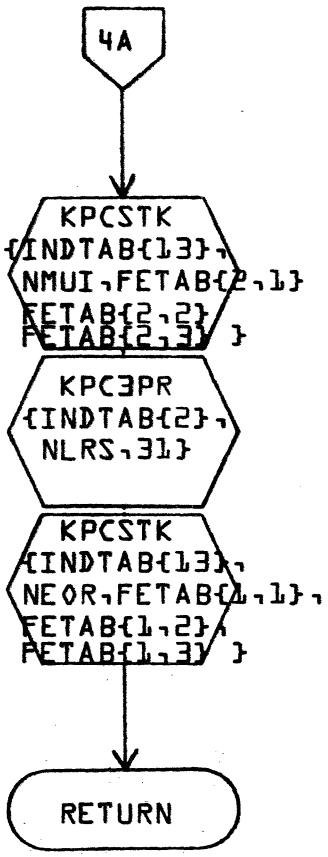
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE FINK		PROJECT MGR.		
		PAGE 2 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FINK		PROJECT MGR.			
		PAGE 3 OF 4	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: FINK		PROJECT MGR.			
FLOWCHART ☒		PAGE 4 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

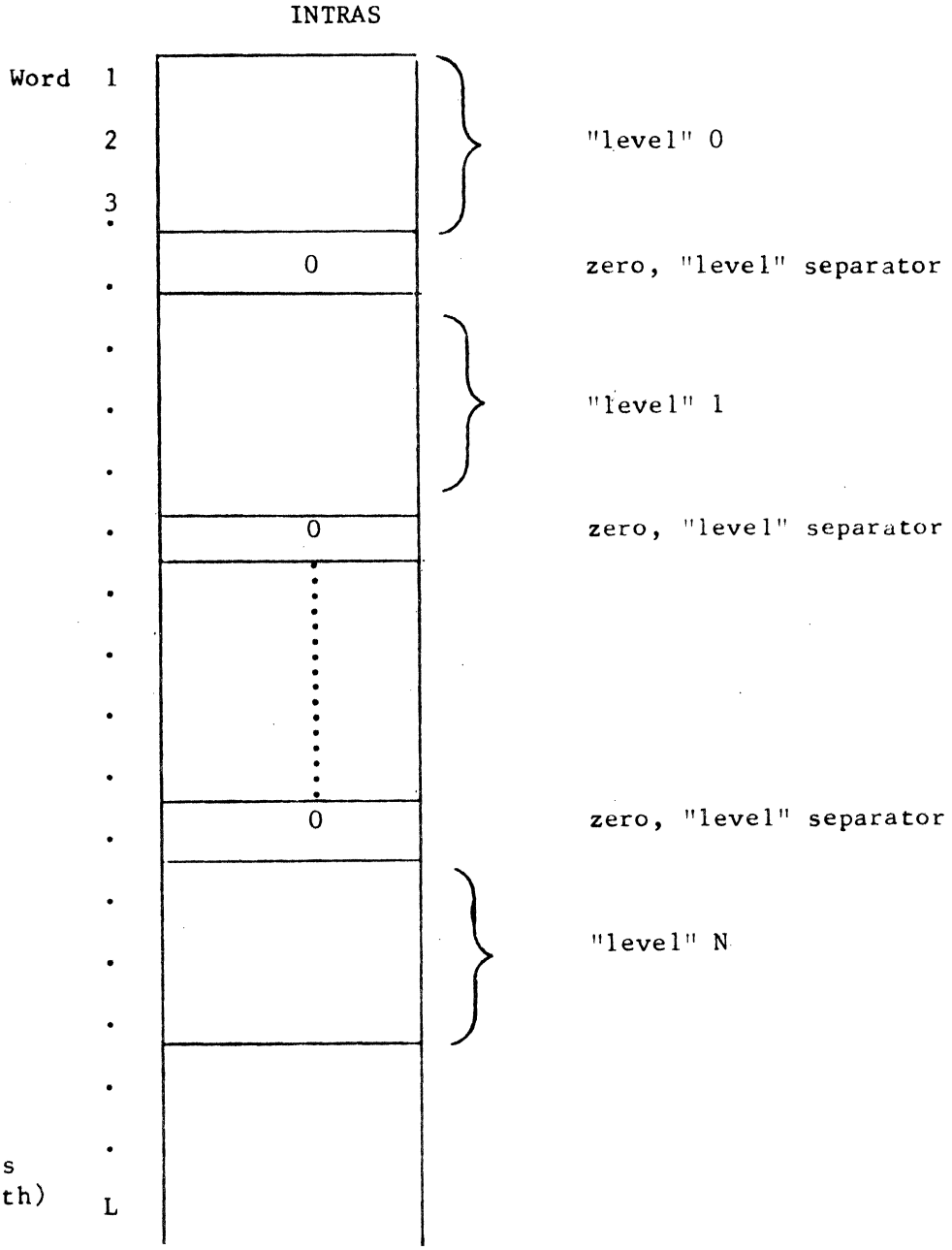
4.4.7 INTRAM

INTRAM is a routine used in conjunction with ACP to generate instructions.

INTRAM, basically, updates the one-dimensional table INTRAX. The major index to the table, INTRAX, is initialized to 0 by PHASE4 prior to entry of each statement processor.

Format of the table is shown:

INTRA-STATEMENT TEMPORARY STORAGE TABLE



Elements are of three types depending upon the type of storage they identify. Information within the elements is generally organized from bottom to top since updating of the table proceeds downward but information is scanned and introduced in a backward direction.

A. Temporary Storage Element

Word 1	P
Word 2	N

where: P is the symbol table pointer of the assigned temporary storage.
N is the element designator.

If N = 3 the temporary storage was assigned to contain the results of a computation whose operator was not inverse.
N = 4 the temporary storage was assigned to contain the results of a computation whose operator was inverse.

B. Operand Element

Word 1	Ps
Word 2	A
Word 3	P
Word 4	N

where: P is the symbol table pointer of the variable.
A is the additive of the base address where the operand is an array element. A = 0 if additive is 0 or if there is no additive.
Ps is the symbol table pointer of the operand subscript where the operand is an array element. Ps = 0 if there is no subscript.
N is the element designator.

If N = 5 the operand was not an inverse.
N = 6 the operand was an inverse.

C. Calling Sequence Label Element

Word 1	P
Word 2	N

where: P is a symbol table pointer assigned previously by the subscript processor to label the cell containing the results of an address computation which must be stuffed into a calling sequence.
N = 7 or 8.

D. Constant (non-symbol table pointer) Calling Sequence Element

Word 1	C	where C is a constant
Word 2	N	N = -7

Parameters

Parameters are common locations KNTROL, IPTR, IADDIT, ISBSCP, and KINTYP.

INTRAM performs 10 different functions depending upon the value of the controlling parameter KNTROL which can assume values -7 and +1 through +11. If KNTROL = ...

1. INTRAS table is updated to next level:
   a. INTRAX is incremented by 1
   b. INTRAS (INTRAX) set to 0
2. INTRAS table is updated back to next lower level.
   a. INTRAX is decremented by 1
3. Temporary storage element of type N = ± 3 is added to current level of table.
   a. INTRAX is incremented by 1 and INTRAS (INTRAX) set to IPTR.
   b. INTRAX is incremented by 1 and INTRAS (INTRAX) set to KNTROL.
4. Temporary storage element of type N = ± 4 is added to current level of table (as per N = 3).
5. Operand element of type N = 5 is added to current level of table.

a. INTRAX is incremented by 1 and INTRAS {INTRAX} set to ISBSCP.
b. INTRAX is incremented by 1 and INTRAS {INTRAX} set to IADDIT.
c. INTRAX is incremented by 1 and INTRAS {INTRAX} set to IPTR.
d. INTRAX is incremented by 1 and INTRAS {INTRAX} set to KNTROL.

6. Operand element of type N = 6 is added to current level of table {as per N = 5}.

7,8. Calling sequence label element is added to current level of table {as per N = 3}.

-7. Calling Sequence constant is added to current level of table {as per N = 3}.

9. Dump instruction string, add/subtract mode. The INTRAS table is scanned in a backward direction beginning at the current position of the index INTRAX at the last word of the latest entry added to the current level.

a. As each element is encountered, an add or subtract instruction is generated which references the operand designated in that element as per the element designator specification of non-inverse respectively. ADD, FAD, or DAD; SUB, FSB, or DSB is decided upon by interrogation of parameter KINTYP. KINTYP is set to 1, 2 or 3 as computation is integer, real or double precision respectively.
b. If the element was temporary storage, and element designator was positive valued, that temporary storage is released by call to the temporary storage allocation routine, TSALOC. {Negative valued element designator indicates that the temporary storage was assigned for a subexpression and therefore should not be released until the use count has been decremented to 0.}
c. If the element was an operand element containing a non-zero subscript pointer, level 0 is searched for a temporary storage element containing that pointer. If found, that temporary storage is also released through TSALOC. {A subscript pointer may be a temporary storage yet not in level 0 because it contains a subexpression result and must not be released.} Upon completion of control function 8, INTRAX points to the 0 level break between the leve just processed and the preceding level of INTRAS.

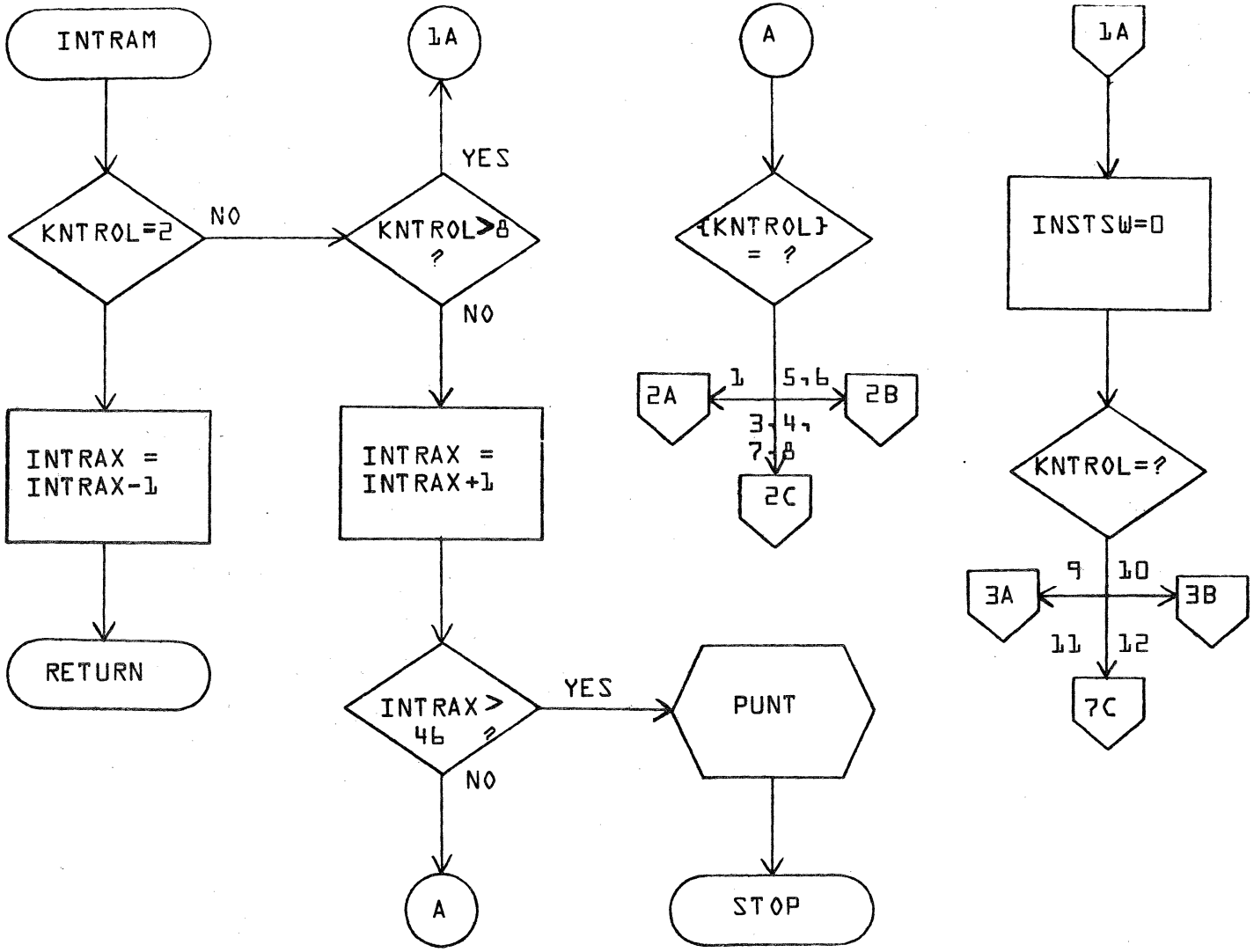
10. Dump instruction string, multiply/divide mode. Exactly the same procedure as 8 with MUI/FMU/DMU, DVI/FDV/DDV, with following exceptions:

    a. Constant, integer operands are checked to determine whether they should be replaced by shifts, and shifts substituted where appropriate.
    b. Instructions for A-Q sign extension are appended preceding integer divides.
    c. Check is made to see if code is being generated for pass 1 of a floating multiply. If so, inverses are ignored and all FDV {or DDV} instructions become FMU's {or DMU's}.

11. Dump parameter string. The INTRAS table is scanned in a backward direction beginning at the current position of the index INTRAX at the last word of latest entry added to the current level. As each element is encountered a calling sequence entry is created.

    a. Temporary Storage Elements. An address constant is generated for the temporary storage as appropriate to run-anywhere switch. The temporary storage is then released by call to TSALOC.
    b. Operand Elements. An address constant is generated for the temporary storage as appropriate to run-anywhere switch.
    c. Calling Sequence Label Element. A label is generated as per the pointer in the element. This is followed by a generated BSS 1.
    d. Calling Sequence Constant. A constant is generated.
    e. IREF bit is set in symbol table to show that operands have been referenced in a calling sequence.

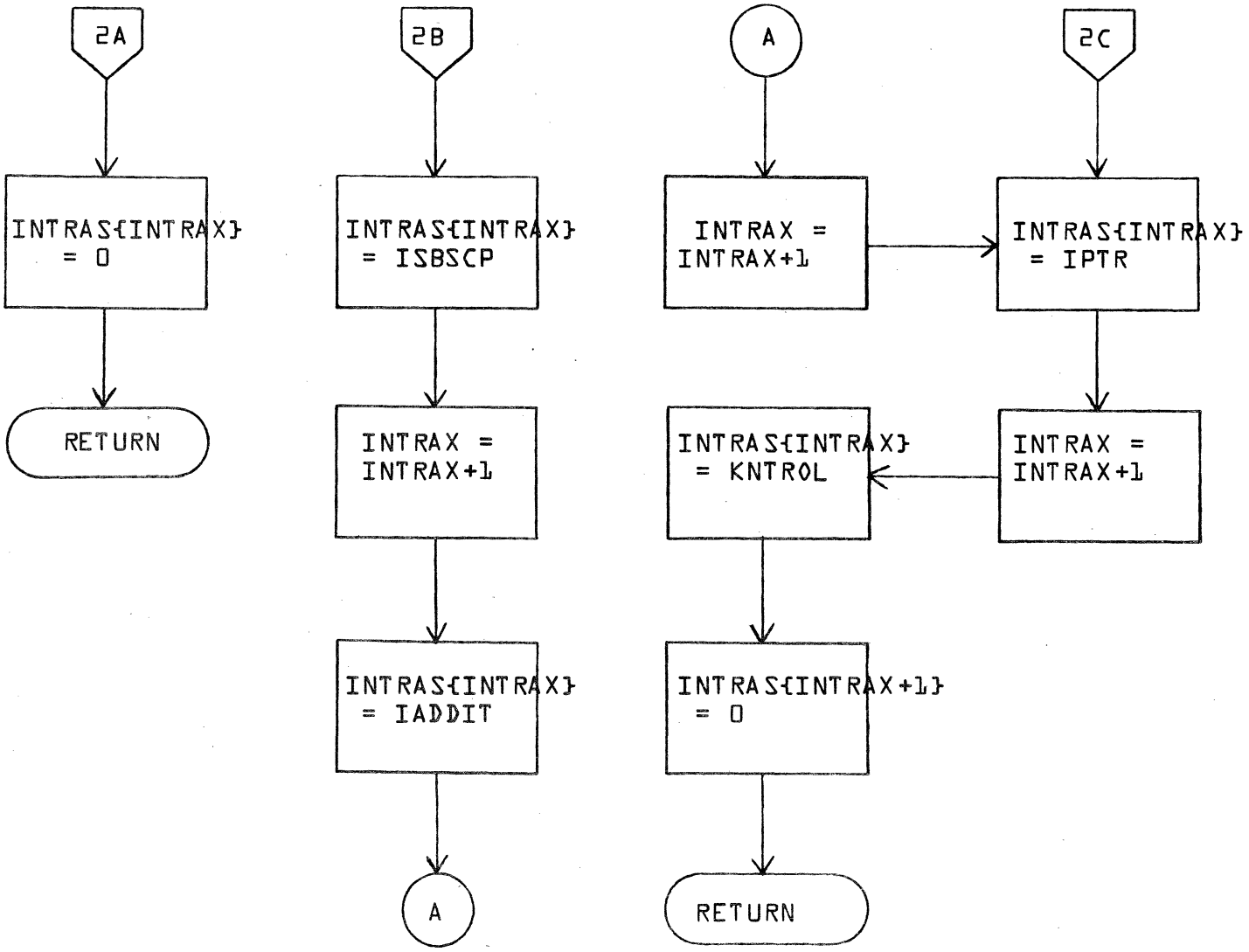
12. Backup over Parameter String. Function is exactly as in 11 except no actual code is output. This is performed whenever a function is done in-line and therefore, there is no need to generate address constants for the parameter list.



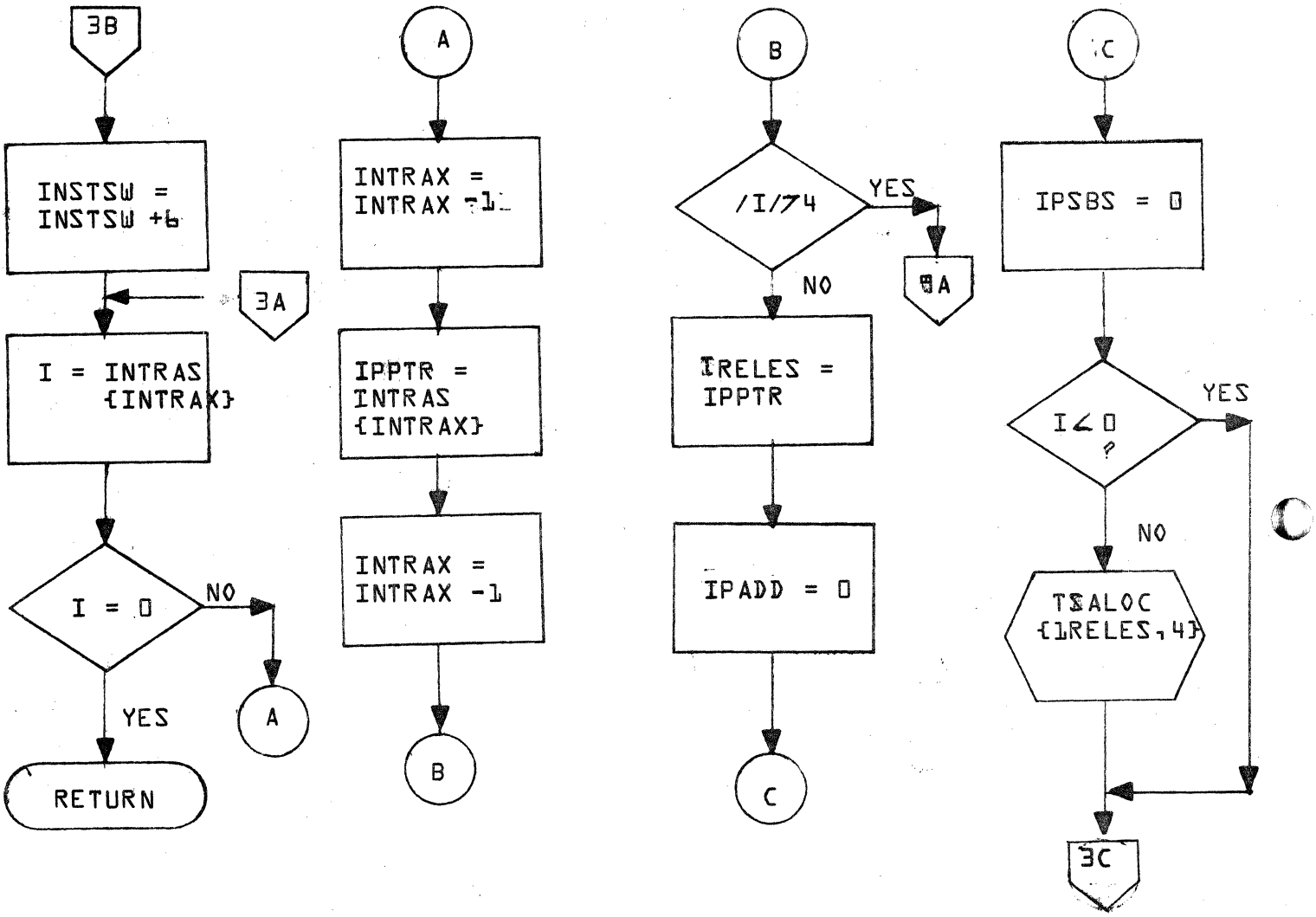
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE INTRAM		PROJECT MGR.		
		PAGE 1 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE INTRAM		PROJECT MGR.		
		PAGE 2 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

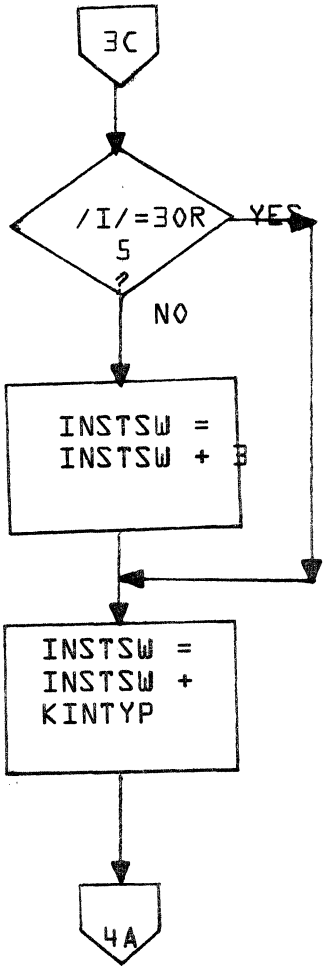


TITLE			DRG. NO.
INTRAm			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF 9

CSD 203

CONTROL DATA CORPORATION

LA JOLLA FACILITY

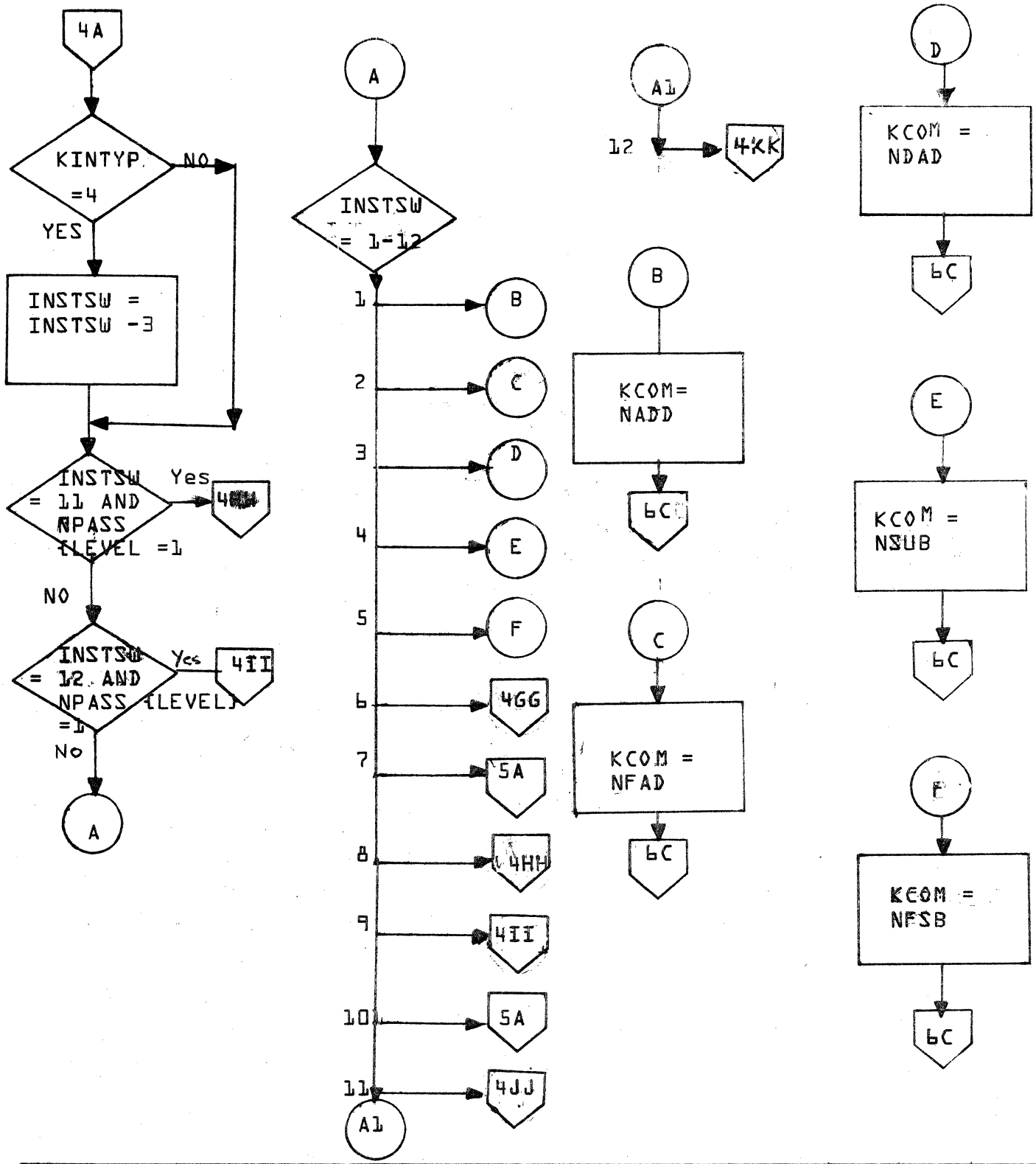


TITLE	INTRAM		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3A OF 9

CSD 203

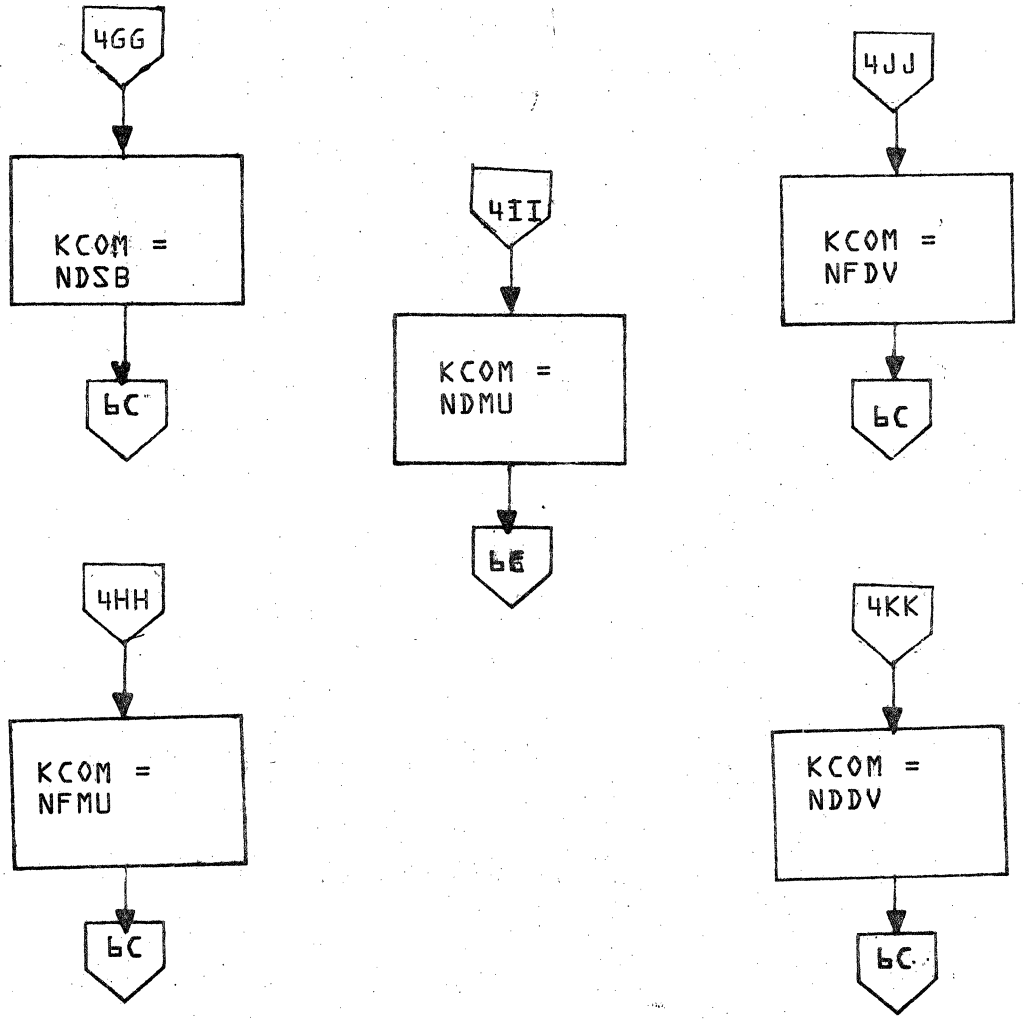
CONTROL DATA CORPORATION

LA JOLLA FACILITY



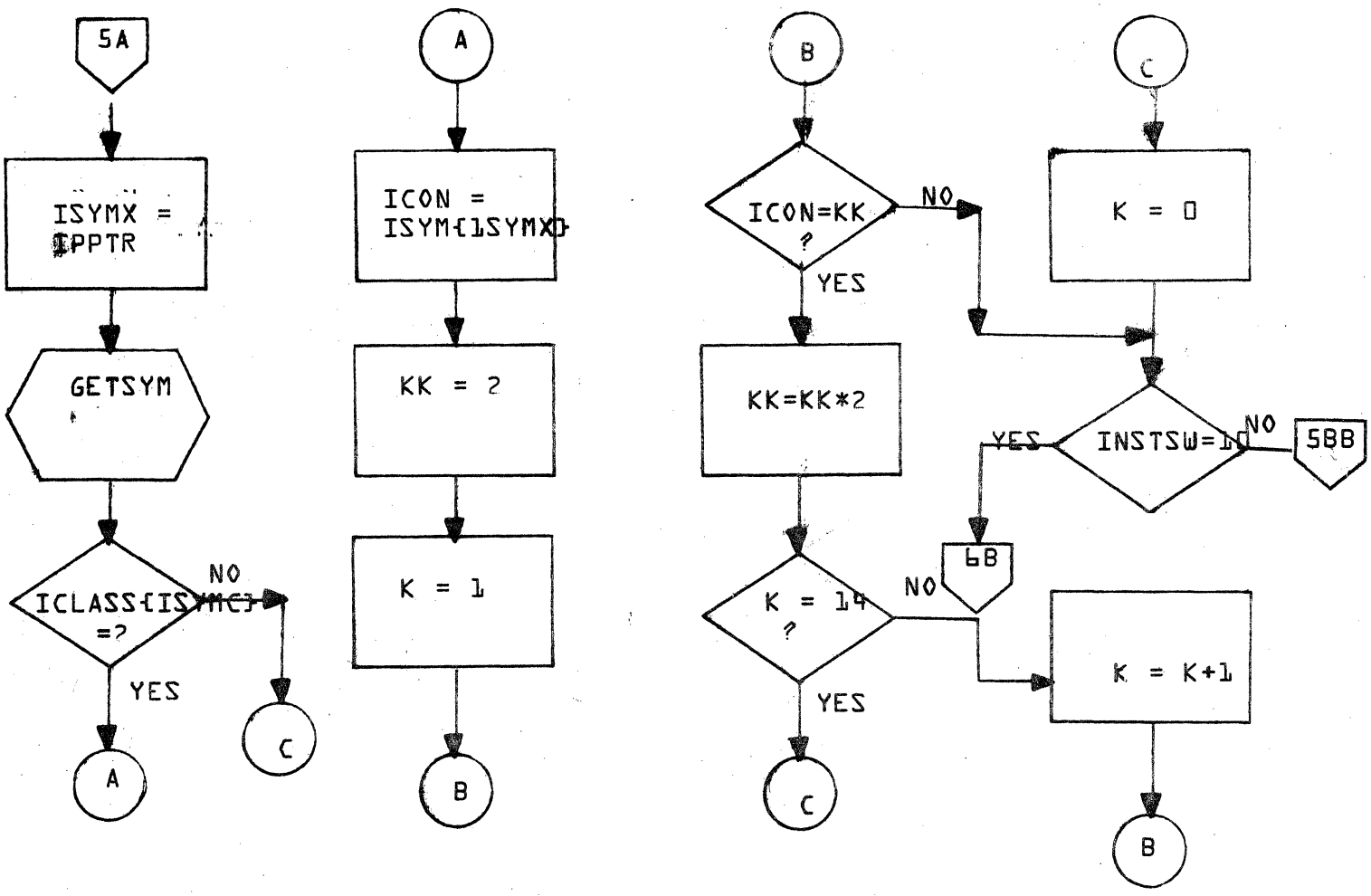
TITLE	INTRAM	DRG. NO.
		REVISION
DRAWN BY	PROJ.	DATE
		SHEET 4 OF 9

CSD 203


4GG
KCOM = NDSB
6C
4II
KCOM = NDMU
6E
4JJ
KCOM = NFDV
6C
4HH
KCOM = NFMU
6C
4KK
KCOM = NDDV
6C

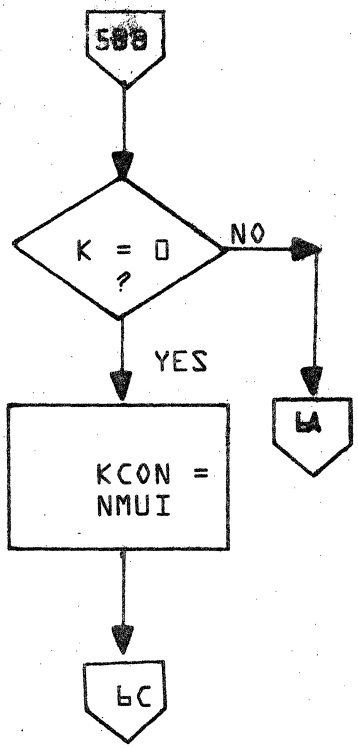
CONTROL DATA

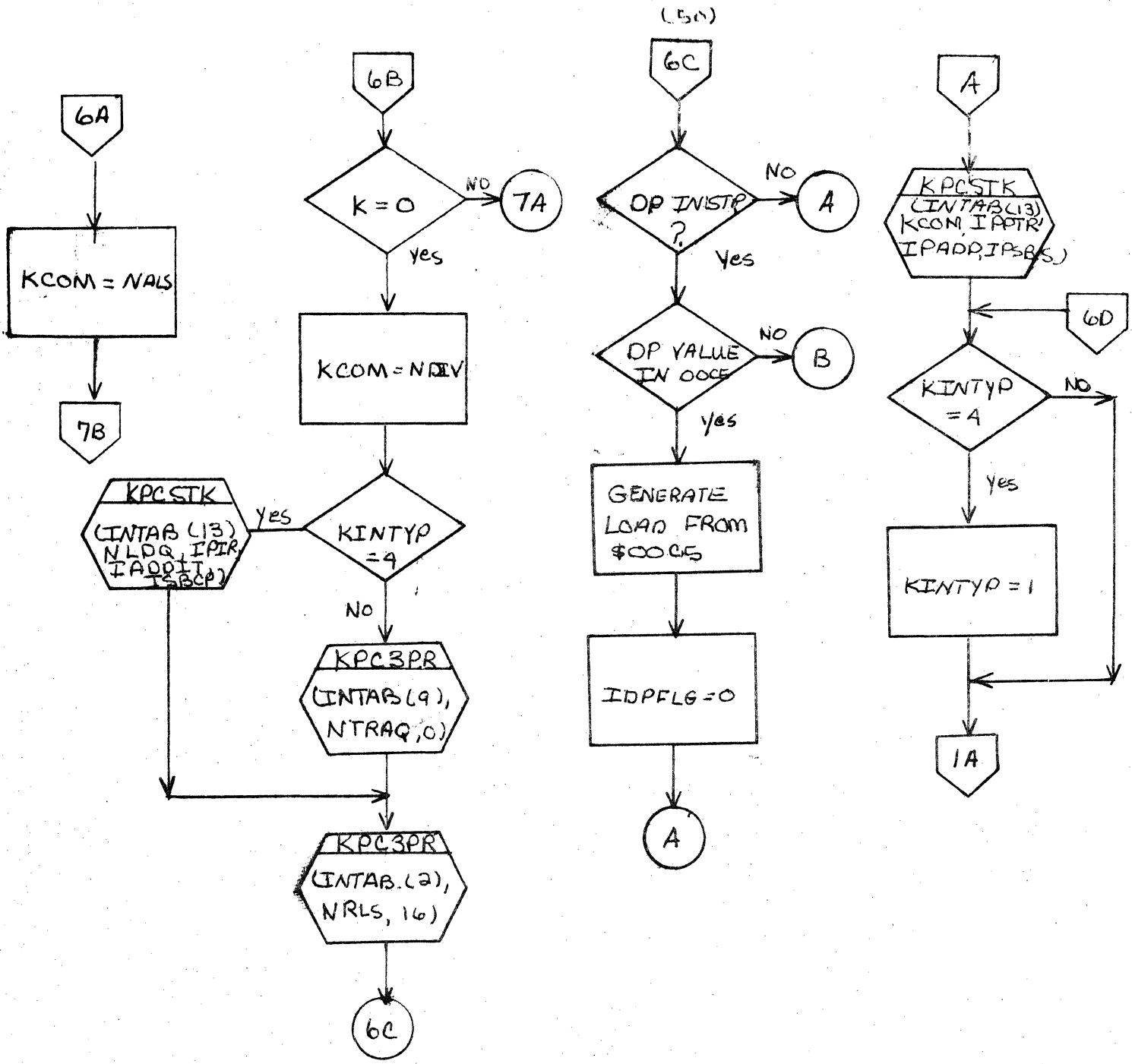
LA JOLLA FACILITY



TITLE	INTRAM	DRG. NO.
		REVISION
DRAWN BY	PROJ.	DATE	SHEET 5 OF 9

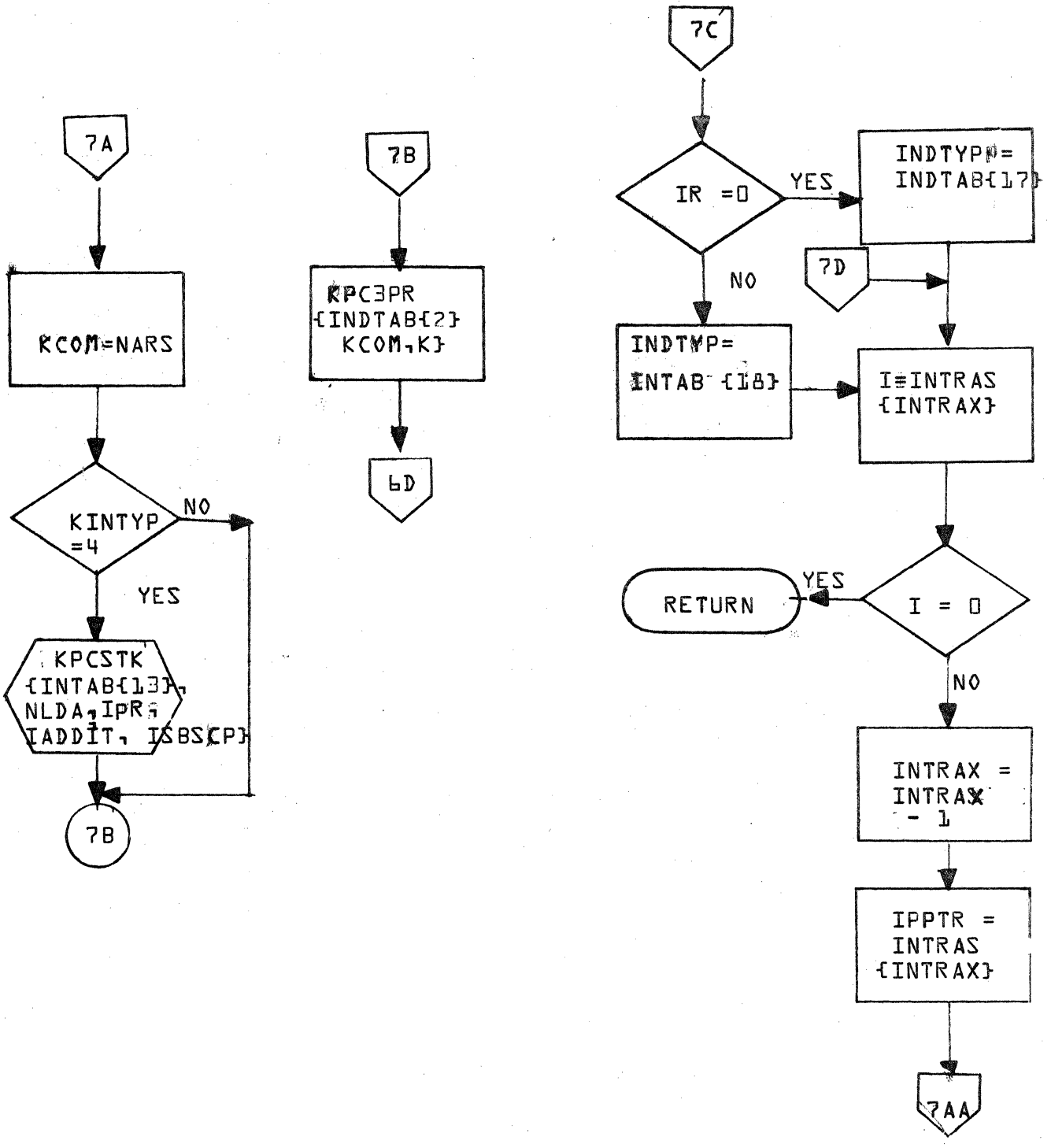
CSD 203


500
K = 0
?
NO
YES
KCON =
NMUI
6A
6C



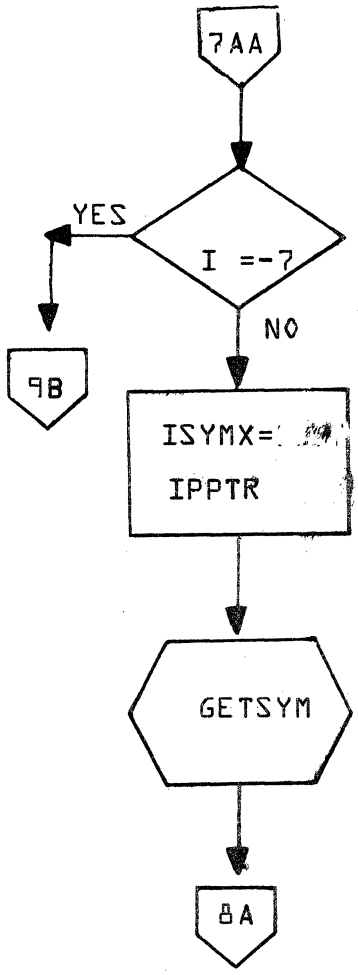
TITLE	INTRAM		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 60 OF 9

CSD 203



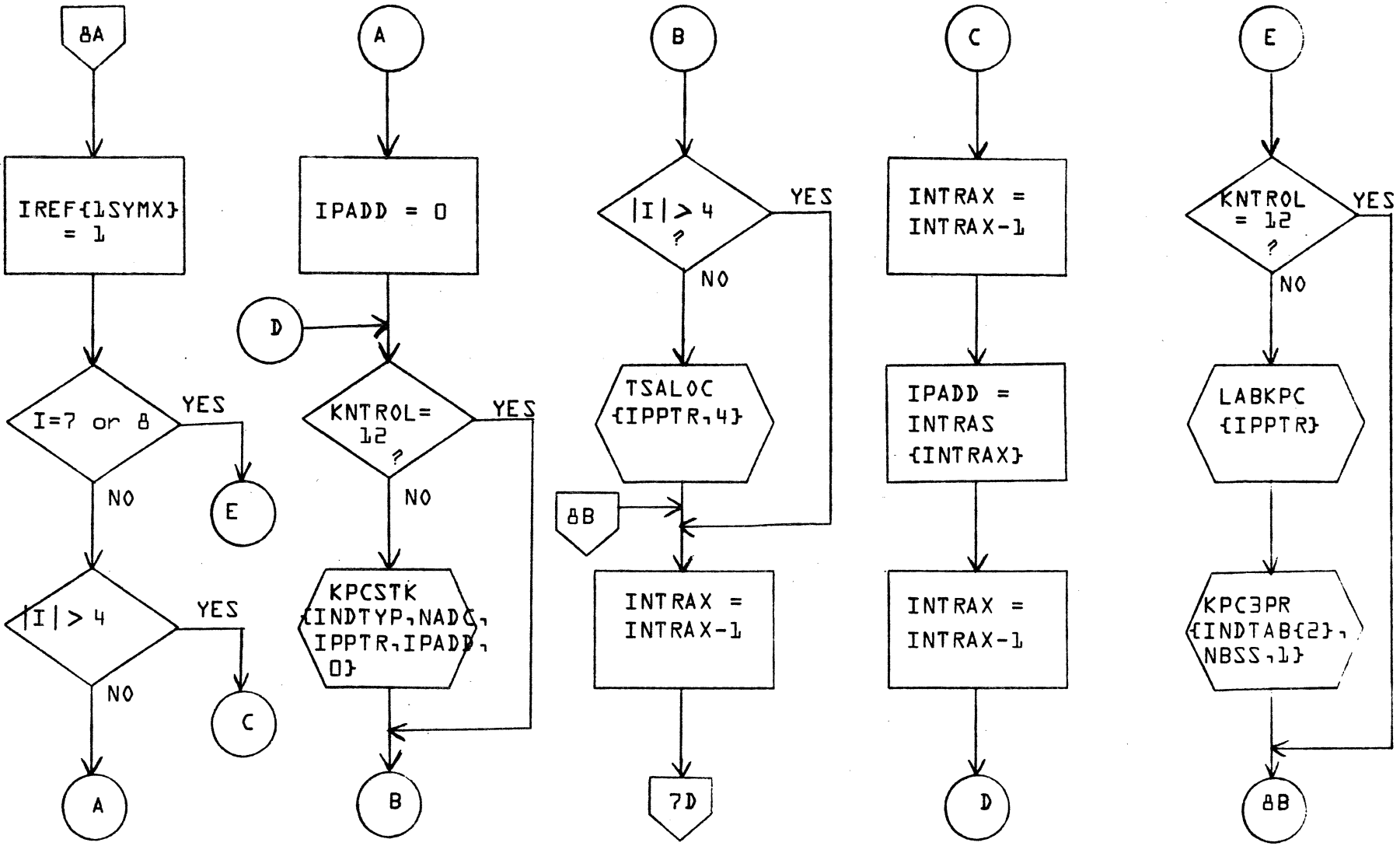
TITLE	INTRAM		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 7 OF 7

CSD 203

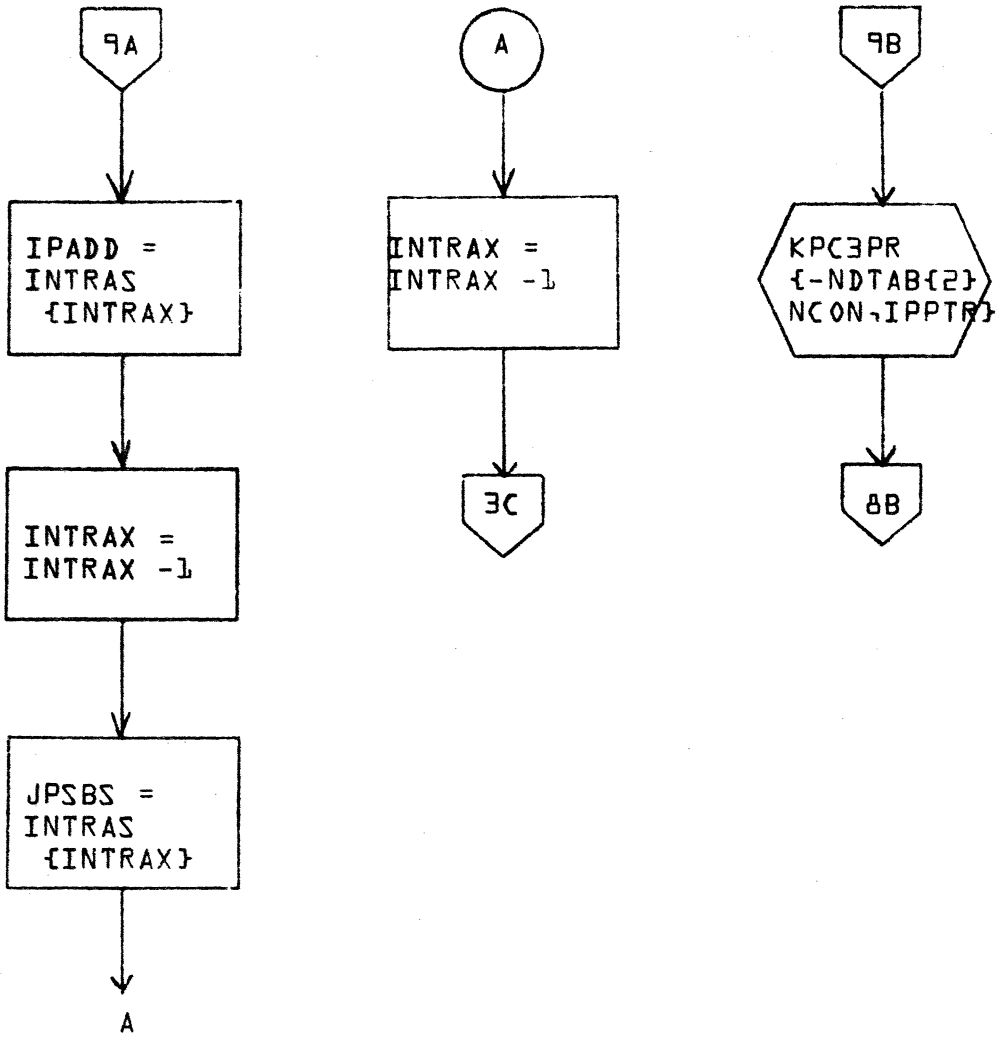


TITLE	INTRAm		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 7A OF 9

CSD 203



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE INTRAM		PROJECT MGR.		
		PAGE 8 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

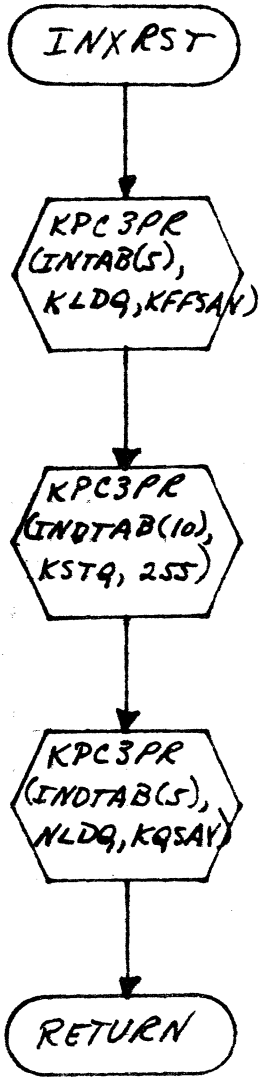


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE Intram	PROJECT MGR.			
	PAGE 9 OF 9	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

4.4.8 INXRST

INXRST is a short subroutine to generate the instruction sequence which restores the Q and I index registers. INXRST is called by END and PHASEB



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE INXRST	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

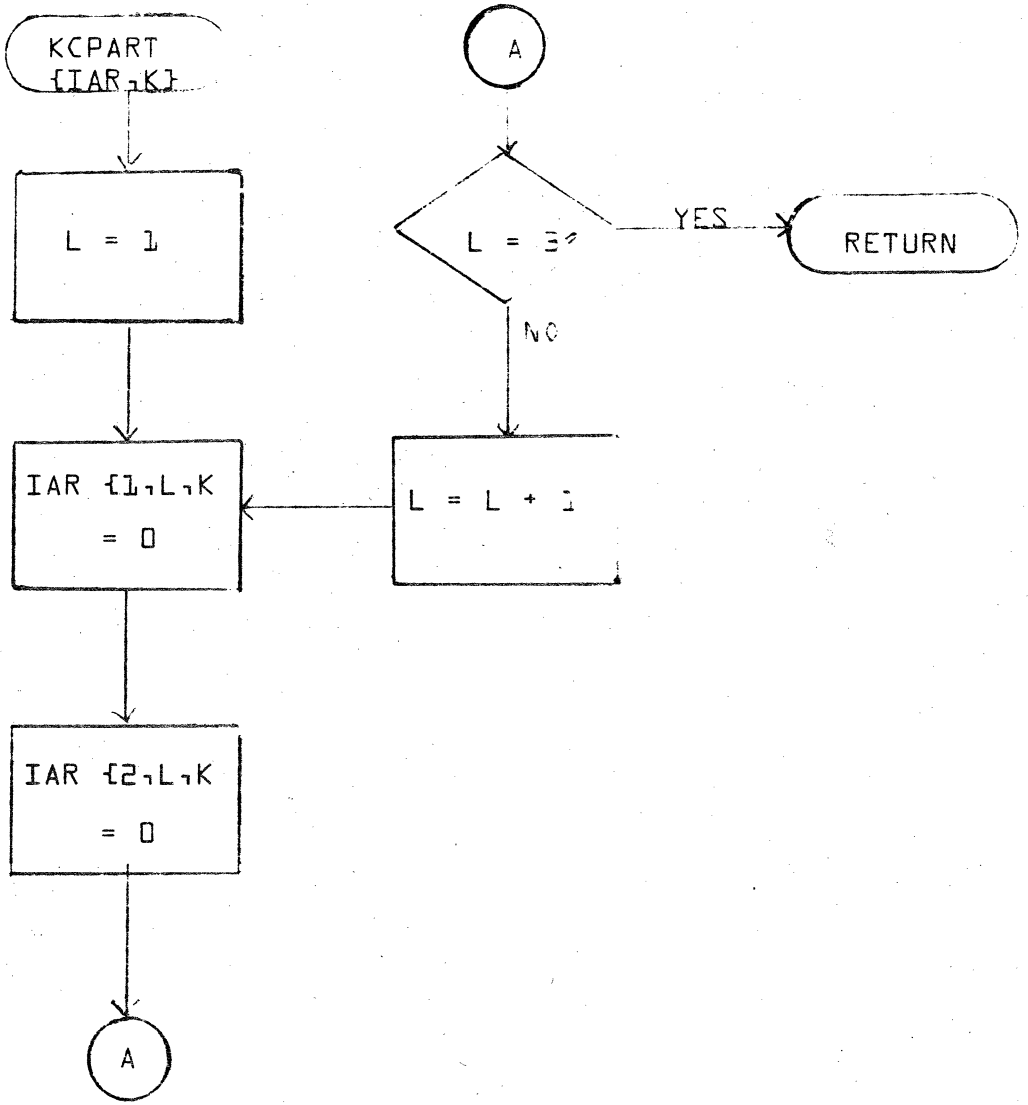
FILED

4.4.9 KCPART

KCPART is used in conjunction with KPCSTK in keeping a memory of the 'current' contents of the accumulator, single precision floating point pseudo accumulator, and double precision floating point pseudo accumulator. KCPART is called to clear either the accumulator portion, sp pseudo accumulator portion, or dp pseudo accumulator portion of the accumulator record table KART.

PARAMETERS:

IAR	address of table KART
K	=1 clear accumulator record
	=2 clear sp pseudo accumulator record
	=3 clear dp pseudo accumulator record



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IME	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE KCPART		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

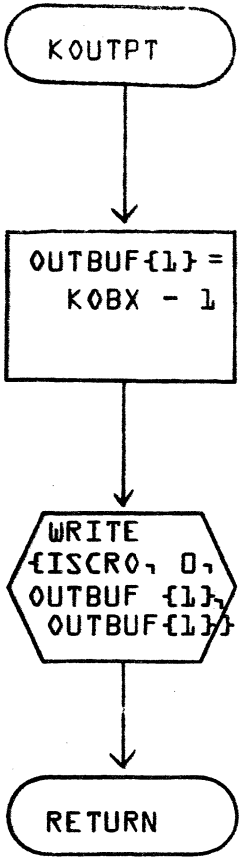
4.4.10 KOUTPT

KOUTPT is the routine called to write a PHASE4 pseudo instruction output record. It is called upon for all pseudo instructions but not for the special records such as DATA and FORMAT.

Steps:

1. Transfer the number of words in this record to OUTBUF(1) (the word in any output record which normally designates the number of words in the record). This value is obtained by subtracting 1 from KOBX the output buffer index.

2. Call WRITE.



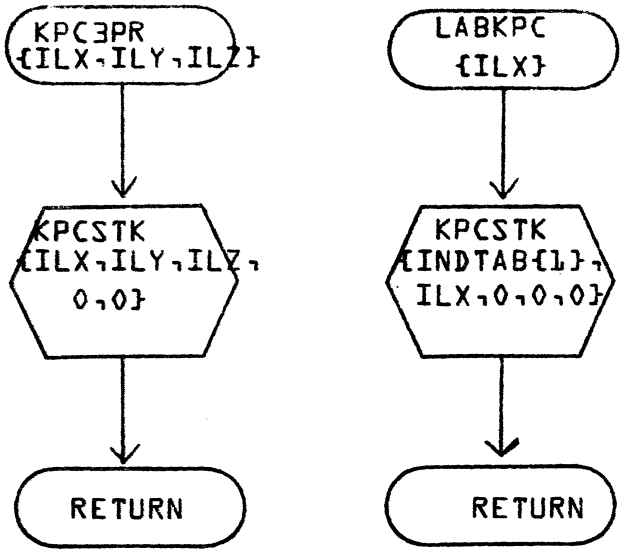
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE KOUTPT		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

4.4.11 KPC3PR and LABKPC

Both of these routines are simply a single call to KPCSTK. Most calls to KPCSTK are made through KPC3PR or LABKPC to reduce the number of parameters, hence the number of words in the call.

KPC3PR handles all calls where the last two of the five KPCSTK parameters are always 0. LABKPC handles all calls to KPCSTK which are requests to output a label. (Only the second parameter varies from one call to the next).



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: KPC3PR & LABKPC		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

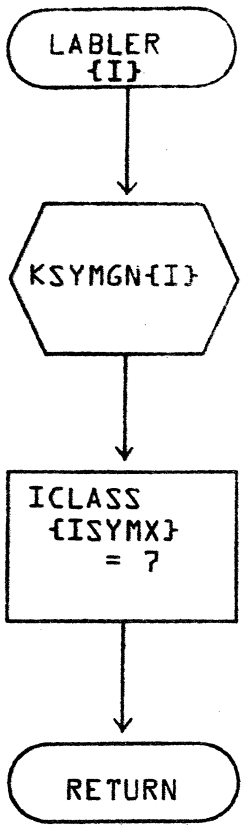
4.4.12 LABLER

LABLER is called by various PHASEB subroutines to create a symbol table entry which represents a label, i.e., a FORTRAN generated label.

Parameters:

I - returns the symbol table pointer of the entry created by LABLER.

Flowchart:

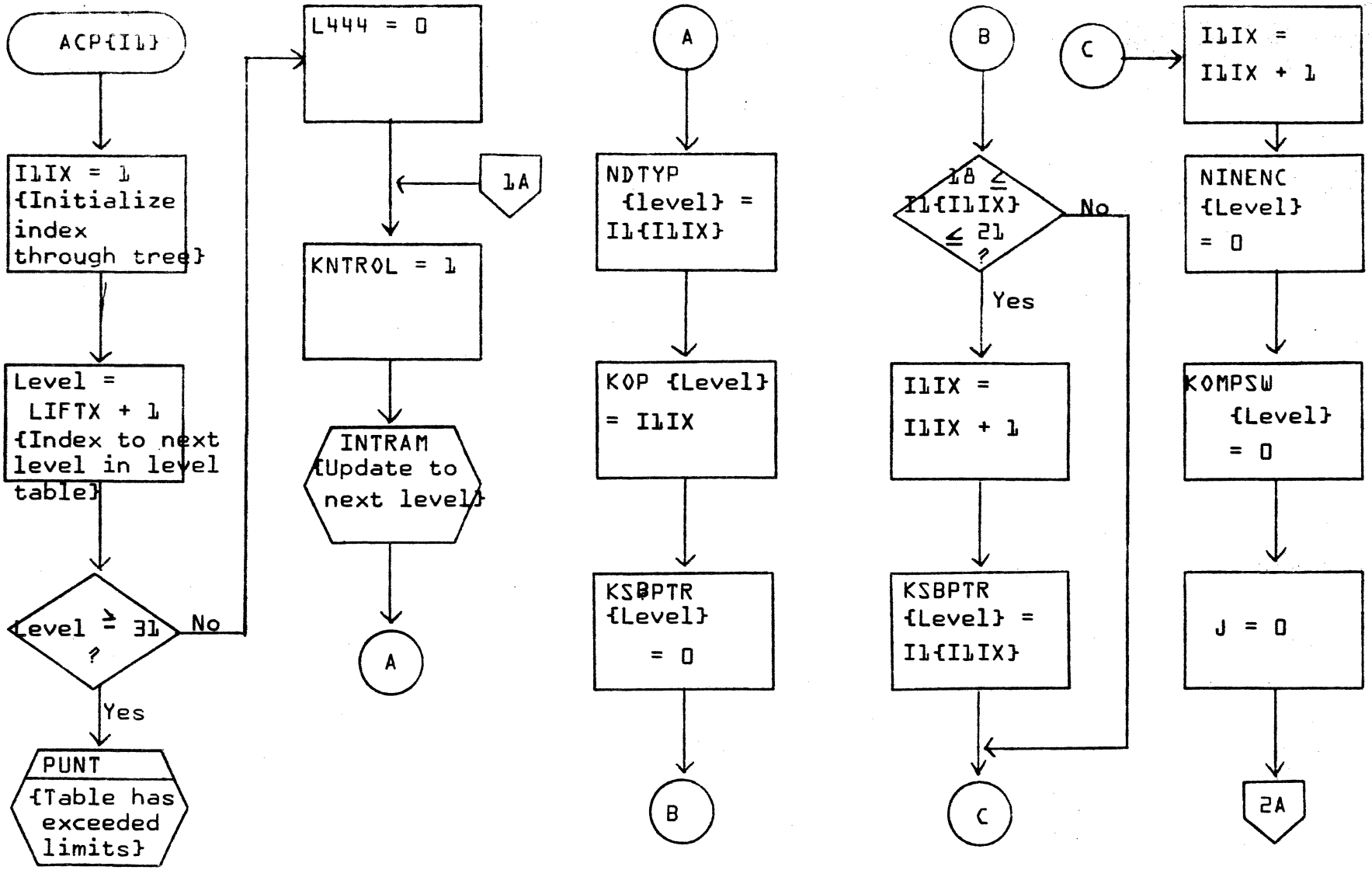


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE LABLER		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

4.4.13 ACP

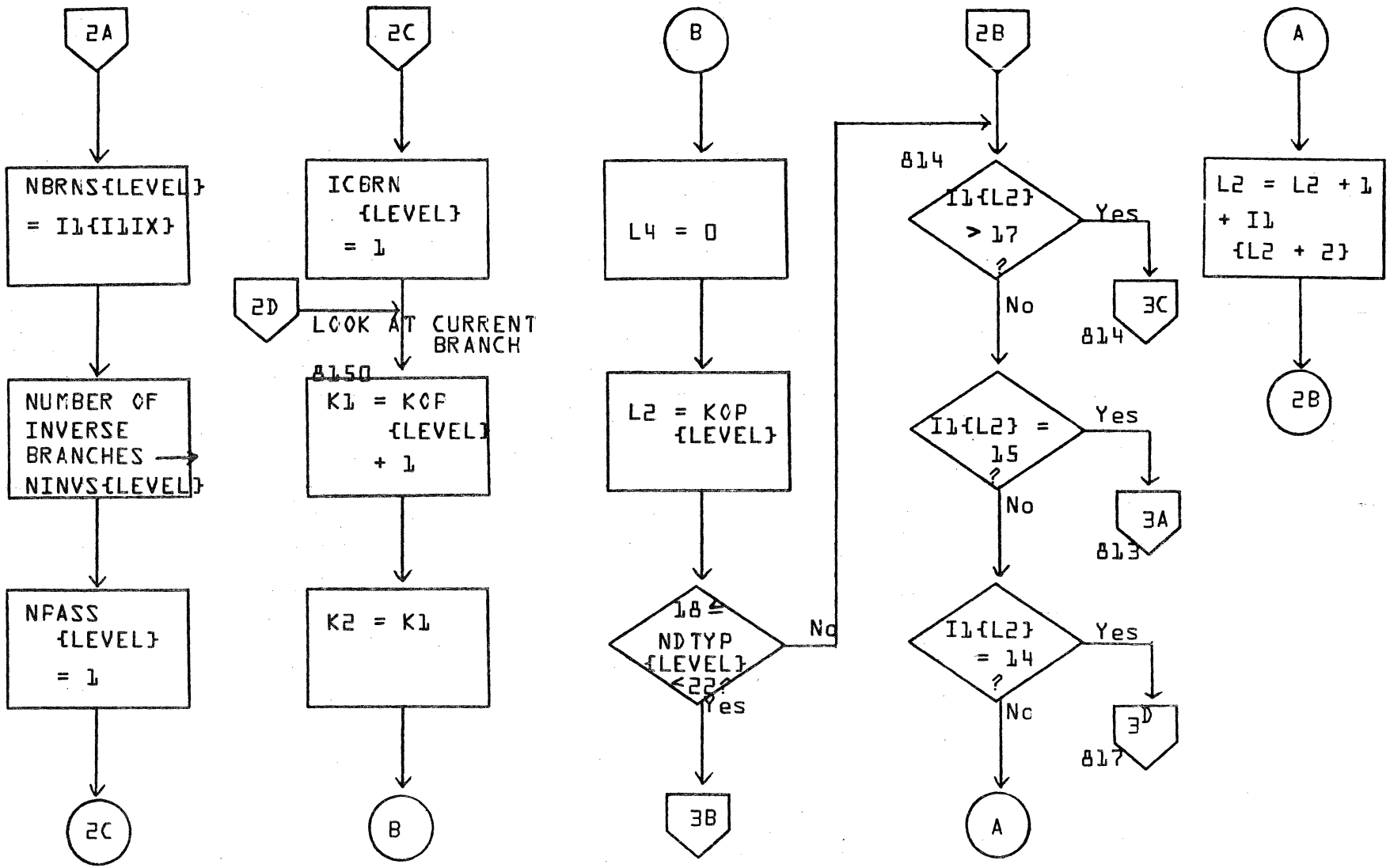
ACP is the routine which actually processes arithmetic trees. ACP scans each tree and generates commands. ACP is called by ASUPER and SUBPR3. ASUPER and SUBPR3 are in turn called by individual statement processes. ACP calls INTRAN or KPCSTK directly to output commands.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: ACP		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 17	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

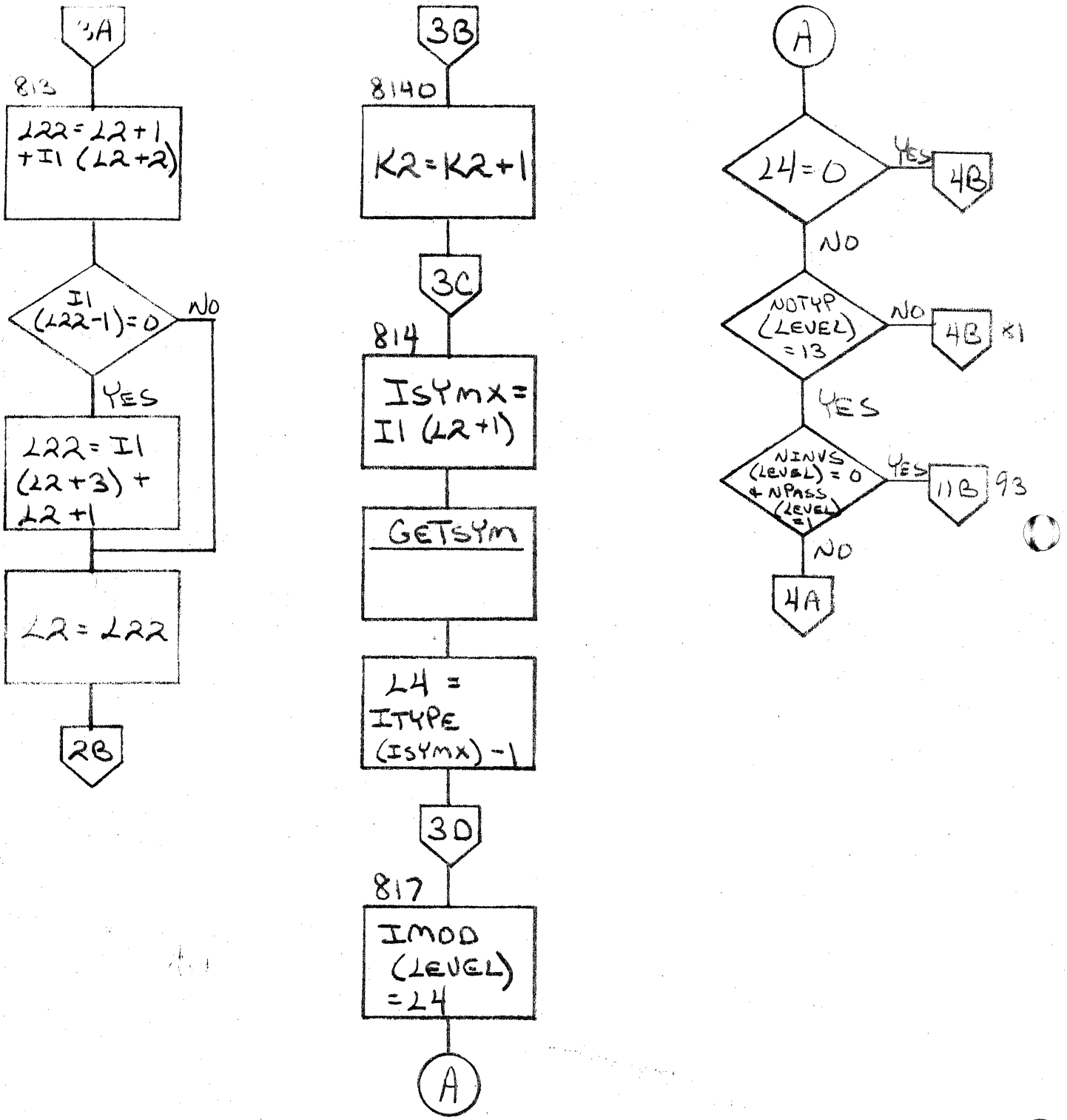
4-130

AA1385 (FORMERLY CA127-1)



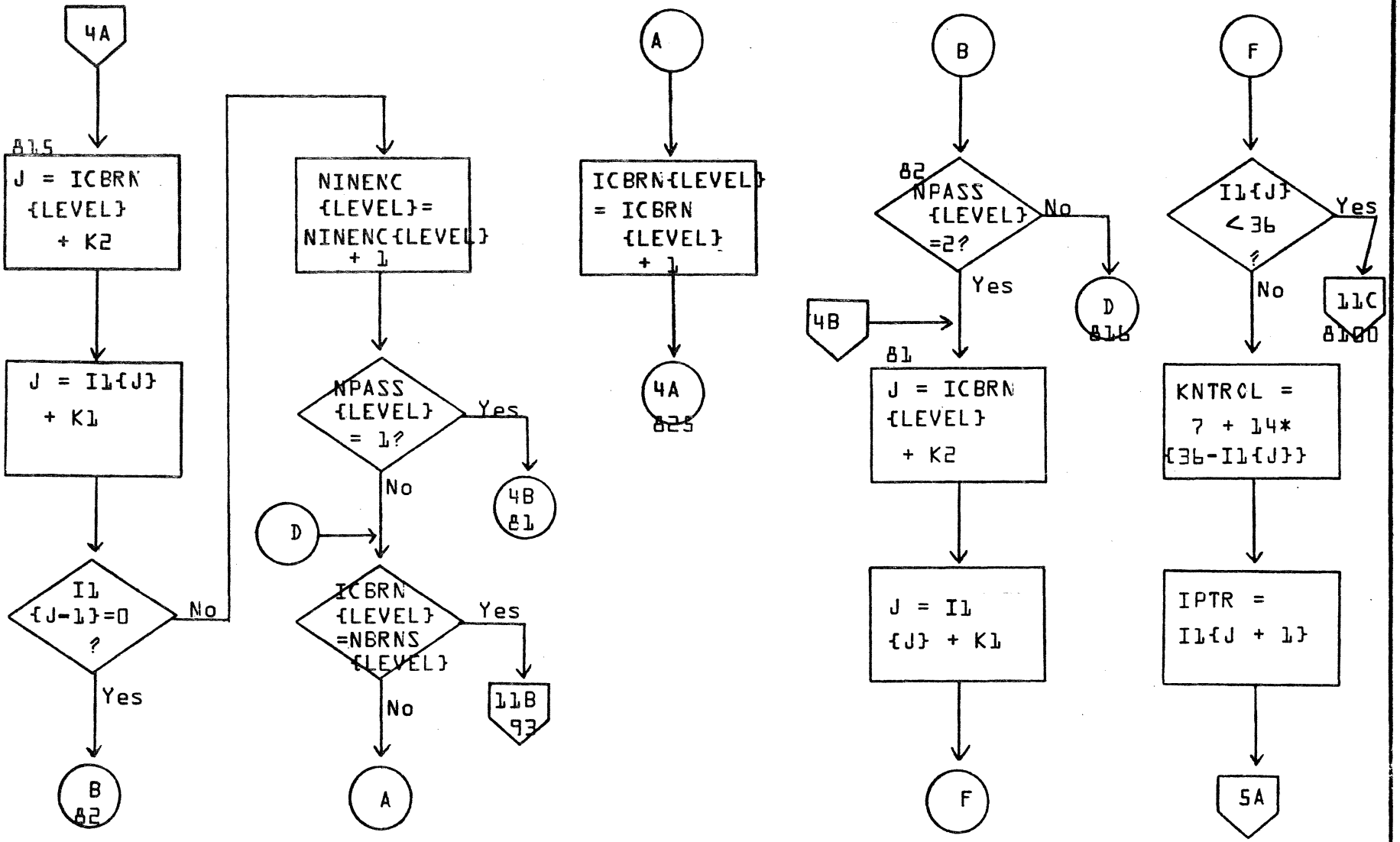
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ACF		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 17	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

4-131

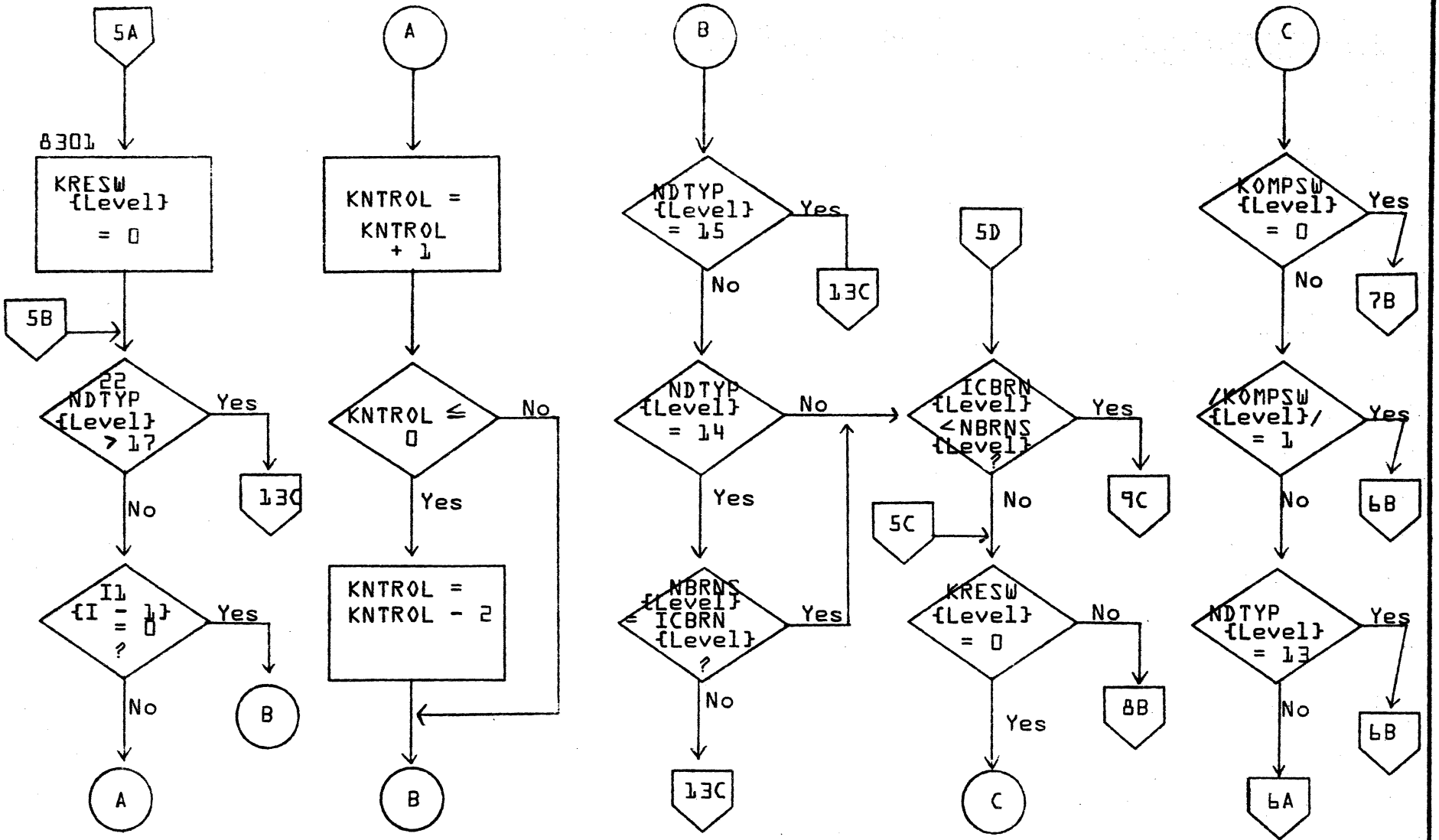


TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF 17

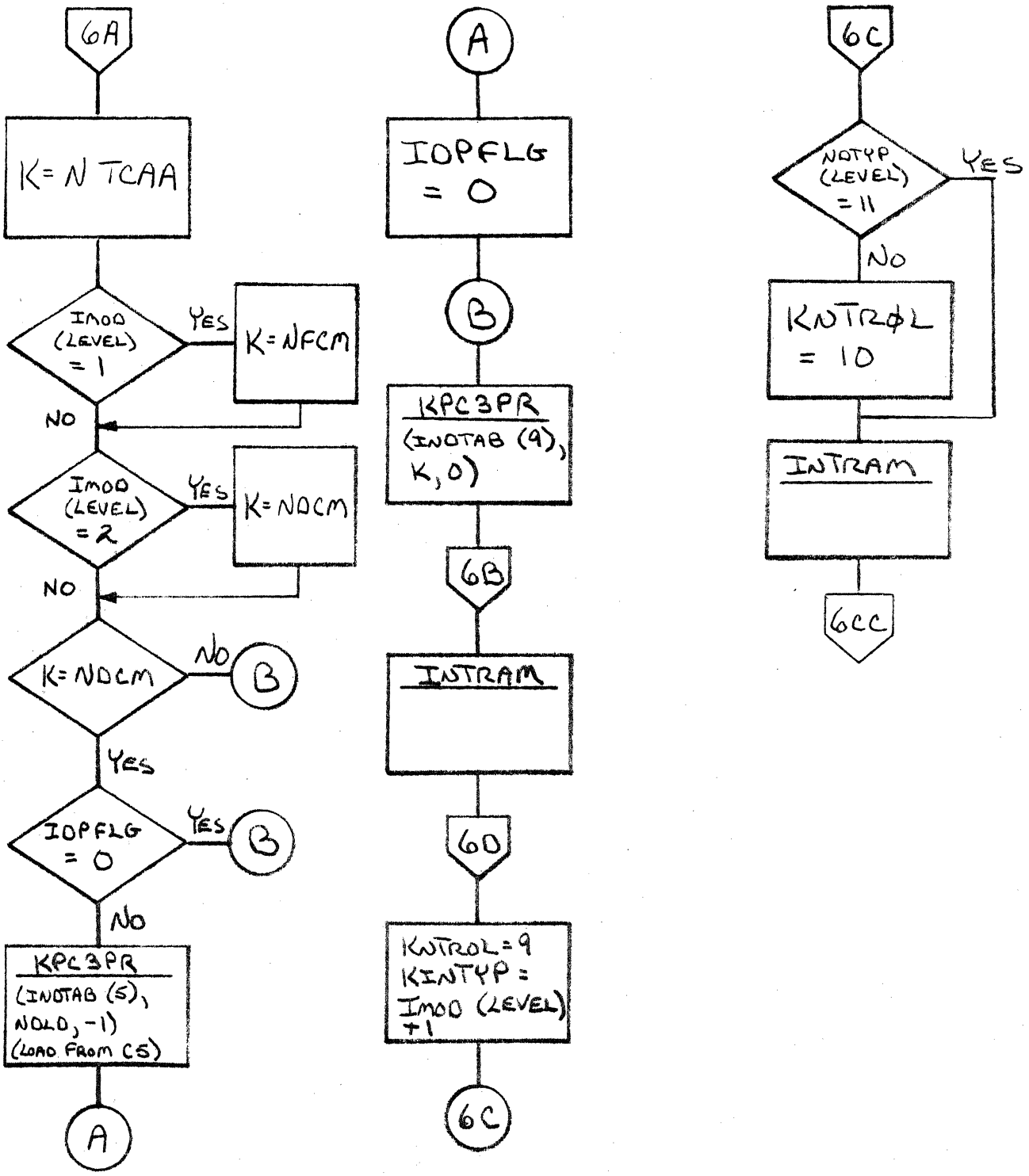
CSD 203



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ACP		PROJECT MGR.		
		PAGE 4 OF 17	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

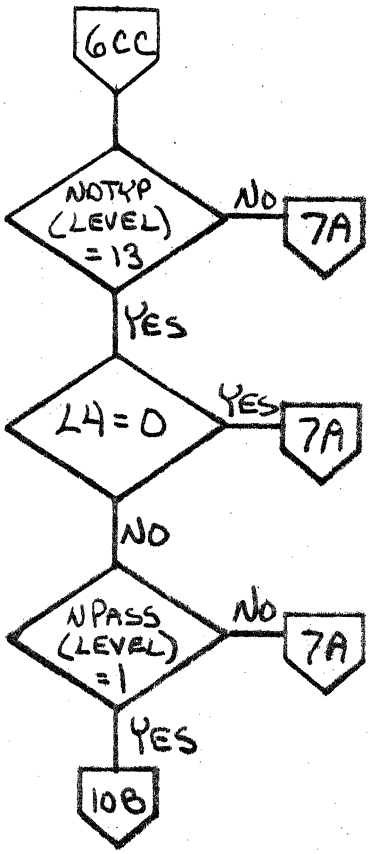


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: ACP		PROJECT MGR.			
FLOWCHART ☐		PAGE 5 OF 17	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



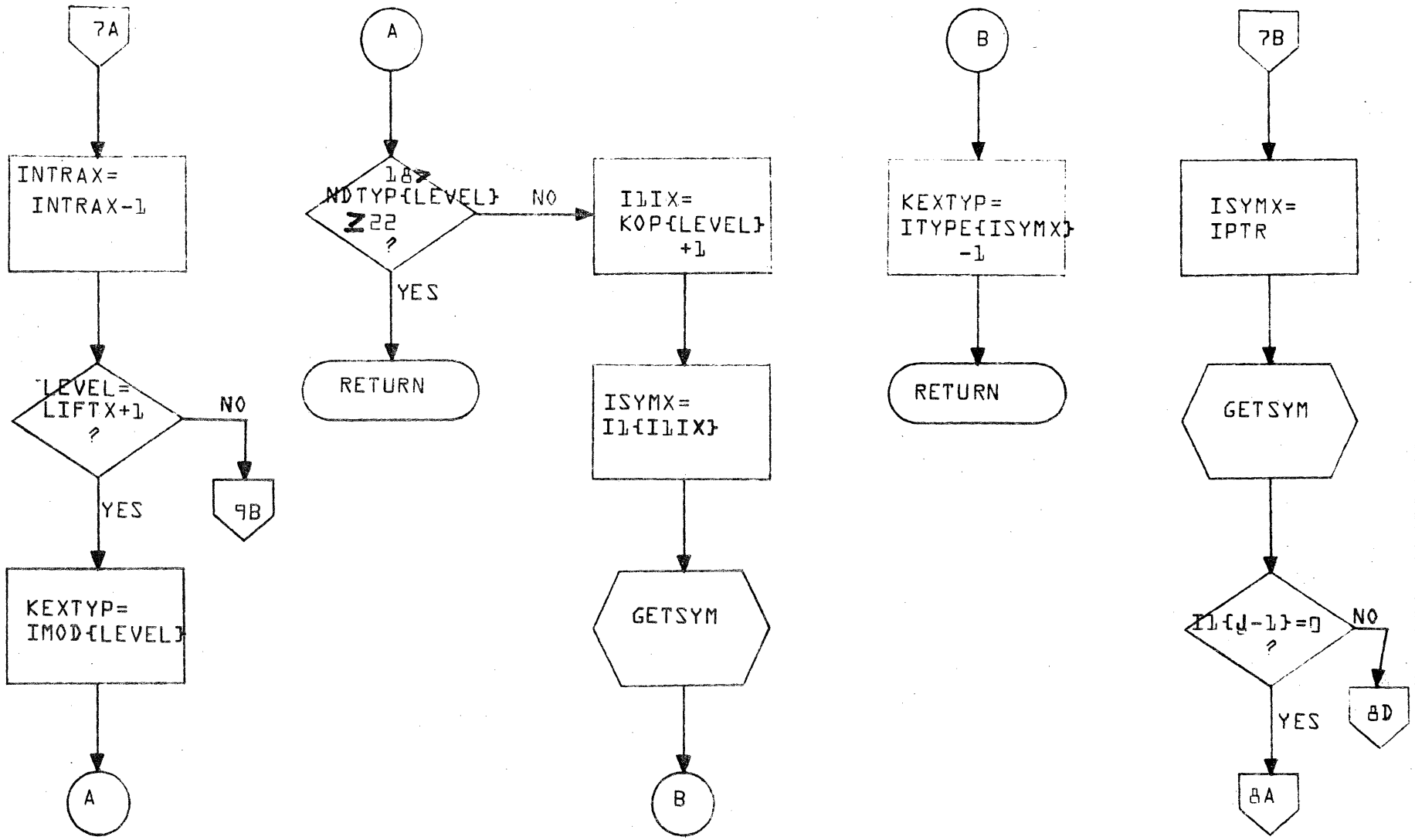
TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 6 OF 17

LA JOLLA FACILITY



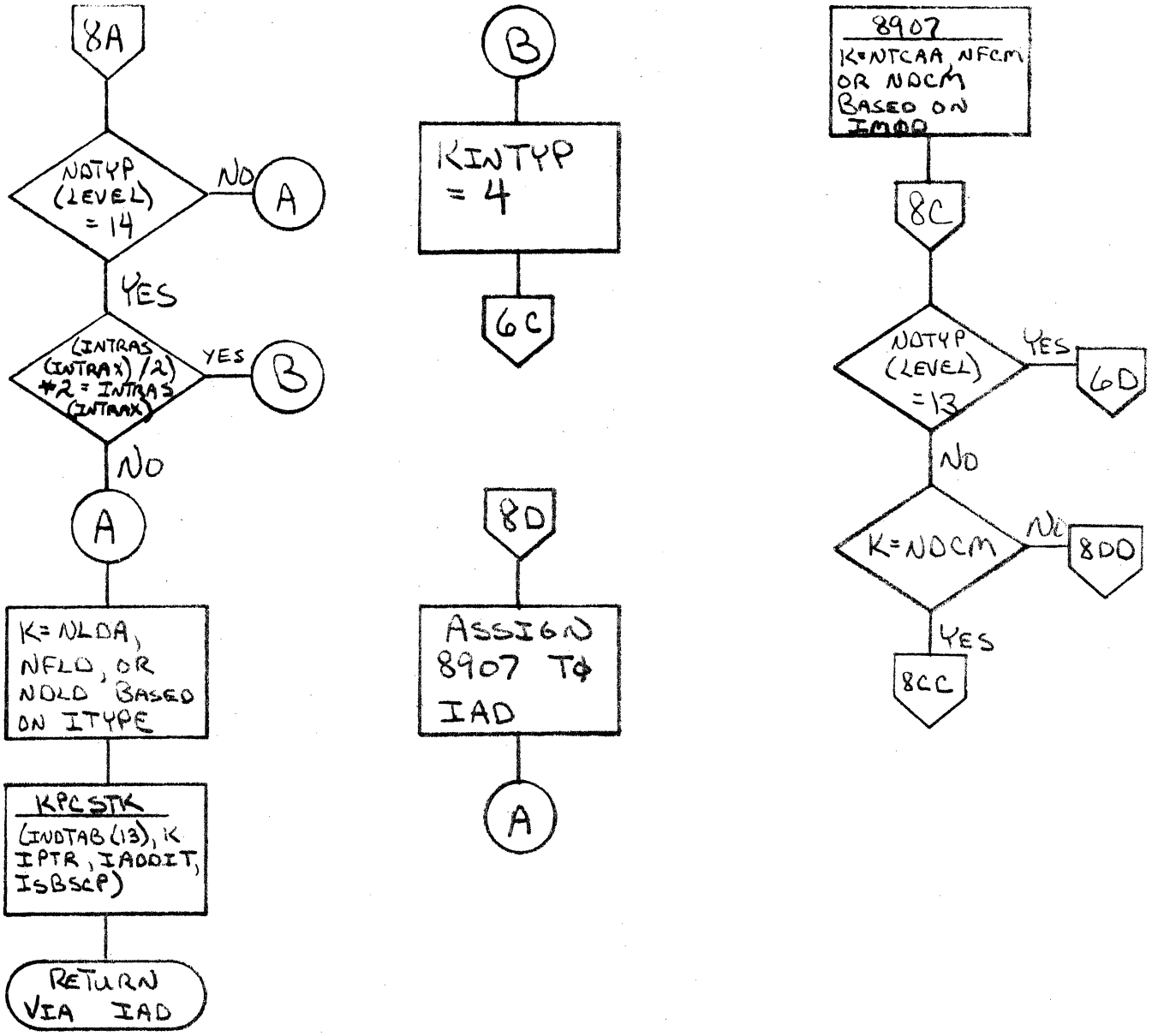
TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 6A OF 17

CSD 203



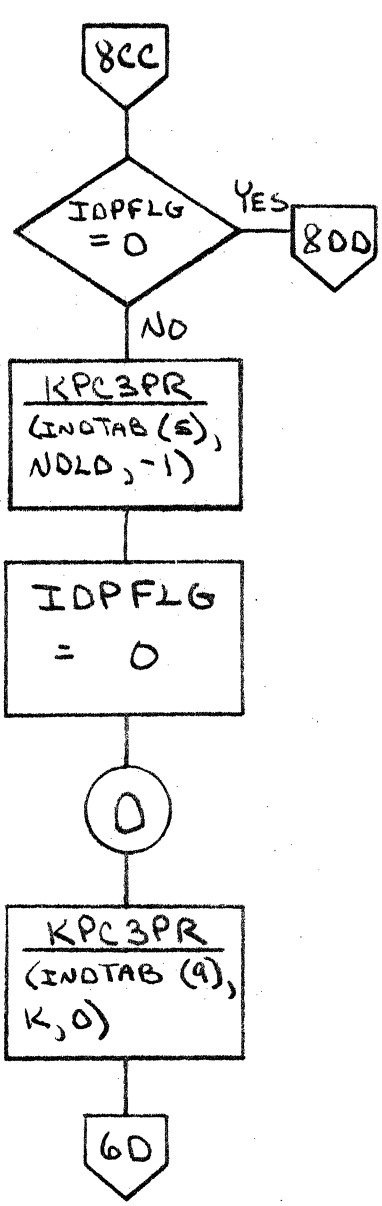
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ACP		PROJECT MGR.		
		PAGE 7 OF 17	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

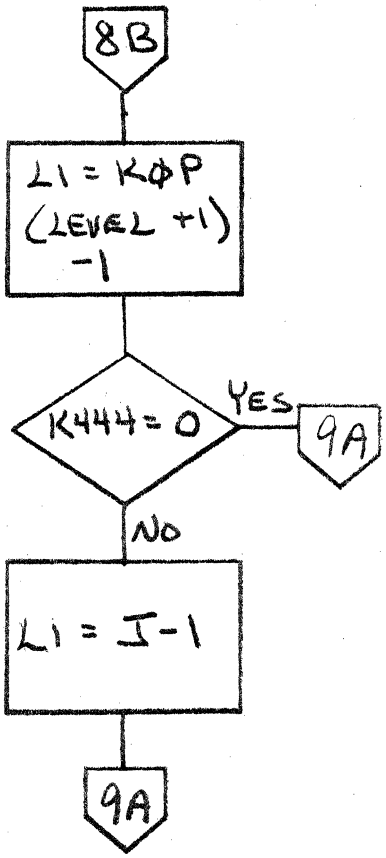
LA JOLLA FACILITY

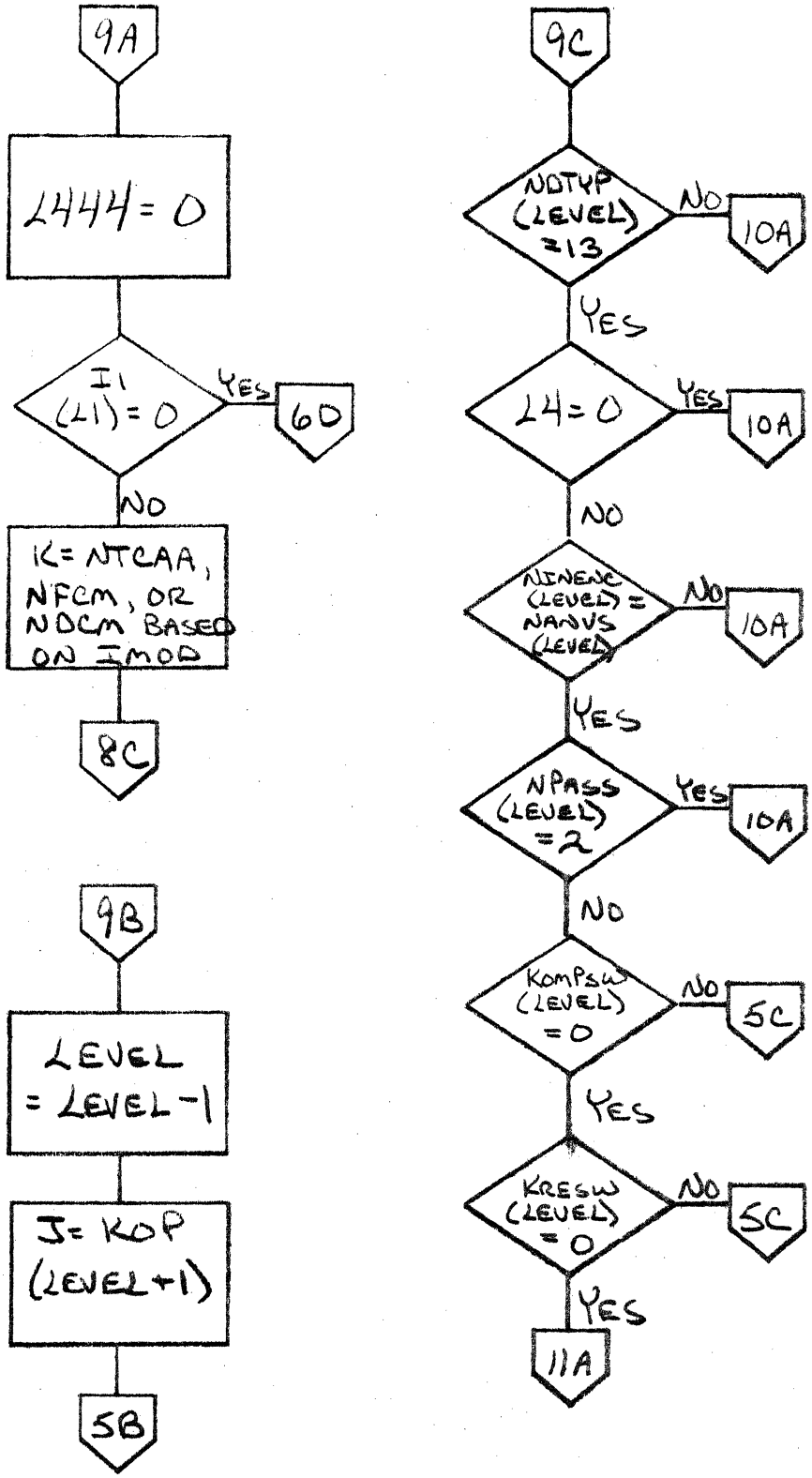


TITLE	ACP		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 8 OF 17

CSD 203

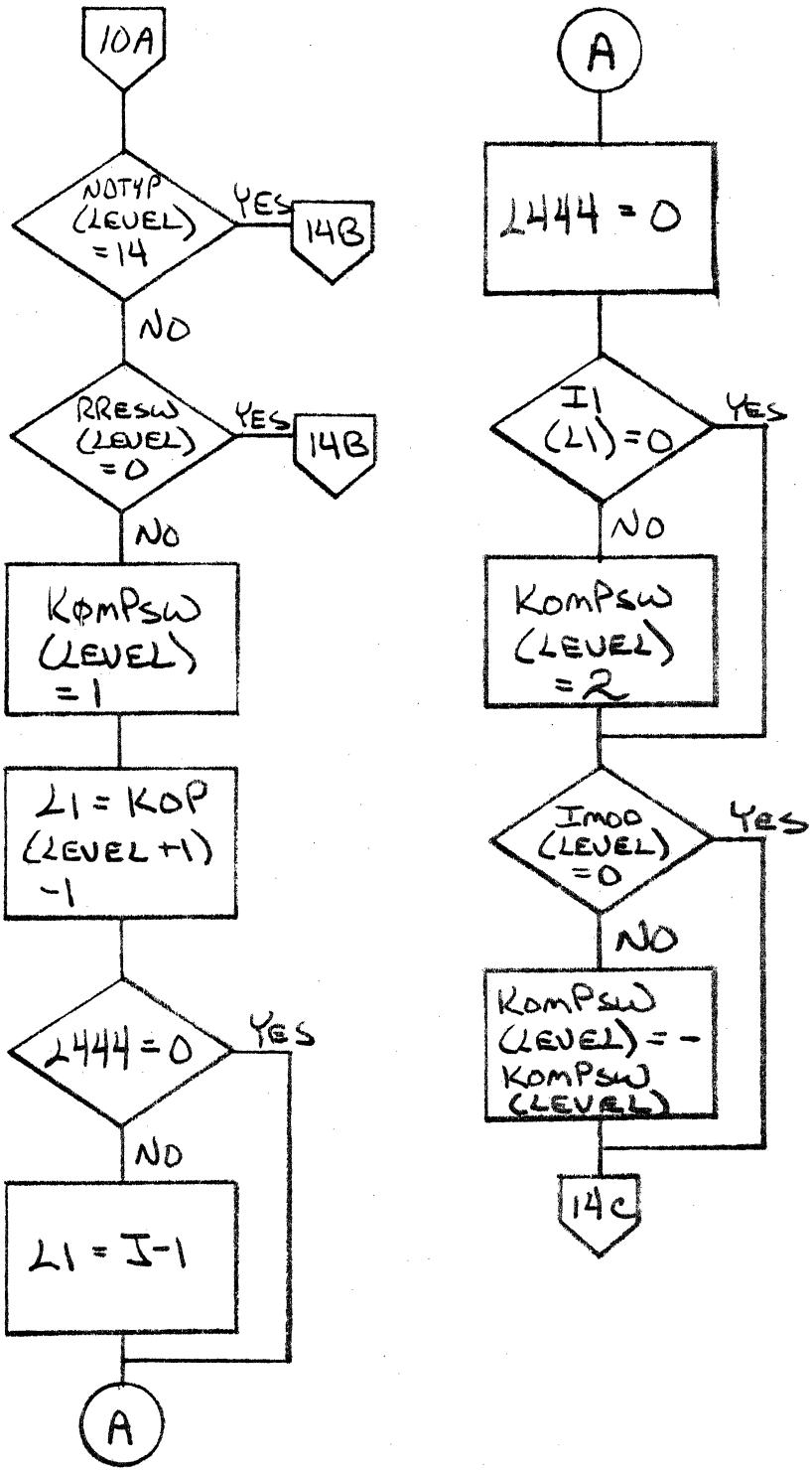

8CC
IDPFLG = 0
YES
8DD
NO
KPC3PR (INOTAB(6), NOLD, -1)
IDPFLG = 0
0
KPC3PR (INOTAB(9), K, 0)
6D


8B
L1 = KΦP (LEVEL+1) -1
K444 = 0
YES
9A
NO
L1 = J-1
9A

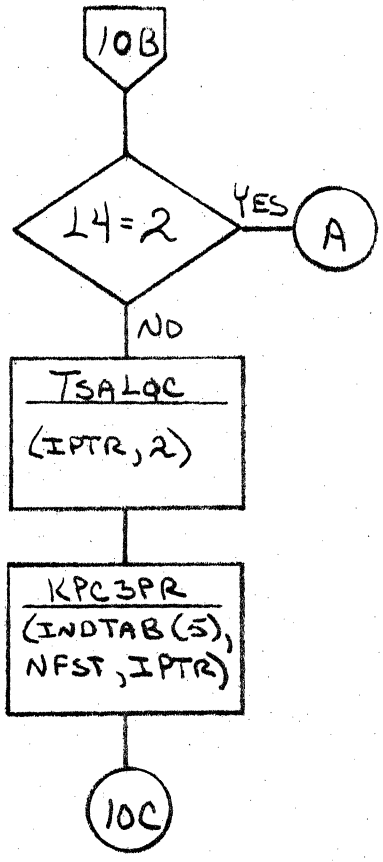


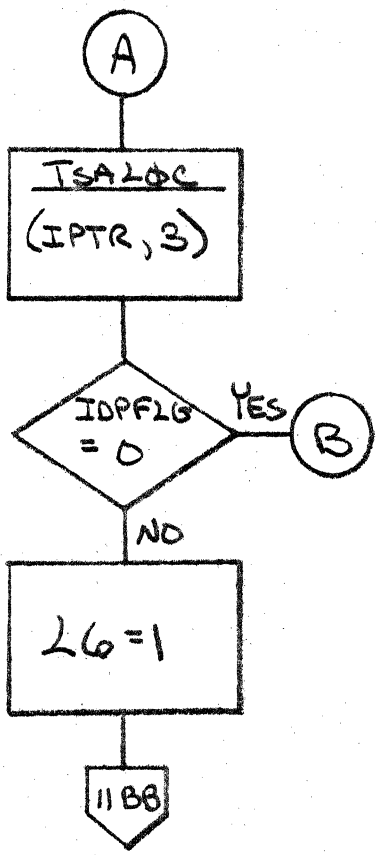
TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 9 OF 17

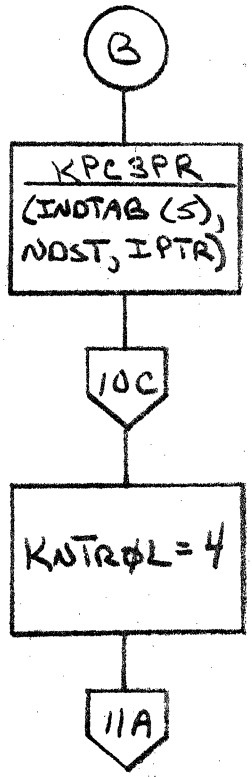
CSD 208


10A
NOTYP (LEVEL) =14
YES
14B
NO
RRESW (LEVEL) =0
YES
14B
NO
KOMPSW (LEVEL) =1
L1 = KOP (LEVEL+1) -1
L444 = 0
YES
NO
L1 = J-1
A
A
L444 = 0
I1 (L1) = 0
YES
NO
KOMPSW (LEVEL) =2
IMOD (LEVEL) =0
YES
NO
KOMPSW (LEVEL) = - KOMPSW (LEVEL)
14C

LA JOLLA FACILITY

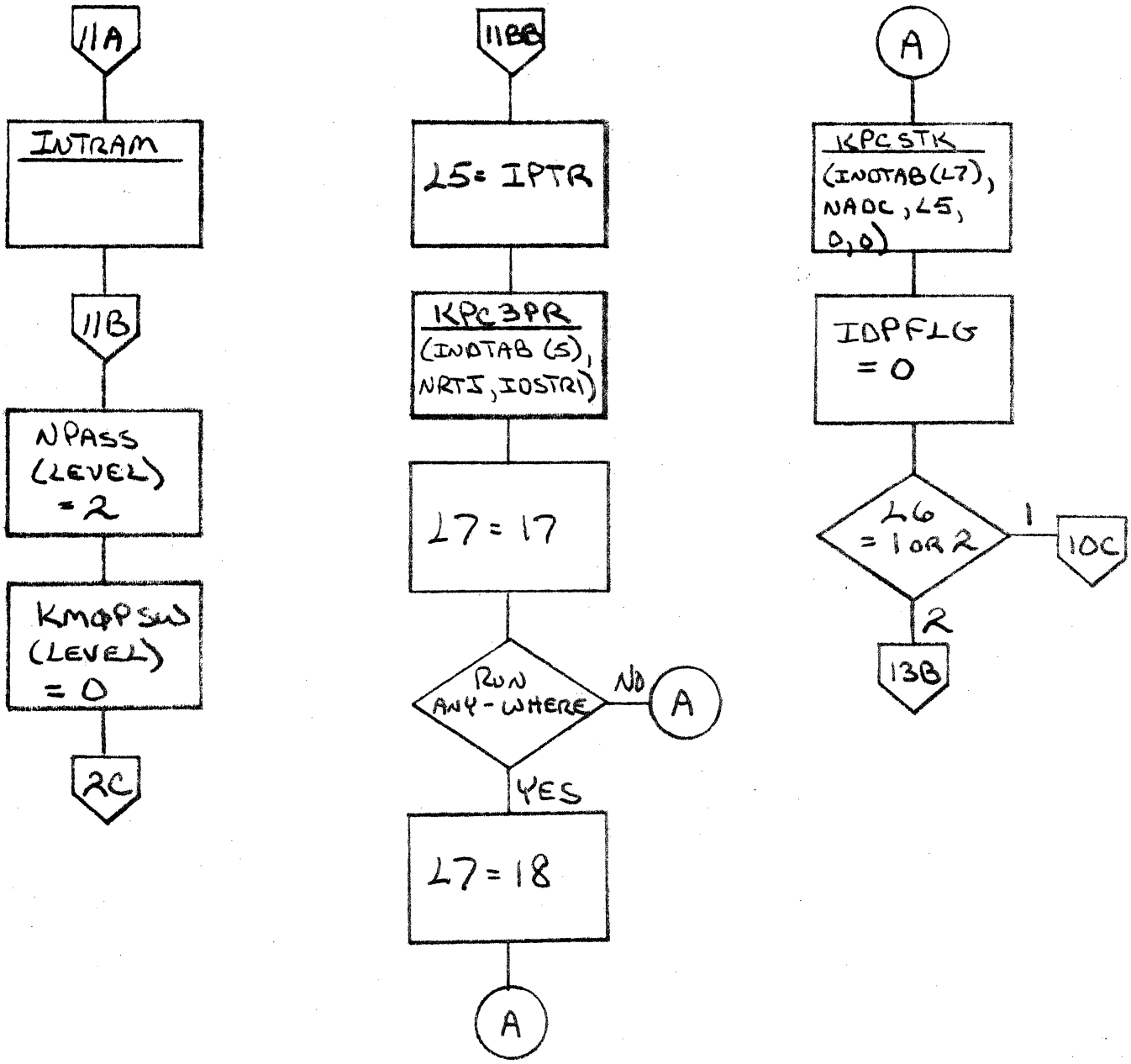






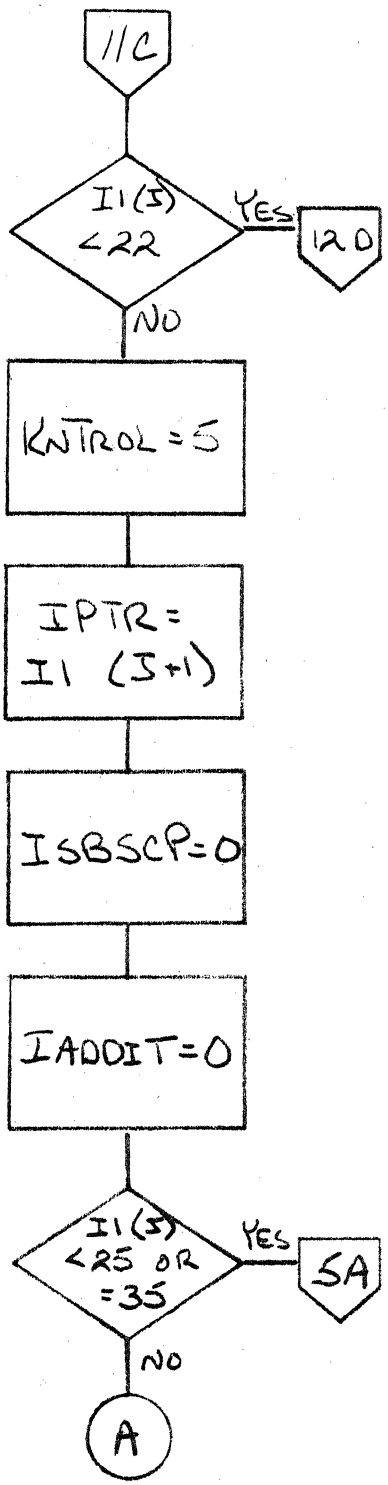
TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 10A OF 17

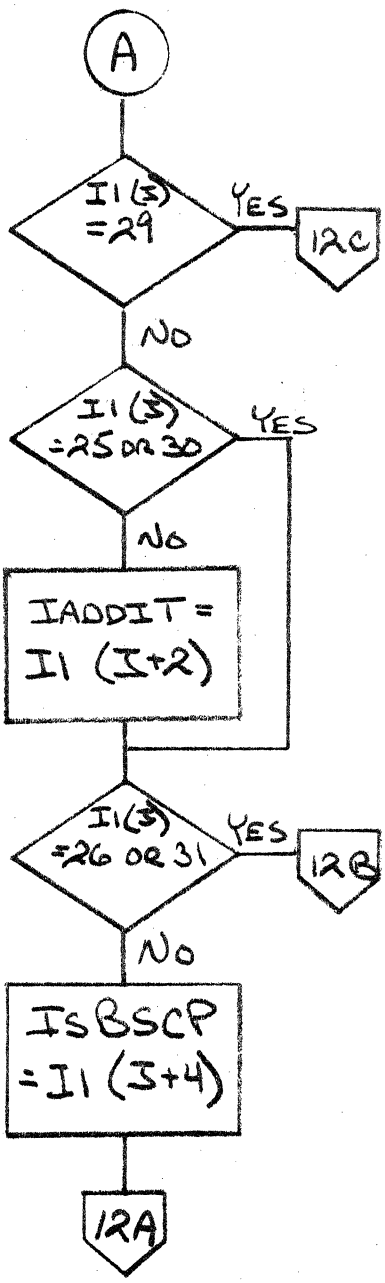
CSD 209



TITLE	ACP		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 11 OF 17

CSD 203



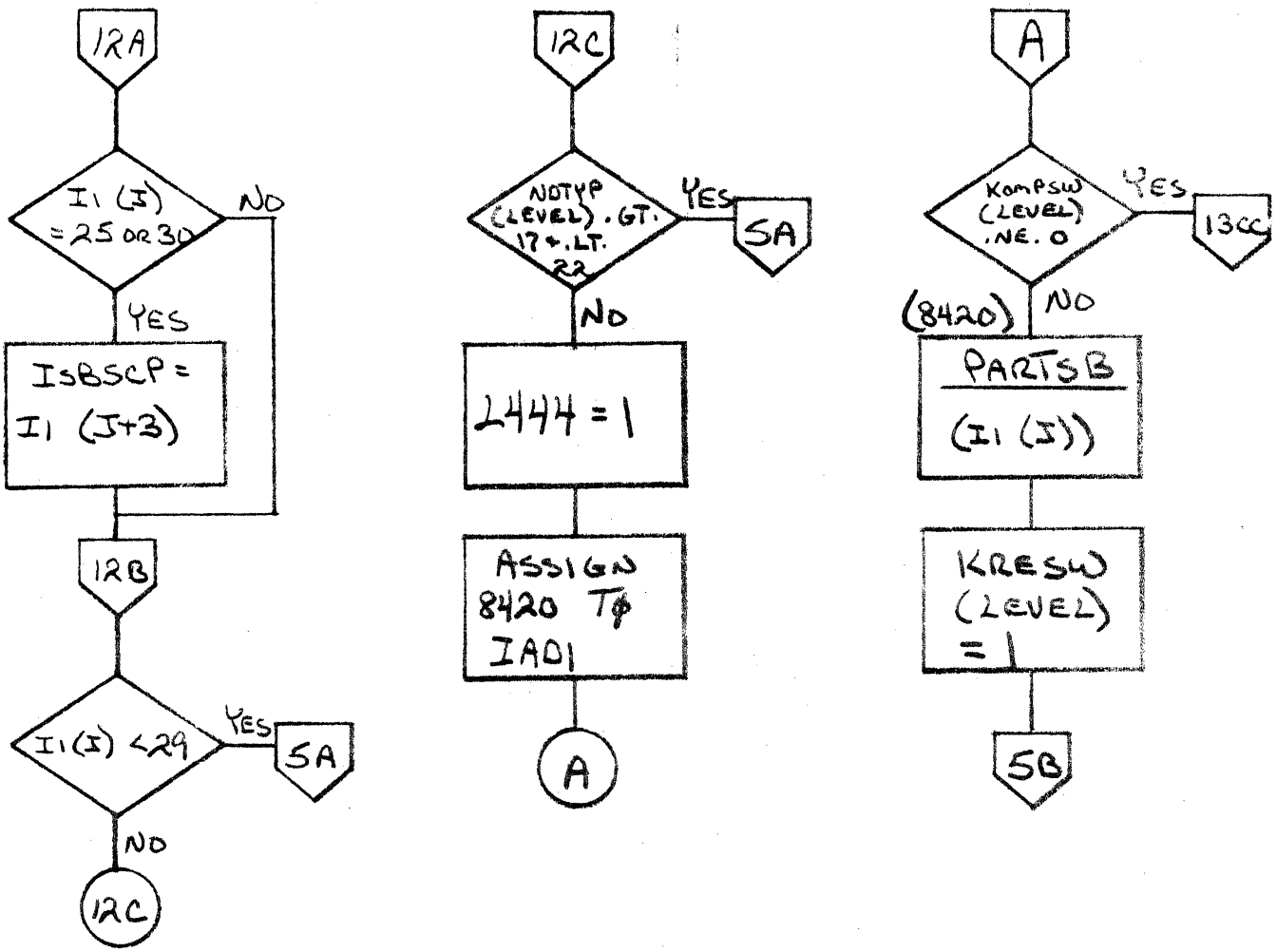


TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 11A OF 17

CSD 209

CONTROL DATA

LA JOLLA FACILITY

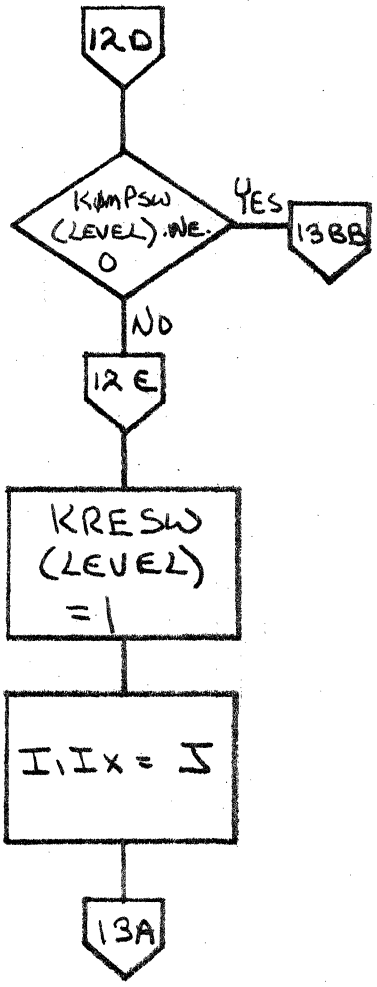


TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 12 OF 17

CSD 208

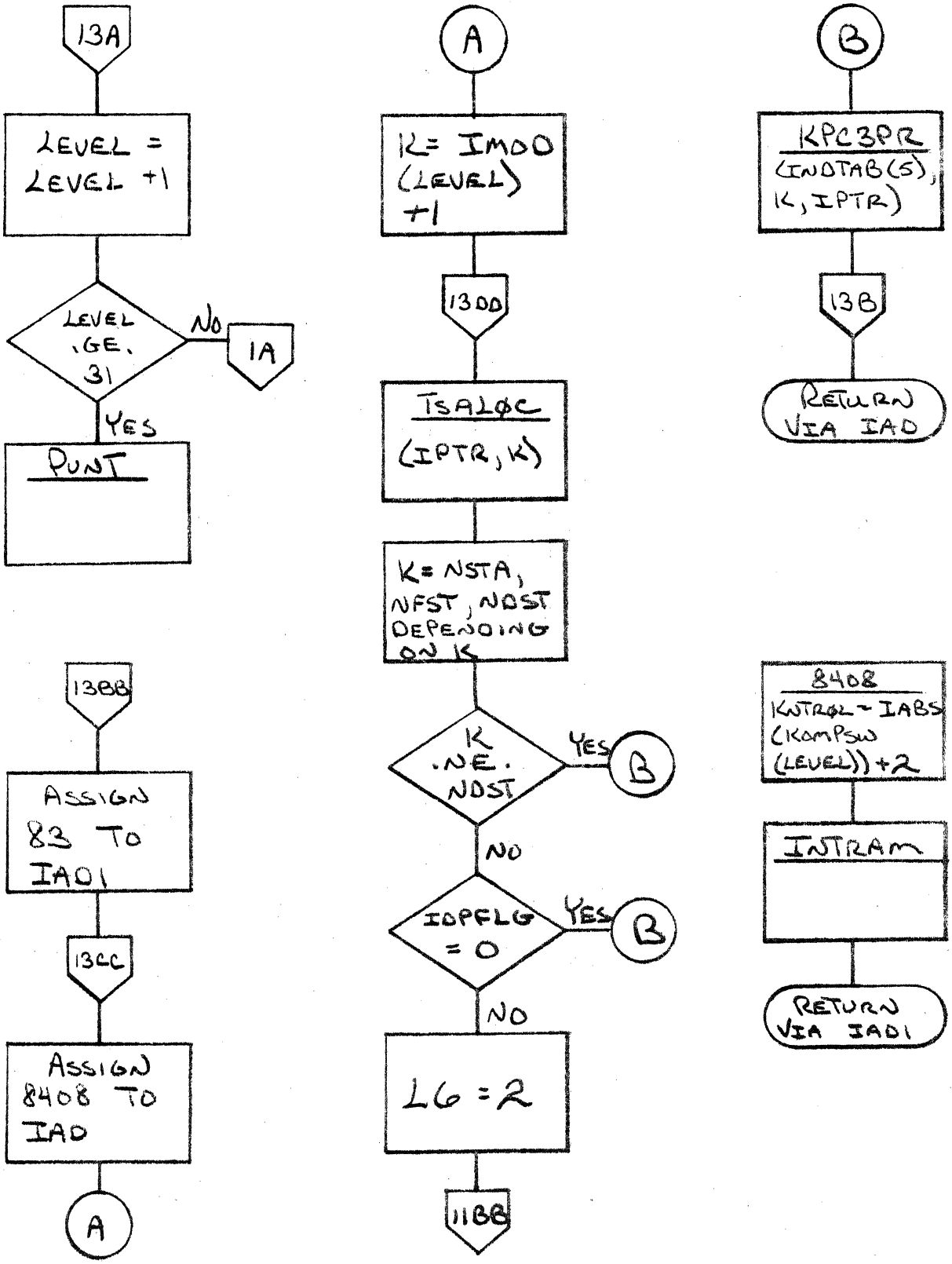
CONTROL DATA

LA JOLLA FACILITY



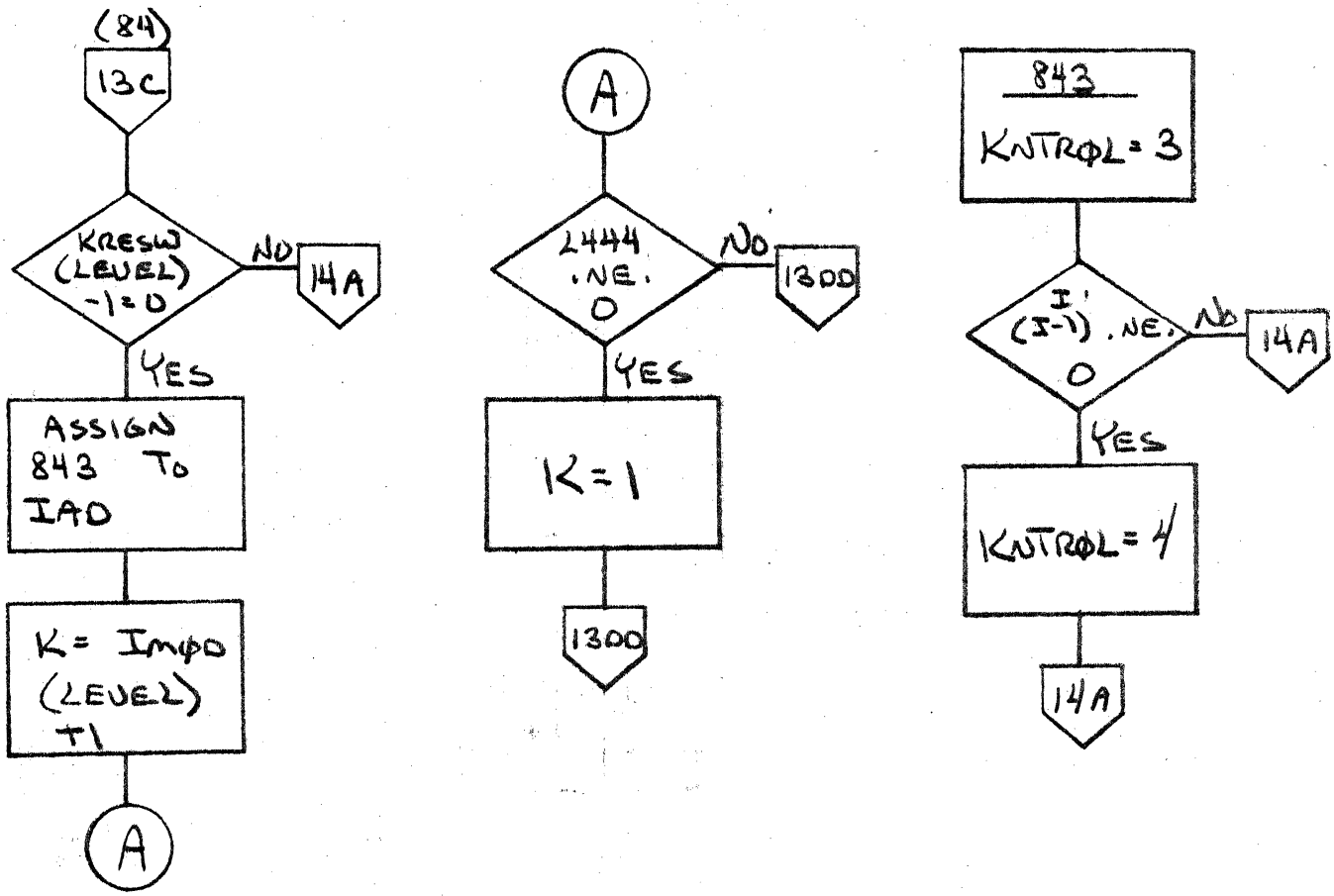
TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 12A OF 17

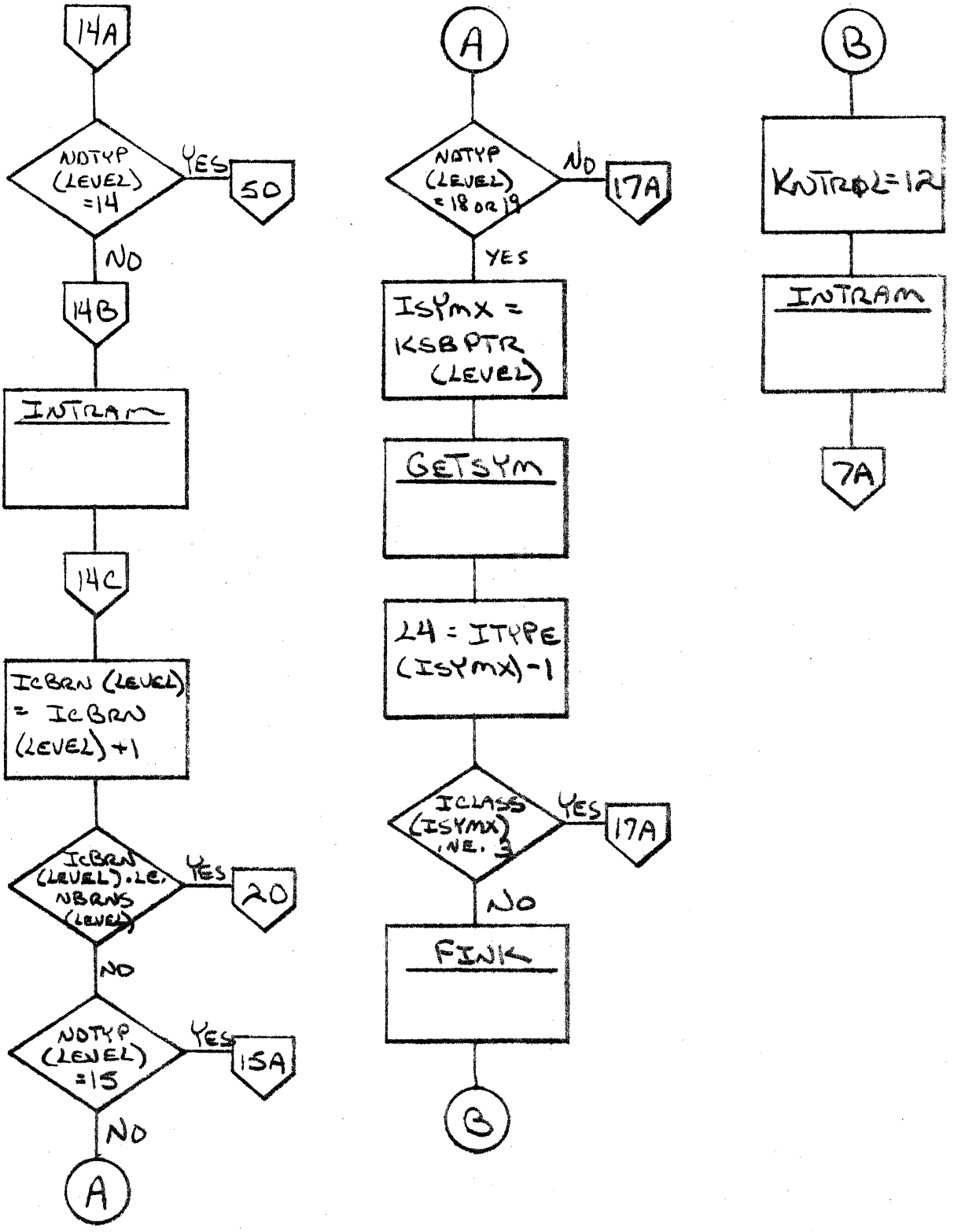
CSD 203



TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 13 OF 17

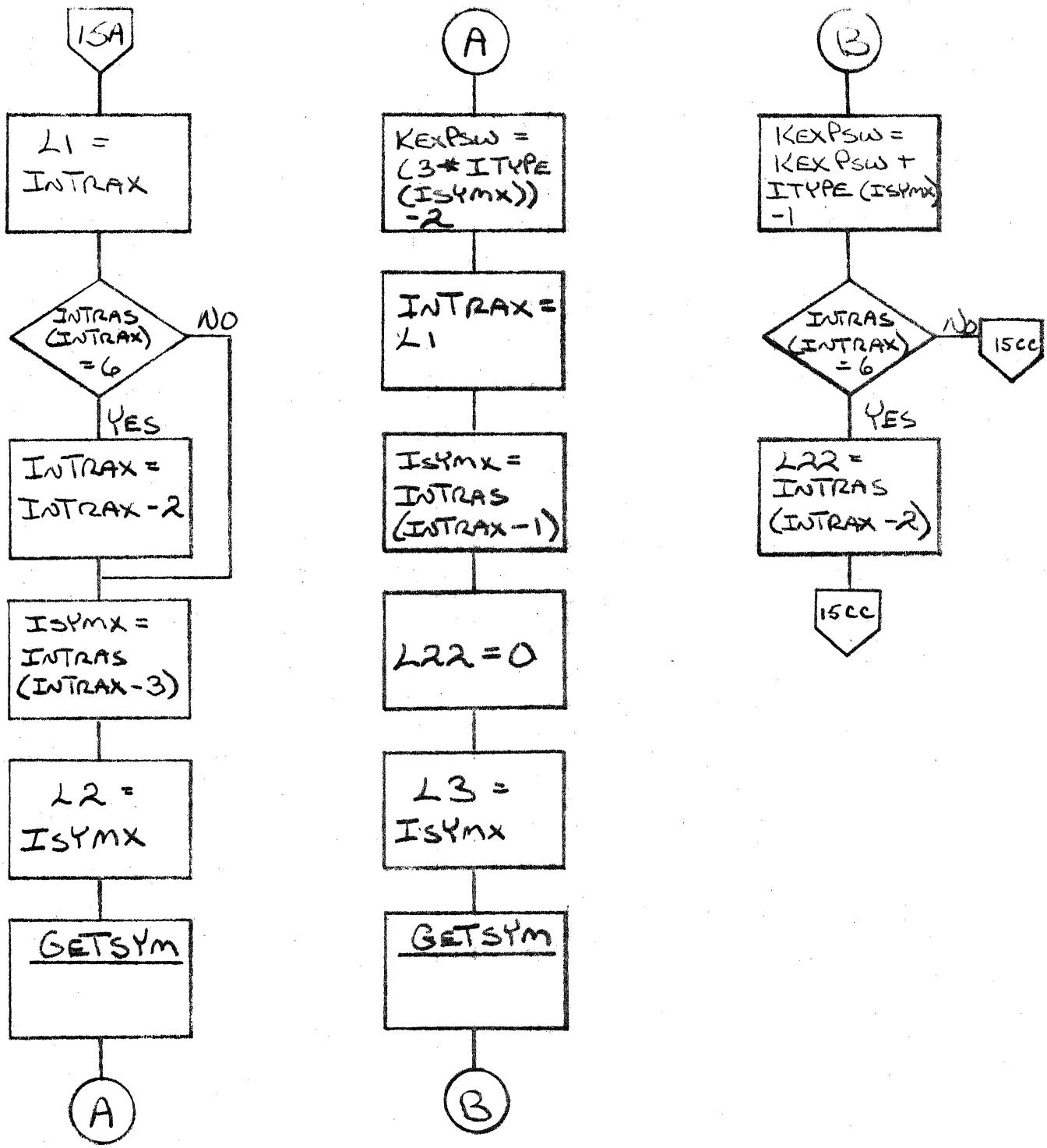
CSD 203


(84)
13C
KRESW (LEVEL) -1=0
NO
14A
YES
ASSIGN 843 TO IAD
K= IMPO (LEVEL) +1
A
A
2444 .NE. 0
NO
1300
YES
K=1
1300
843
KNTRØL=3
I' (I-1) .NE. 0
No
14A
YES
KNTRØL=4
14A



TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 14 OF 17

CSD 203

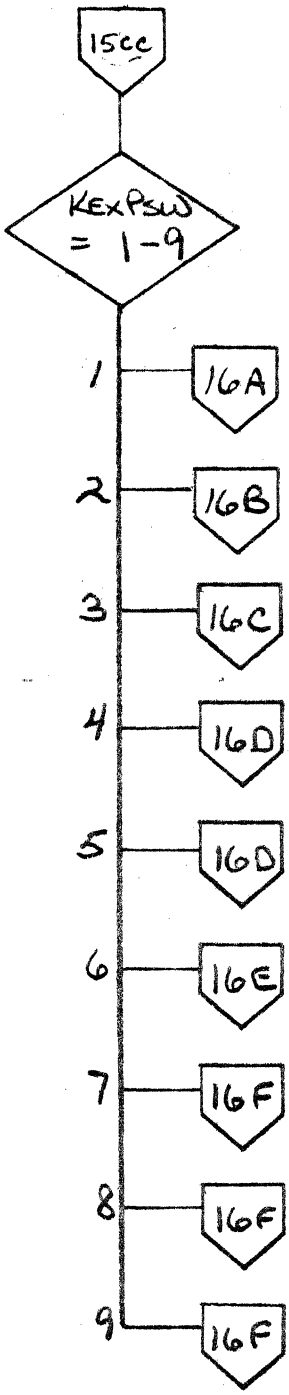


TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 15 OF 17

CSD 203

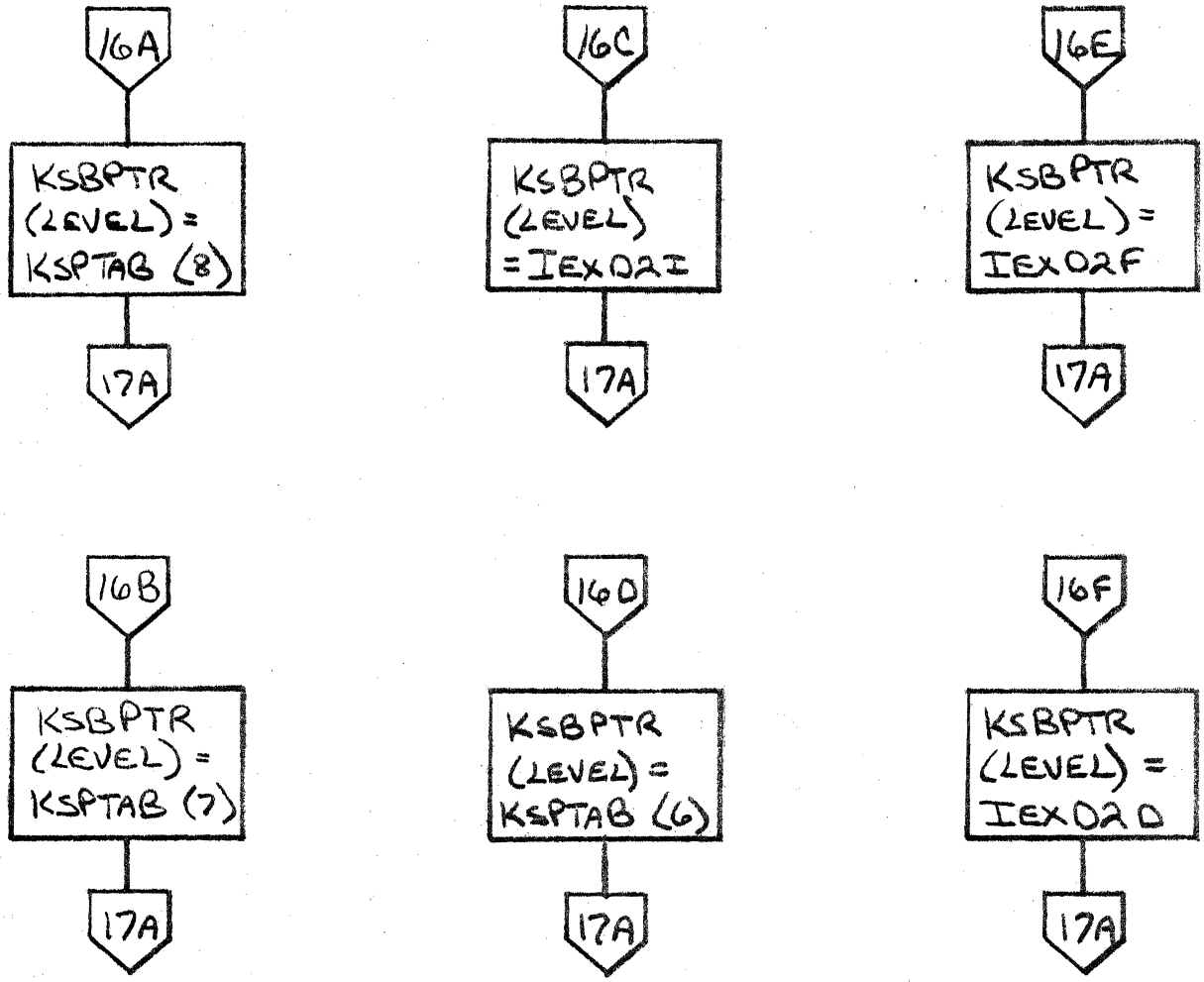
CONTROL DATA

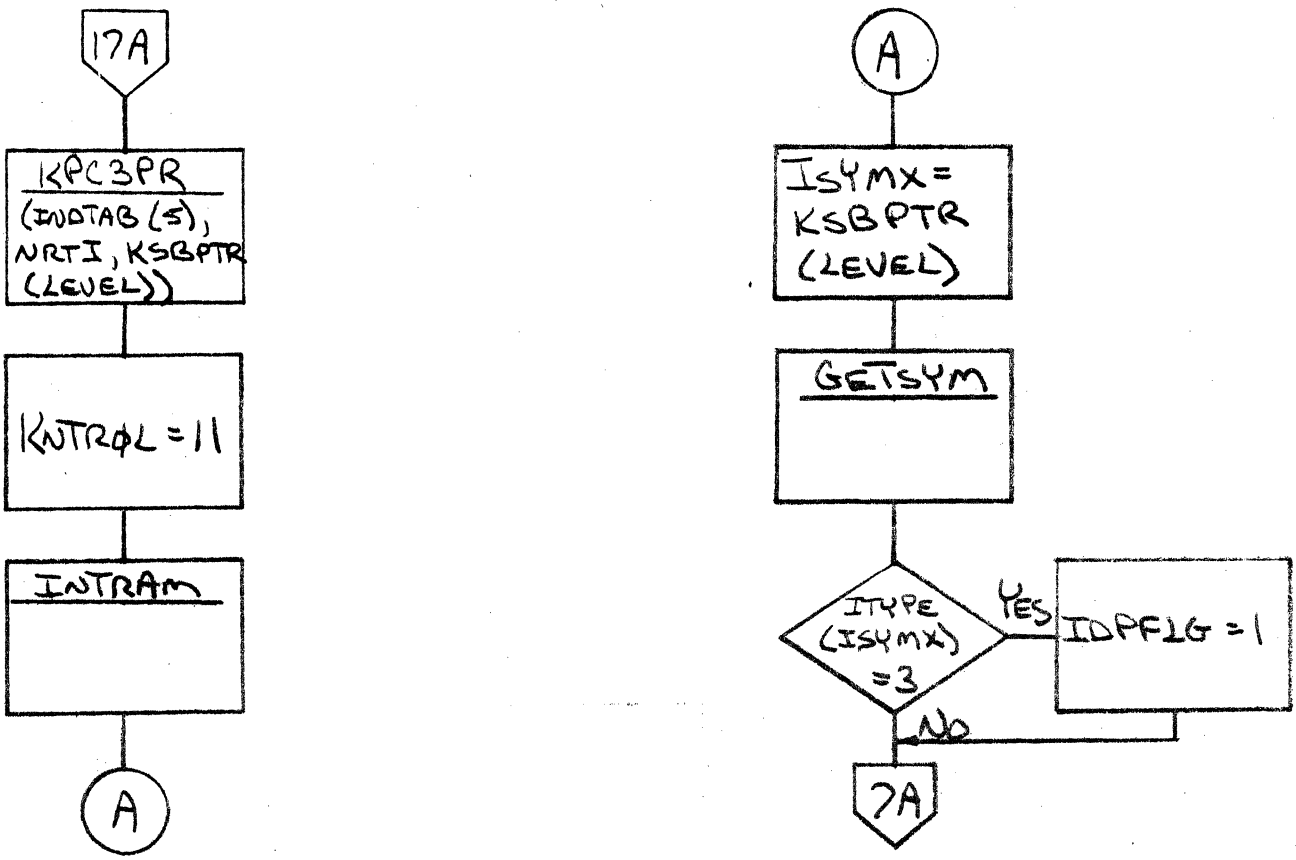
LA JOLLA FACILITY



TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 15A OF 17

CSD 203


16A
KSBPTR (LEVEL) = KSPTAB (8)
17A
16C
KSBPTR (LEVEL) = IEX02I
17A
16E
KSBPTR (LEVEL) = IEX02F
17A
16B
KSBPTR (LEVEL) = KSPTAB (7)
17A
16D
KSBPTR (LEVEL) = KSPTAB (6)
17A
16F
KSBPTR (LEVEL) = IEX02D
17A

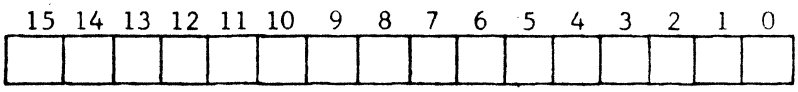


TITLE ACP			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 17 OF 17

CSD 203

4.4.4 PARTSB

PARTSB is called upon to generate instructions which load a byte or signed byte into the accumulator. Recalling the bit designations of the 1700 16-bit word:



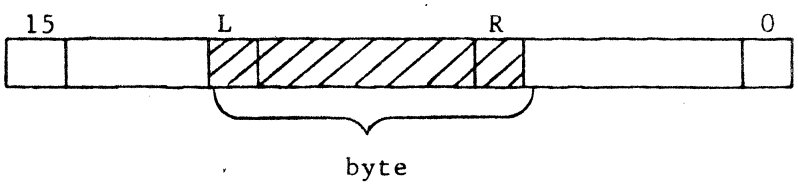
Bytes are delineated in the symbol table by

IPART(ISYMX) = 1, byte
= 2, signed byte

IPARTR(ISYMX)= the bit number of the right-most (least significant bit of the byte)

IPARTL(ISYMX)= the bit number of the leftmost (most significant bit of the byte)

In a given byte:



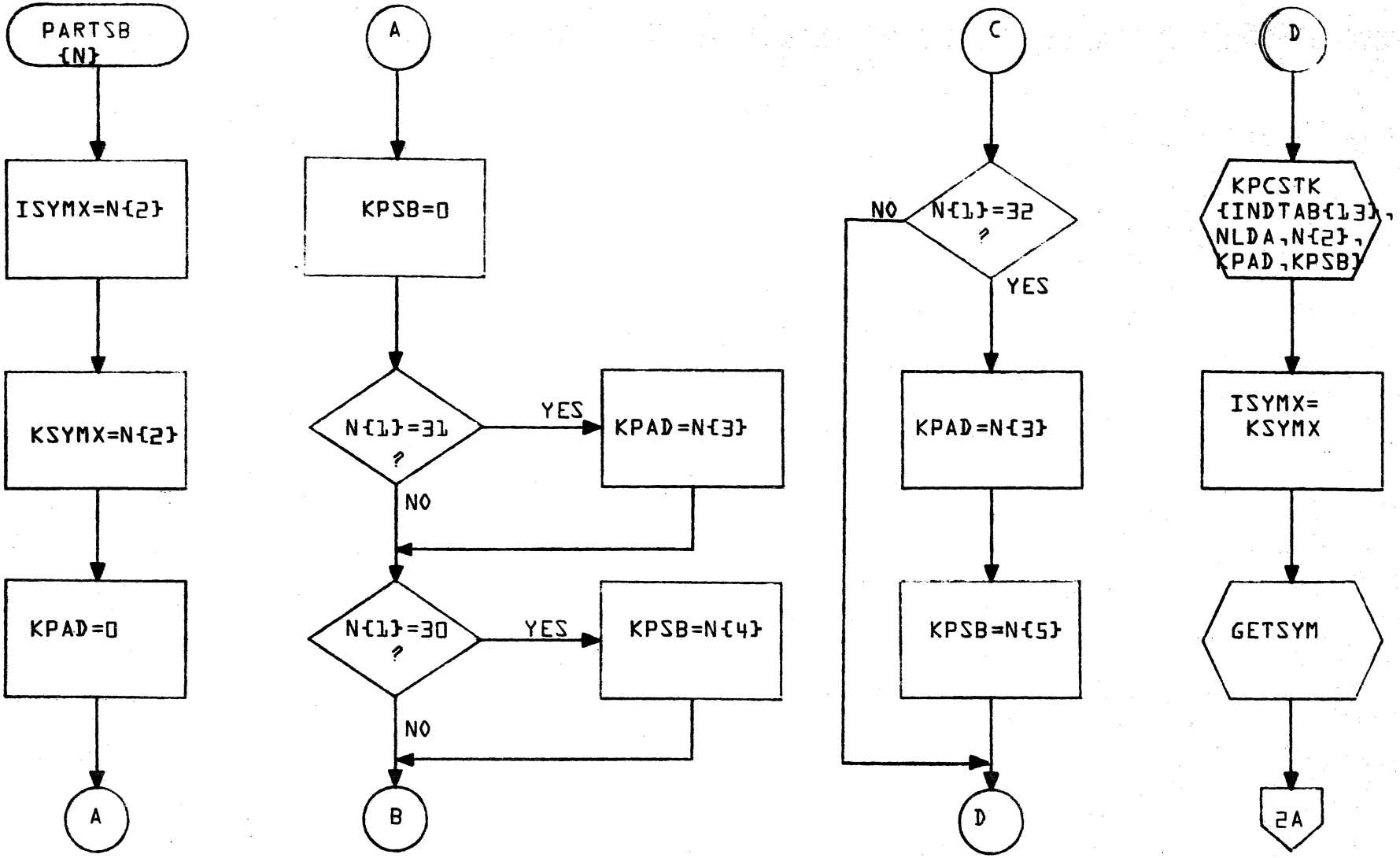
two basic instruction sequences are used:

1. Byte (unsigned - except for the nominal case, L = 15, R = 0)

	LDA	X	where X is the word containing the byte
	ARS	m	where m = IPARTR(ISYMX) (omitted when m = 0)
	AND	MASK	where the mask is a generated constant such that bits 15 through bit (L-R+1) are 0 and bits (L-R) through 0 are 1. The generated constants for the masks are entered in the symbol as they are required.

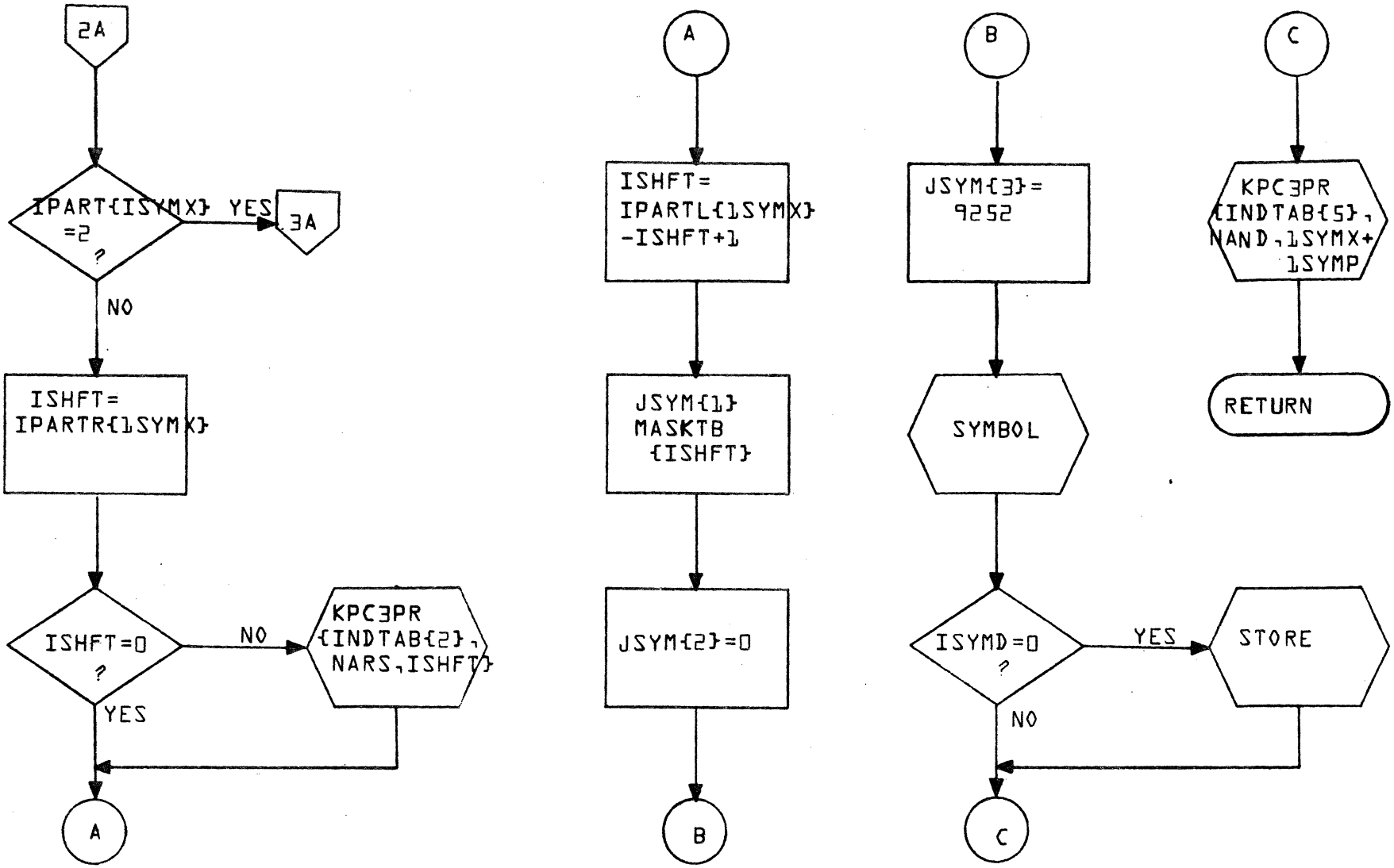
2. Signed Byte	LDA	X	where X is the word containing the byte
	ALS	m	where m = 15-IPARTL(ISYMX) (omitted when m = 0)
	ARS	n	where n = 15-IPARTL(ISYMX)+IPARTR(ISYMX)

Flowcharts:

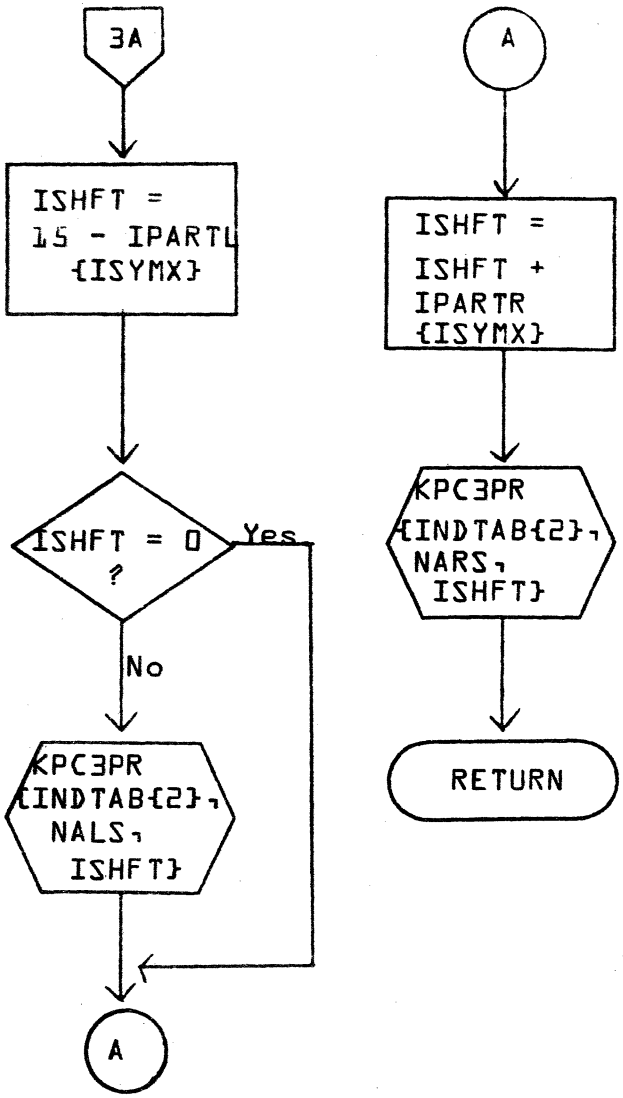


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PARTSB	PROJECT MGR.			
	PAGE 1 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

ERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PARTSB		PROJECT MGR.		
		PAGE 2 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PARTSB		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

4-150

4.4.15 READIR

READIR is the PHASE B subroutine for reading an input record. It is called by PHASE B.

Parameters:

I = the index to the input buffer (INBUFF(I)) designating the starting location to begin input.

J = output parameter contains index to the input buffer (INBUFF(J)) designating the beginning of non-header information of the record just read in.

Operation:

1. Call READ and input a record beginning at INBUFF(I) specifying 203-I words. Since the buffer is 202 words long all inputs are through the last word of the input buffer even though most records do not require that much space.

2. Compute J

    1. If there is notstatement label, J = I+5.

    2. If there is a statement label

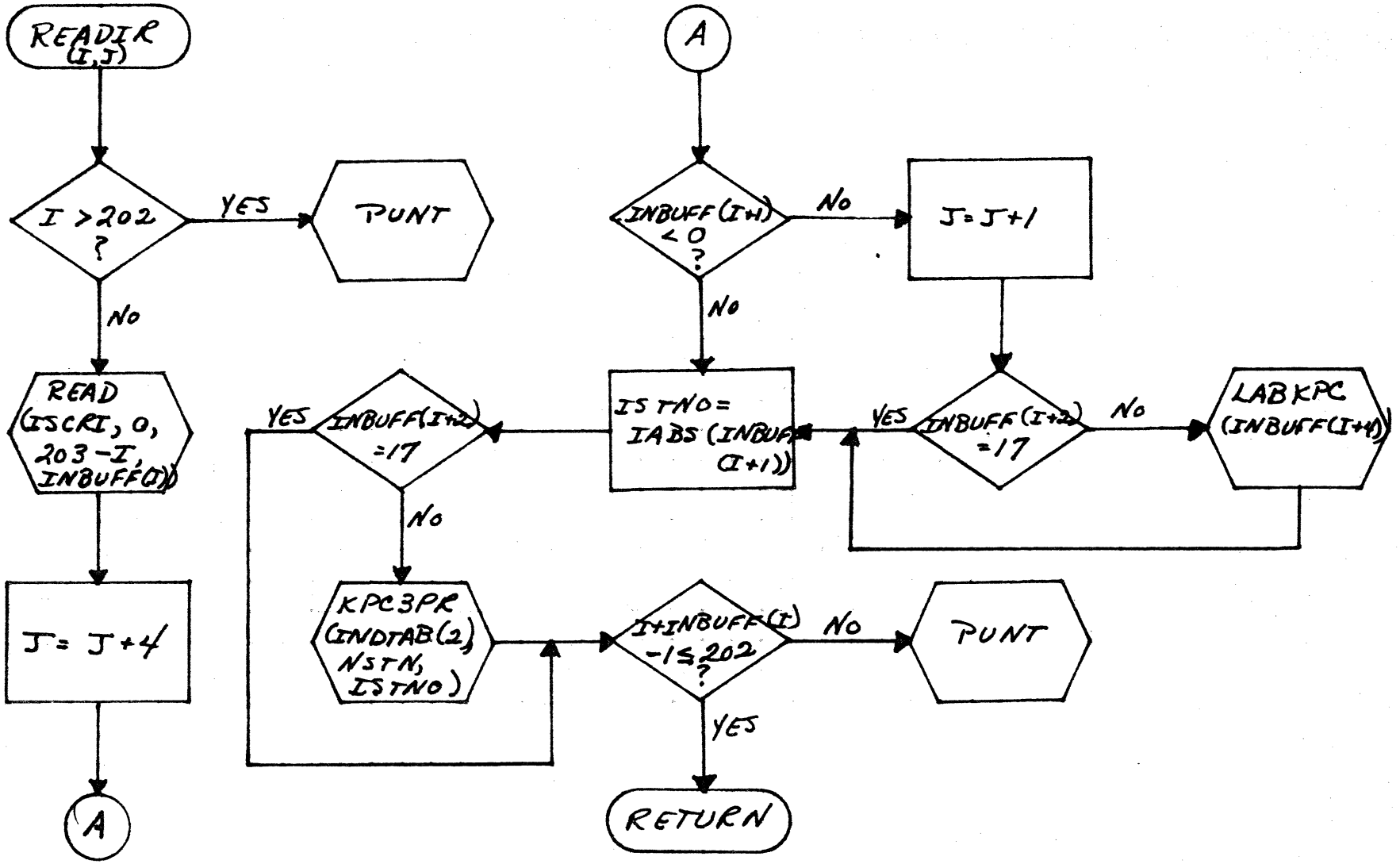
        a. J = I+6
        b. output the label.

    3. Update ISTNO to new statement number.

    4. Output STN pseudo instruction (gives PHASE C indication that a new statement is beginning and what the statement number is.

    5. Check to insure that the statement just read was not too large for the available input buffer. If so call PUNT.

Flowcharts:



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
FLOWCHART ☒	DOCUMENT TITLE READIR		PROJECT MGR.			
DECISION TABLE ☐		PAGE 1 OF 1	PROJECT NAME			
OTHER ☐	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

4.4.16 SUBPR1 and SUBPR2 subscript processors.

SUBPR1 and SUBPR2 are called upon by ASUPER to process subscripts. (See discussion of subscript processing in 4.4.2, ASUPER).

Both programs check for operand types 26 (increment only subscripted variable) and 31 (increment only subscripted partial variable). No processing is performed on these two types. What then remains are types 25, 27, 28, 30, 32, and 33, variable only, variable and increment, and complex subscripted partial/non partial variables.

Parameters:

I - input parameter to both SUBPR1 and SUBPR2 represents address of array which begins at the operand "type" code of a "tree" operand. I(1) = type code.

4.4.16.1 SUBPR1

SUBPR1 processes subscripts which subscript variables that are not part of calling sequences.

The subscript expression is then passed on to SUBPR3 which generates the instruction, which evaluate the subscript and leave the result in the accumulator.

SUBPR3 sets its parameter

I1 = 0 if no actual evaluation was required. (i.e. operand was type 25, 27, 30, or 32 and the variable in the subscript expression was not a partial.

I1 = 1, or 2 if instructions for evaluation were actually required. (i.e., operand type was 28 or 33 (complex) or the variable in the subscript expression was a partial).

SUBPR1 then references the returned parameter I1. If I1 = 0, processing is complete. If I1 $\neq$ 0, then:

1. A temporary storage location is allocated.
2. A store instruction is generated to store the result of the subscript evaluation.
3. The variable type in the subscript expression is set to 24 (non-subscripted, non-partial variable).
4. The temporary storage cell is added to INTRAS through a call to INTRAM (at this point INTRAS is at level 0, the level which contains subscript assignments. See discussion in 4.4.7, INTRAM)

4.4.16.2 SUBPR2

SUBPR2 is the analogous processor to SUBPR1 but processes subscripts which subscript variables that are part of calling sequences.

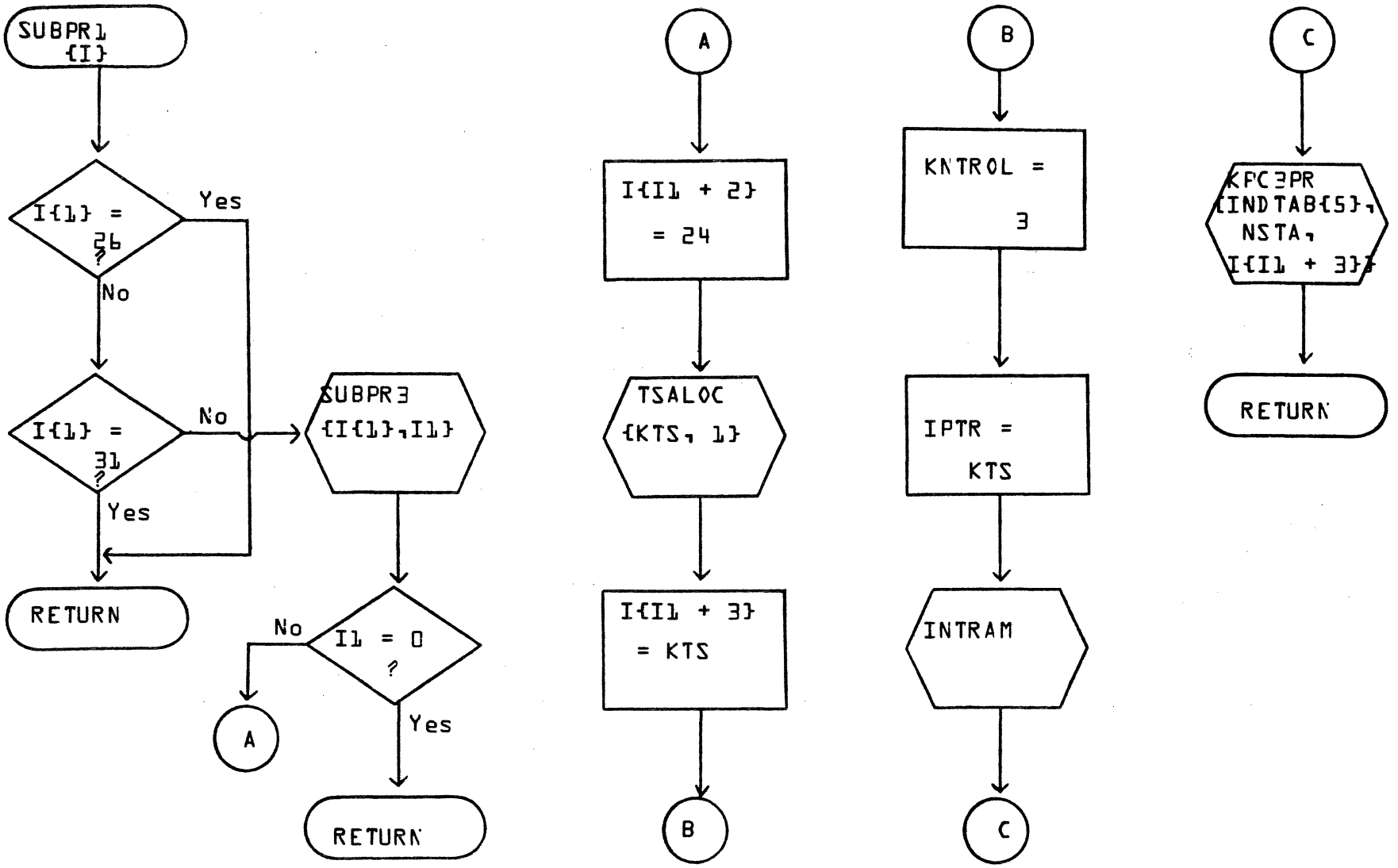
As in SUBPR1, the subscript expression is passed on to SUBPR3 which generates the instruction which evaluates the subscript and leaves the result in the accumulator.

Then I1 is tested as in SUBPR2.

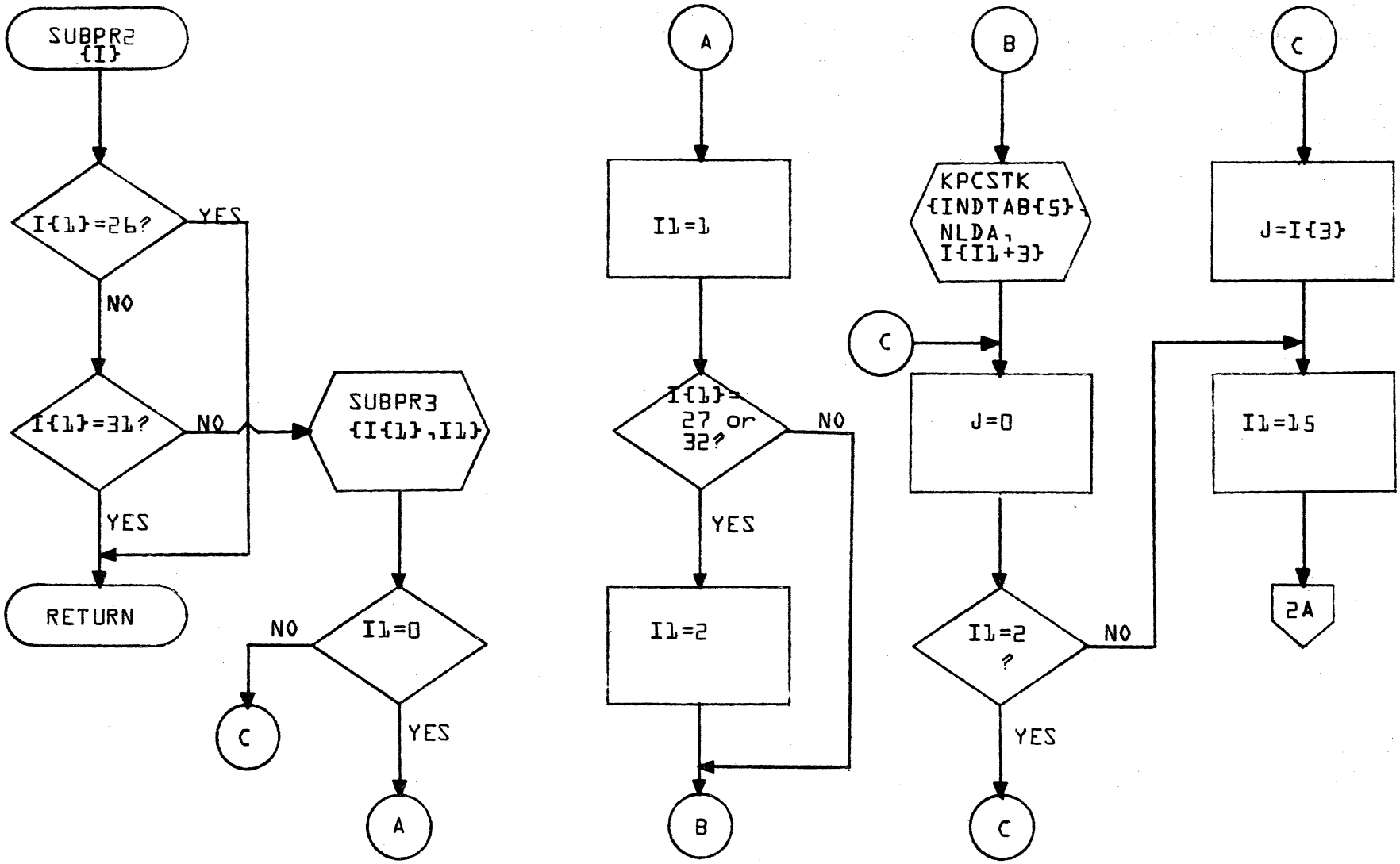
If I1 = 0 no actual evaluation was required but the value of the subscript must still be in the A register so an LDA command is generated to load the subscript variable.

Then an ADD instruction is generated to add the address of the subscripted variable to the value of the subscript. (In the run-anywhere case the relocation constant is also added).

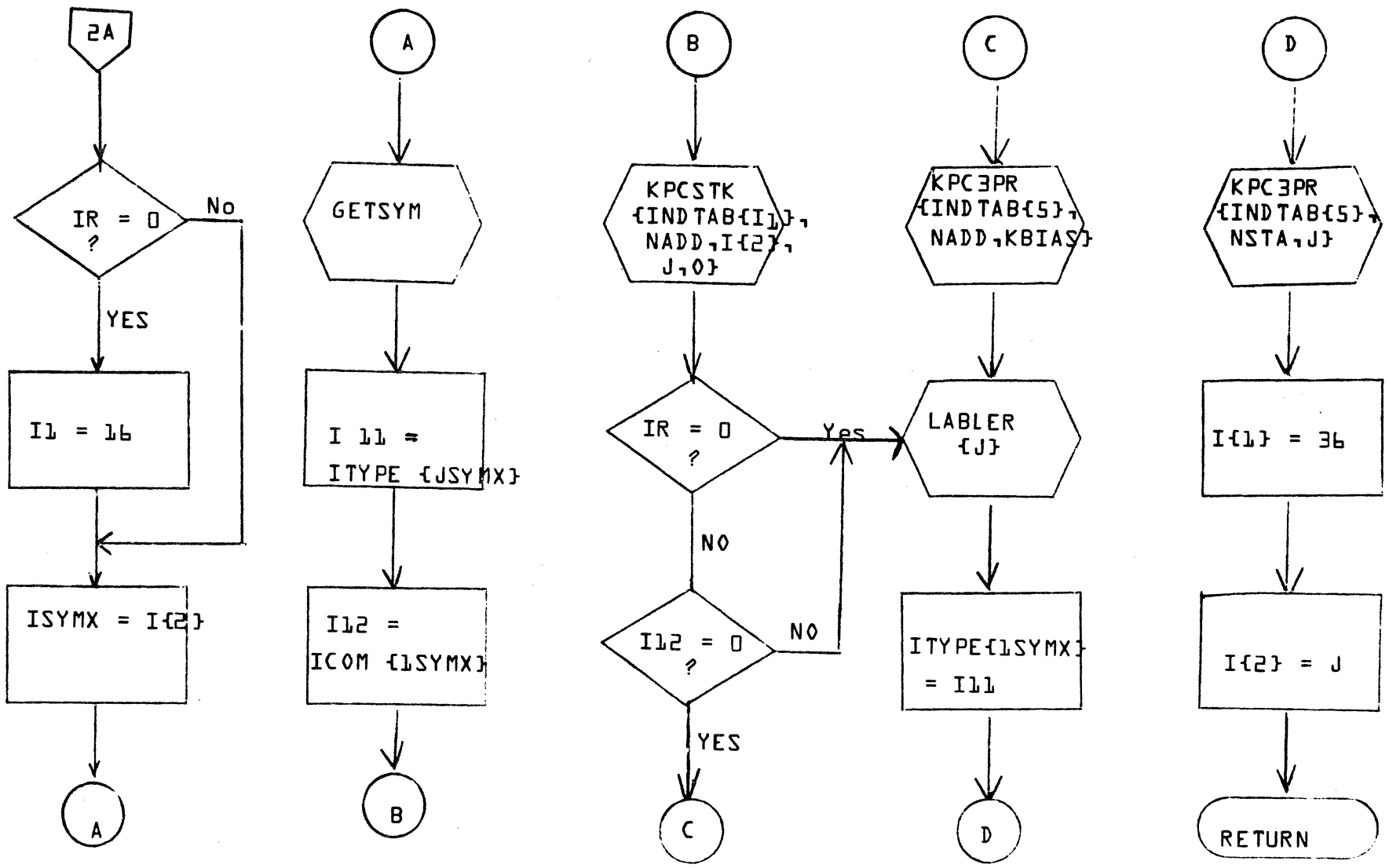
Finally, a label is created and a store instruction referencing that label is generated. The subscripted variable operand is then modified to become a "calling sequence label element" (type 36).



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SUBPR1		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBPR2		PROJECT MGR.			
		PAGE 1 OF 2	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



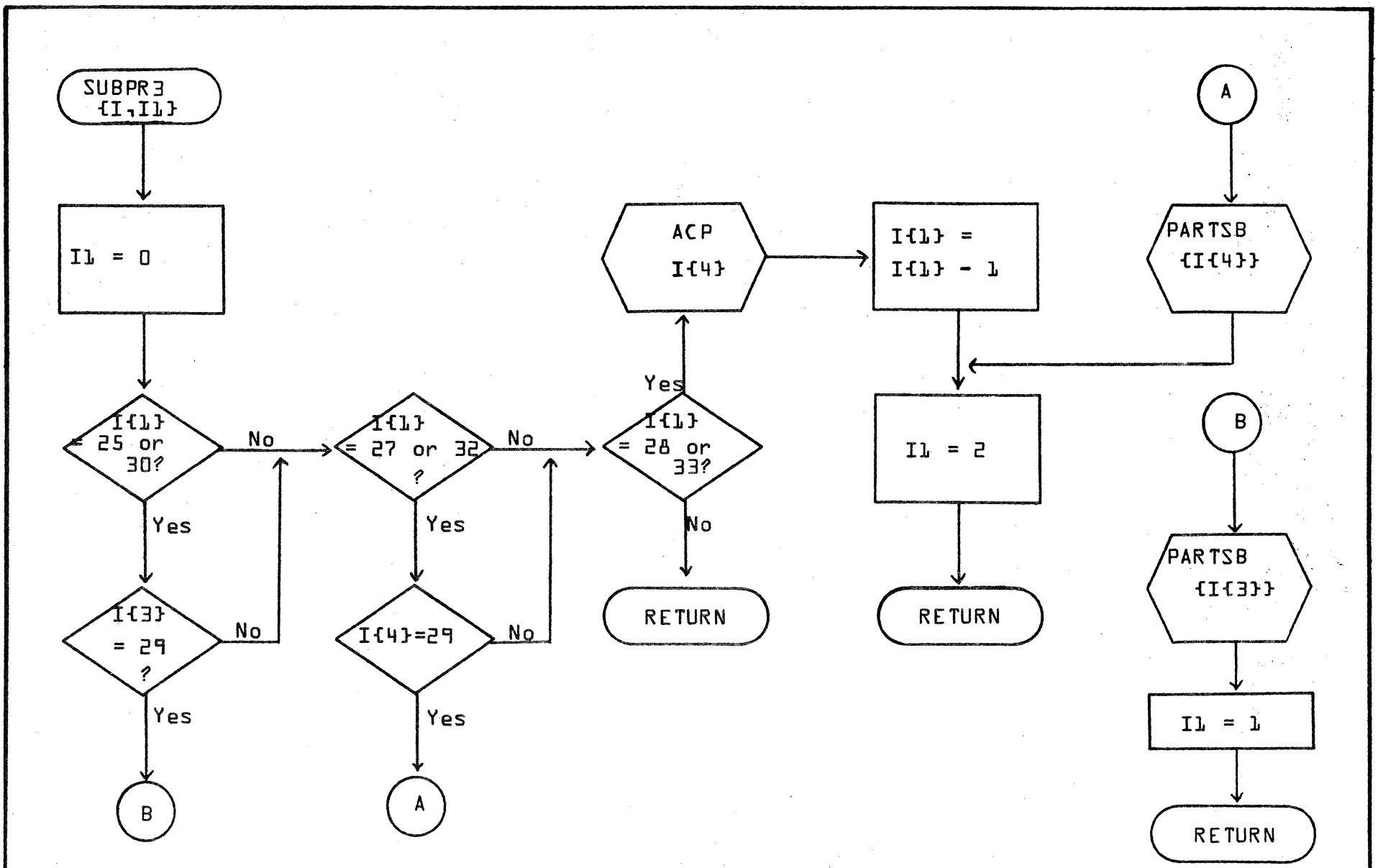
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBPR2	PROJECT MGR.			
	PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			

4.4.17 SUBPR3

SUBPR3 is called by SUBPR1 and SUBPR2 to generate the instructions for evaluating a subscript. Code generation is performed in three cases and the output parameter I1 is set accordingly:

1. Partial variable only subscript, I1 = 1.
2. Partial variable with increment subscript, I1 = 2.
3. Complex subscript I1 = 2.

When no instructions are generated, I1 = 0.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SUBPR3		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

4.4.18 TSALOC

TSALOC controls temporary storage allocation by updating and monitoring KTCAT, temporary cell allocation table.

Table structure is shown:

KTCAT (20, 2)

PTR	A

where each row is a two word entry and:

PTR = the symbol table entry

A = available indicator:

0 = integer temporary storage (unavailable)
1 = real temporary storage (unavailable)
2 = double precision temporary storage (unavailable)
3 = integer temporary storage (available)
4 = real temporary storage (available)
5 = double precision temporary storage (available)

Parameters

KCELL input or output parameter containing symbol table pointer to temporary storage cell.

N request parameter =

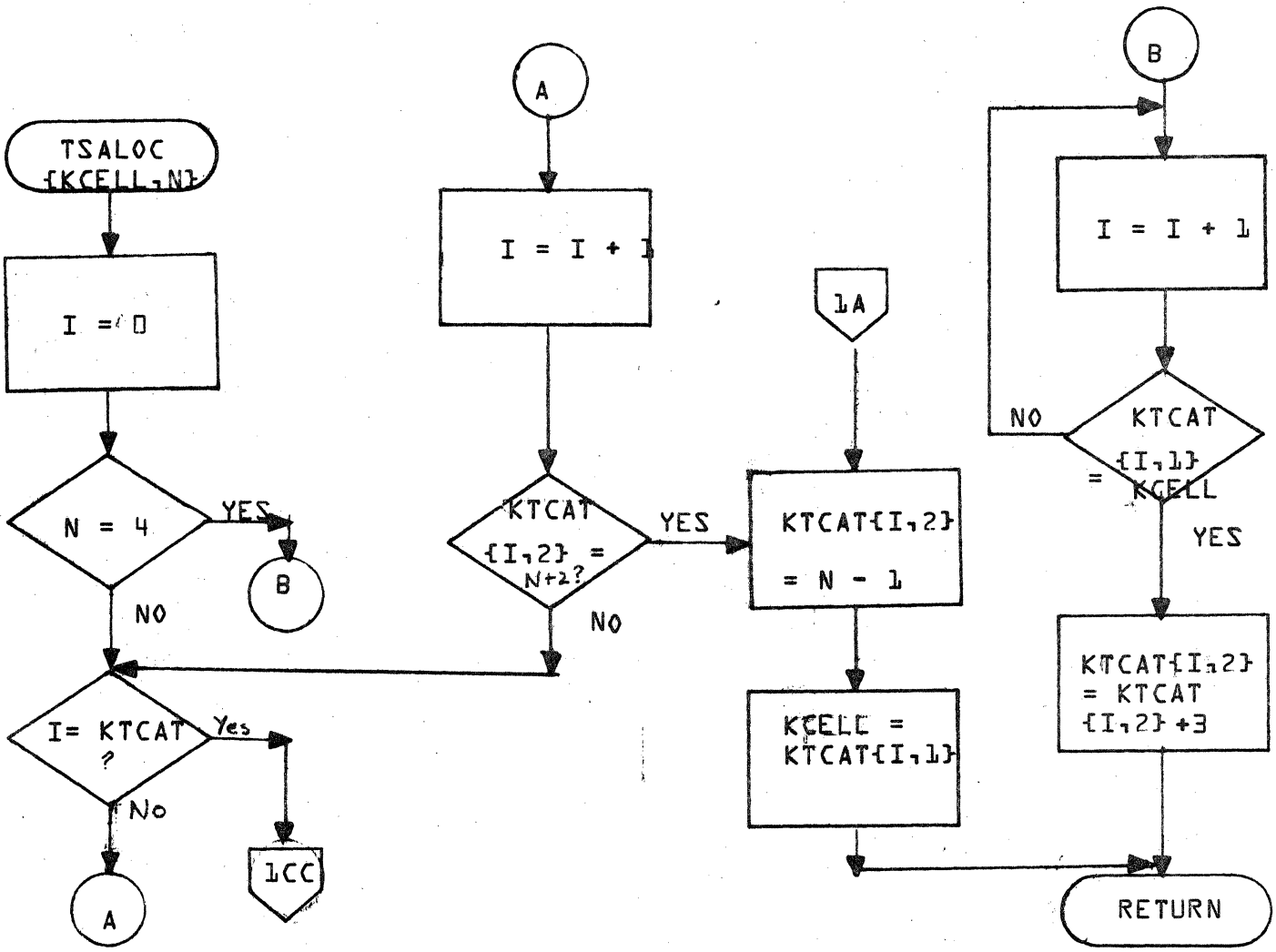
1 integer temporary storage requested. TSALOC searches KTCAT for an available integer temporary storage {A=3}. If one is found, the pointer is passed back in KCELL and availability indicator is set to 0. If none is found a new one is created and KTCAT is updated.

2 real temporary storage requested. TSALOC searches KTCAT for an available real temporary storage {A=4}. If one is found, the pointer is passed back in KCELL and availability indicator is set to 1. If none is found, a new one is created and KTCAT is updated.

3 double precision temporary storage requested. TSALOC searches KTCAT for an available double precision temporary storage {A=5}. If one is found, the pointer is passed back in KCELL and availability indicator is set to 2. If none is found, a new one is created and KTCAT is updated.

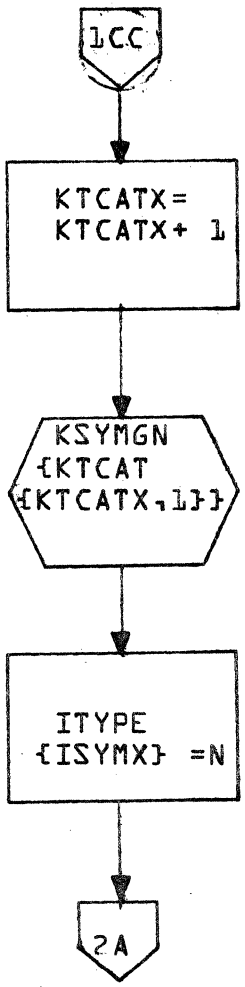
4 release request. KCELL contains pointer of temporary storage cell to be released. It is looked up in KTCAT and availability indicator is set to available:

0 → 3
1 → 4
2 → 5



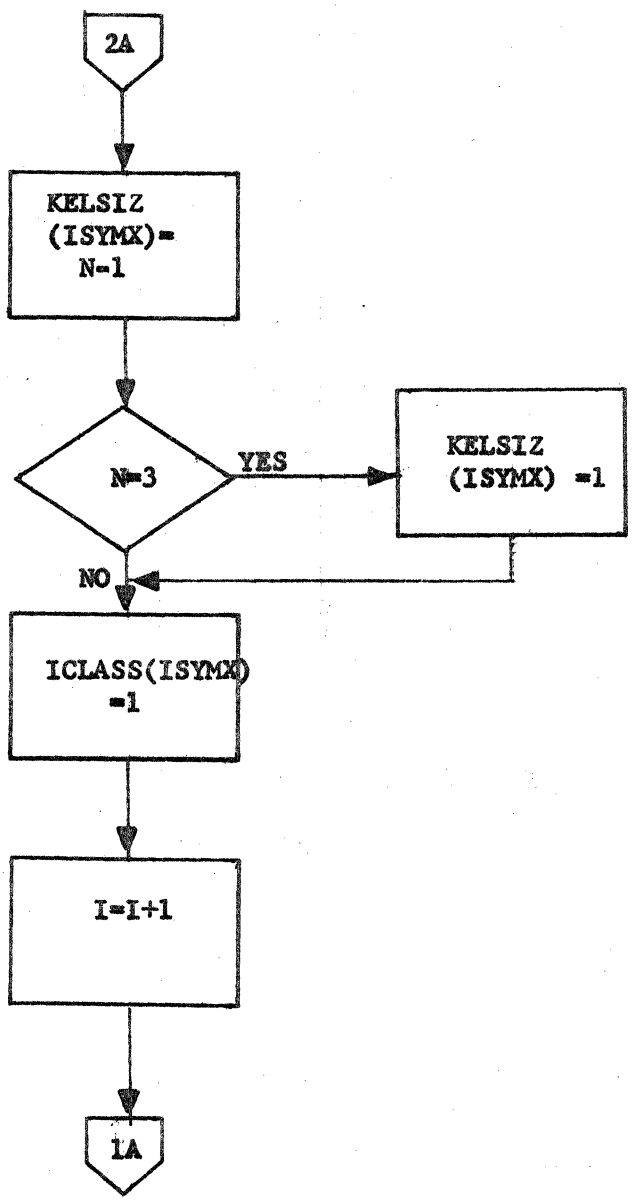
TITLE	TSALOC		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 1 OF 2

CSD 203



TITLE	TSALOC		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 1A OF 2

CSD 203



TITLE	TSALOC		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2 OF 2

CSD 203 ANALYSIS

4.4.19 HELEN

I. General

Locations of several types of data and storage cells are not established implicitly in the source program. They include locations of:

1. Variable Arrays
2. Variables
3. Constants
4. Format information
5. Temporary cells assigned by Phase B to contain intermediate computations.

The emphasis of 1700 FORTRAN on conserving core space and optimizing relative addressing, necessitates location of this data in relation to the rest of the program such that space and speed are minimized.

II. Implementation

All programmer defined arrays, variables, constants, etc., except format specifications will be output at the top of each program by Phase B Under certain conditions Phase C will take them from their place at the top of the program and relocate them within the program body.

A. Format Information

See NOPROC

B. Arrays

1. Though placement of an array within the program body may shorten references to it, far more references may be destroyed by so doing.
2. Arrays are most often referred to in loops. Though there may be a great deal more references made to array elements during the logical sequence of a program's operation there will be far fewer physical references; that is, less space will be devoted to array references than to single variable references.
3. Considering the non-subscripted references to arrays, it is likely that because of their size at least some of the elements would not be accessable through 1-word commands regardless of the placement of the array.

4. Equivalence considerations may make optimum assignment of array positions a highly difficult analysis. Therefore, arrays will be located at the beginning of the program.

C. Constants, Variables, Temporary Cells

1. Phase B

Phase B will set a flag for a reference to any location which is a candidate for becoming a 1-word command. The flag KRFCNT, will be kept in a portion of each symbol table entry. Before execution of Phase C, those counts will be set to 0 which refer to any types of data which have already been located by Phase B {only special cases are located by Phase B}.

2. Phase C

Phase C will take all those with the flag set and relocate them in the body of the program such that they immediately follow the first unconditional jump after the first reference which defines the location {i.e., STA, etc.}. Phase C will also revise references in the following manner:

a. The first reference to a cell which falls outside of a 127 cell distance between the cell and its reference is made a two-word command with a relocatable address operand.
b. Every subsequent reference which falls outside of the mentioned range but within range of the relocatable address operand will be made a relative-indirect reference to that operand.
c. When a subsequent reference is found which is outside of both ranges, the process begins over again as in 1.
d. Variables and constants used only in Calling Sequences or others whose references are never candidates for 1-word commands end up with KRFCNT = 0.

D. Temporary Cells

1. Both one-word, two-word and three-word temporary cells will be generated by Phase B to contain integer, real and double precision intermediate results respectively. Temporary storages are the result of three major functions of Phase B.

a. Storage of intermediate computations in arithmetic expressions. Such storage is reusable from one expression to the next and is never needed between expressions.
b. Special temporary storages for such things as saving indexes, is in use from beginning to end and never usable for anything else. At any rate, the locations for these types of storage are decided upon by Phase B.

2. Reuse

Most temporary cells will be used for a brief time during computations and then become unused. It is therefore advantageous to reuse temporary cells as much as possible rather than create new ones.

Temporary storage which is in use or available for use will be kept in a table KTCAT, which will cross-reference their names (symbol table pointers) with information about their type and use. The table will be regularly updated in the following manner:

a. As temporary cells are needed, the table will be searched for an unused temporary cell of the same type (i.e., integer, real or double precision). If one is found, it will be noted in use. If one is not found, a new one will be generated and noted in use.
b. As temporary cells terminate use, the table will be updated to show that that cell is no longer in use. (See TSALOC 4.4.18, for further discussion.)

3. Placement

a. Types

1. Two-word Temporary Storage

It is likely that in most cases all references to two-word temporary storage will occur only in single precision floating point calling sequences. If so, it will be advantageous (as in FORMAT and in most arrays) to place these cells outside of the program body.

2. One-word Temporary Cells

   It is likely that in most cases all references to one-word temporary storage will occur in instructions which are candidates for one word command and proper placement may improve efficiency to a very great extent.

3. Three-word Temporary Storage

   {Same as 3.a.1 on page 4-166}

b. Overlapping

It is conceivable that when a real {two-word} temporary storage becomes free for reuse, that it could be reused for two one-word temporary cells.

Considerations:

1. The ranges of two one-word temporary cells would have to very closely coincide such that they could be made consecutive cells. Such an analysis as this is not practical in Utility FORTRAN;

at any rate such a situation is rare.

2. If it were done, the method of placement of one-word cells (as the second word of constant commands) would be precluded and placement would become less efficient.
3. A typical program will not have 1 and 2-word temporary storages in equal quantities but rather will have mostly one or the other.

Conclusion:

Very little is gained by overlapping. FORTRAN does not attempt to overlap re-use of one-word and two-word temporary storage.

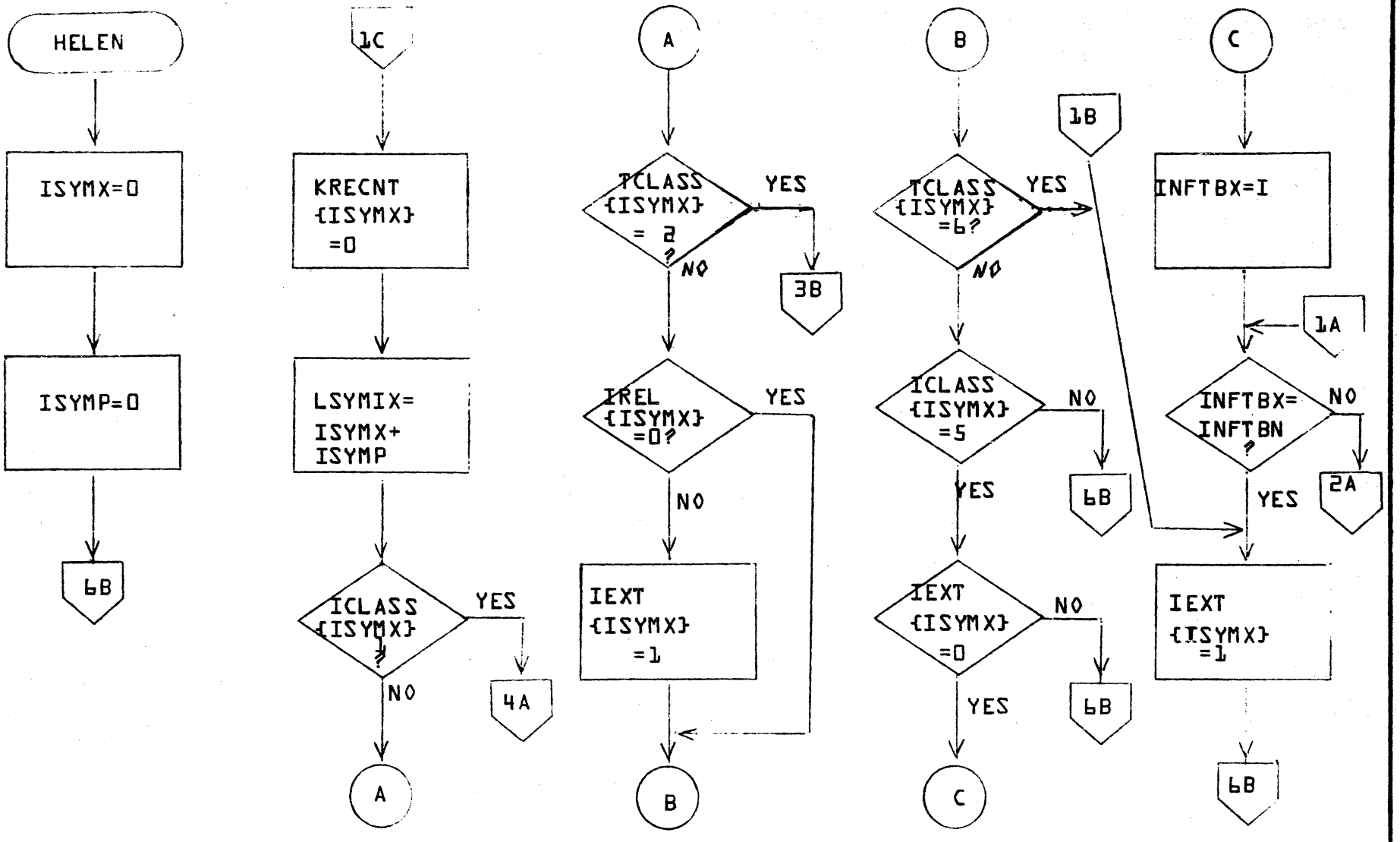
III. Responsibilities of HELEN

The responsibilities of subprogram HELEN, then, are as follows:

A. To output BSS commands of arrays, variables, including equivalence considerations, and CON commands of constants.

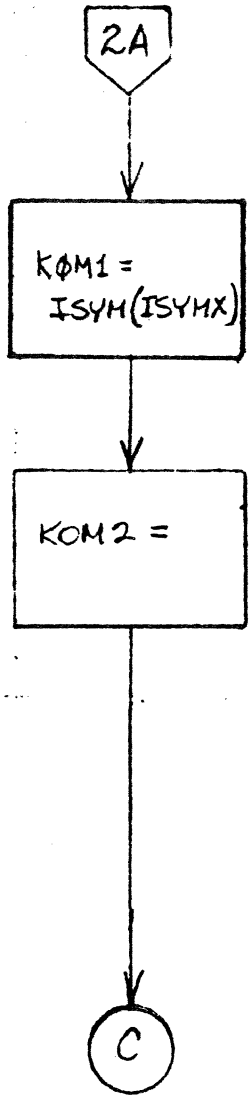
B. To process functions.

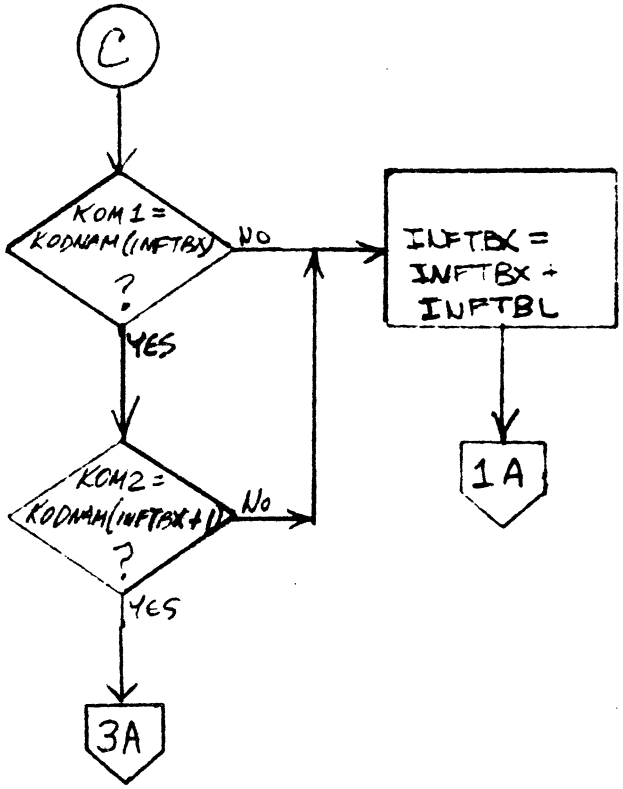
1. Replace coded names of standard library functions with their library name.
2. Set up index for in-line functions for use by FINK (see 4.4.6) in generating in-line function instructions.



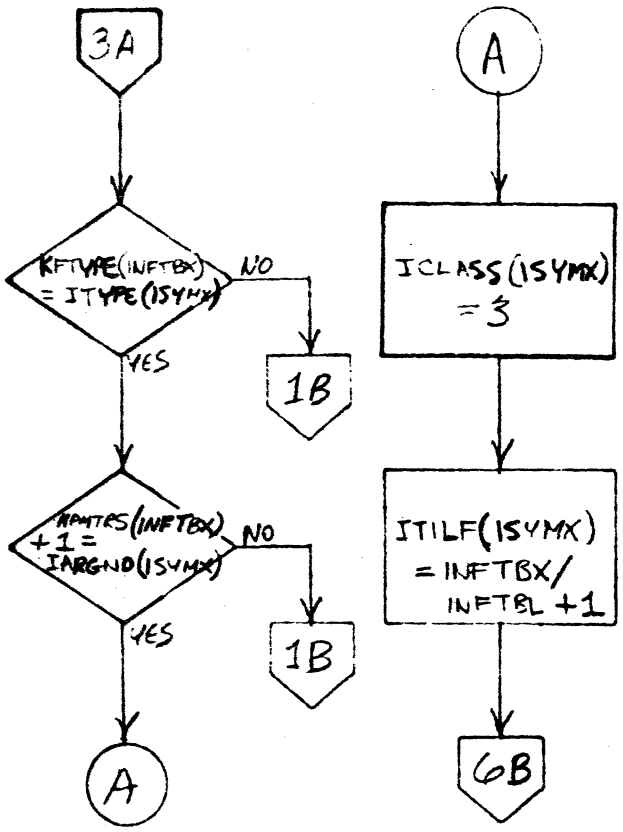
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE HELEN		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 5/67	TASK NAME			

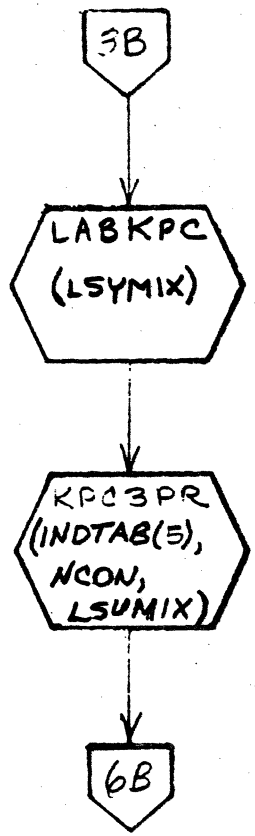
AA1385 (FORMERLY CA127-1)



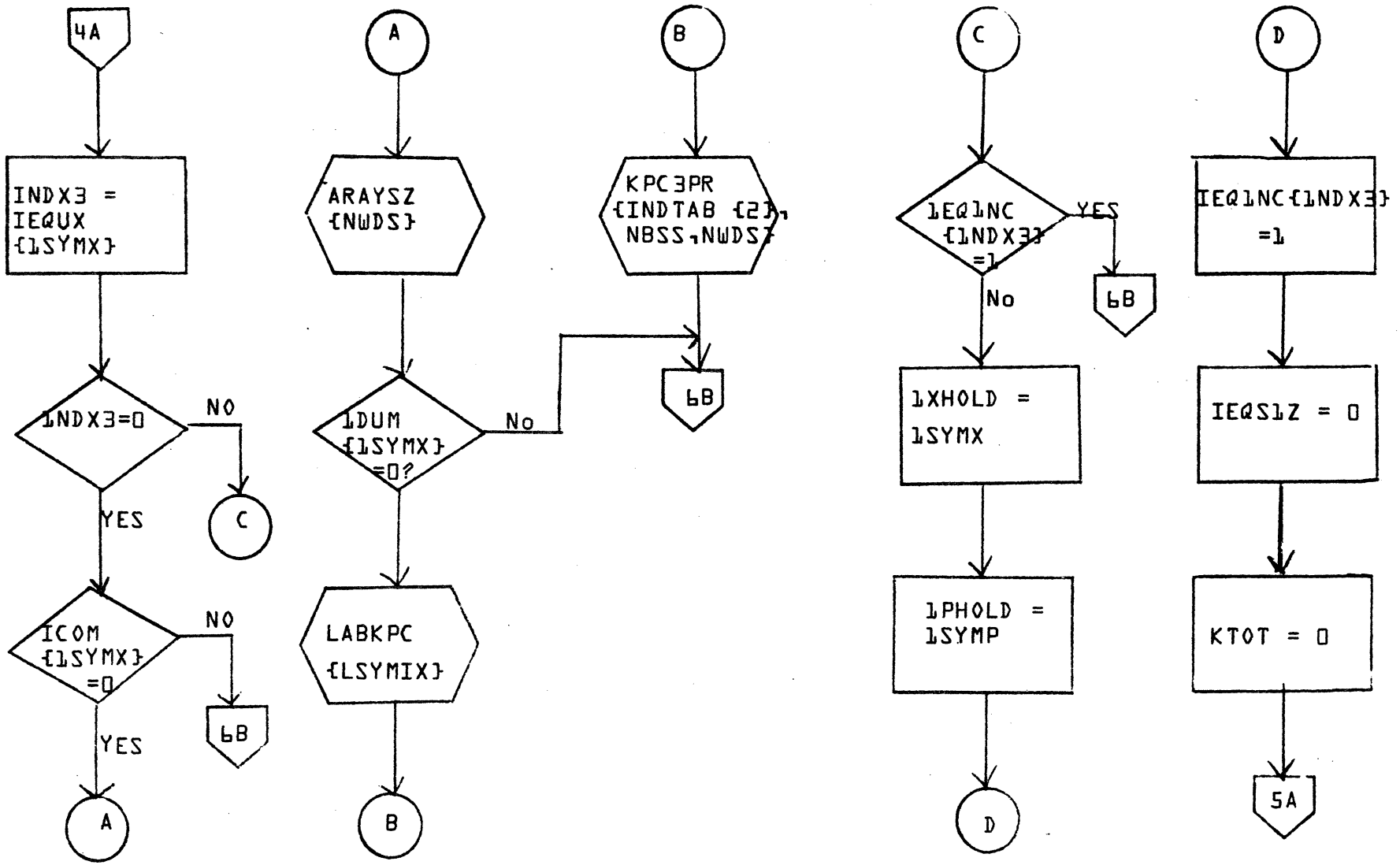


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE HELEN	PROJECT MGR.			
	PAGE 2 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			

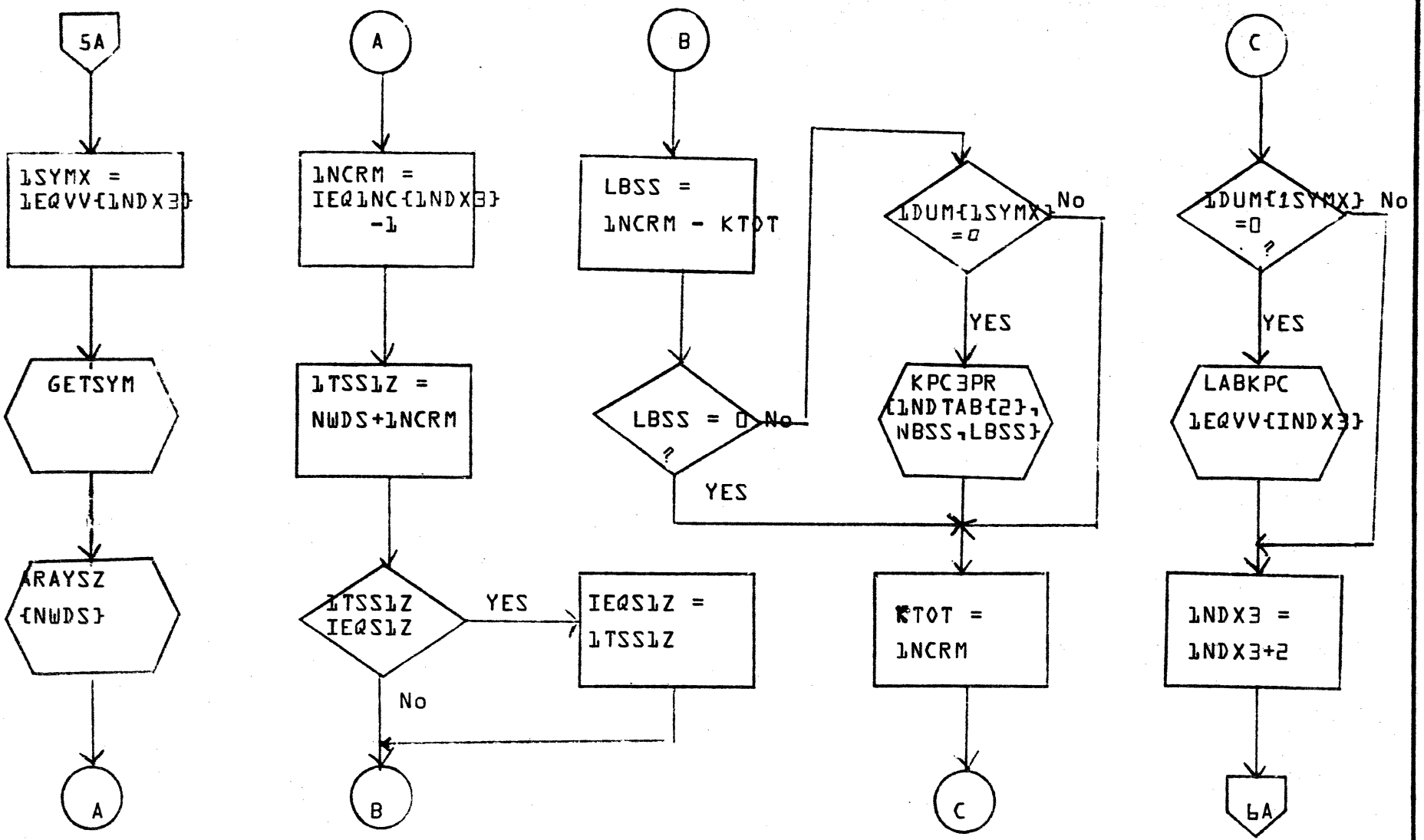




CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE HELEN		PROJECT MGR.		
		PAGE 3 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-67	TASK NAME		

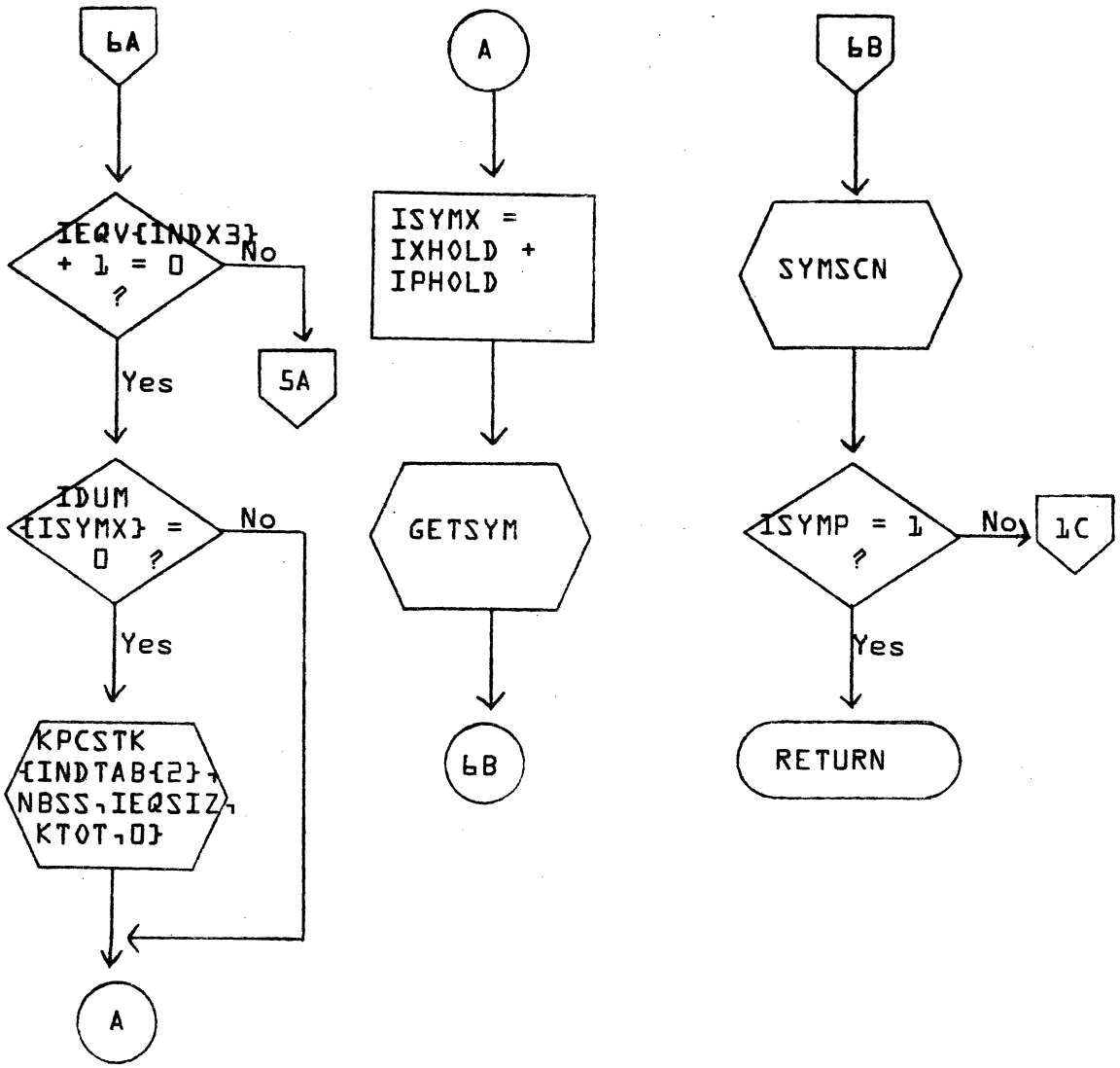


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE HELEN		PROJECT MGR.		
		PAGE 4 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3/67	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE HELEN		PROJECT MGR.		
		PAGE 5 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3/67	TASK NAME		

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE HELEN		PROJECT MGR.			
		PAGE 6 OF 6	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

4.4.20 DUMMY

Subroutine interface for programs calling subroutines or functions generated by FORTRAN or calling sequences generated by FORTRAN.

I. Calling Sequences

A. General

Calling sequences written in assembly language and intended to communicate with FORTRAN-generated subroutines, and those generated by FORTRAN from statements or expressions such as: CALL SUB (P_1,P_2,P_3,...P_n) or FUNC(P_1,P_2,P_3,...,P_n), will have the following code:

LOC	RTJ*	SUB
LOC+1		address of parameter 1
LOC+2		address of parameter 2
LOC+3		address of parameter 3
⋮		
LOC+N		address of parameter N
LOC+N+1		(return address)

(If the RTJ requires a 2-word relative command, then the first parameter will be at address LOC+2, etc., but this fact has no bearing on the logic of the interface.)

In all cases, parameters are addresses of:

1. variables
2. constants
3. storage locations containing results of computed parameters
4. in the case of I/O calls, actual constants rather than addresses

The CALL below contains examples of each of the first three types of parameters respectively.

```
CALL SUB (VAR, 3.5, A+B)
```

Addresses of parameters occur in consecutive locations following the RTJ command, one cell per address in the order that the parameters appear in the CALL or FUNCTION statement, left to right.

B. Floating Point

Floating point calling sequences include other special data interspersed in the parameter list. Consult floating point specifications for differences. Tney are, however, germane to the discussion which follows.

C. Run Anywhere Case

Parameters may consist of two types of addresses:

1. Absolute addresses

   a. All variables which are in COMMON will be generated as absolute addresses.
   b. All variables which are also parameters to the routine which contains the calling sequence will be computed at object time and placed into the calling sequence as absolute addresses. (Excludes floating point calling sequence parameters.)

2. Relative Addresses

   These will be generated by FORTRAN for all variables which do not fall into the categories for absolute addresses. This need arises from the necessity to compute effective addresses of parameters at object time. Shifting the location of blocks of code destroys the integrity of any absolute addressing which references variables internal to that block. What will be termed "self relative" addresses (which is what is meant by a relative address in a calling sequence) will be determined in the following manner:

   The address of the parameter (for example parameter 3) minus the location of the "self relative" address in the calling sequence (LOC+3), or:

   self relative address of parameter 3=(address of P_3) - (address LOC+3)

3. Bit 15

   Only the fifteen low order bits of a parameter (14-0) are absolutely necessary to express a parameter whether absolute or relative.

   a. Non-floating point calls

      In all calling sequences (except floating point) bit 15 then, is used as a flag to distinguish between the two addressing modes in that:

bit 15 = 0, if this parameter is an absolute address.
bit 15 = 1, if this parameter is a relative address. In cases of forward references, the bit would have to be set; in backward references, the bit would already be set. Thus part of the computation of a relative address (see entry routine) involves masking bit 15 off and extending bit 14 to the left.

b. Floating point calls

Floating point calls employ a different method of distinguisying between relative and absolute addresses. Bit 15, for relative addresses, is artificially set to 0. Absolute address already have bit 15 = 0.

D. Non-Run Anywhere Case

1. Non-floating point calling sequences

Parameters will consist only of absolute addresses as mentioned in 3a but will never require object-time computation, since programs will never enter memory except through the loader. Parameters still must be plugged by the entry routine but no computation of effective address is required.

2. Floating point calling sequences

The contrasting design of the floating point package now takes advantage of the indirect addressing ability of the 1700. Parameters which are indirect addresses are given a 1 in bit position 15. Its precise use will be outlined under paragraph B.

II. Subroutine or Function Entry

A. Entry routines will have the following general responsibilities:

1. Picking up parameters and, if necessary, biasing the ones which are relative and plugging the addresses wherever necessary.
2. Computing and setting up the proper return address to the calling program.
3. Initializing the execution of the main part of the subroutine or function.

B. The task of properly linking up instructions and other references to external parameters begins at the start of instruction generation. The Subroutine Argument Table (KSUBAT) and the Statement Function Argument Table (KSFAT) are built and updated in the following manner.

1. Initializing

   a. Function Subprograms

      The arguments are all placed in KSUBAT and made empty. (I.e., the table is initialized to show no increments.)

      Since the name of the function subprogram logically applies to the value of the function, but physically must apply to the entry point, the following procedure will be used to create the distinction:

      Two separate symbol table entries will be created for that name by PASS1 and both will be passed on to PHASE 4. The symbol table entry which refers to the entry point label will be differentiated from the symbol table entry which refers to the label of the function value by having bit 15 of ISYM(1) set to 1. In the latter ISYM(1) will be 0. In fact, all symbol table entries which are entry points (i.e., to include names of subroutines as well) will have this bit set.

   b. Statement Functions

      Statement Functions do not compute their values into labeled locations nor are they external to the calling programs. Thus, only a single symbol table entry is used and bit 15 is 0.

      All arguments (including the additional one if function is floating point) are initialized into the Statement Function Argument Table (KSFAT) in precisely the same manner as described for function subprograms.

   c. Subroutines

      Subroutine name entries in the symbol table will also have bit 15 set as noted above. Since subroutines pass results back through parameters they need no extra label nor do they need an extra parameter. Otherwise all arguments are initialized into KPCT precisely in the same manner as described for function subprograms.

2. Updating (This is the function of subroutine DUMMY)

2a. Updating KSUBAT

Each time an instruction operand or a calling sequence operand is encountered which is an argument to the subroutine being compiled, it is paired with the additive (or increment) of that reference and considered unique from any other argument (or the same argument but different additives).

The unique argument additive is looked up in KSUBAT to see if it exists. If not, it is added to the table such that the additives are sorted lowest to highest. Also, a temporary storage cell is assigned to it and the reference to it is modified to a reference to the temporary storage cell as already described.

When a temporary storage cell is assigned to replace argument references a special bit, KDUMY, is set in the symbol table entry for that temporary storage.

2b. Updating KSFAT

Updating KSFAT follows exactly the same procedure as outlined for KSUBAT, except that the instruction operands and calling sequence operands concerned are not those external to the program being compiled but rather external to the Statement Function being compiled (i.e., dummy arguments of the function).

3. Phase C

Phase C must perform the following functions for both run-anywhere and non-run-anywhere:

a. When the first reference is encountered to a temporary cell (of the type described above, i.e., KDUMY = 1) it is made a two word command such that the operand is the contents of the address located in the second half of this instruction. The address of the second word of the instruction is kept and cross-referenced with the temporary cell.

b. All subsequent references (less than 128 cells distant) are made indirect references to the address portion of that command.

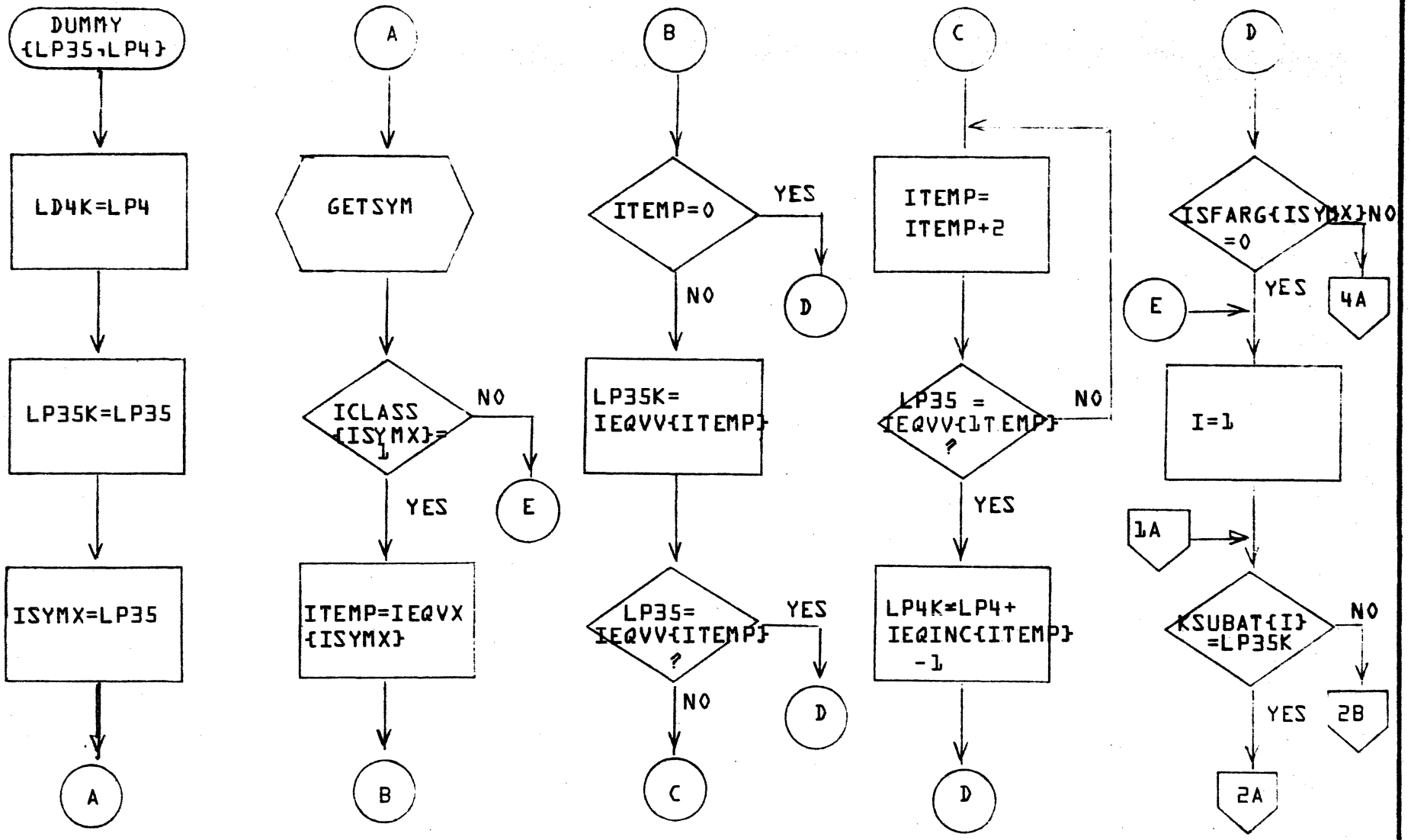
c. When a reference is encountered outside of the range the process begins over again as in step 1.

d. Whenever a reference is discovered in a calling sequence, the address of that parameter's location in the calling sequence is noted as in 1. Subsequent references may then refer back to this location subject to the same restrictions.

e. When PhaseC encounters the pseudo-instruction PST, the operand portion will contain a pointer to a temporary cell (of the type described above). At this point, PHASE5 must generate a STA command into each one of the addresses cross-referenced with that temporary cell. Phase B will already have generated all the preliminary code to compute the address of the parameter and its increment.

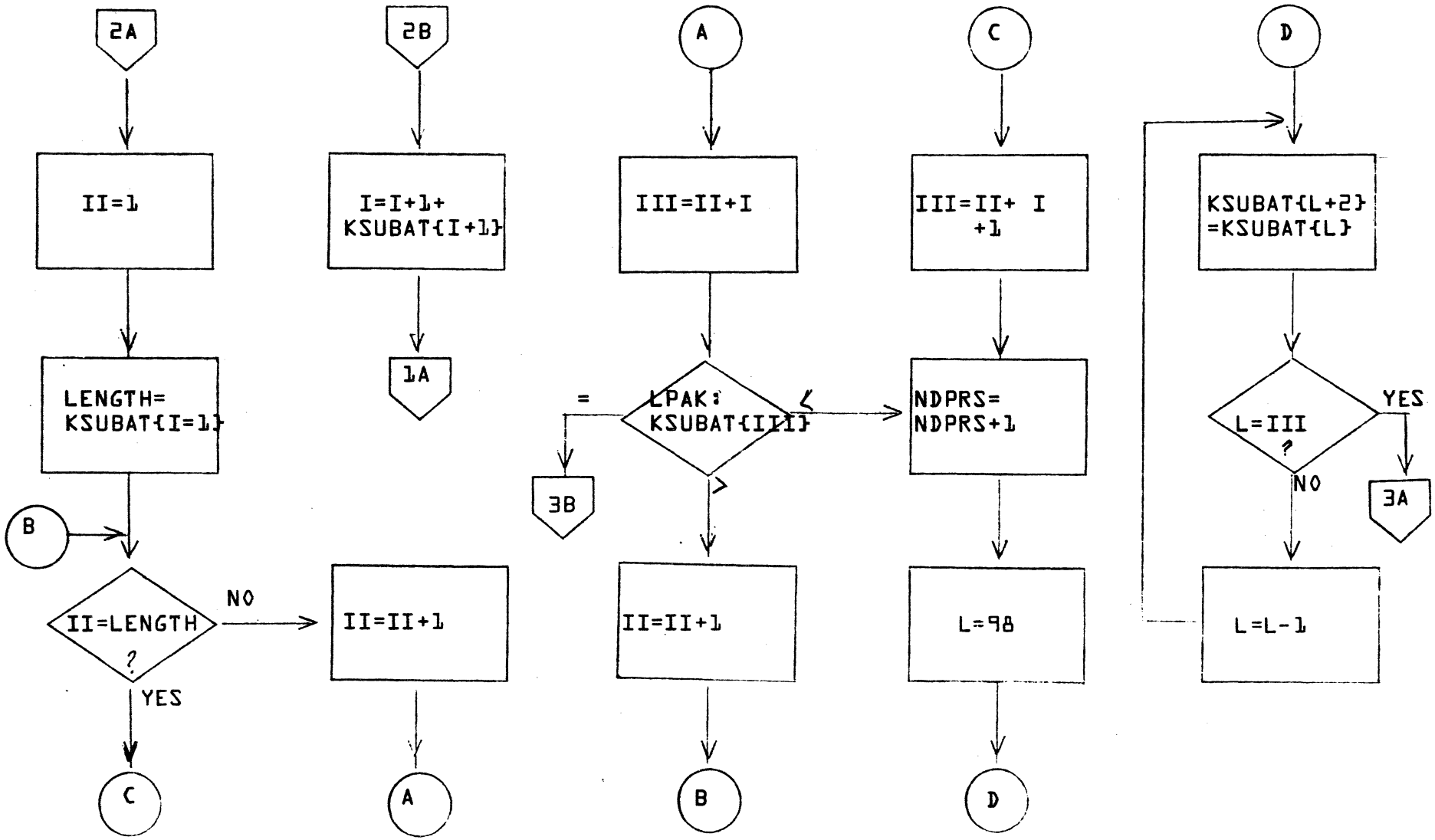
4. Entry Routines

The entry routine is generated for each Statement Function and again for the whole Subroutine or Function Subprogram. (See ENTCOD 4.4.4.)

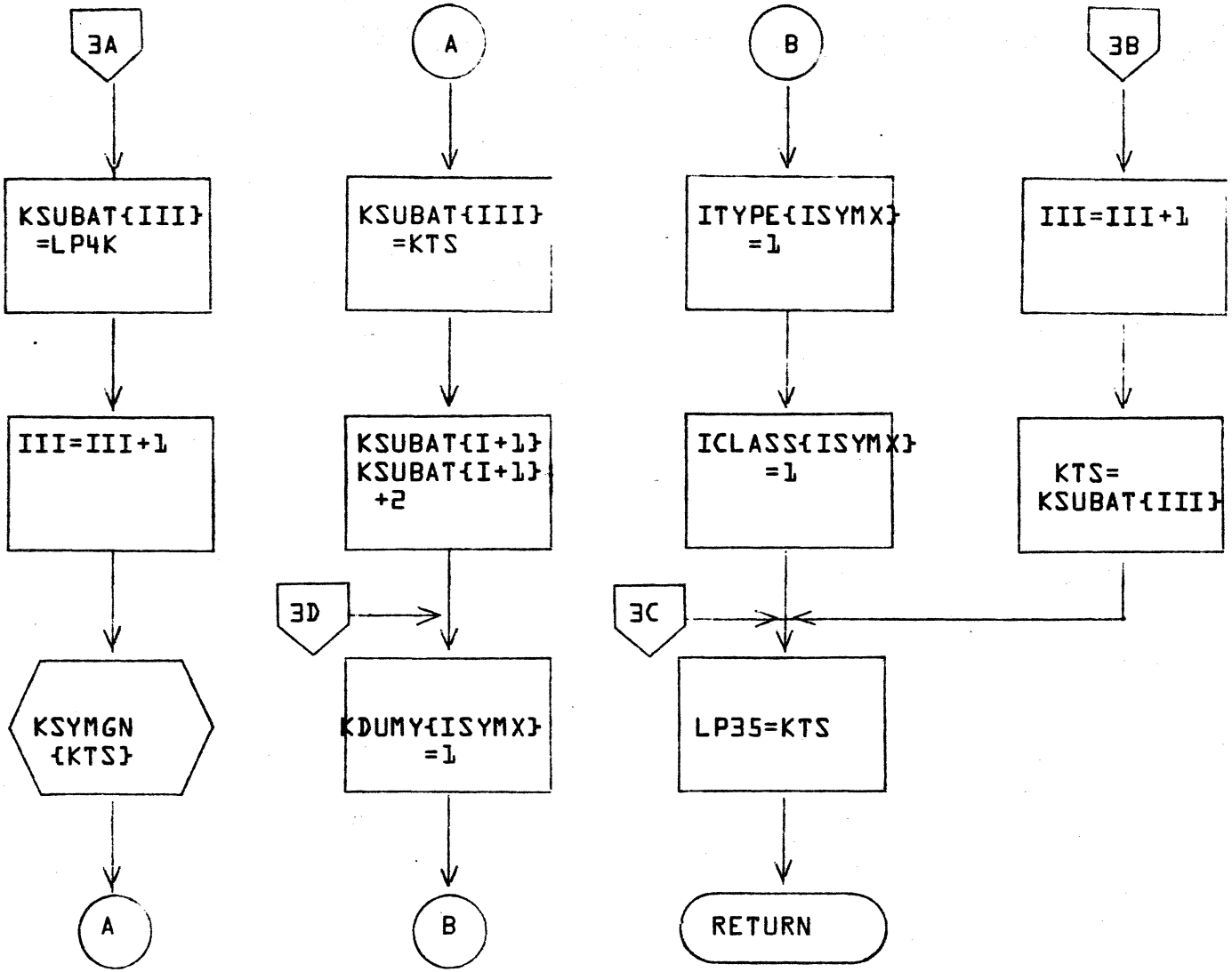


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE DUMMY PAGE 1 OF 4 NUMBER ISSUE DATE DRAWN BY DATE 3-67	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

AA1385 (FORMERLY CA127-1)



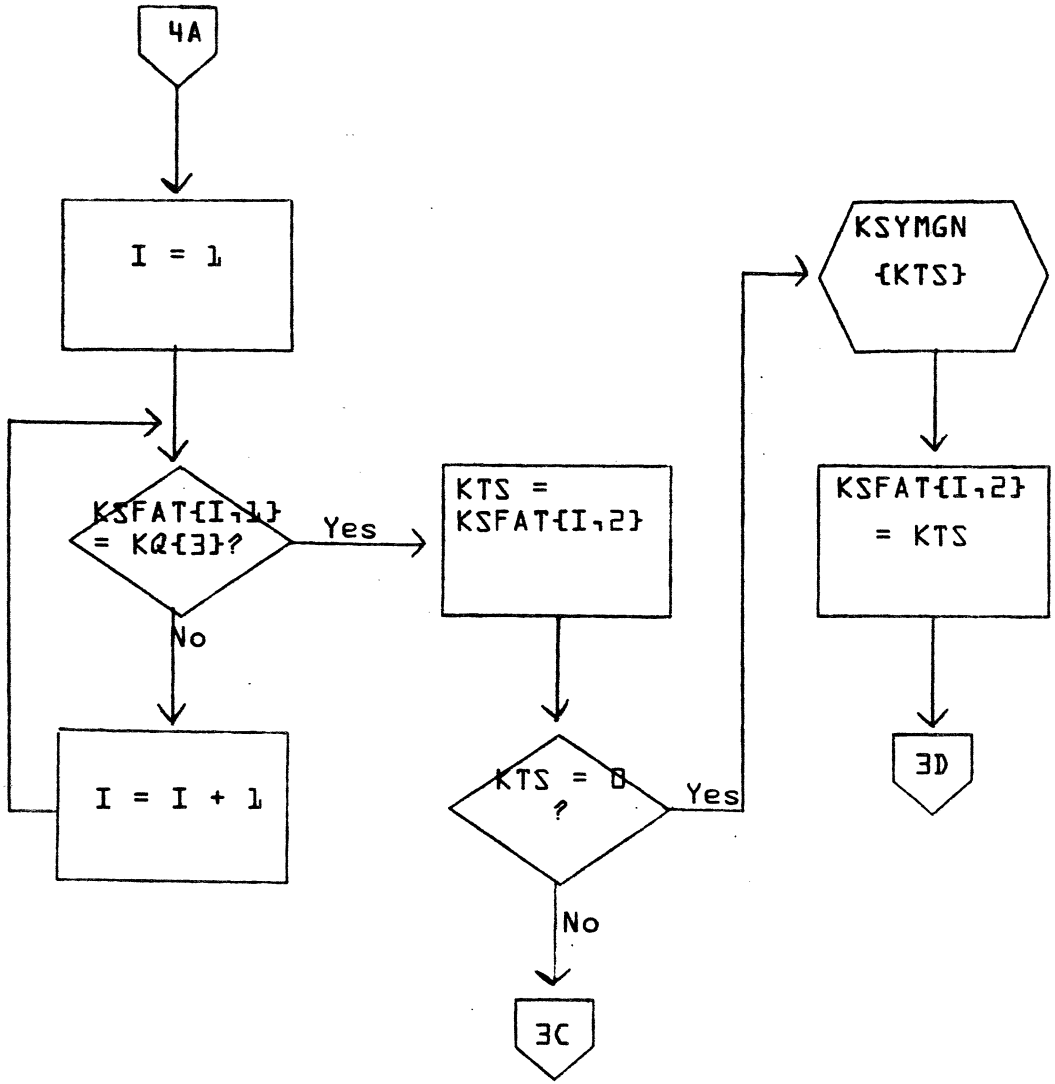
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DUMMY		PROJECT MGR.		
		PAGE 2 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE 3/67	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DUMMY		PROJECT MGR.		
		PAGE 3 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3/67	TASK NAME		

AA1385 (FORMERLY CA127-1)

4-182



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DUMMY	PROJECT MGR.			
	PAGE 4 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK. NO.			
	DRAWN BY DATE	TASK NAME			

KPCSTK

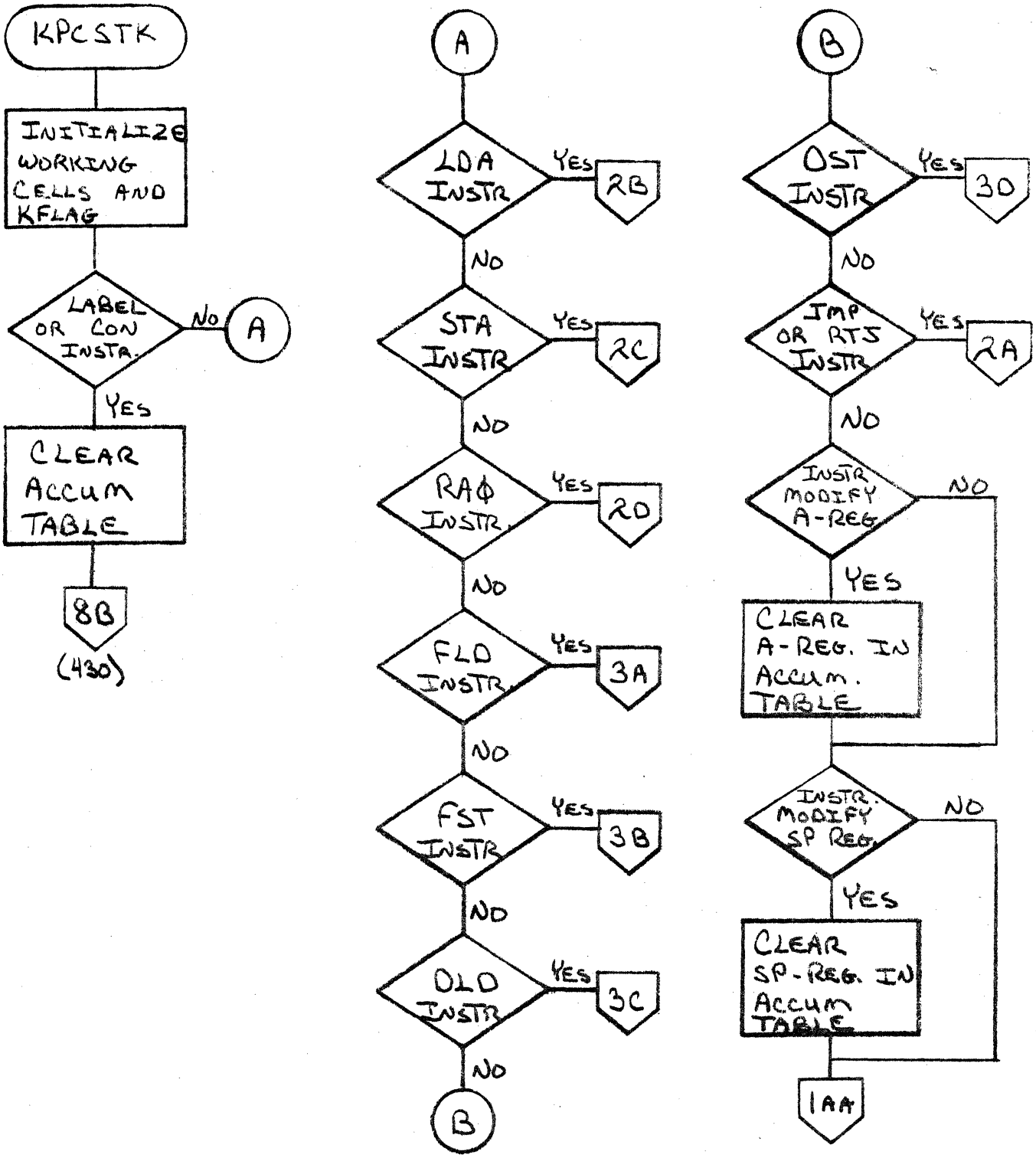
KPCSTK outputs pseudo code. KPCSTK updates the accumulator memory table (KART). KPCSTK adjusts addresses to reflect additives and subscripts. Some optimization is performed by checking sizes of constants (conversion of LDA commands to ENA wherever possible).

Input parameters:

1. Indicator entry from INDTAB
2. Operation code
3. Operand (constant or symbol table pointer)
4. Additive value to operand
5. Symbol table pointer to variable subscript of operand

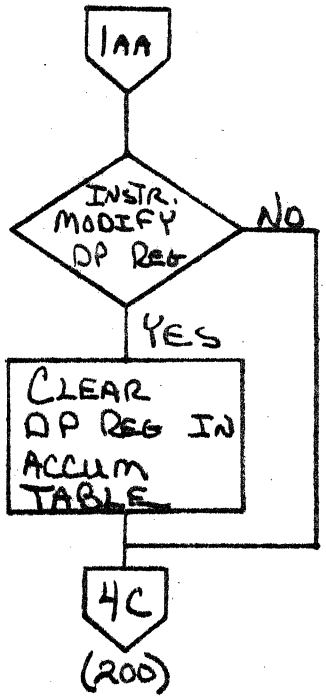
KPCSTK calls:

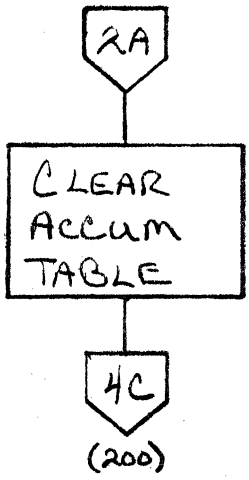
KPOUT
FCMSTK
DUMMY
GETSYM
KCPART

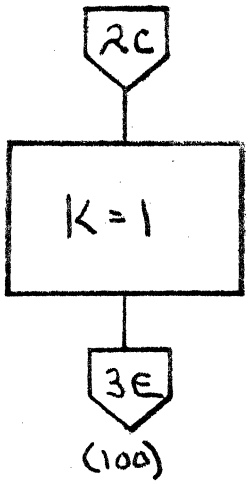


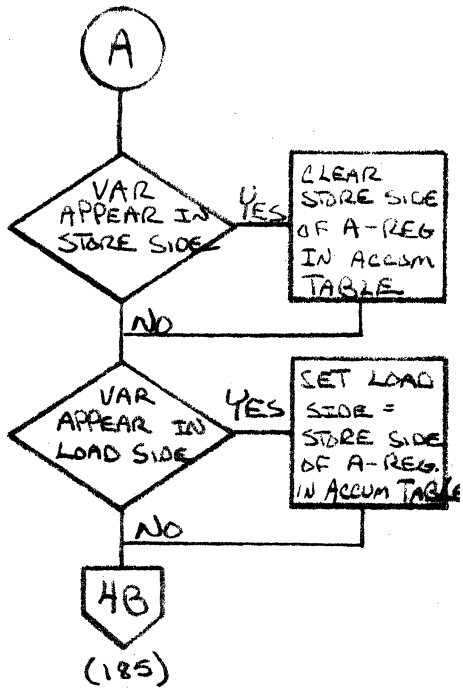
TITLE	KPCSTK		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 1 OF 9

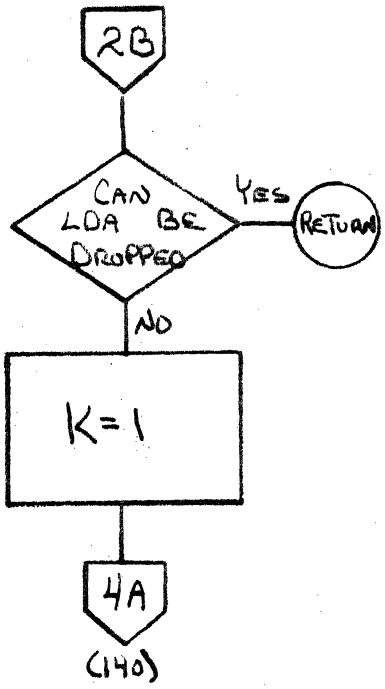
CSD 203

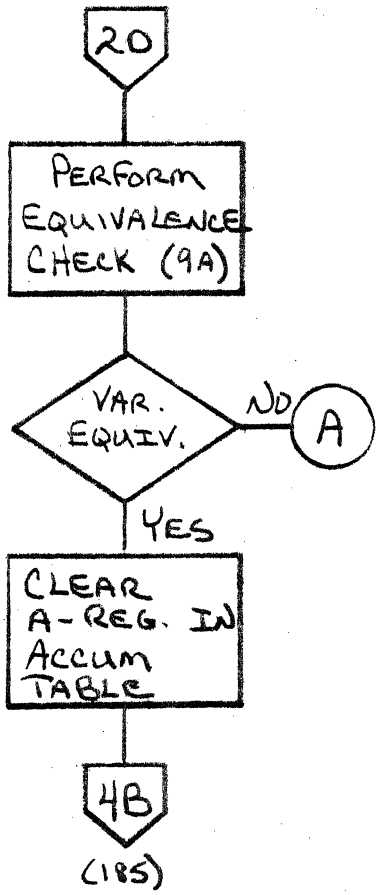

1AA
INSTR. MODIFY OP REG
NO
YES
CLEAR OP REG IN ACCUM TABLE
4C
(200)





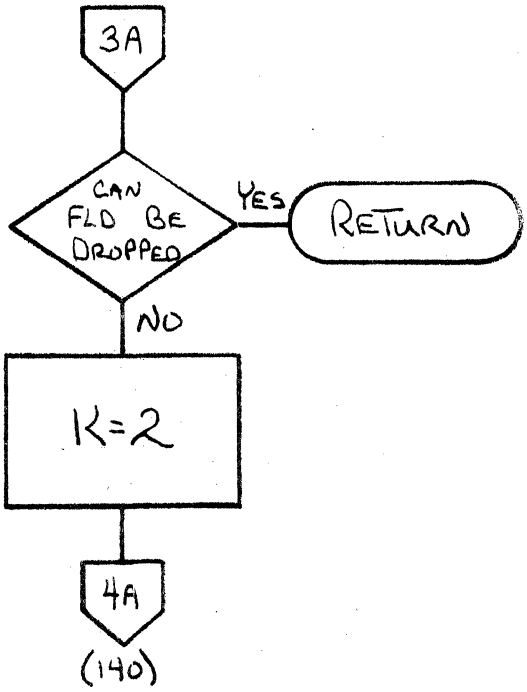


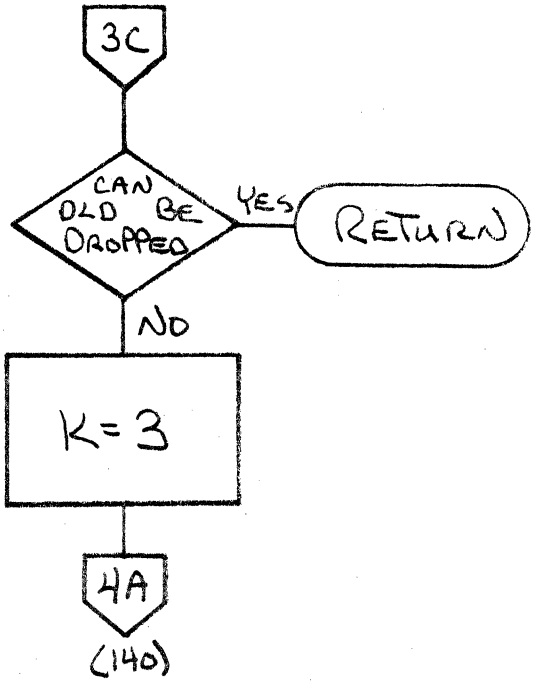


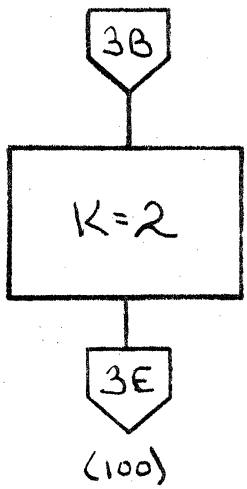


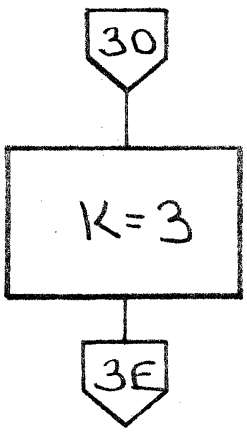
TITLE KPC STK			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2 OF 9

CSD 203



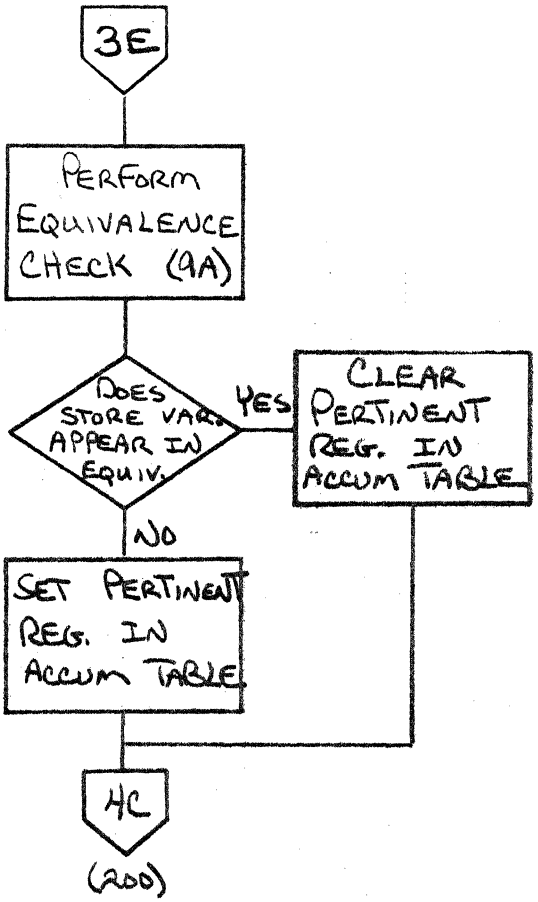


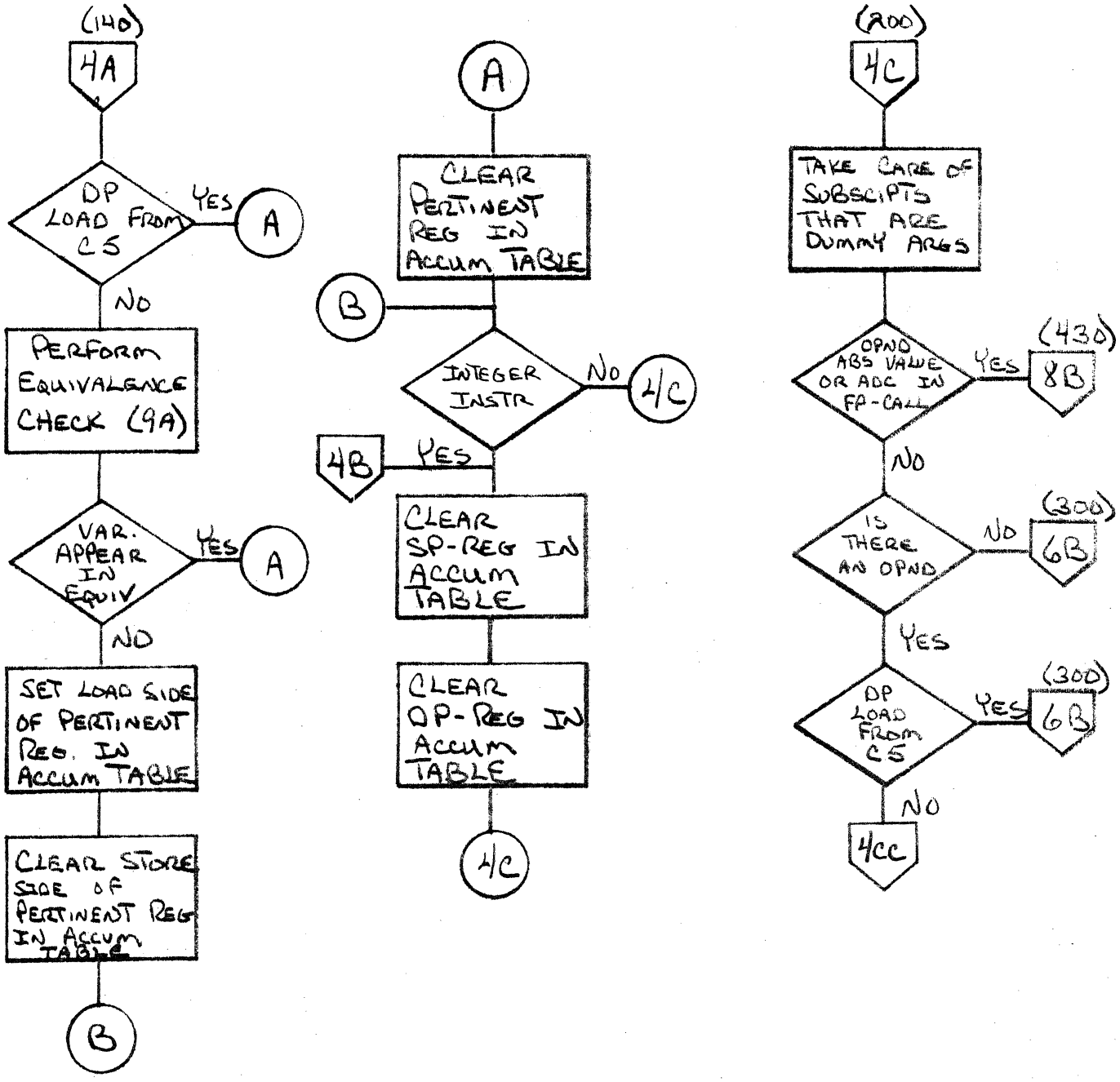




TITLE KPCSTK			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 3 OF 9

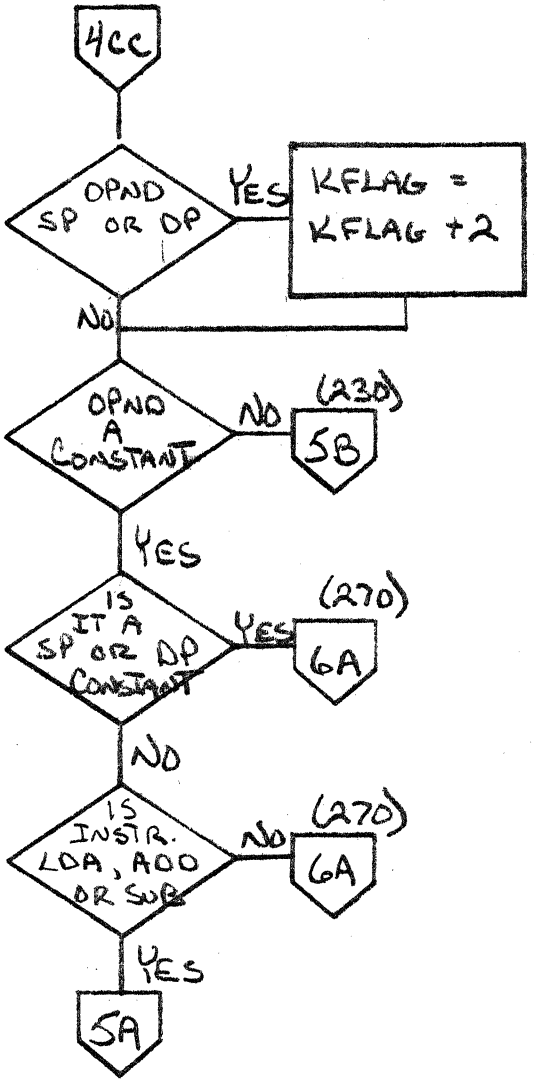
CSD 203


3E
PERFORM EQUIVALENCE CHECK (9A)
DOES STORE VAR. APPEAR IN EQUIV.
YES
CLEAR PERTINENT REG. IN ACCUM TABLE
NO
SET PERTINENT REG. IN ACCUM TABLE
4C
(200)



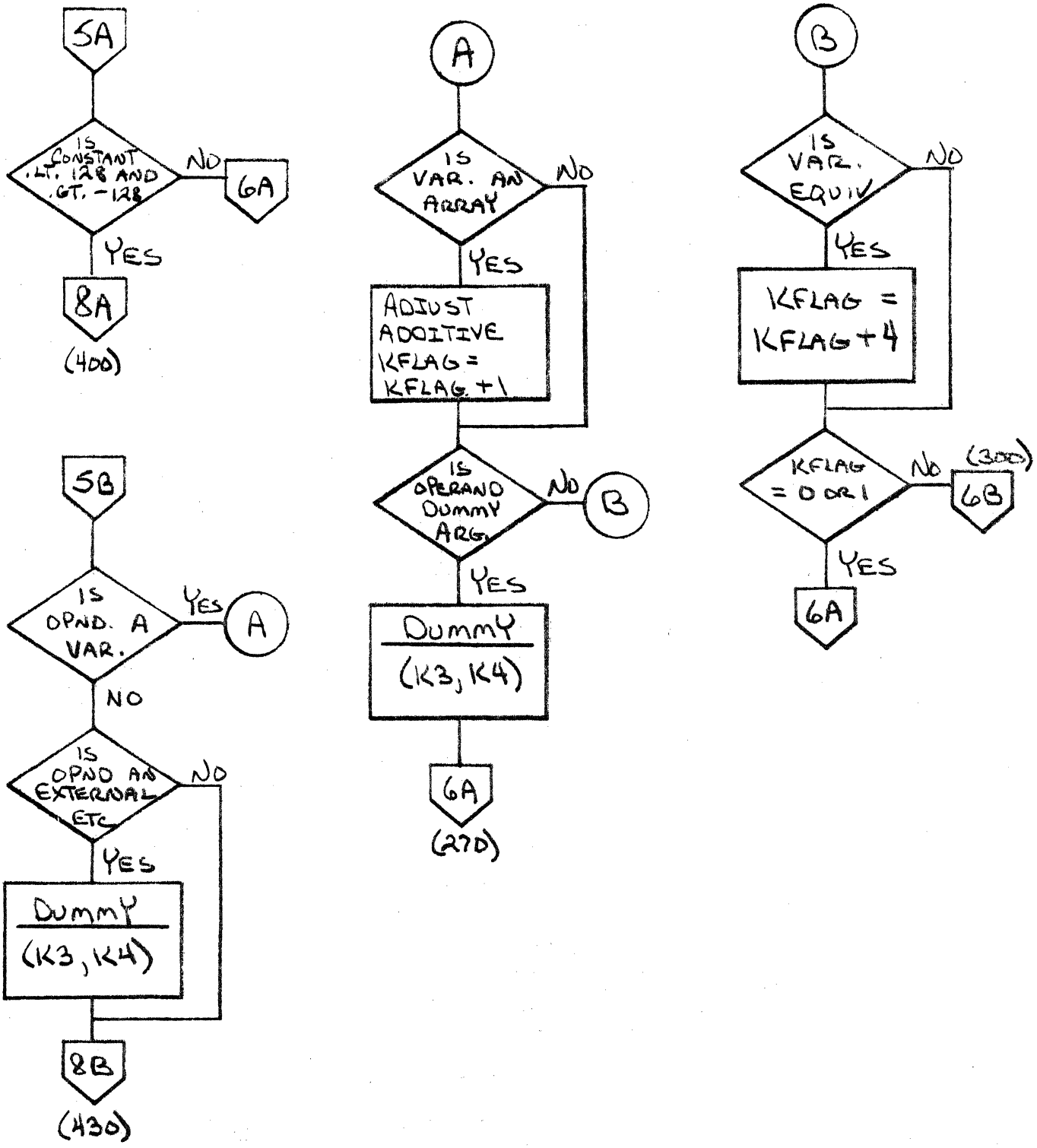
TITLE	KPCSTK		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 4 OF 9

CSD 203



TITLE KPCSTK			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 4A OF 9

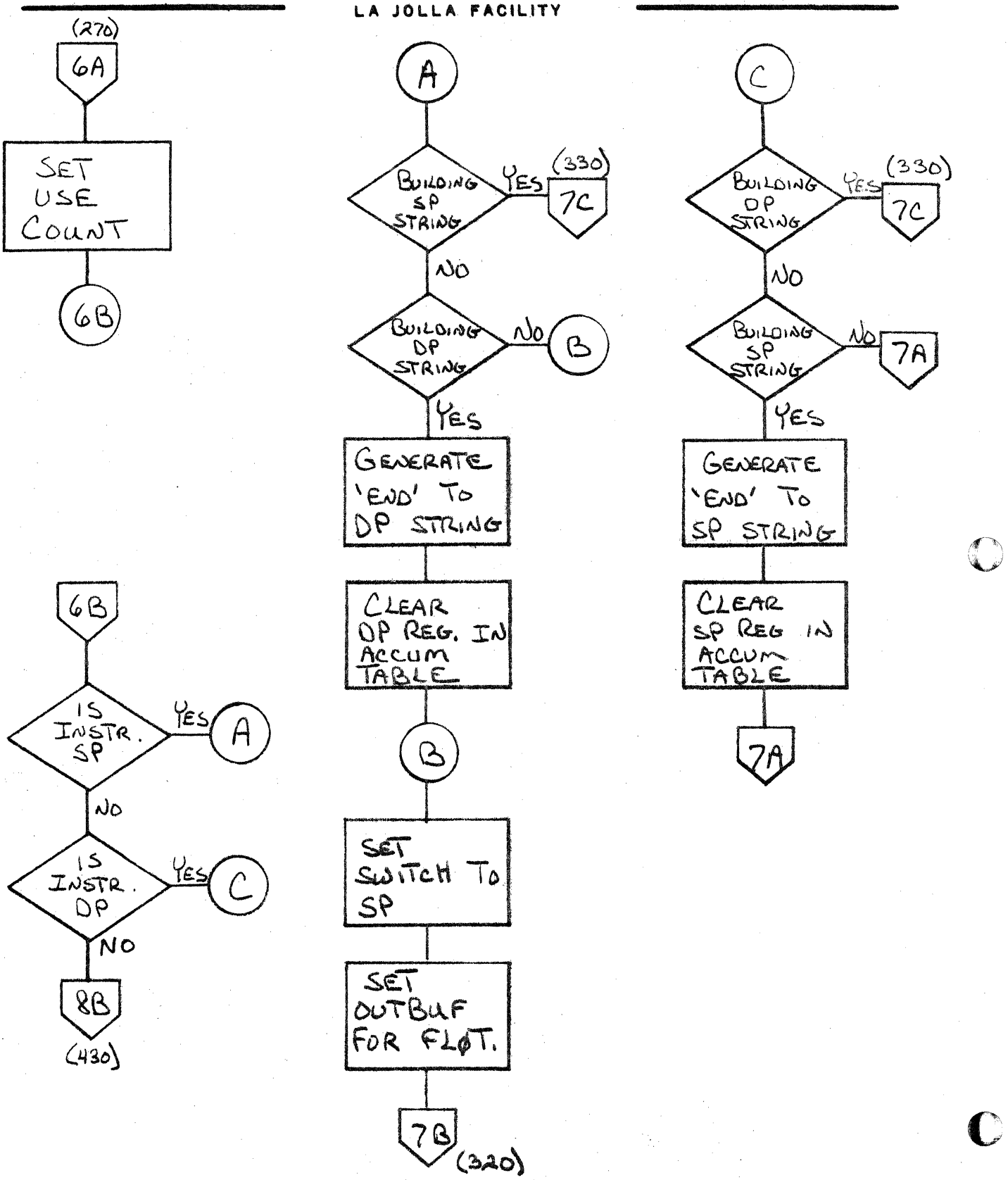
CSD 203



TITLE	KPCSTK		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 5 OF 9

CSD 203

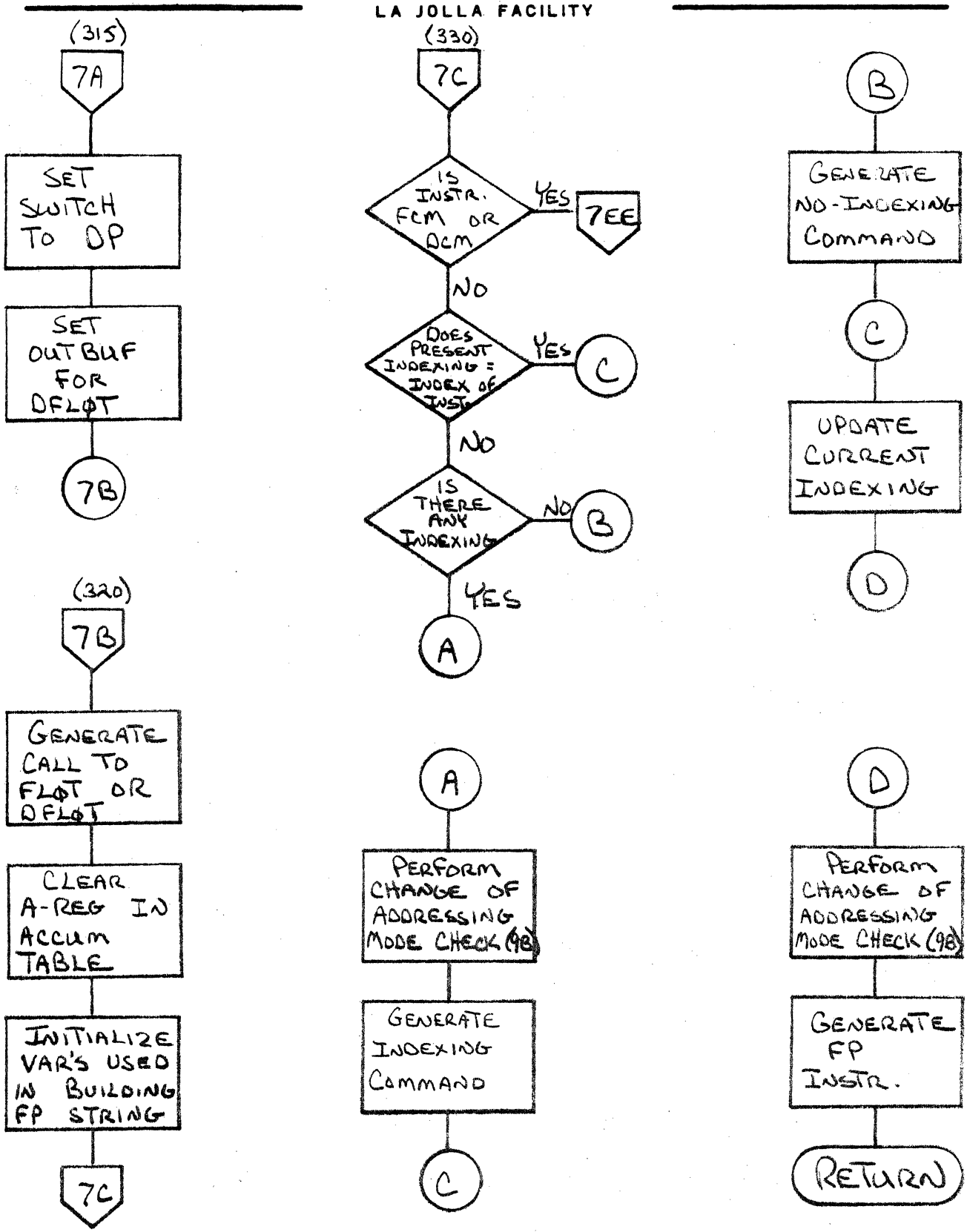
LA JOLLA FACILITY



TITLE	KPCSTK		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 6 OF 9

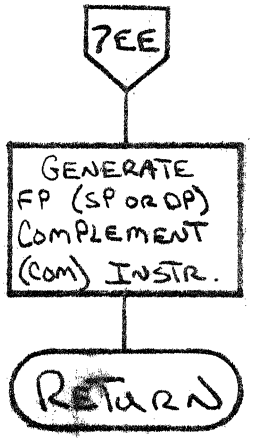
CSD 203

LA JOLLA FACILITY

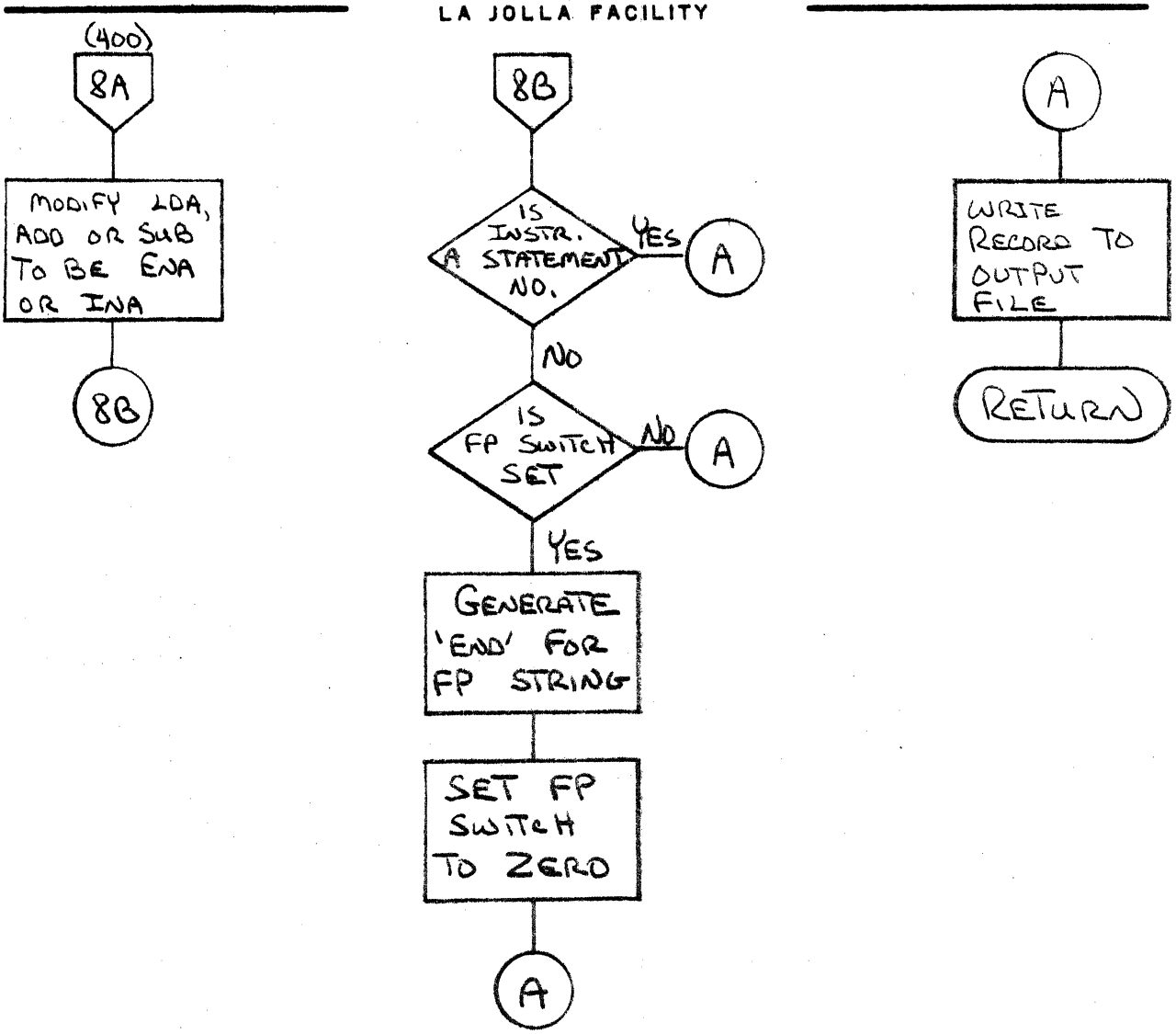


TITLE KPCSTK			DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 7 OF 9

CSD 203


7EE
GENERATE FP (SP OR DP) COMPLEMENT (COM) INSTR.
RETURN

LA JOLLA FACILITY

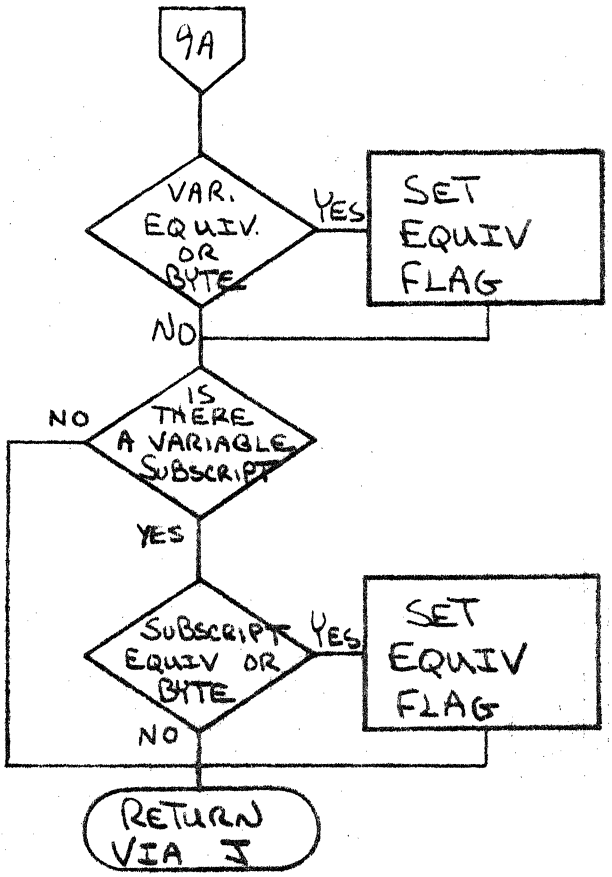


TITLE	KPCSTK		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 8 OF 9

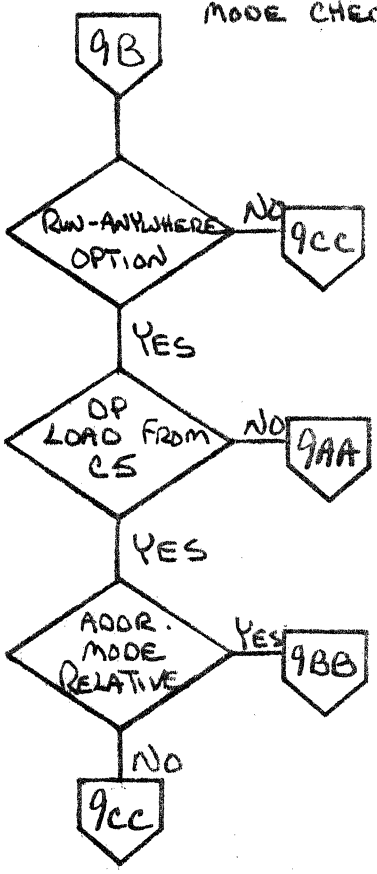
CSD 203

LA JOLLA FACILITY

(190 - EQUIVALENCE CHECK)

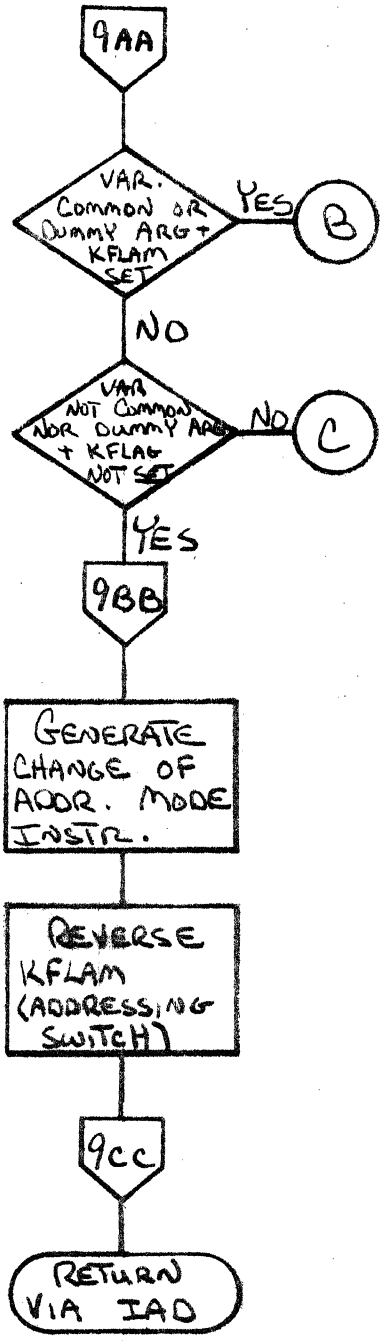


(380 - CHANGE OF ADDRESSING MODE CHECK)



TITLE	KPCSTK		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 9 OF 9

CSD 203


9AA
VAR. COMMON OR DUMMY ARG + KFLAM SET
YES
B
NO
VAR NOT COMMON NOR DUMMY ARG + KFLAG NOT SET
NO
C
YES
9BB
GENERATE CHANGE OF ADDR. MODE INSTR.
REVERSE KFLAM (ADDRESSING SWITCH)
9CC
RETURN VIA IAD

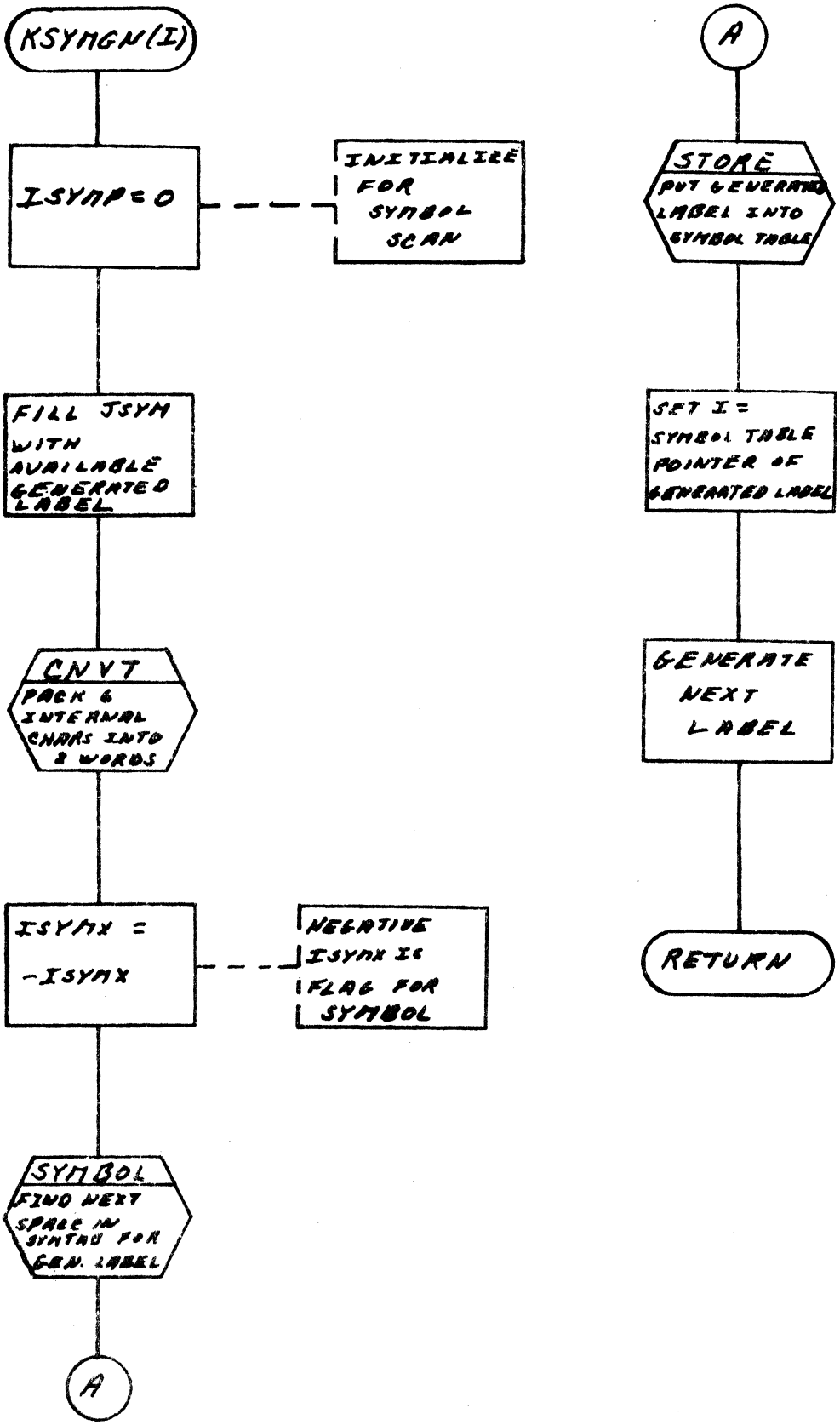
4.4.21 KSYMGN

KSYMGN generates variable names used internally in the compilation of a routine as labels, temporary storage, or dummy parameters. The variables are stored in the symbol table and the pointer to the symbol table entry is passed to the calling routine in the parameter I.

The generated names are of the form .hhhhh where h denotes a hexadecimal digit and the first generated name is .00000.

KSYMGN calls:

CNVT
STORE
SYMBOL


KSYMGN(I)
ISYMP=0
INITIALIZE FOR SYMBOL SCAN
FILL JSYM WITH AVAILABLE GENERATED LABEL
CNVT
PACK 6 INTERNAL CHARS INTO 2 WORDS
ISYMX = -ISYMX
NEGATIVE ISYMX IS FLAG FOR SYMBOL
SYMBOL
FIND NEXT SPACE IN SYNTAU FOR GEN. LABEL
A
A
STORE
PUT GENERATED LABEL INTO SYMBOL TABLE
SET I = SYMBOL TABLE POINTER OF GENERATED LABEL
GENERATE NEXT LABEL
RETURN

Chapter 5

TABLE OF CONTENTS

5.2 Assembler Output Phase (continued)

5. Assembler Phases C and D/E

Phases C and D/E are the assembler phases of the FORTRAN compiler. Their function is to assign storage locations, assign index registers, optimize use of one-word addressing and produce relocatable binary and list output.

Phase C is the input phase of the assembly process, and Phase D/E is the output phase. Subroutine PHASEC is the main processor for the input phase. The intermediate pseudo-instruction output of phase C is written on a disk scratch file, from which it is read by PHASE6, the main processor for the output phase D/E.

5.1 Assembler Input Phase C

Three types of records are handed to the assembler by the previous pass; pseudo instructions, data strings and format records.

5.1.1. Pseudo Instructions

Pseudo instruction records consist of 1700 commands, pseudo commands, labels and statement numbers. Subroutine PHASEC processes all pseudo instruction records except storage reference instructions, conditional skip instructions, BSS pseudo instructions, constants and labels. Those items not processed by subroutine PHASEC are processed by calls to the following subroutines:

	Subroutine
Storage Reference	CL12
Conditional Skip	SKIP
BSS	BSS
Constant	CON
Label	LABIN or LABEL

The initial pseudo instruction records read in the input pass contain variables, constants and arrays to which the assembler must assign storage locations. Subroutine LABIN performs this function.

When the first executable command is encountered, excluding the jump at the top of a main program, pseudo instructions are sent to the Index Optimization Subroutine, IXOPT. If an instruction is indexed, the subroutine decides which index to use. If the instruction is not indexed, it is checked for its possible effect on subsequent indexing.

An output record is constructed for each pseudo instruction and stored in the output buffer. Instructions with absolute operands are completely processed in the input phase. The output record is flagged as ready for output. Instructions with relocatable operands are partially processed. Final assignment of the operand address is made after analysis in the output phase. Address modes are identified in the input phase. All instructions with two-word relative addressing mode are flagged as one-word candidates. Instructions with no address mode field and those with a mode other than two-word relative are flagged as fixed length but not ready for output.

Pseudo instruction records are placed in the output buffer by subroutine INOUT. When an END record is encountered control is passed from PHASEC to the output phase.

5.1.2 Data Strings

Data strings are completely processed in subroutine DATAST. A separate output record, identified as an ORG record, is constructed for each datum in the same format as the pseudo instruction record.

5.1.3 Format Records

Format records are completely processed in subroutine PHASEC ORG records containing two ASCII characters are constructed in the same format as the pseudo instruction record.

5.1.4 Record Formats

5.1.4.1 Input Record Instruction Codes

5.1.4.2 Breakdown of Input Record Indicator

5.1.4.3 Input Pseudo Instruction Record Formats

5.1.4.4 Output Record Formats

5.1.4.5 Assembler Instruction Type Codes

5.1.4.6 Breakdown of Output Record Indicator

5.1.4.7 Phase C and D/E Common Blocks

INPUT RECORD INSTRUCTION CODES

Instruction	Code
BSS	1
ADC	2
CON	3
END	4
PARAMETER STORE	5
STATEMENT NO.	6
JMP	10
RTJ	11
LDA	12
AND	13
STA	14
STQ	15
RAO	16
ADD	17
SUB	18
LDQ	19
DVI	20
ADQ	21
MUI	22
EOR	23
KLDQ (I register load)	28
KSTQ (I register store)	29
ENA	30
INA	31
ENQ	32
INQ	33
LRS	34
LLS	35
QRS	36
QLS	37
ARS	38
ALS	39

Instruction	Code
Conditional Jump	
A Jump ≠ 0	40
A Jump = 0	41
A Jump < 0	42
A Jump ≥ 0	43
A Jump ≤ 0	44
A Jump > 0	45
Q Jump ≠ 0	46
Q Jump = 0	47
AAQ A	60
TCQ A	61
TRA Q	62
TCA A	63
LAQ Q	64

BREAKDOWN OF INPUT RECORD INDICATOR

Bit No.	Holder Name	Meaning
15	NL	Not used in input phase.
14	NT	This record is a label
13	NA	This record has an additive
12	NS	This record has an index
11	ND	Use indirect addressing mode
10	NR	Use two-word constant Addressing Mode
9	NQ	Assign index to Q register
8	NF	Not used in input phase
7-2		
Instruction code ≠ 2	NTYPE	
	= 0	Addressing mode to be assigned.
	= 1	Absolute operand. Use one-word direct addressing mode.
	= 2	Absolute operand. Use two-word constant addressing mode.
	= 3	Not used
	= 4	Operand is relocatable Address constant
	= 5	Operand is absolute Address constant
Instruction code = 2	NTYPE = 0 thru 5	Instruction is address constant type 0 through 5. See description of address. Constant types under Subroutine PHASE 5 (5.1.5.)
1-0	NOWI	Number of words in the record minus 2.

INPUT PSEUDO INSTRUCTION RECORD FORMATS

(Assembly Input Phase)

INSTRUCTION	WORD 1	WORD 2	WORD 3	WORD 4	WORD 5
Holder Name		NOPC	NOPT	NADD	NXPT
LABEL	Indicator	Label Pointer			
BSS	Indicator	Instr Code	Size		
ADC	Indicator	Instr Code	Address Pointer	Additive	
CON	Indicator	Instr Code	Constant		
END	Indicator	Instr Code			
Parameter Store	Indicator	Instr Code	Operand Pointer		
Statement Number	Indicator	Instr Code	Statement No.		
Storage Reference Instr					
Type Code ≠ 1 or 2	Indicator	Instr Code	Operand Pointer	Additive	Index Pointer
Type Code = 1 or 2	Indicator	Instr Code	Operand		
Register Reference and Shift Instructions	Indicator	Instr Code	Operand		
Inter-Register Instr	Indicator	Instr Code			

OUTPUT RECORD FORMATS

(Assembly Input Phase to Assembly Output Phase)

INSTRUCTION	WORD 1	WORD 2	WORD 3	WORD 4	WORD 5
Holder Name		NOPC	NOPT	NADD	NXPT
LABEL	Indicator	Label Pointer			
BSS	Indicator	Size			
ADC	Indicator	Address Pointer	Additive		
CON	Indicator	Constant	2nd word of a Floating Point Constant		
END	Indicator	0			
Statement Number	Indicator	Statement No.			
Storage Reference Instr.					
Type Code = 1	Indicator	Binary Instr			
Type Code ≠ 1	Indicator	Bits 15-8 Binary Instr. Bits 7-0 *Operand Code	Operand or Operand pointer	Additive	Index Pointer
Register, Reference, Shift and Inter-register instructions	Indicator	Binary Instr			
Skip Instructions	Indicator	Binary Instr	0		
ORG	Indicator	Address Pointer	Data Word		
Continuation ORG	Indicator	Data Word			

* Operand Code = 0 NOPT = pointer to operand in symbol table
= 1 NOPT = pointer to operand in IDVTAB
= 2 NOPT = pointer to operand in NDSTAB
= 3 NOPT = Absolute operand

ASSEMBLER INSTRUCTION TYPE CODES

Code	Addressing Mode	Instruction
1	One-word direct	Storage Reference
2	One-word indirect	
3	One-word relative	
4	One-wrod relative, indirect	
5	Two-word constant, 2nd word is an absolute address constant	
6	Two-word constant, 2nd word is a relocatable address constant	
7	Two-word absolute	
8	Two-word absolute, indirect	
9	Two-word relative	
10	Two-word relative, indirect	
11		Register reference and shifts
		Conditional Skips
12		A register $\neq 0$
13		A register $= 0$
14		A register < 0
15		A register ≥ 0
16		A register ≤ 0
17		A register > 0
18		Q register $= 0$
19		Q register $\neq 0$
		Conditional Jumps
20		A register $\neq 0$
21		A register $= 0$
22		A register < 0
23		A register ≥ 0
24		A register ≤ 0

25	A register 0
26	Q register = 0
27	Q register ≠ 0
30	AAQ A
31	TCQ A
32	TRA Q
33	TCA A
34	LAQ Q
40	BSS
41	ADC Type 1
42	ADC Type 2
43	ADC Type 3
44	ADC Type 4
45	ADC Type 5
46	Integer Constant
47	Real Constant
48	Double Precision Constant
50	Data ORG
51	Data Continuation ORG
52	Program ORG
53	Program Continuation ORG
59	Statement No.
60	END

Breakdown of Output Record Indicator

Bit No.	Holder Name	Meaning
15	NL	Skip this record
14	NT	This record is a label
13	NA	Operand address has an additive.
12	NS	Operand address is indexed.
11	ND	This record is ready for output.
10	NR	This record is fixed length
9	NQ	Flag for address constant sub-type.
8	NF	Flag for address constant sub-type.
7-2	NTYPE	Assembler instruction type code. See list.
1-0	NOWI	Number of words in the record minus 2.

LIST OF HOLDERS, COUNTERS, FLAGS BUFFERS AND TABLES

IN PHASE C and D/E COMMON BLOCKS

NAME	USE
ICFLAG	Change flag set in ADMAX
ICOUNT	Program counter
ICT	Program counter increment
IDECR	Program address decrement set in ADMAX
IDVTAB(312)	Table of address holders built in CHOP
IFIX	Fixed buffer indicator set in AMOUT
INBUFF(203)	Input buffer
INCT	Index for INBUFF and NOBUFF. In phases D/E, circular buffer pointer to NOBUFF (always increases)
ISCOUN	Holder for ICOUNT saved from input phase
ITCOUN	Holder for ICOUNT saved from output phase
IUSE	Flag indicating use of I register in object program
IV(5)	Temporary storage
IXC	Occurrence flag for index to be assigned set in CHKWD and IXOPT
IXF	Holder for contents of object time I register
IXFLAG	Decision flag set in CHKWD
IXQ	Holder for contents of object time Q register
IXS1(10)	Table of saved indexes built in IXOPT
IXS2(10)	Table of occurrence flags for saved indexes built in IXOPT

NAME	USE
IXS3(10)	Table of equivalence table indexes for saved indexes built in IXOPT.
IXSX	Index for IXS1, IXS2 and IXS3.
LRS	Flag indicating instruction is LRS 16.
NA	Additive flag extracted from record indicator.
NADD	Additive to instruction operand address.
NADX	Index for IDVTAB.
NASCI	Holder for alpha output word used in SETPRT.
NATYPE	Assembly instruction type code in input phase. Index to alpha opcode table in output phase.
NBFL	Flag for last word in binary output record.
NBINC	Punch buffer index.
ND	Indirect flag extracted from record indicator in input phase. Flag indicating instruction is ready for output in output phase.
NDFL	Temporary holder for "ready for output" flag used in CL12.
NDSTAB(500)	Table of dummy parameter address holders built in CHOP.
NDXJ	Temporary holder for ICOUNT.
NF	Flag in the record indicator used in PHASE 5 and AMOUT to indicate subtype of address constant.
NFTAB(160)	Table of futures built in AMOUT.
NL	Indicator flag to ignore this record. Used in AMOUT.

NOBW	Number of binary words currently in the punch buffer
NOCT	NOBUFF index used in input phase In output phase D/E, next available entry in circular buffer.
NOBUFF(210) C (1152) D/E	Output buffer passed from the input phase to the output phase
NODS	Index to NDSTAB
NOFS	Index to NFTAB
NOPC	Instruction code in input phase Partial or complete binary word in output phase
NOPT	Pointer to operand location in the symbol table
NOUTOT	Total number of words in NOBUFF In Phases D/E, last entry in circular buffer
NOWI	Number of words in the input record extracted from the record indicator
NOWO	Number of words in the output record. Used in the input phase
NPCBF(57)	Punch buffer
NPTBF(55)	Print buffer
NPTBFX	Print buffer index
NQ	Flag extracted from the record indicator indicating the index for this instruction has been assigned to the Q register. Used in the output phase as a flag for an ADC sub-type.
NR	Two word instruction flag extracted from the record indicator in the input phase. Fixed length flag in the output phase.

NAME	USE
NRCD(5)	Buffer for output record. Used in the input phase.
NRC1	Address relocation holder
NRC2	Address relocation holder for second word of a two-word instruction
NRM	Program relocation holder
NS	Flag extracted from the record indicator indicating the instruction address is indexed
NSETX	Print buffer index
NSUMD	Total number of storages in IDVTAB
NSUMDS	Total number of storages in NDSTAB
NSTYPE	Flag set when first executable instruction of the object program is encountered
NT	Flag extracted from the record indicator indicating this instruction is a label
NTYPE	Holder for address type extracted from the record indicator in the input phase. Used for assembler instruction type in the output phase.
NSUMVS	Total number of storages in NFTAB
NW2	Holder for second binary word of a two-word instruction
NW2FL	Flag indicating a two-word instruction
NX	Index for INBUFF and NOBUFF used in BKDWN

NAME	USE
NXCON	Holder for hex or decimal output word used in SETPRT
NXPT	Pointer to index address in the symbol table.
IHEAD	Pointer to 'beginning' of circular buffer (always increases).
INOB	Size of NOBUFF, used to reference circular buffer in phases D/E
INC	Actual subscript in NOBUFF, calculated from INCT modulo INOB, by subroutine INDEX

5.1.5 Subroutine PHASEC

Subroutine PHASEC is the first subroutine entered in the FORTRAN Assembler. Symbol table location holders, other than common, are backgrounded to $7FFF_{16}$. The first input record is read here. Subsequent records are usually read in subroutine INOUT. They may be read in other routines if a forward scan is initiated at the bottom of the current input buffer.

If the input record is a data string or a format record the subroutine branches immediately to process the record. If the input record is a pseudo instruction record, subroutine BKDWN is called to determine the type of pseudo instruction.

Following are descriptions of the types of pseudo instructions encountered and their processing.

LABEL - Subroutine CHKWD is called to destroy any indexes saved in subroutine IXOPT since they will not be valid after a label is encountered. If this is a statement label, subroutine LABEL is called. If this is a variable or constant label and the first executable instruction has not been encountered, subroutine LABIN is called. If the first executable instruction has been encountered, subroutine LABEL is called.

BSS - Call subroutine BSS

ADC - If the address constant is an external and if this is the first encounter, the program counter is put in the symbol table. If this is not the first encounter, the output record is set up without recording the program counter.

If the address constant is not an external, it may be any one of five types.

Type 1 - An entry in a table of addresses set up for a computed GO TO. Address is relative to referenced location computed in 16 bit arithmetic.

Type 2 - A parameter in the calling sequence to a floating point subroutine when using relative addressing. By examining the run anywhere switch and the setting of the dummy argument identification bit in the symbol table, type 2 is broken down into four sub-types.

Parameter is not a dummy argument

Address is relative to referenced storage computed in 15 bit arithemtic with bit 15 set to zero.

Parameter is a dummy argument

Run anywhere program - address is relative to referenced storage computed in 15 bit arithmetic with bit 15 set to one.

Non-run anywhere program - If the parameter has been referenced, this address is the location of the previous reference with bit 15 set. If the parameter has not been referenced, this location will be stuffed with the computed address of the dummy argument.

Type 3 - Parameter in a calling sequence to a subroutine other than floating point. Address is relative to referenced location computed in 15 bit arithmetic with bit 15 set to one.

Types 4 & 5 are parameters in a calling sequence to a subroutine other than floating point or to a floating subroutine when absolute addressing is used.

Type 4 is the relocatable address of the referenced location. Type 5 is the absolute address of the referenced location.

The actual address is not computed in subroutine PHASEC bits NQ and NF in the indicator are used to flag the setting of bit 15 and the type of address to be used.

CONSTANT - Call subroutine CON

PARAMETER STORE - This is a STA command with a special type code that tells the assembler the A register contains a computed address to be stored in a dummy parameter address holder. The first holder is always in the symbol table entry for this dummy parameter. The dummy parameter table is searched for any other holder that may have been created to maximize the use of one-word relative, indirect addressing. A store Address record is built for each holder. The first Store Address record contains a pointer to the entry in the symbol table. Subsequent records contain a pointer to the entry in the dummy parameter table.

STATEMENT NO. - These are numbers generated consecutively by the FORTRAN Compiler for each FORTRAN source record. The output record applies to list output only.

STORAGE REFERENCE COMMAMD - Call subroutine CL12. On return from subroutine CL12, if the storage reference command is an unconditional jump, the symbol table is searched for variable and constant holders and dummy parameter address holders that are ready for output. Variable and constant holders are ready for output when they have been used as the operands in a store type command. Dummy parameter address holders are ready for output after the first reference. Holders are identified as ready for output in Subroutine CHOP. LABEL and BSS records are constructed for each holder to be output. The current program location is entered in the symbol table.

The dummy parameter table is also searched for any additional dummy parameter address holders to be output. A BSS record is constructed for each holder to be output. The current program location is entered in the dummy parameter table.

REGISTER, SHIFT and INTER-REGISTER COMMANDS - Since these three types of commands have absolute operands, the binary output word is completely constructed. The output record is flagged "ready for output".

If the command is an LRS 16 the assembler examines the next record to see if it is an indexed DVI. If it is, and if the index in the DVI record is not in the I register holder, subroutine QXLD is called to construct the necessary index load and store commands to be inserted in the output buffer prior to the Long Right Shift Command. The index in the DVI command is placed in the I register holder.

SKIP - Call Subroutine Skip

END - The symbol table and the dummy parameter table are searched for holders to be identified as described under Storage Reference Commands. A dummy ENA record is output to avoid a possible BSS as the last command in the program.

In the 16K version of 1700 Fortran, the current symbol table page is written and the page size is changed from 2400 to 960. Control passes to the assembler output phases D/E.

Subroutine BLDUP and INOUT are called to complete processing of the output record. Subroutine COUNT is called if the instruction to be output will occupy a storage location in the binary object program.

Records placed in the output buffer from subroutine PHASEC are flagged as "fixed length" or "ready for output". Bits NR and ND in the indication are used for this purpose. A fixed length instruction that is not ready for output is one with a forward storage reference that cannot be defined until after analysis in the output phase.

5.1.5.1 Flow Chart of Subroutine PHASEC

5.1.6 Subroutine BKDWN

The function of subroutine BKDWN is to extract and store the information in the pseudo instruction record. In the Assembly Input Phase, these are the records constructed in the previous FORTRAN Compiler pass. In the Assembly Output Phase, these are the records constructed in the Assembly Input phase.

See 5.1.4 for a detailed list of the input record breakdown.

5.1.6.1 Flow Chart of Subroutine BKDWN

5.1.7 Subroutine BLDUP

Subroutine BLDUP sets up the records that are output from the assembler input phase. See 5.1.4 for output record format.

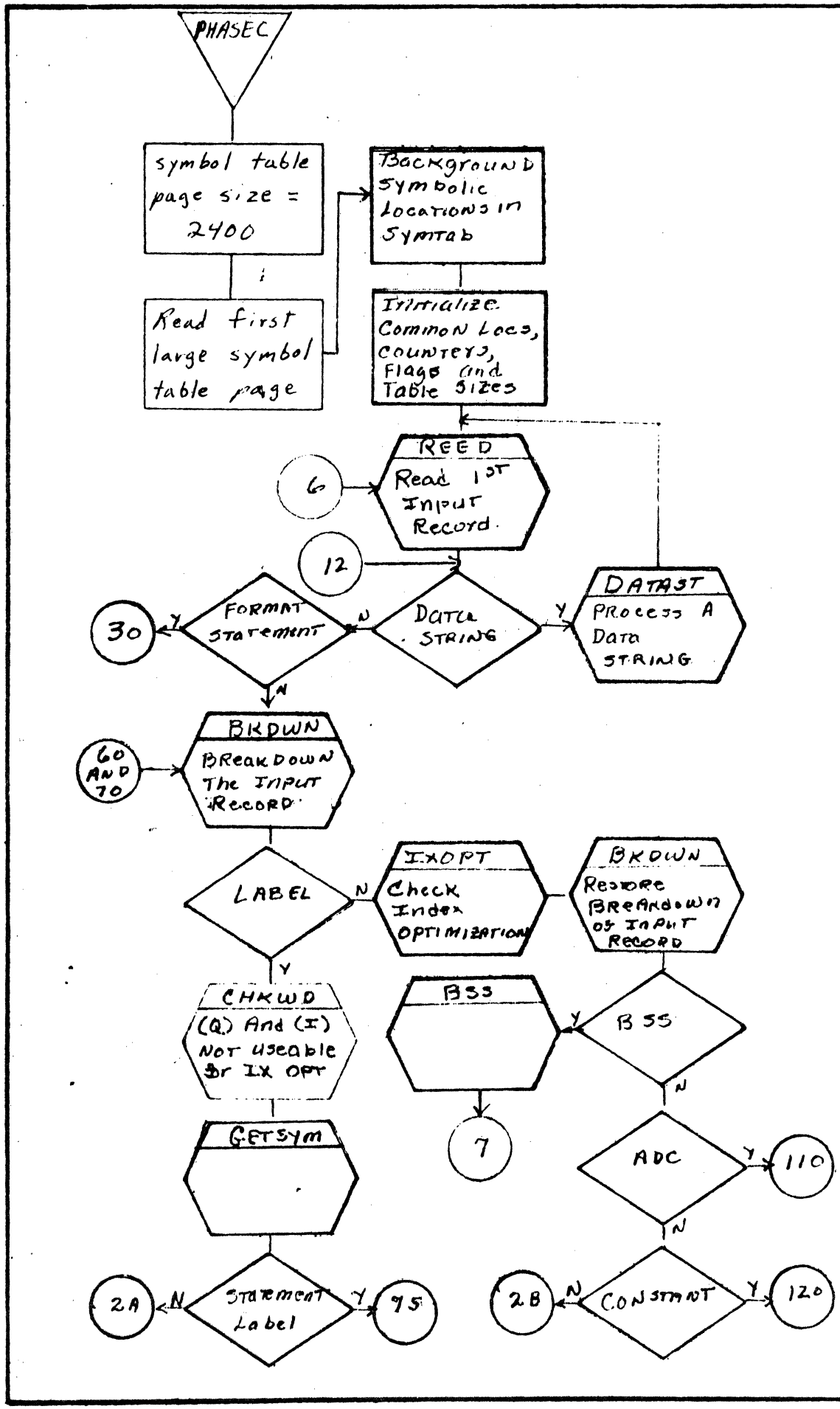
5.1.7.1 Flow Chart of Subroutine BLDUP

5.1.8 Subroutine BSS

Subroutine BSS constructs a BSS output record. The programs counter is incremented by the size of the BSS. The record is flagged ready for output.

A BSS size of minus one in the input record identifies the I register holder which may be eliminated in the assembler output phase. In this case, the output record is not flagged ready for output.

5.1.8.1 Flow Chart of Subroutine BSS

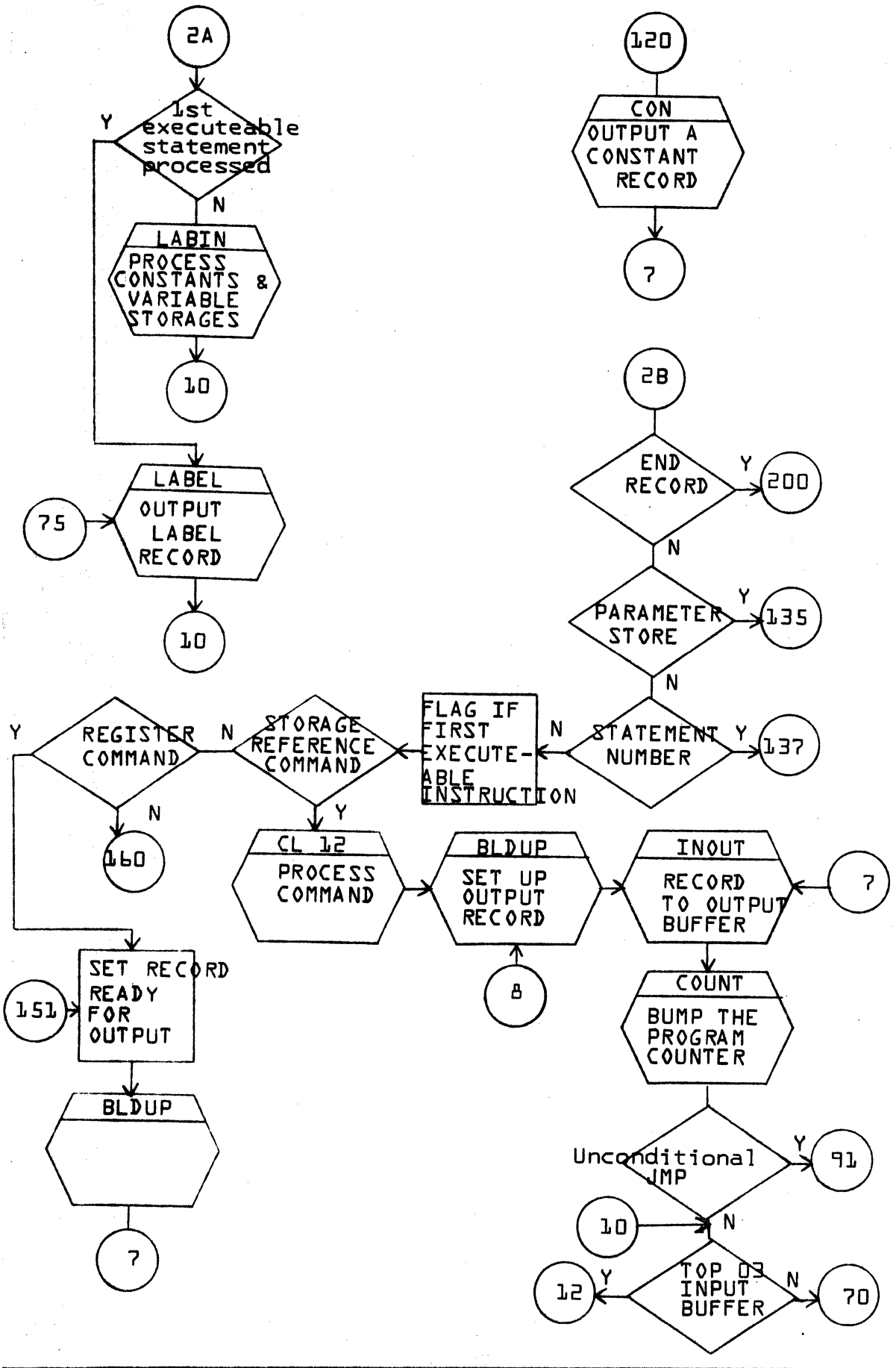


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	PHASEC	PAGE 1 OF 8	ISSUE DATE
NUMBER	5.1.5.1		

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR
PROJECT NAME
TASK NO.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

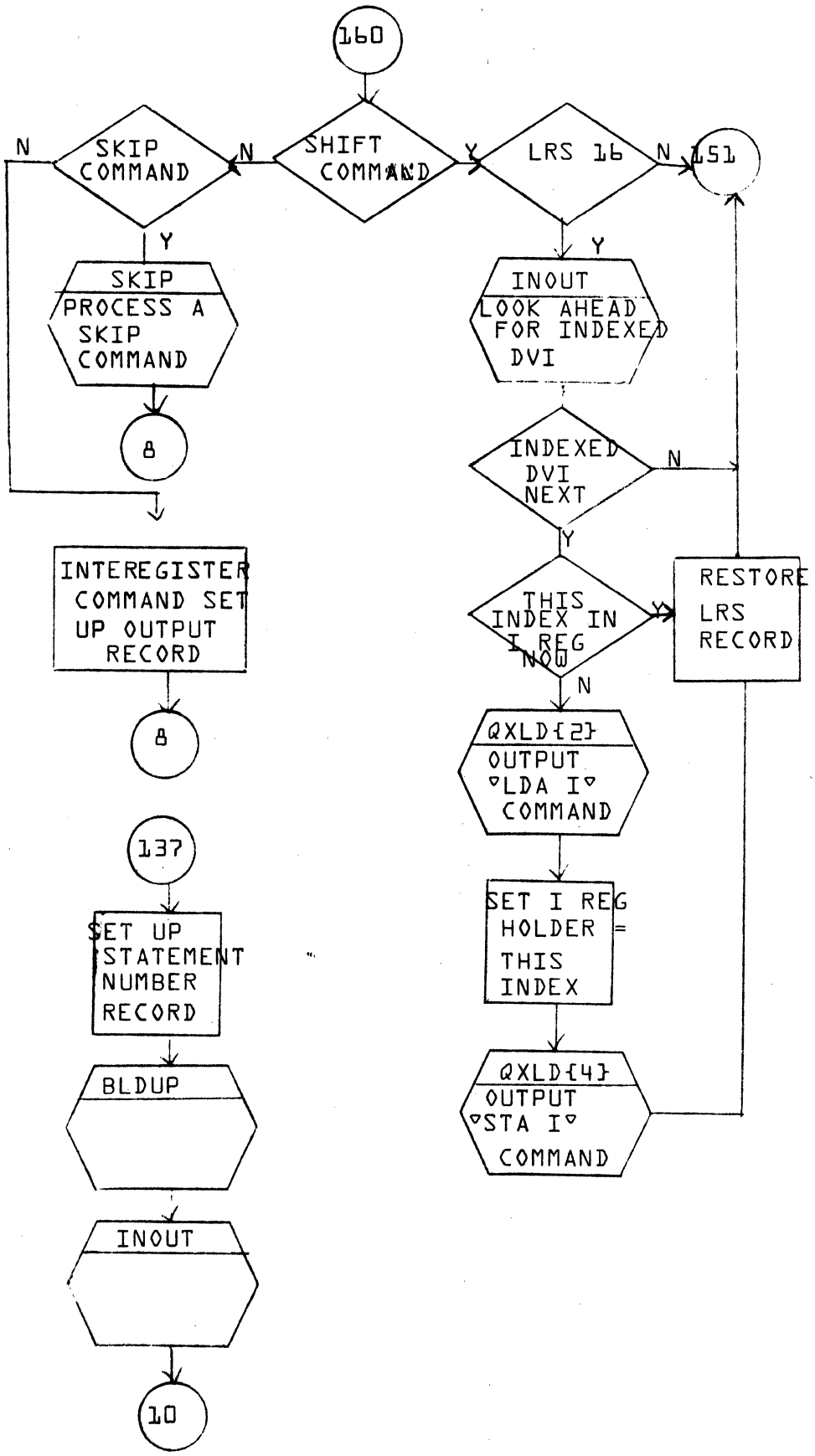
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: PHASE C
PAGE 2 OF 8
NUMBER: 5.1.5.1
ISSUE DATE:
DRAWN BY:
DATE:

AA1385 (FORMERLY CA127-1)

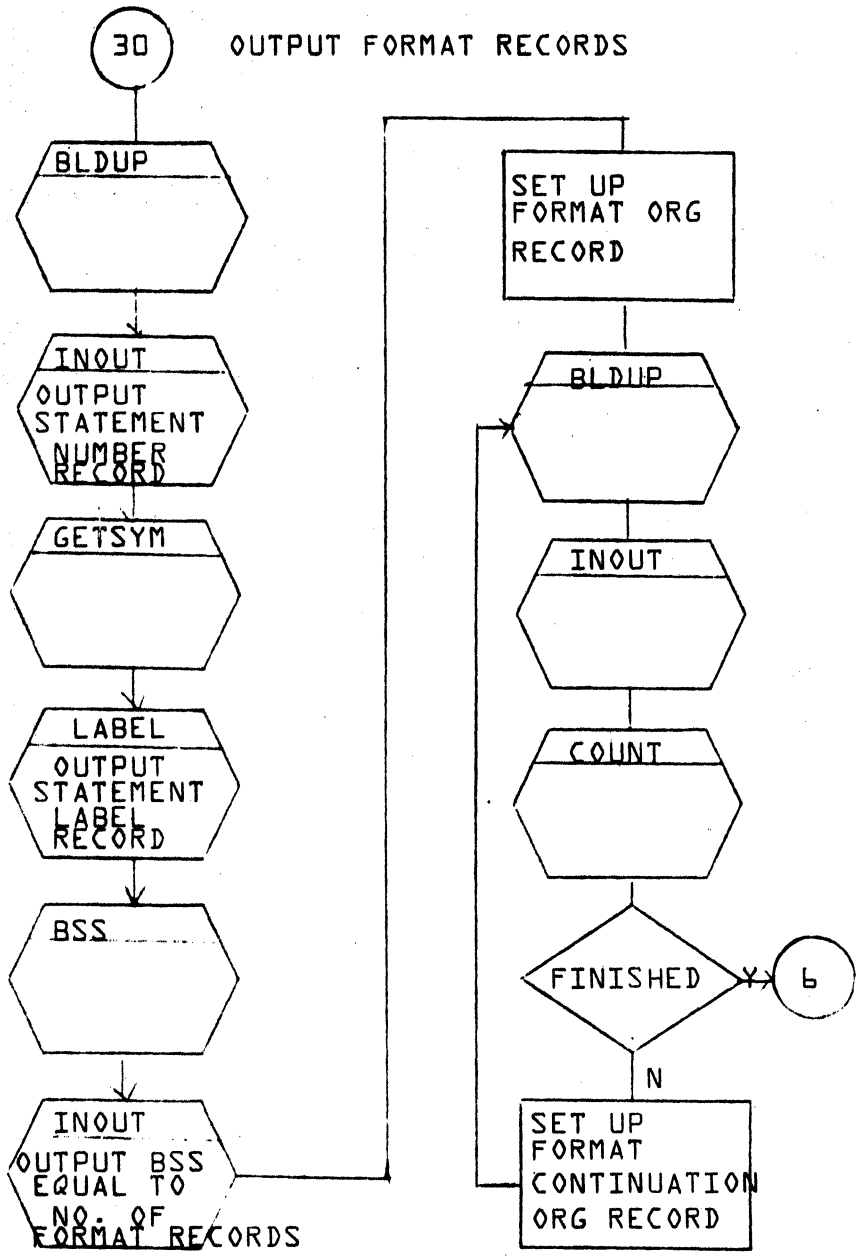


REV	APPROVED	DATE

PROJECT NO.			
PROJECT MGR.			
PROJECT NAME			
TASK NO.			
TASK NAME			
DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	Phase C	PAGE	3 OF 8
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: PHASE C
PAGE 4 OF 8
NUMBER
ISSUE DATE
DRAWN BY
DATE

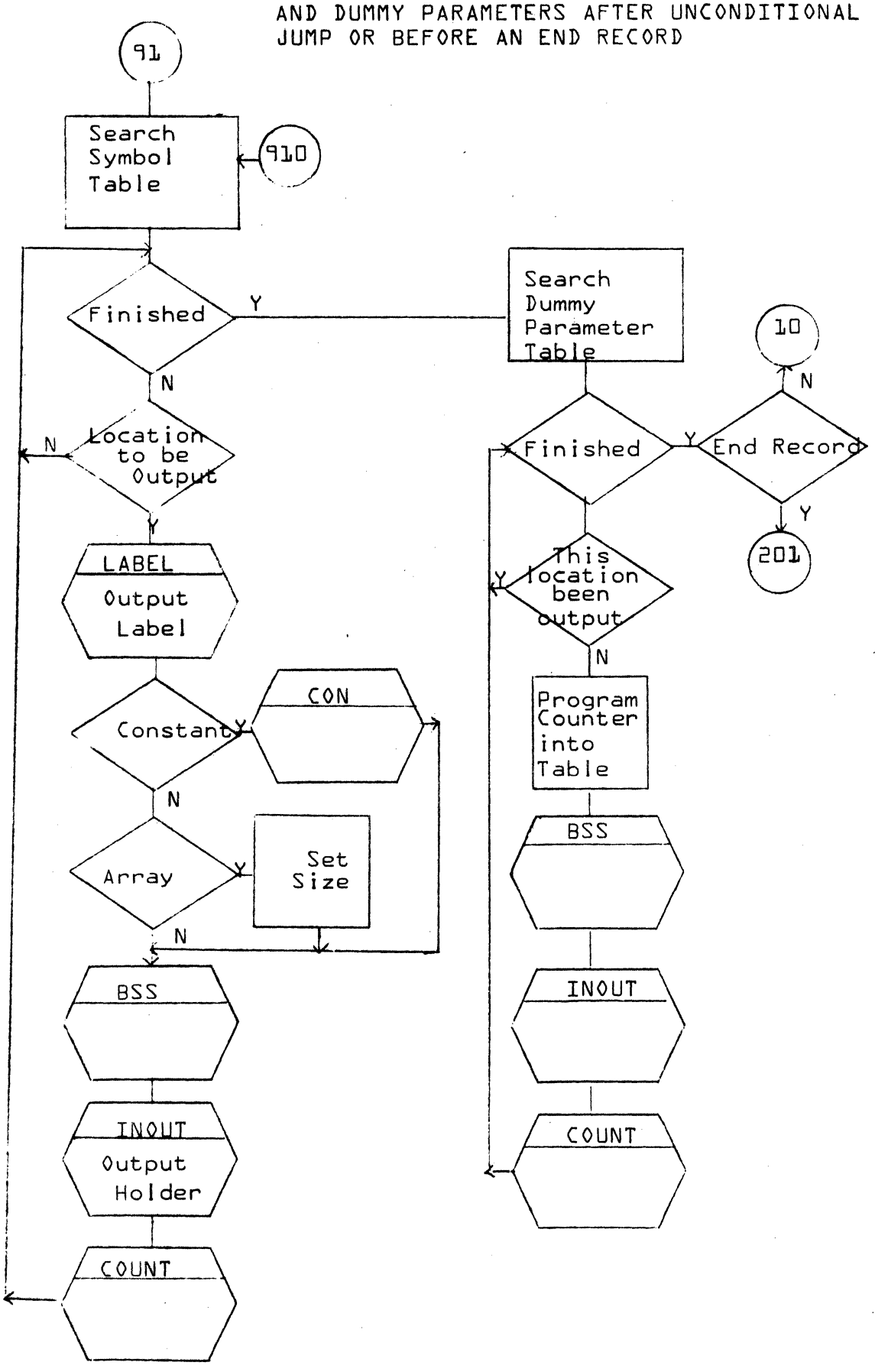
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

A B C D

OUTPUT HOLDERS FOR VARIABLES, CONSTANTS AND DUMMY PARAMETERS AFTER UNCONDITIONAL JUMP OR BEFORE AN END RECORD

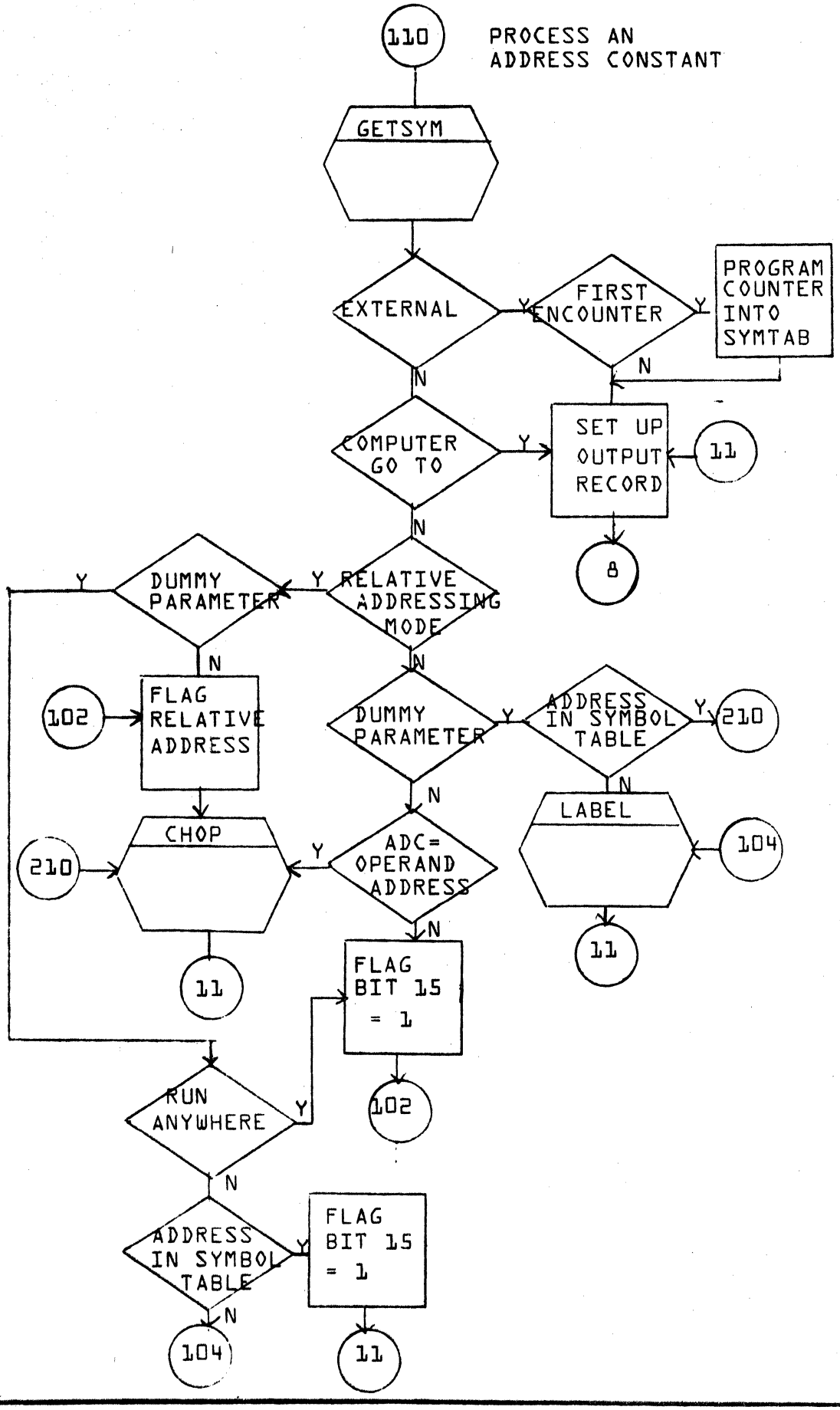


REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: Phase C
PAGE 5 OF 8
NUMBER: 5.1.5.1
ISSUE DATE
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER



REV	APPROVED	DATE			
			PROJECT NO.		
			PROJECT MGR.		
			PROJECT NAME		
			TASK NO.		
			TASK NAME		

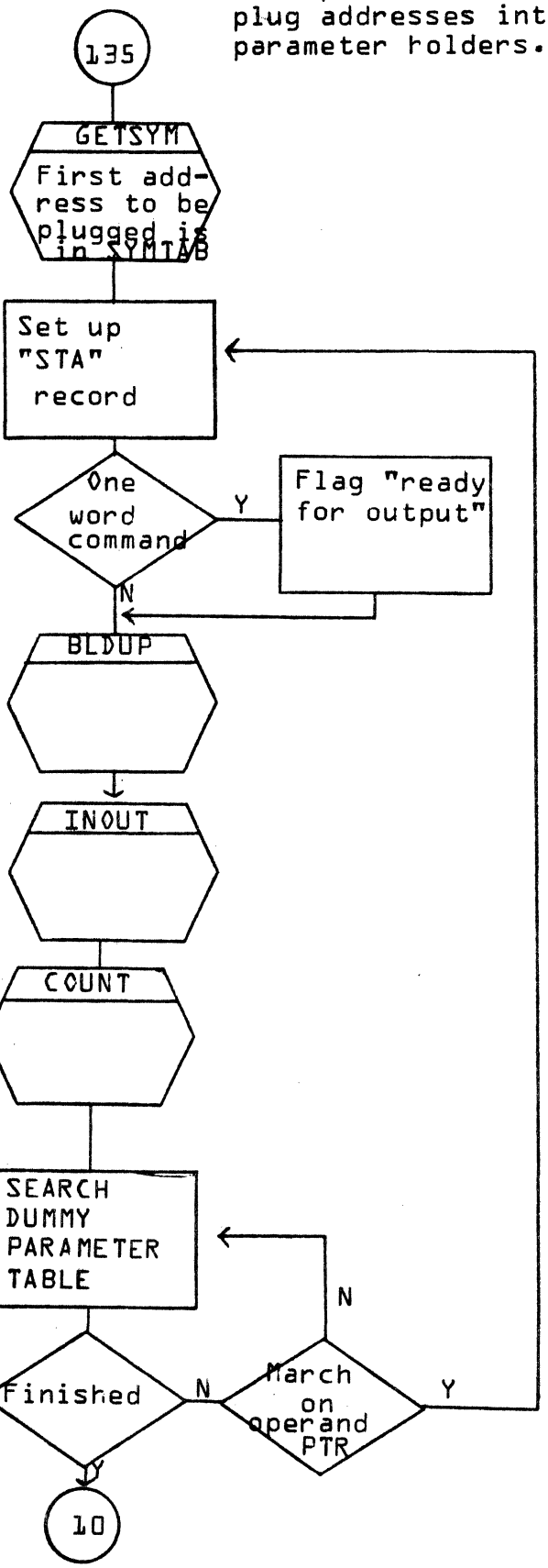
DOCUMENT CLASS IMS
MACH. TYPE 1700
DOCUMENT TITLE PHASE C
PAGE 6 OF 8
ISSUE DATE
NUMBER
DATE
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

Set up store commands to plug addresses into dummy parameter holders.

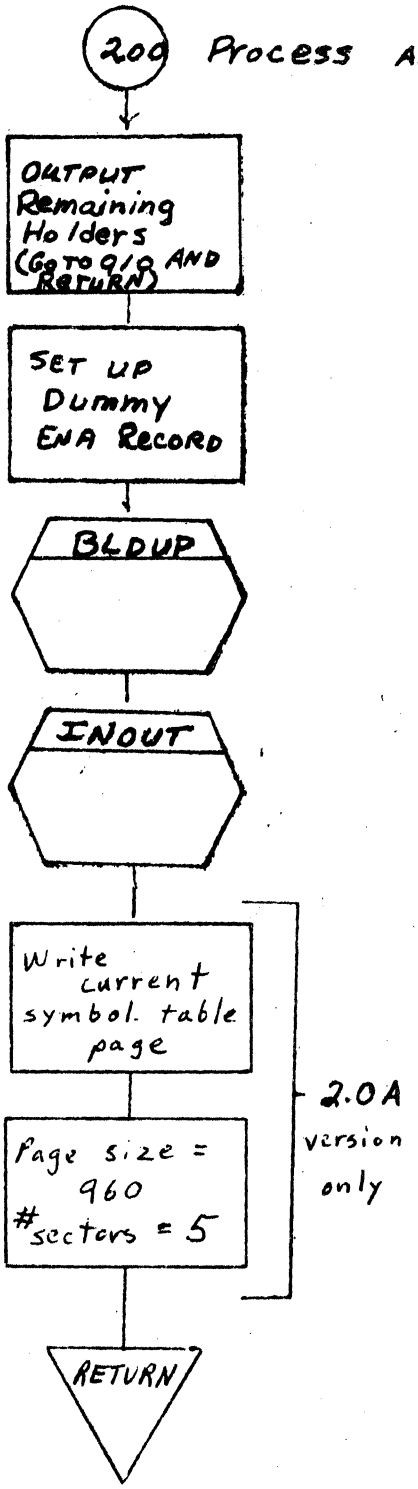


REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: PHASE 5
PAGE 7 OF 8
NUMBER: 5.1.5.1
ISSUE DATE
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER



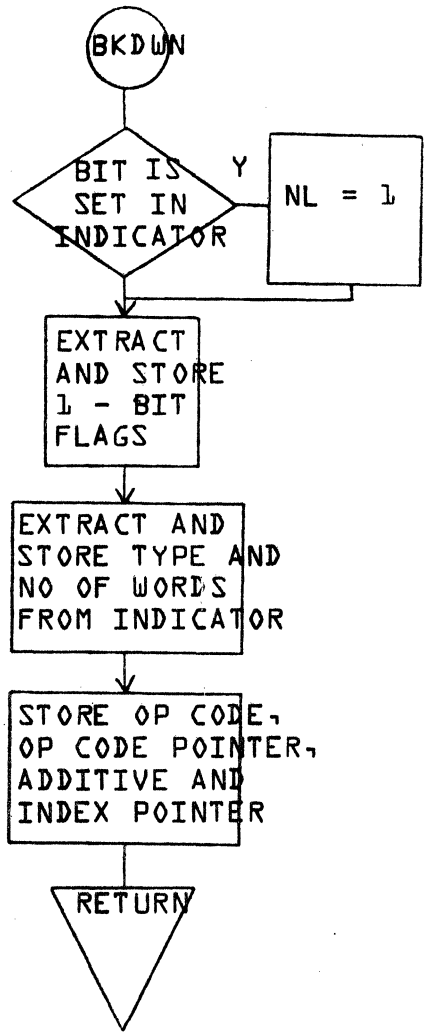
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS: IMS
MACH TYPE: 1700
DOCUMENT TITLE: PHASE C
PAGE 8 OF 8
NUMBER: 5.1.5.1
ISSUE DATE:
DRAWN BY: N TOLBOTT
DATE: 10/31/66

PROJECT NO.
PROJECT MGR
PROJECT NAME
TASK NO.
TASK NAME

REV	APPROVED	DATE

AA1365 (FORM) A127-11

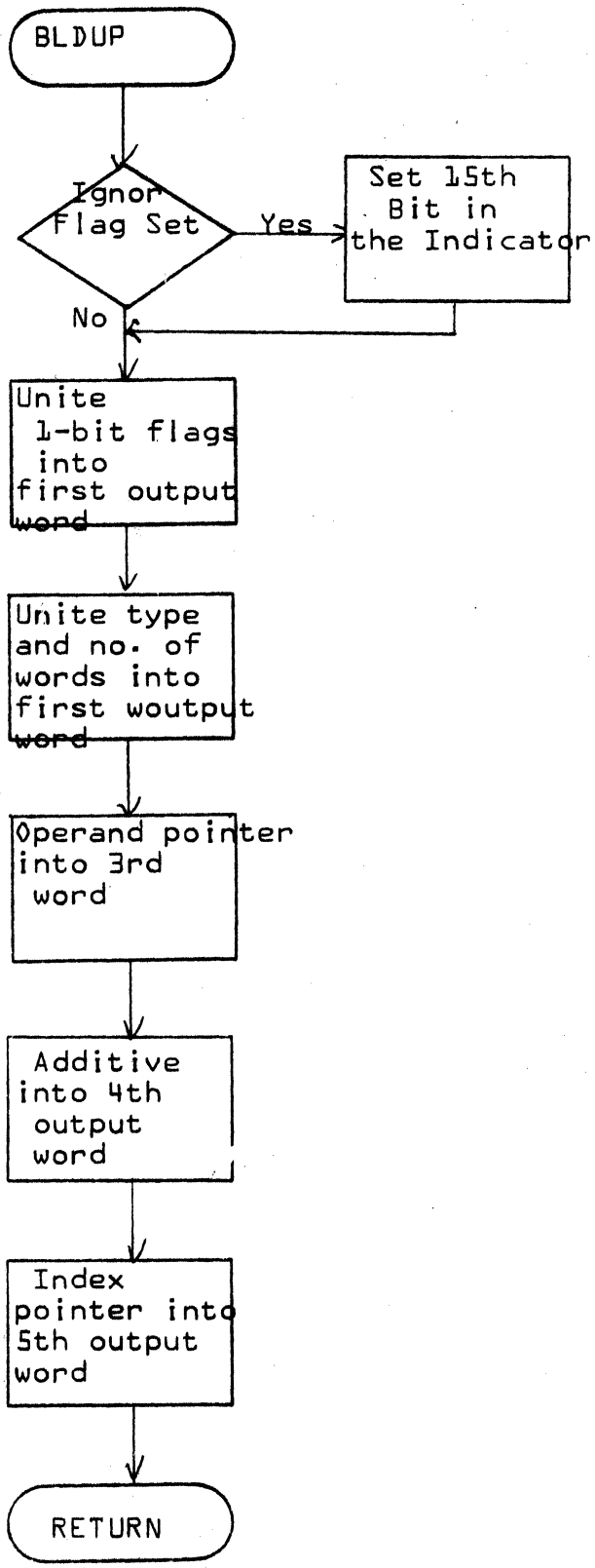


REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	Subroutine Bkdwn
PAGE	1 OF 1
NUMBER	5.1.6.1
ISSUE DATE	
DRAWN BY	
DATE	
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

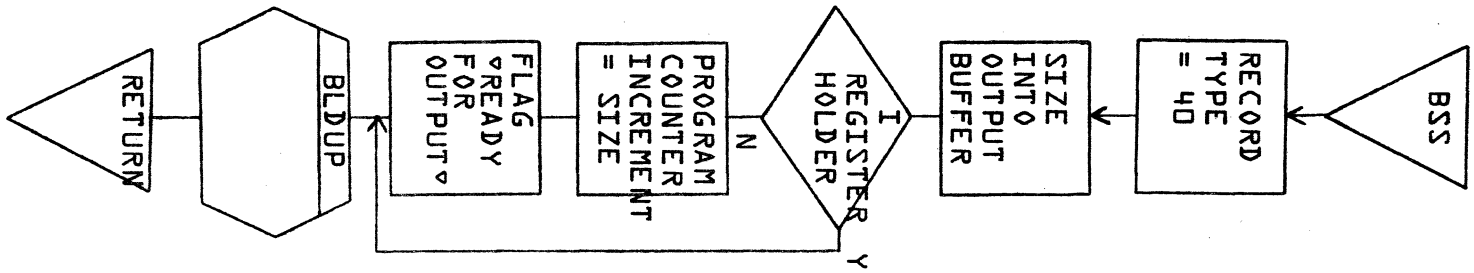
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: Subroutine BLDUP
PAGE 1 OF 1
NUMBER
ISSUE DATE
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV | APPROVED | DATE

PRINTED IN U.S.A.

CA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE Subroutine BSS	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER 5.1.8.1 ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

5.1.9 Subroutine CHKWD

Subroutine CHKWD is used in the input phase of the FORTRAN Assembler. It checks all pseudo instructions to determine what effect, if any, they will have on current or future index register assignments. CHKWD is called from subroutine PHASE 5 whenever a label record is encountered and from subroutine IXOPT in both its primary and secondary scans of the input buffer.

The following pseudo instruction records may affect index register assignment:

Label - All index holders are cleared

Return Jump - Any index holders containing addresses in the COMMON area are cleared. If an address currently to be assigned to an index register is in the COMMON area, the decision flag is set.

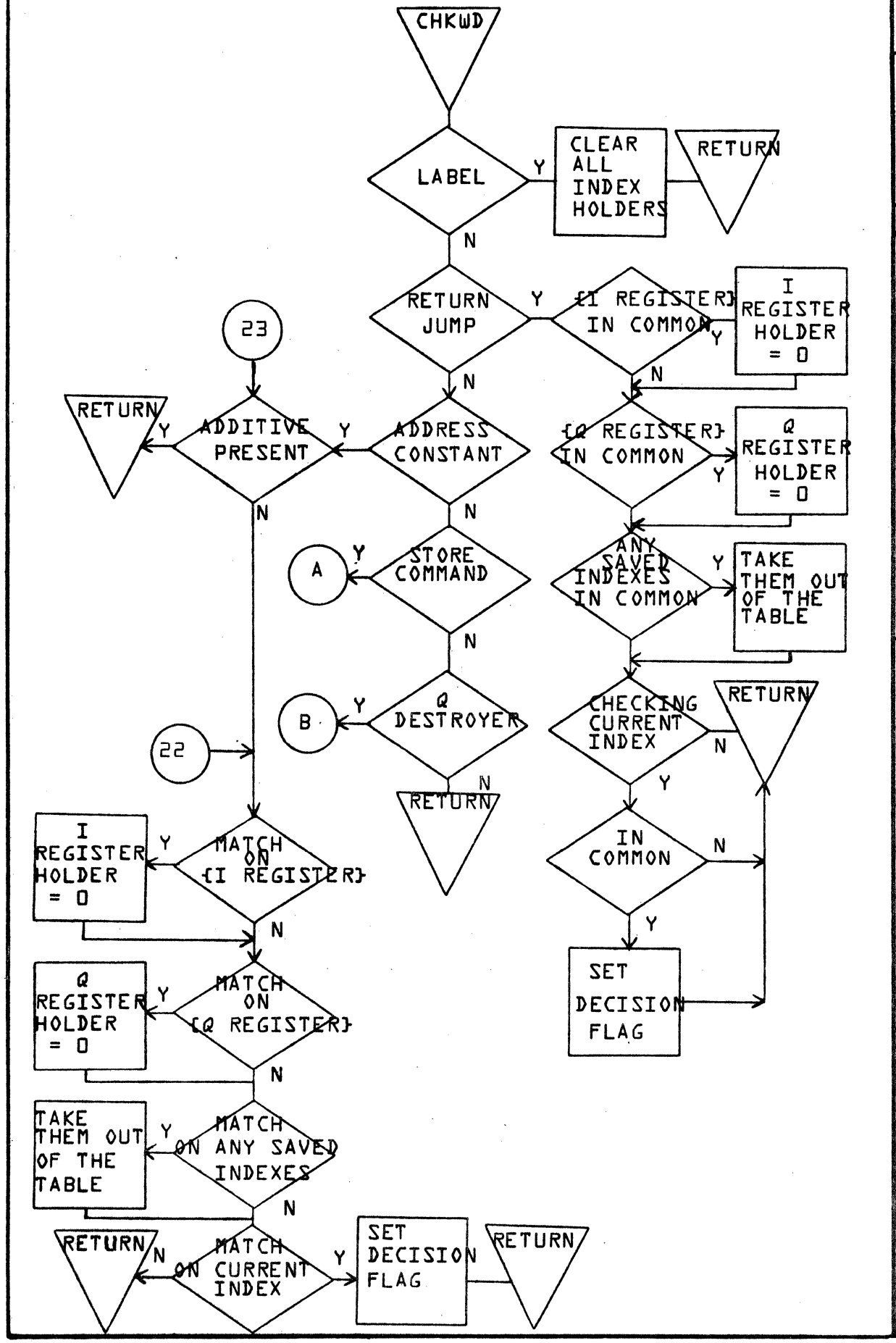
Address Constant - Any index holders containing this address constant are cleared. If an address currently to be assigned to an index register is this address constant, the decision flag is set.

Store Instruction - Any index holders containing the address of the operand of the store instruction are cleared. If the operand of the store instruction is in an equivalence chain, any index holders containing addresses in the same equivalence chain are cleared. If an address currently to be assigned to an index register is the same as the operand in the store instruction or is in the same equivalence chain, the decision flag is set.

Q Destroying Instruction - The Q register index holder is cleared and the occurrence flag is set.

The occurrence flag is set when an instruction affects index register assignment but is not decisive. The decision flag is set when an instruction is encountered that decides the index register assignment. These flags have meaning only if there is an address currently to be assigned to an index register.

5.1.9.1 Flow Chart of Subroutine CHKWD

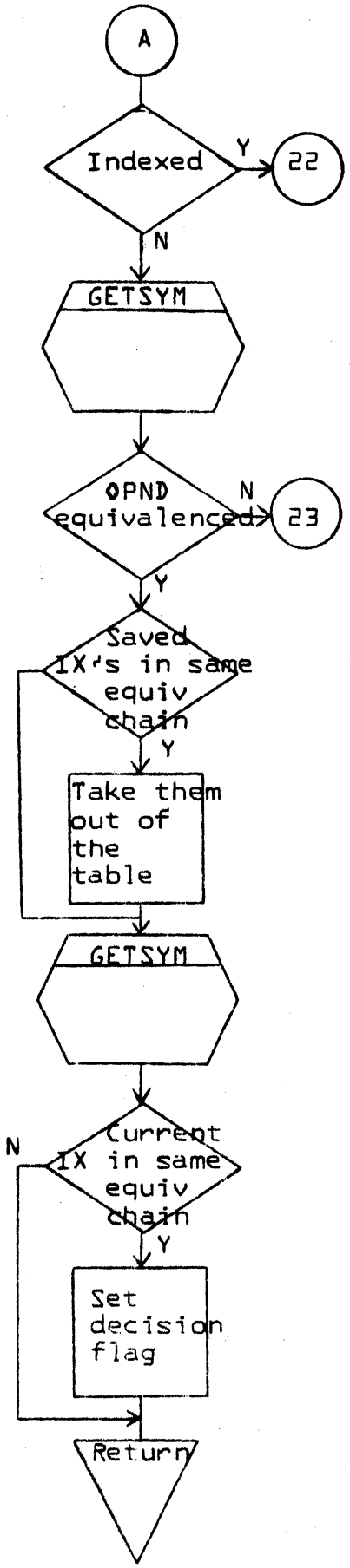


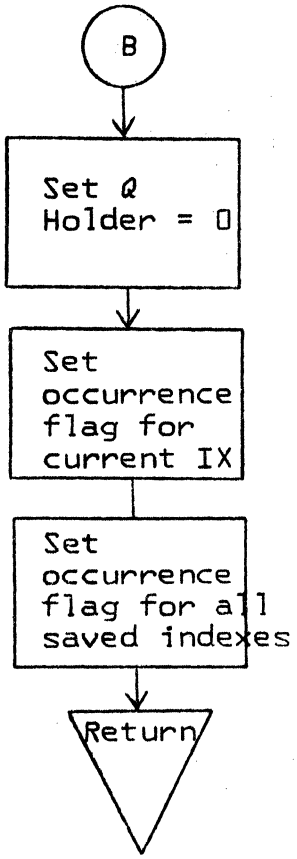
REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: Subroutine Chkwd
PAGE 1 OF 2
NUMBER: 5.1.9.1
ISSUE DATE:
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER





CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	Subroutine Chkwd	PAGE	2 OF 2
NUMBER	5.1.9.1	ISSUE DATE	
DRAWN BY		DATE	

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

AA1385 (FORMERLY CA127-1)

5.1.10 Subroutine CHOP

The function of subroutine CHOP is to flag all defined operands for which no holder has been generated, to build tables of address holders, and to determine the addressing mode of all storage reference commands for which no address mode has been set in the previous pass.

An operand to be located within the body of the object program is flagged as defined if it is used in a store instruction or if it is an address constant. Holders for defined operands are generated following the next encountered unconditional jump instruction or at the end of the program.

An operand whose address is not within the body of the object program that is, COMMON locations, externals, functions and dummy parameters, is flagged as defined on first encounter. Holders for the operand addresses are generated following the next encountered unconditional jump instruction or at the end of the program.

Subroutine CHOP builds two tables of address holders, IDVTAB and NDSTAB.

Entries are made in IDVTAB for instructions referencing operands within the body of the object program when neither the operand holder nor any previous table entry for this operand can be reached with one-word relative direct or indirect addressing mode. The address of the referencing instruction and the address of the operand holder are entered in the table. Addresses of instructions referencing operands located in COMMON and a pointer to the operand address are also entered in IDVTAB on first encounter of the operand reference if the use count is greater than one. Subsequent references to the same operand are entered in IDVTAB if no previous reference can be reached with one-word, relative, indirect Addressing Mode.

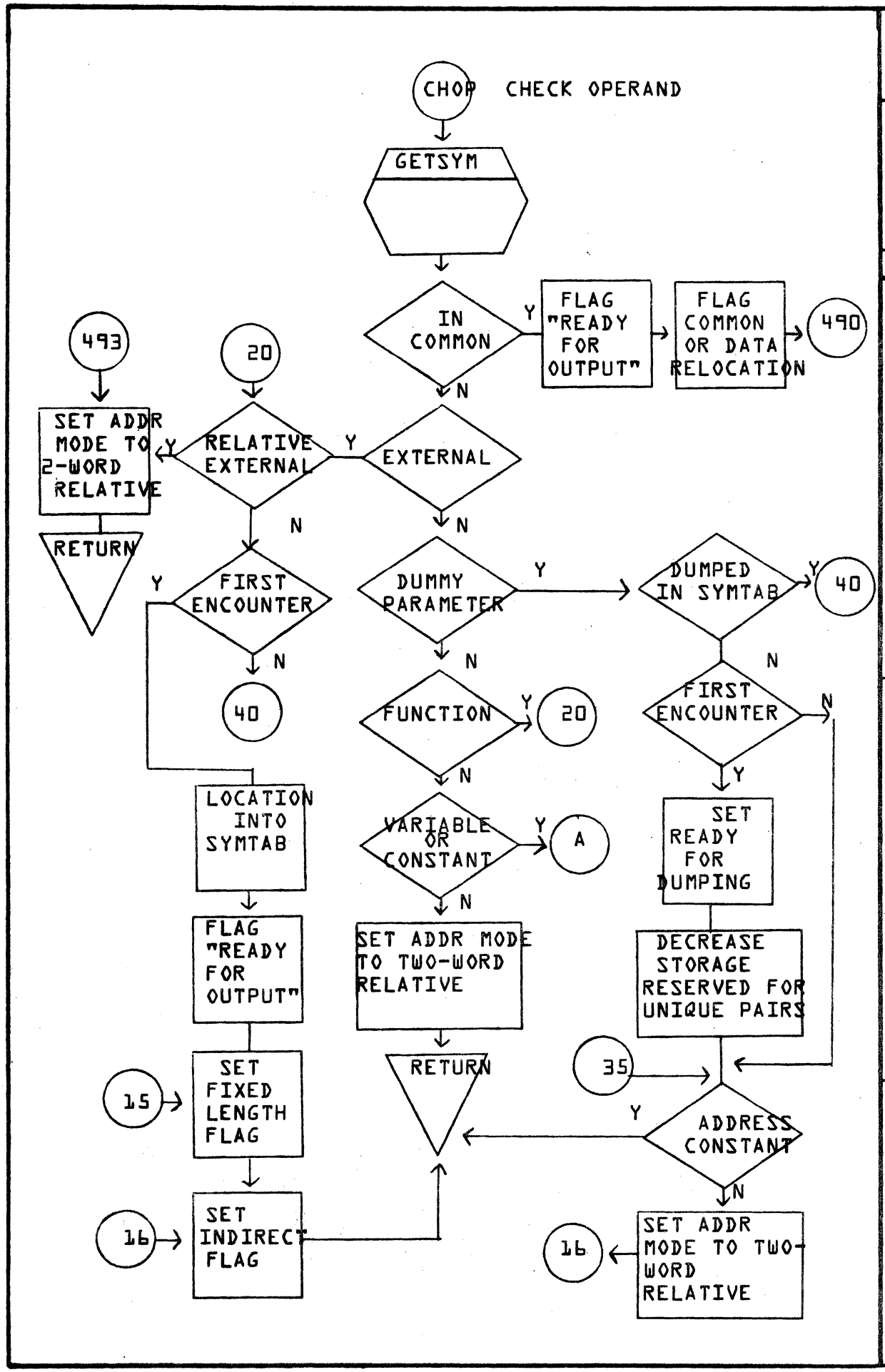
Entries are made in NDSTAB for instructions referencing operands outside the body of the object program (excepting COMMON locations) when neither the operand address holder defined in the symbol table nor any previous table entry for this operand holder can be reached with one-word, relative, indirect addressing mode. Entries consist of the address of the referencing instruction and a pointer to the operand address.

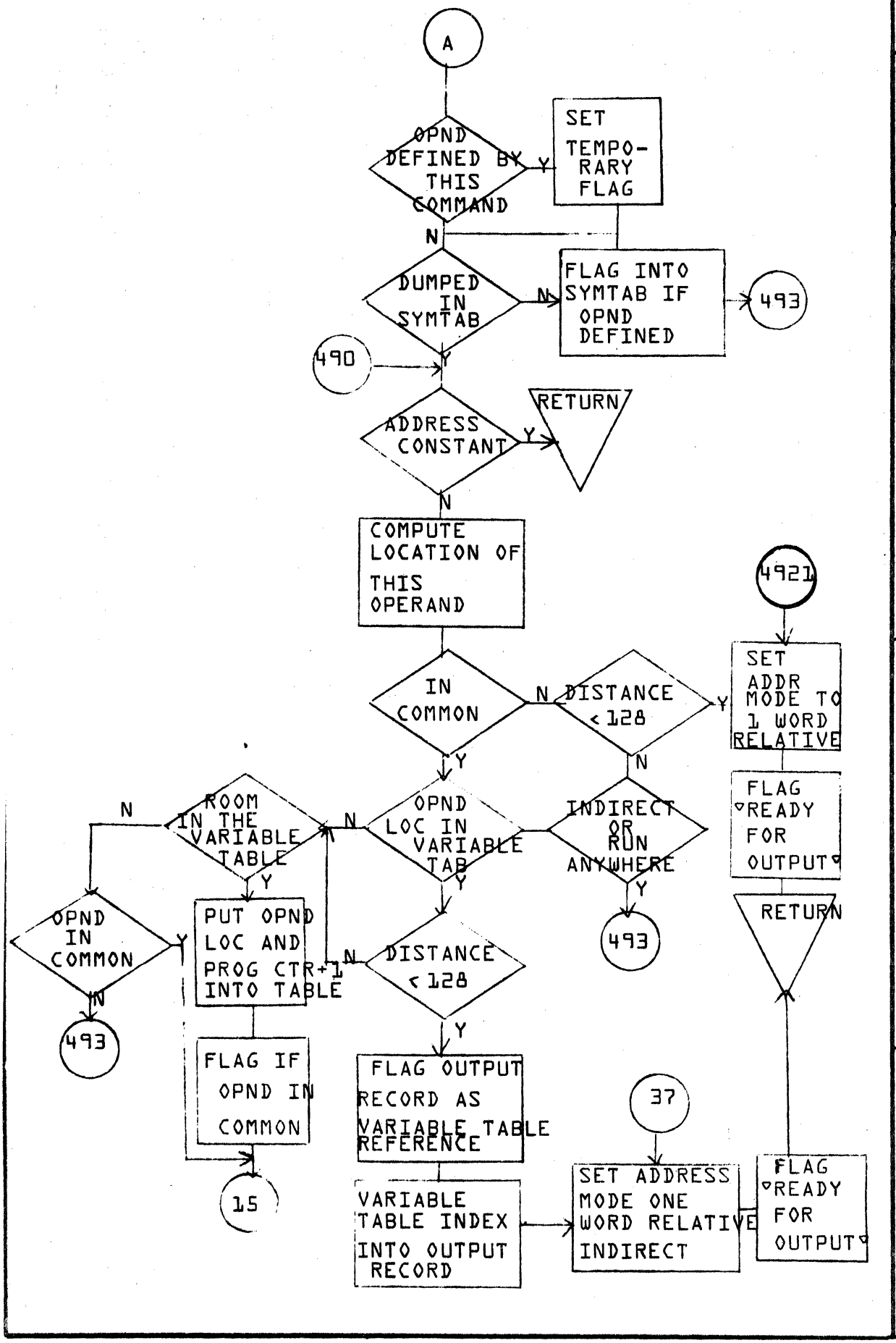
Addressing modes for instructions referencing operands located within the body of the object program are made one-word, relative, direct if the operand holder can be reached or one-word, relative, indirect if a table entry for this operand can be reached. If this instruction address is entered in IDVTAB, the addressing mode is two-word absolute. If the operand has not been defined, the addressing mode is two-word relative, direct.

Addressing modes for instructions referencing operand addresses outside the body of the object program are made one-word, relative, indirect if a previously defined address holder can be reached. If this instruction is entered in NDSTAB, the addressing mode is two-word, relative, indirect. In the case of an operand located in COMMON, the addressing mode is two-word absolute if the instruction is entered in IDVTAB.

5.1.10.1 Flow Chart of Subroutine CHOP.

5.1.10.2 Formats of IDVTAB and NDSTAB


CHOP CHECK OPERAND
GETSYM
IN COMMON
Y
FLAG "READY FOR OUTPUT"
FLAG COMMON OR DATA RELOCATION
490
N
EXTERNAL
Y
RELATIVE EXTERNAL
Y
SET ADDR MODE TO 2-WORD RELATIVE
RETURN
493
20
N
FIRST ENCOUNTER
Y
N
40
LOCATION INTO SYMTAB
FLAG "READY FOR OUTPUT"
SET FIXED LENGTH FLAG
15
SET INDIRECT FLAG
16
DUMMY PARAMETER
Y
DUMPED IN SYMTAB
Y
40
N
FUNCTION
Y
20
N
VARIABLE OR CONSTANT
Y
A
N
SET ADDR MODE TO TWO-WORD RELATIVE
RETURN
FIRST ENCOUNTER
N
Y
SET READY FOR DUMPING
DECREASE STORAGE RESERVED FOR UNIQUE PAIRS
35
ADDRESS CONSTANT
Y
N
SET ADDR MODE TO TWO-WORD RELATIVE
16
REV
APPROVED
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
DOCUMENT CLASS IMS
MACH. TYPE 1700
DOCUMENT TITLE SUBROUTINE CHOP
PAGE 1 OF 3
NUMBER 5.1.10.1
ISSUE DATE
DRAWN BY
DATE
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

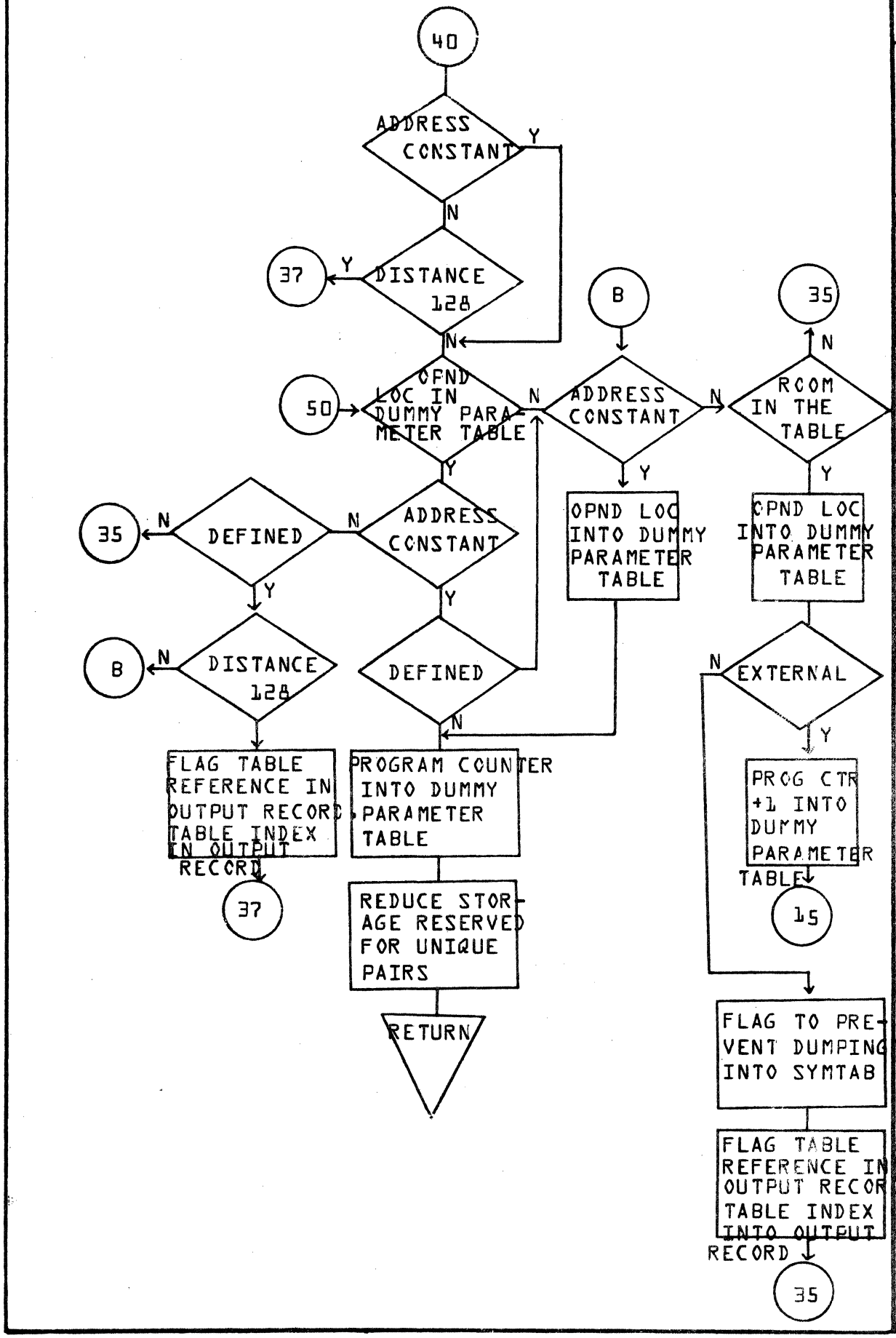
DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE CHOP
PAGE	2 OF 3
NUMBER	
ISSUE DATE	
DRAWN BY	
DATE	

CONTROL DATA CORPORATION

SOFTWARE DOCUMENT

SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

AA1385 (FORMERLY CA127-1)



REV | APPROVED | DATE

PROJECT NO. | PROJECT MGR. | PROJECT NAME | TASK NO. | TASK NAME

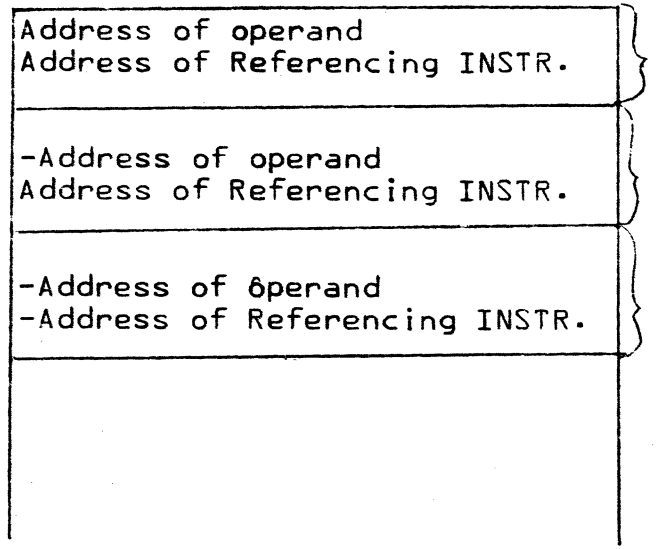
DOCUMENT CLASS: IMS | MACH. TYPE: 1700 | PAGE 3 OF 3

DOCUMENT TITLE: SUBROUTINE CHOP

NUMBER: 5.1.10.1 | ISSUE DATE | DATE | DRAWN BY

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

FORMAT OF IDVTAB

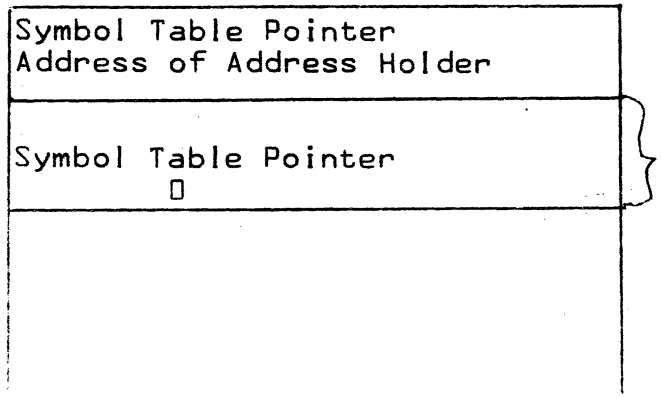


operand is within the body of the Program

operand is in un-labelled Common

operand is in Labelled Common

FORMAT OF NDSTAB



Points to Symbol Table entry for this Dummy parameter or External

Address Holder to be Generated after Next unconditional Jump.

CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE FORMAT OF IDVTAB		PROJECT MGR.			
FLOWCHART ☒	AND NDSTAB	PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

5.1.11 Subroutine CL12

Subroutine CL12 processes all storage reference instructions. If the operand of the instruction is an absolute address or a constant the binary output word is completely constructed and the output record is flagged ready for output. If the operand is a reference to a relocatable storage, the operation code field and the address mode field of the binary word are constructed. The address mode is determined either by the previous pass or by a call to Subroutine CHOP.

Special type codes are assigned to compiler generated instructions to save, restore and hold the I register for use in Subroutine ADMAX in the output phase of the Assembler.

5.1.11.1 Flow Chart of Subroutine CL12

5.1.12 Subroutine CON

Subroutine CON generates holders for constants as directed either by the previous pass or by Subroutine LABIN and CHOP in the Assembly input phase.

5.1.12.1 Flow Chart of Subroutine CON

5.1.13 Subroutine DATAST

Subroutine DATAST processes a string of data. Holders for data defined within the body of the object program are generated at the beginning of the object program by subroutine LABIN. Locations of data defined in labeled COMMON are assigned by the previous compiler pass.

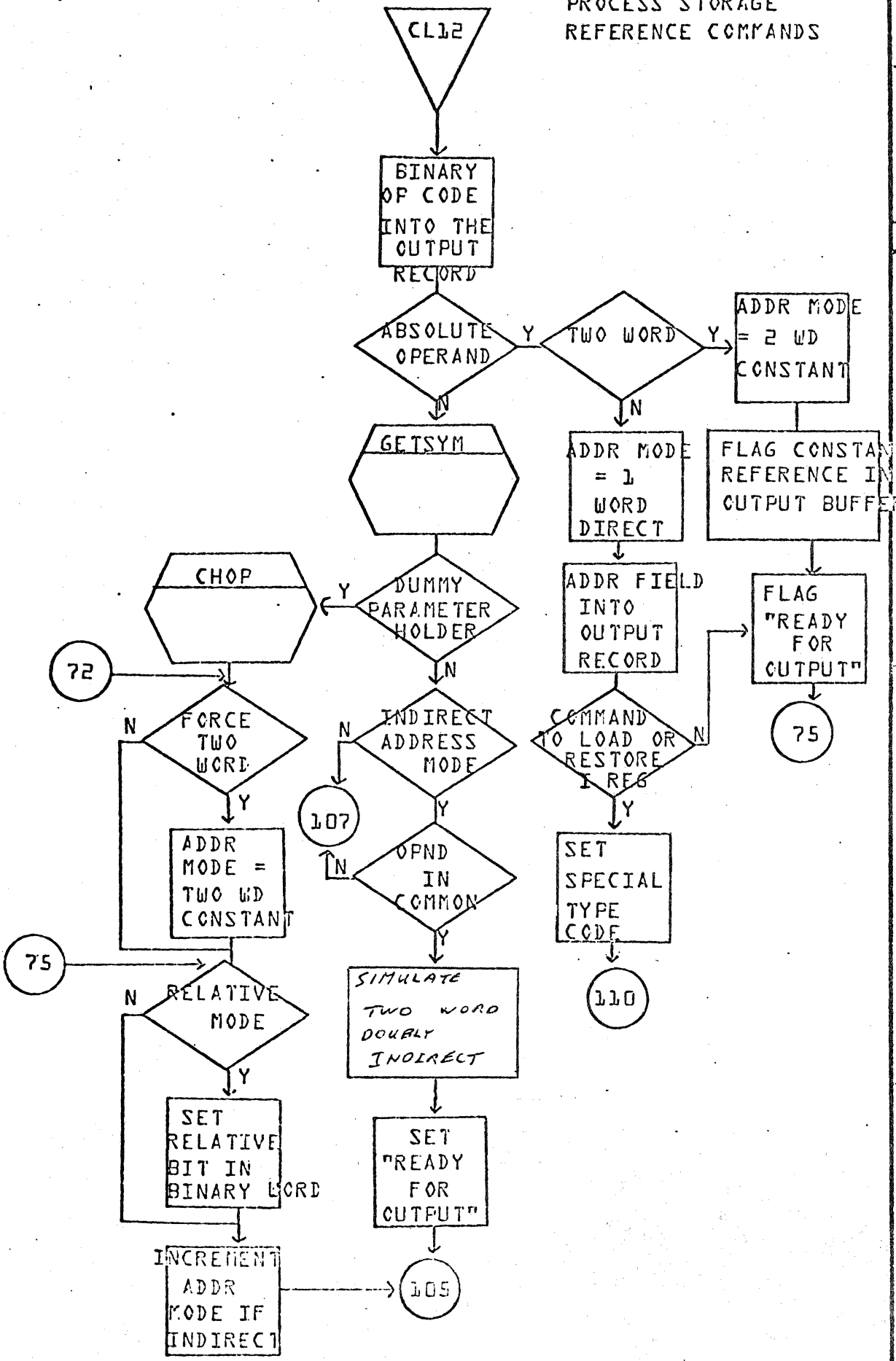
An ORG record is constructed for each datum containing a pointer to the location of the array, the datum and address additive. ORG records are typed as program ORG or data ORG according to their location. A data word that is in a string of consecutively located data is typed as a program continuation ORG or a data continuation ORG according to the location of the array. ORG records are flagged ready for output.

5.1.13.1 Flow Chart of Subroutine DATAST

5.1.14 Subroutine INOUT

Subroutine INOUT is the connecting subroutine between the input phase and the output phase of the FORTRAN Assembly pass. It reads records into the input buffer and writes records into the output buffer until the output buffer is full or an END record is encountered.

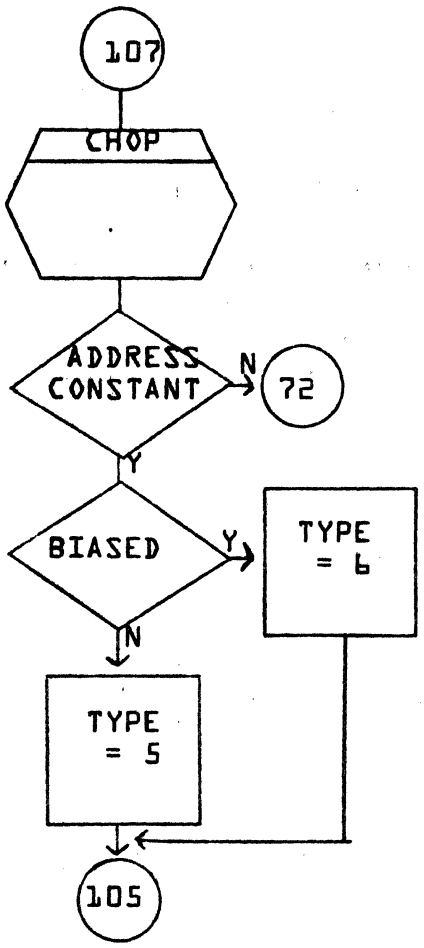
PROCESS STORAGE REFERENCE COMMANDS

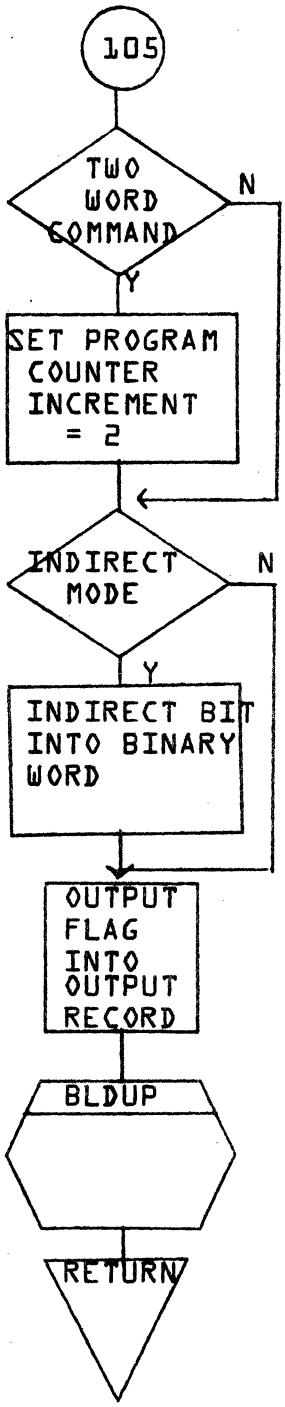


REV	APPROVED	DATE			
DOCUMENT CLASS	IMS	MACH. TYPE	1700	PROJECT NO.	
DOCUMENT TITLE	SUBROUTINE CL12		PAGE 1 OF 2	PROJECT MGR.	
NUMBER	5.1.11.1	ISSUE DATE		PROJECT NAME	
DRAWN BY		DATE		TASK NO.	
				TASK NAME	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

AA1385 (FORMERLY CA127-1)



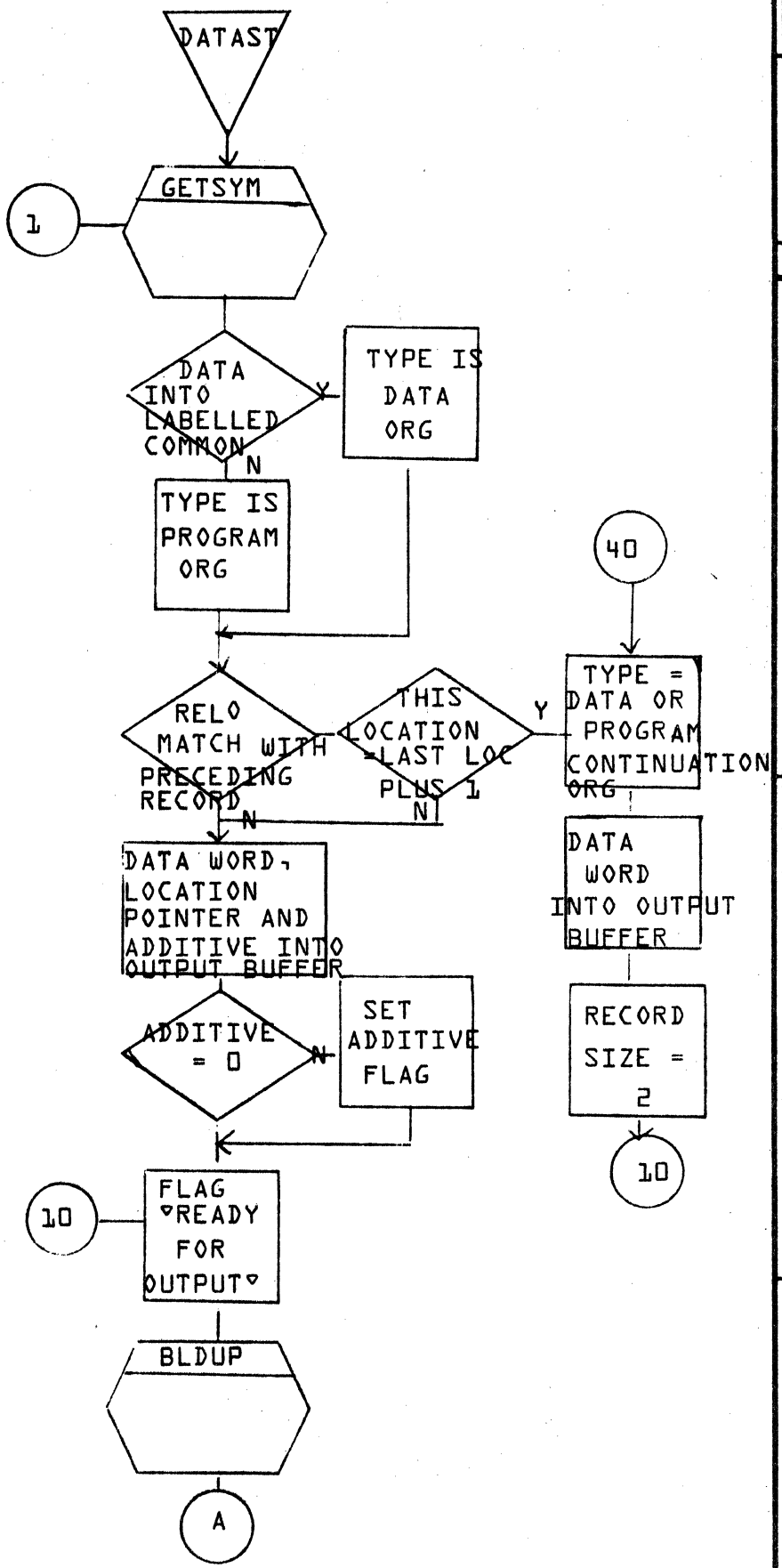


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE CL12		PAGE 2 OF 2
NUMBER	5.1.11.1	ISSUE DATE	
DRAWN BY		DATE	

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME



REV	APPROVED	DATE

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE DATAST	PAGE	1 OF 2
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

PROJECT NO.		TASK NO.	
PROJECT MGR.		TASK NAME	
PROJECT NAME			

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

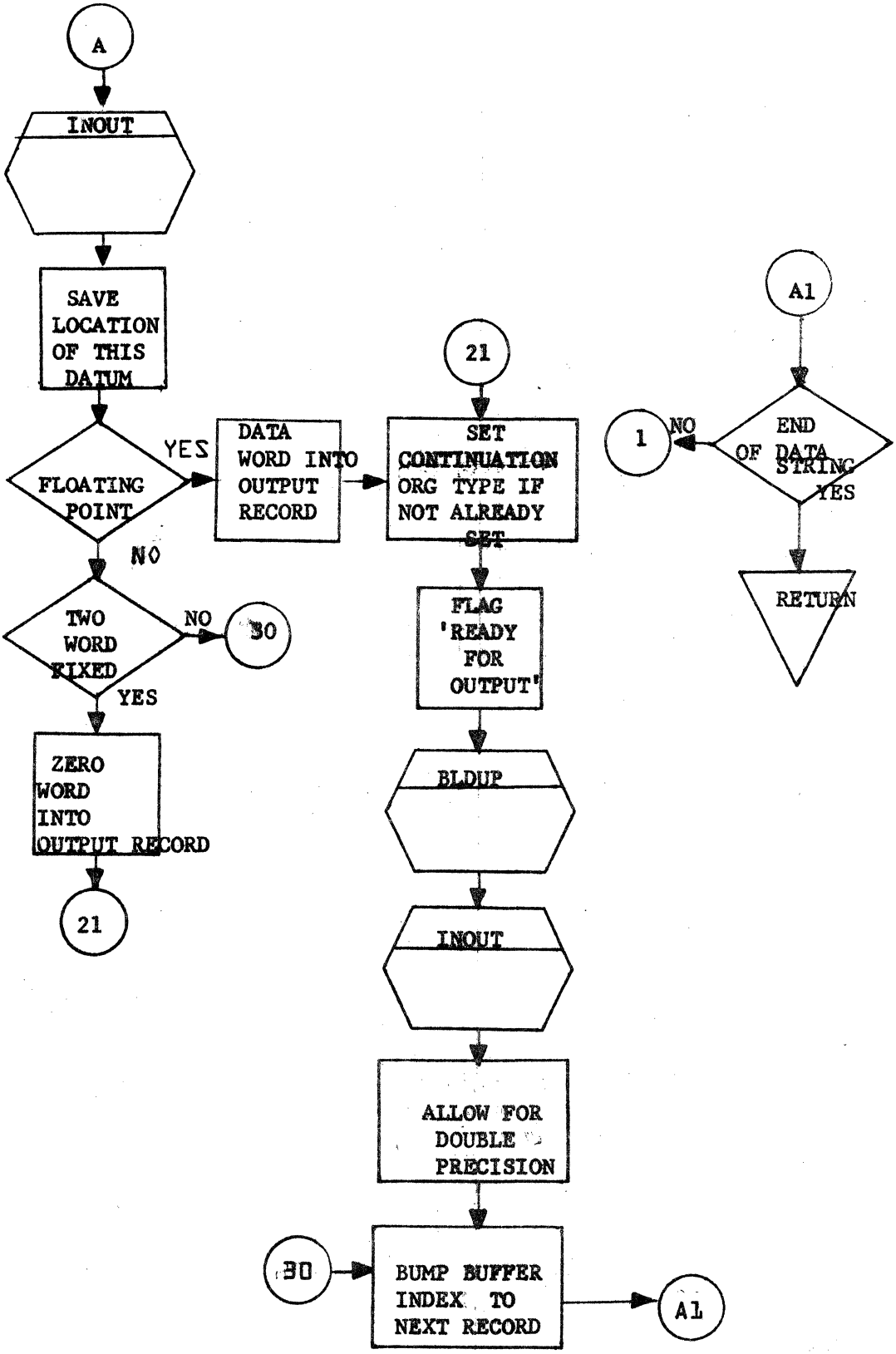
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

AA1385 (FORMERLY CA127-1)

A B C D

CONTROL DATA

LA JOLLA RESOURCE CENTER



TITLE	SUBROUTINE DATAST		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 2 OF 2

CSD 203 ANALYSIS

When the output buffer is full or an END record is encountered, program control passes to the Assembly output phase through a call to Subroutine AMOUT. On return from Subroutine AMOUT program control is passed back to the assembly input phase unless an unprocessed END record is present in the output buffer. In this case, program control is returned to the output phase.

Subroutine INOUT also reads forward in the input buffer or re-fills the input buffer if necessary, to check for an indexed divide instruction following a long right shift instruction.

5.1.14.1 Flow Chart of Subroutine INOUT

5.1.15 Subroutine IXOPT

The function of Subroutine IXOPT is to assign index registers to indexed instructions. It is called from subroutine PHASE 5 for all pseudo instruction records except labels.

If the instruction being processed is not indexed, Subroutine CHKWD is called to determine its possible effect on future indexing. If the instruction is a store address with no additive present, the operand pointer is saved in the A register holder. Otherwise, the A register holder is cleared.

If the instruction being processed is indexed, any of the five following conditions will result in index register assignment without further analysis of the input buffer.

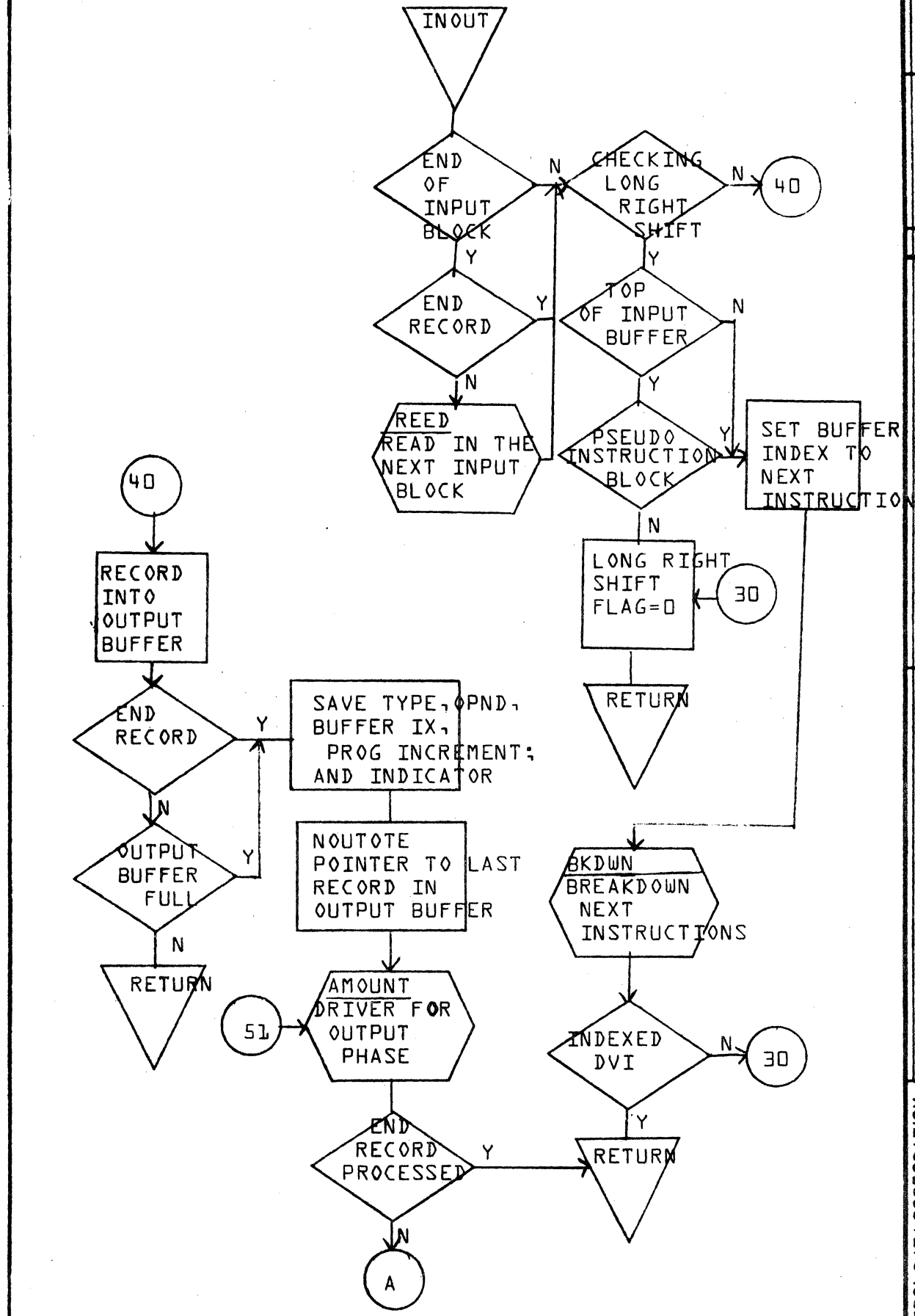
1. This index has been assigned to the Q register by the previous compiler pass.
2. This instruction is an indexed DVI. The index is assigned to the I register.
3. This index is currently in the Q register holder. It is assigned to the Q register.
4. This index is currently in the I register holder. It is assigned to the I register.

No indexing instructions are generated in the above four cases.

5. Subroutine CHKWD sets the decision flag. This index is assigned to the Q register. An LDQ instruction is generated unless the index is in the A register holder. In this case a TRA Q instruction is generated.

If none of the above five conditions is true, the input buffer is scanned forward until enough information is obtained to make the assignment.

The forward scan stops and the index register under consideration for assignment is assigned to the Q register when one of the



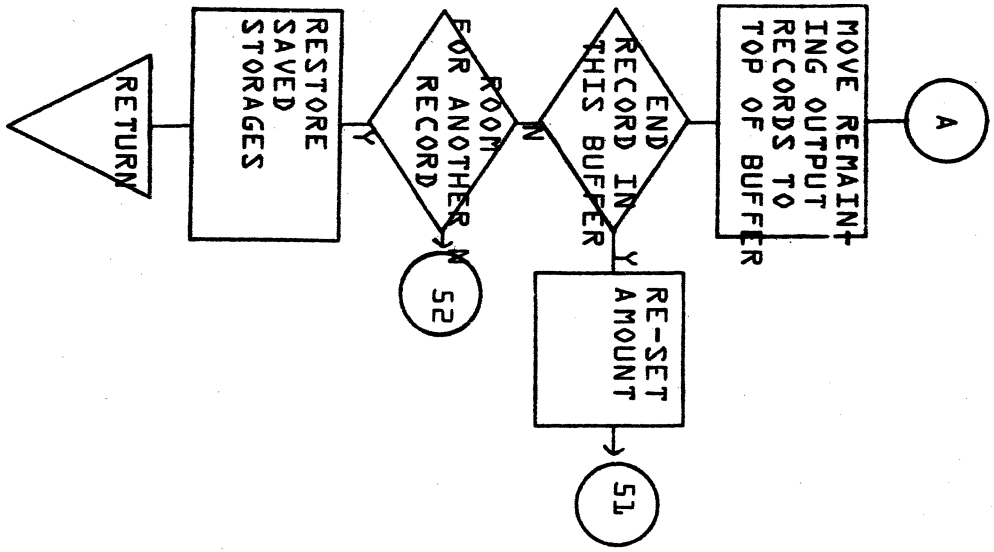
REV	APPROVED	DATE

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE INOUT	PAGE 1 OF 2	
		ISSUE DATE	
NUMBER		DATE	
DRAWN BY			

PROJECT NO.		PROJECT MGR.	
PROJECT NAME		TASK NO.	
TASK NAME			

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBROUTINE INOUT	PROJECT MGR.			
	PAGE 2 OF 2	PROJECT NAME			
	NUMBER 5.1.14.1 ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

following conditions is true.

1. The decision flag is set in Subroutine CHKWD.
2. The same register as that in the I register holder appears.
3. The same register as that in the Q register holder appears. A STQ I instruction is generated before the index load instruction.
4. A different index appears twice. An index is counted as appearing twice only if its occurrence flag has been set since its last appearance.
5. A DVI instruction with no index or with an index different from the one under consideration is encountered.
6. A label record is encountered.
7. The scan reaches the end of the input buffer with no decision.

An LDQ instruction is generated except when the index is in the A register holder. In this case a TRA Q instruction is generated.

The forward scan stops and the index under consideration for assignment is assigned to the I register when one of the following conditions is true.

1. The index under consideration appears again. It is counted as appearing again only if its occurrence flag has been set.
2. A DVI is encountered with an index the same as the one under consideration for assignment.

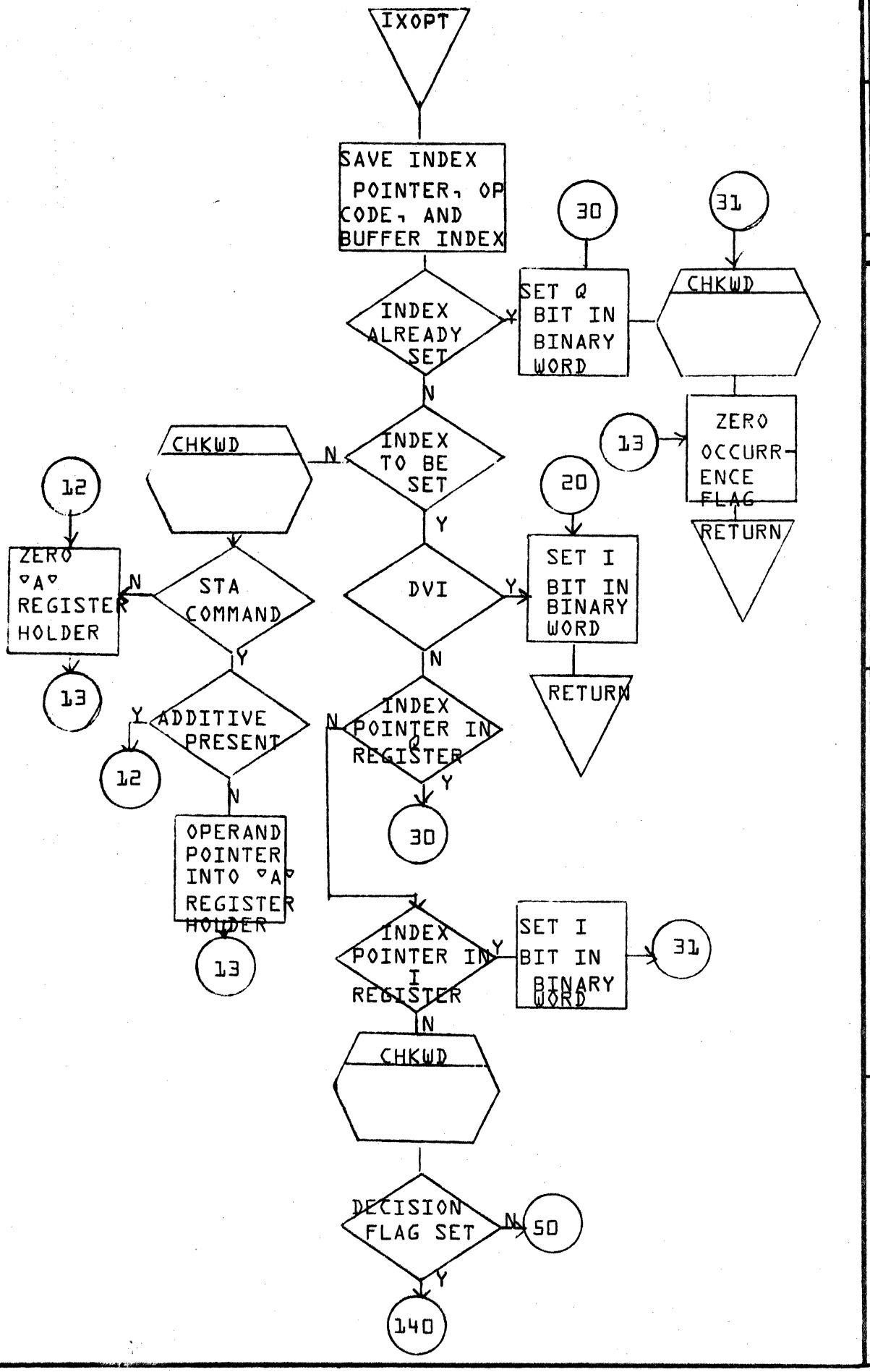
An STA I instruction is generated if the index under consideration is in the A register holder. LDA, STA I instructions are generated if the instruction being processed is an LDA. Otherwise, the instructions generated are LDQ, STQ I.

The occurrence flag is set for the index under consideration if a Q destroying instruction is encountered in the forward scan of the input buffer. The occurrence flags for the index under consideration and all saved indexes are set when an index appears that has not previously been encountered either in the buffer or in the Q or I register holders.

5.1.15.1 Flow Chart of Subroutine IXOPT

5.1.16 Subroutine LABEL

Subroutine LABEL constructs output records for statement and variable labels. The program counter is stored in the symbol table entry for the label being processed. The record is flagged ready for output.

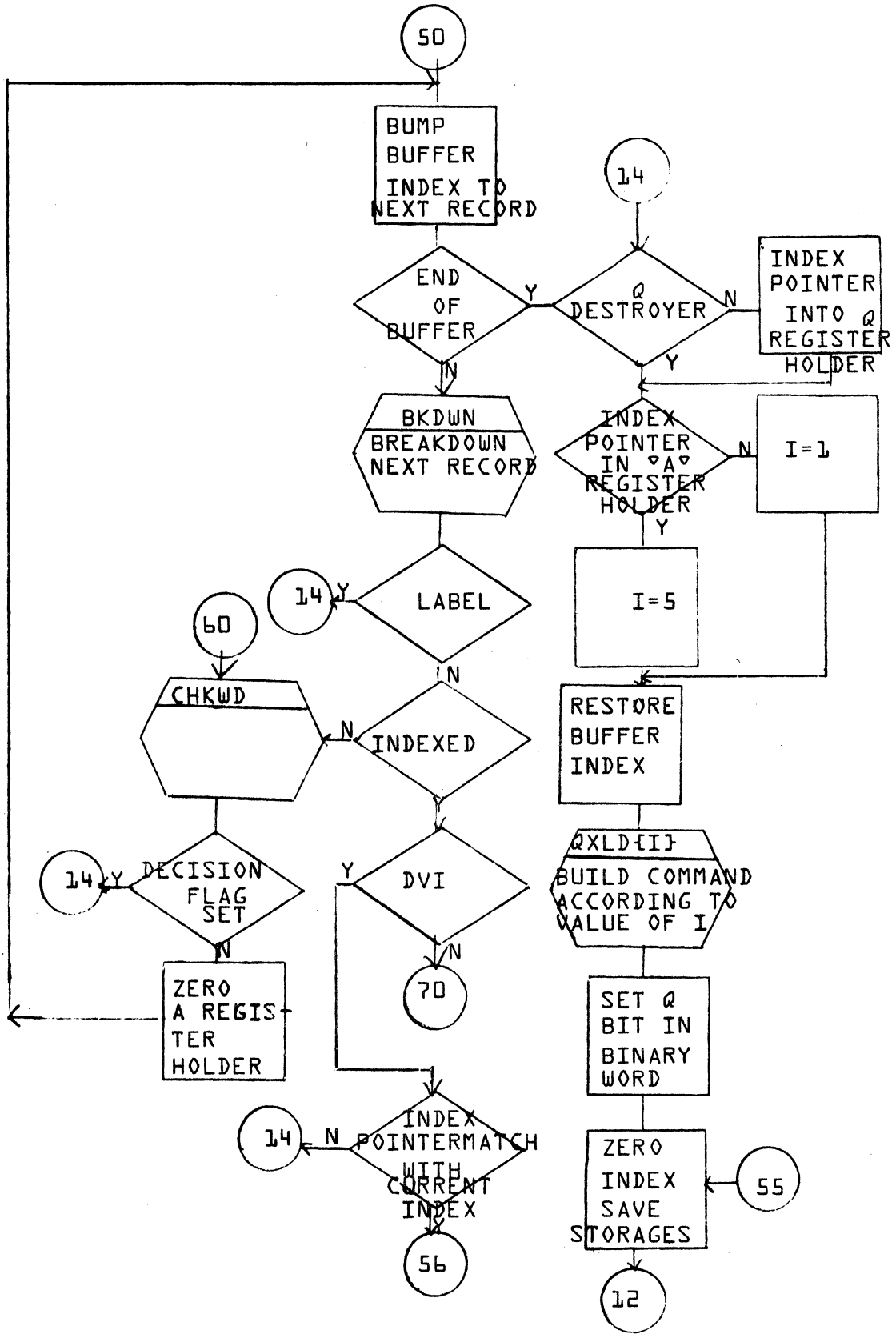


REV	APPROVED	DATE	
PROJECT NO.			
PROJECT MGR.			
PROJECT NAME			
TASK NO.			
TASK NAME			
DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE IXOPT	PAGE	1 OF 4
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

AA1385 (FORMERLY CA127-1)

A B C D

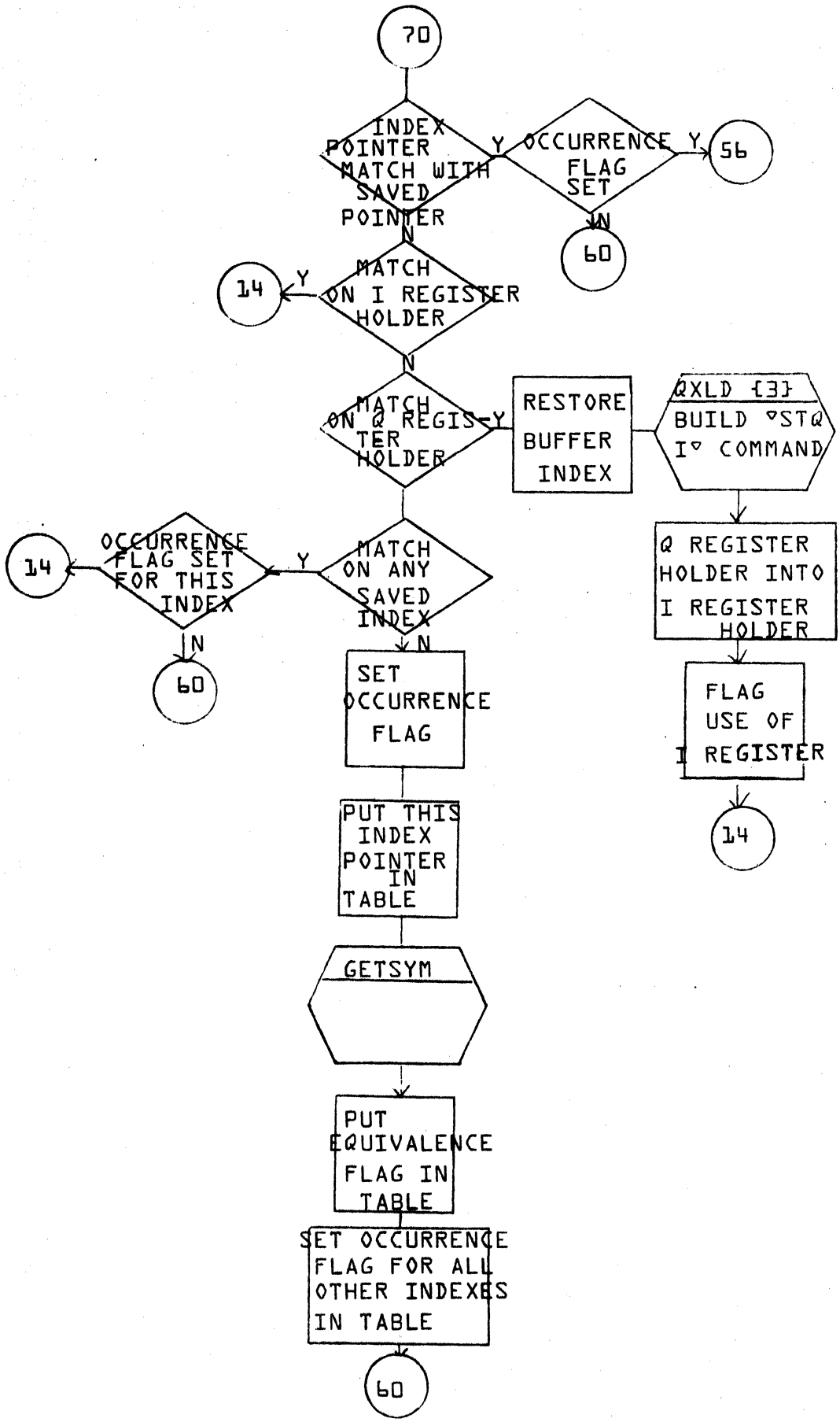


REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	
DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE IXOPT
PAGE	2 OF 4
NUMBER	
ISSUE DATE	
DRAWN BY	
DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐



REV	APPROVED	DATE

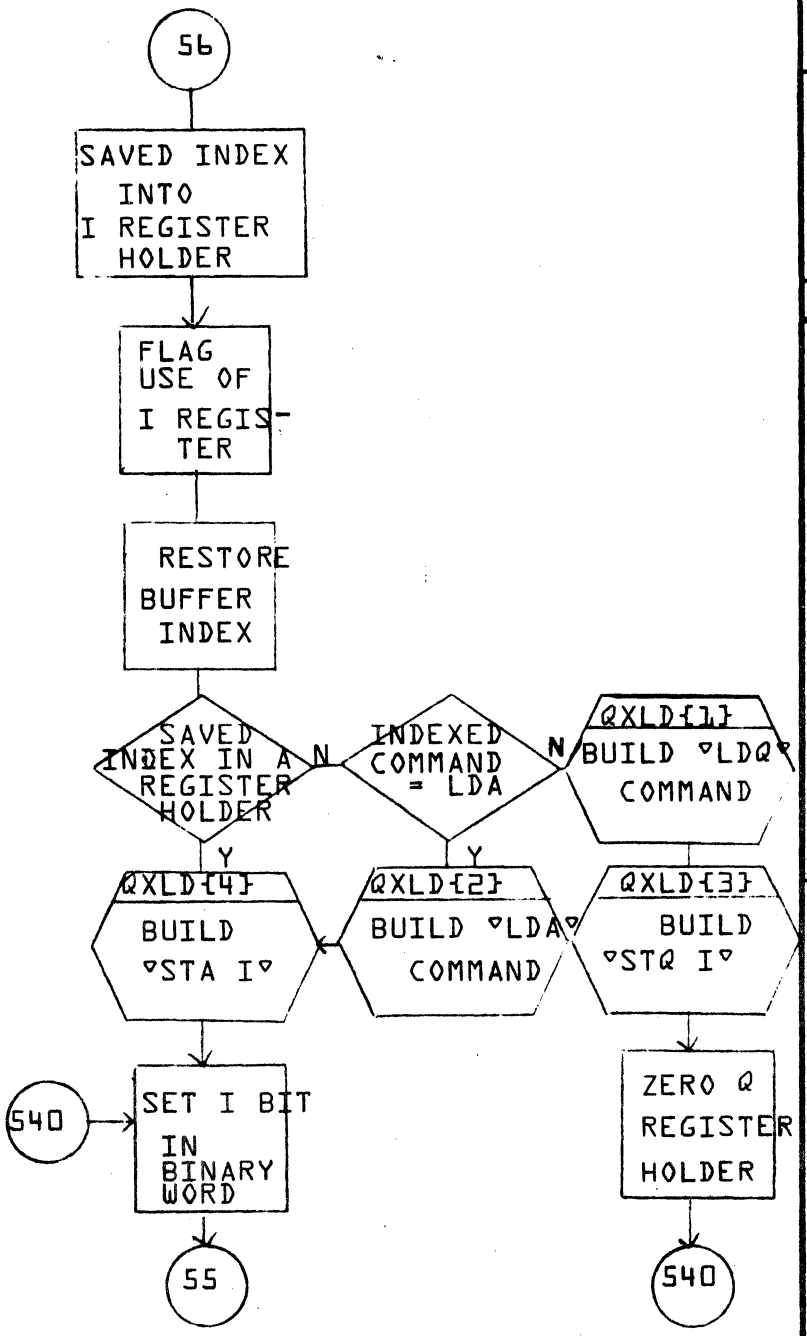
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: SUBROUTINE IXOPT
PAGE 3 OF 4
ISSUE DATE
NUMBER
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

AA1385 (FORMERLY CA127-1)



REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART ☒
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE IXOPT
PAGE	4 OF 4
NUMBER	
ISSUE DATE	
DRAWN BY	
DATE	
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

5.1.16.1 Flow Chart of Subroutine LABEL

5.1.17 Subroutine LABIN

Subroutine LABIN is called from subroutine PHASE C to assign storage locations to certain variables holders, constants and arrays. LABIN is called only if the first executable statement of the object program, excluding the jump at the beginning of a main program, has not been encountered.

Storage locations are assigned for all equivalenced arrays, all DATA arrays and all other arrays which occupy more than twenty locations. Arrays occupying twenty locations or less are assigned storage locations if their use count is less than one third the size of the array.

Storage locations are also assigned to variable holders which are used only once and to referenced constants.

All other variable holders, constants and arrays are ignored by subroutine LABIN. Storage assignments will be made for these items after they are used in the object program.

5.1.17.1 Flow Chart of Subroutine LABIN

5.1.18 Subroutine QXLD

The function of Subroutine QXLD is to generate indexing instructions.

According to the input parameter, QXLD will generate a LDA, LDQ, STA I, STQ I or TRA Q. If the instruction to be generated is a LDA or LDQ, Subroutine CHOP is called to determine the addressing mode. Subroutines BLDUP, INOUT and COUNT are called to output the instruction and bump the program counter.

5.1.18.1 Flow Chart of Subroutine QXLD

5.1.19 Subroutine SKIP

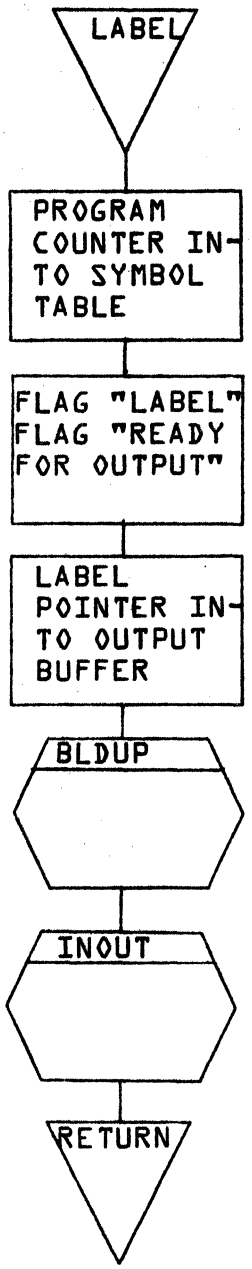
Subroutine SKIP generates one or two conditional skip instructions and a two-word relative unconditional jump instruction for every conditional jump instruction encountered in the input buffer.

The skip records are flagged as fixed langth. The jump instruction is a candidate for reduction to one word or elimination in the Assembler output phase.

Below are listed the conditional jump instructions received from the previous compiler pass and the corresponding instructions generated by the assembler:

Conditional Jump	Instruction Generated
A Jump $\neq 0$	SAZ 2 JMP
A Jump $= 0$	SAN 2 JMP
A Jump < 0	SAP 2 JMP
A Jump ≥ 0	SAM 2 JMP
A Jump ≤ 0	SAZ 1 SAP 2 JMP
A Jump > 0	SAZ 3 SAM 2 JMP
Q Jump $= 0$	SQN 2 JMP
Q Jump $\neq 0$	SQZ 2 JMP

5.1.19.1 Flow Chart of Subroutine SKIP



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

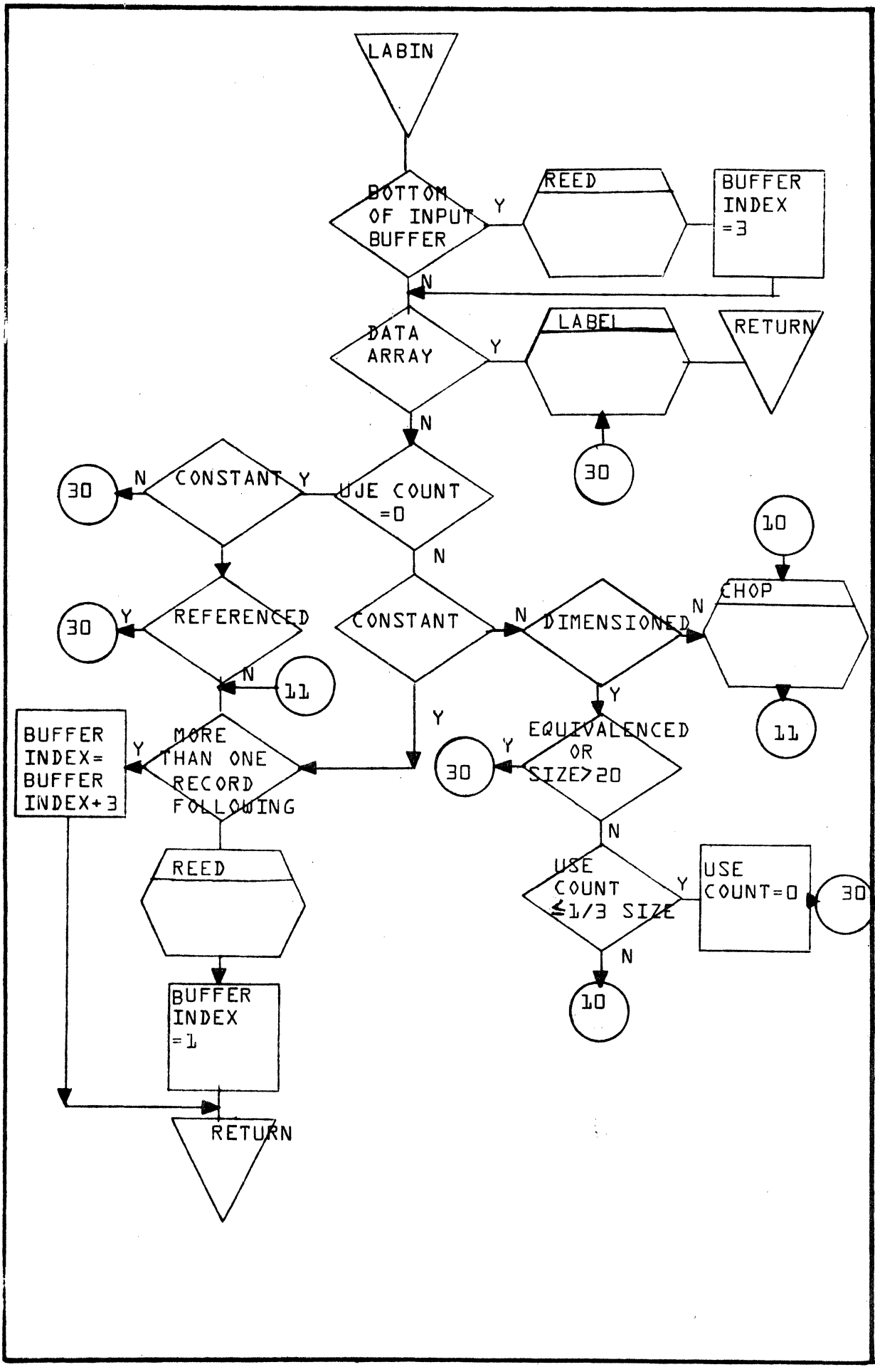
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: SUBROUTINE LABEL
PAGE 1 OF 1
NUMBER: 5.1.16.1
ISSUE DATE:
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV	APPROVED	DATE

AA1385 (FORMERLY CA127-1)

A B C D

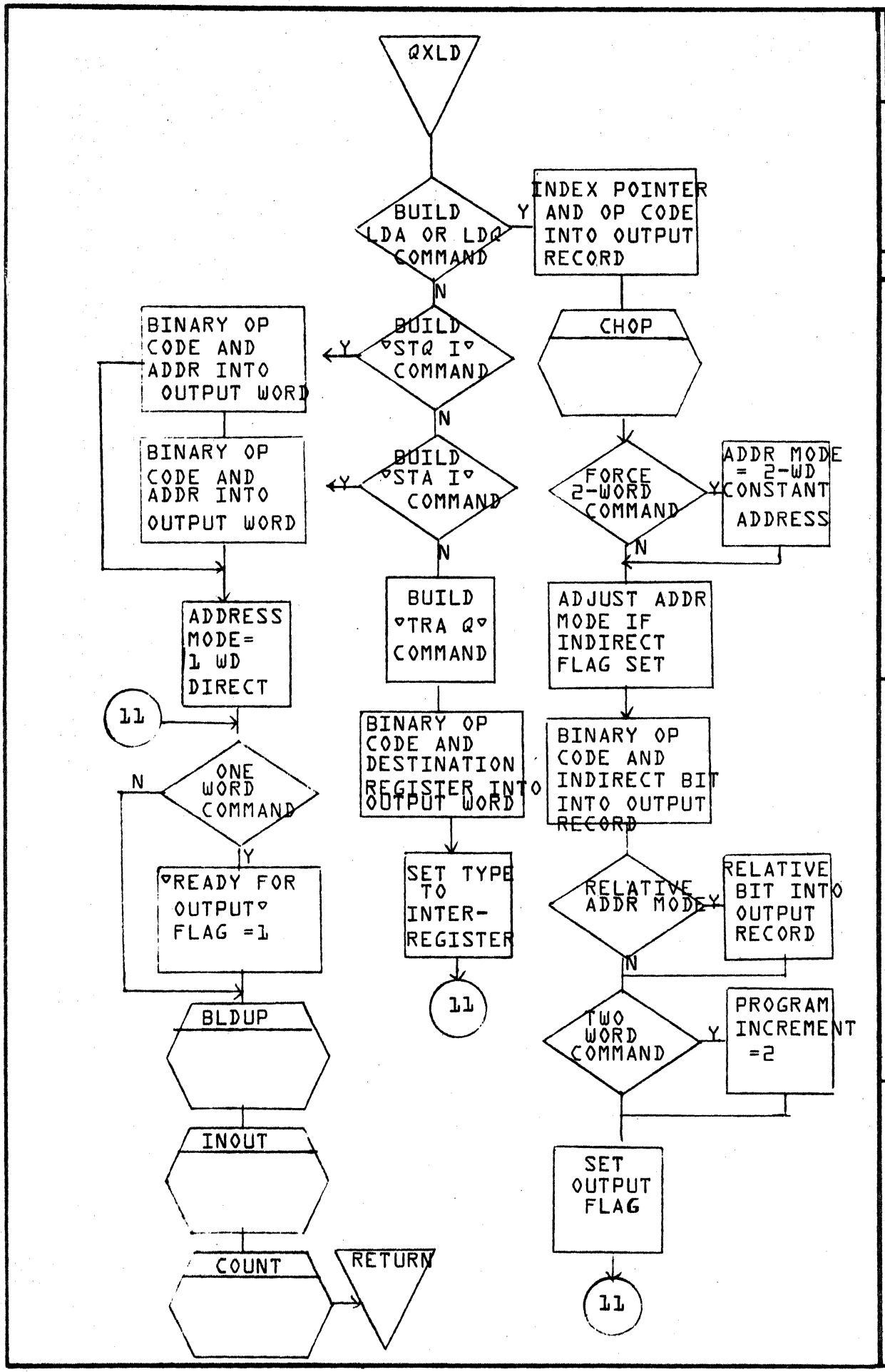


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS
DOCUMENT TITLE	SUBROUTINE LABIN
NUMBER	
DRAWN BY	
MACH. TYPE	1700
PAGE	1 OF 1
ISSUE DATE	
DATE	

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME



REV	APPROVED	DATE

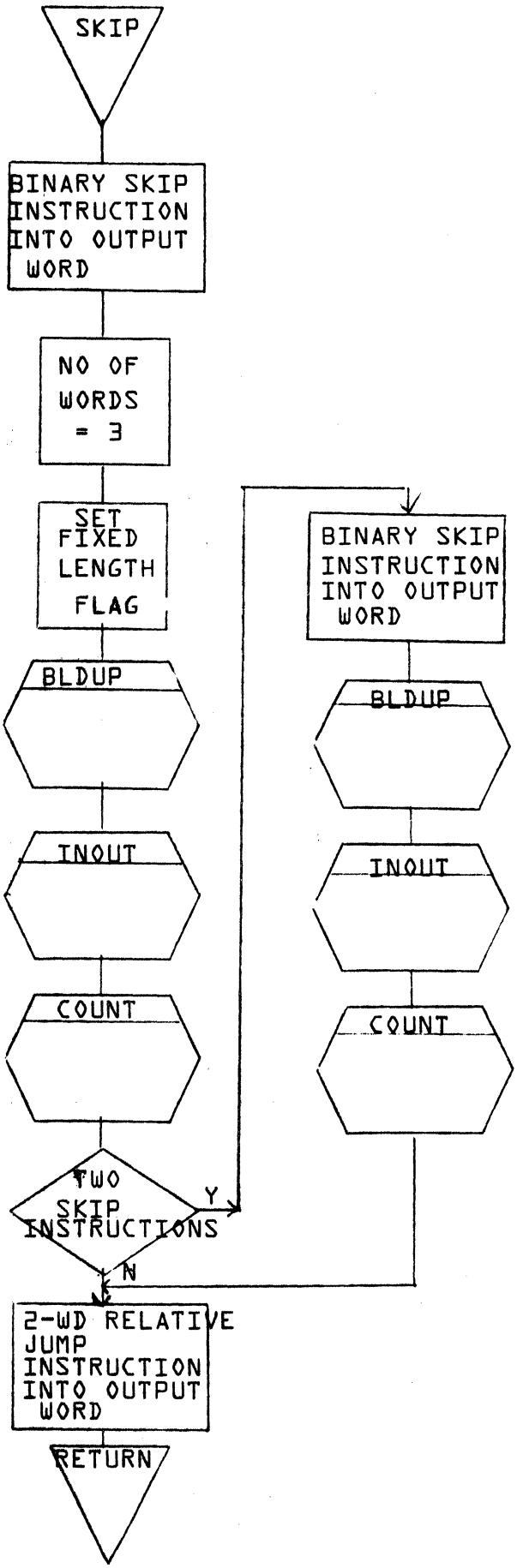
PROJECT NO.				
PROJECT MGR.				
PROJECT NAME				
TASK NO.				
TASK NAME				

DOCUMENT CLASS	IMS	MACH. TYPE	1700	
DOCUMENT TITLE	SUBROUTINE QXLD	PAGE 1 OF 1		
NUMBER		ISSUE DATE		
DRAWN BY		DATE		

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

AA1385 (FORMERLY CA127-1)

A B C D



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	SKIP SUBROUTINE	PAGE	1 OF 1
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

REV	APPROVED	DATE

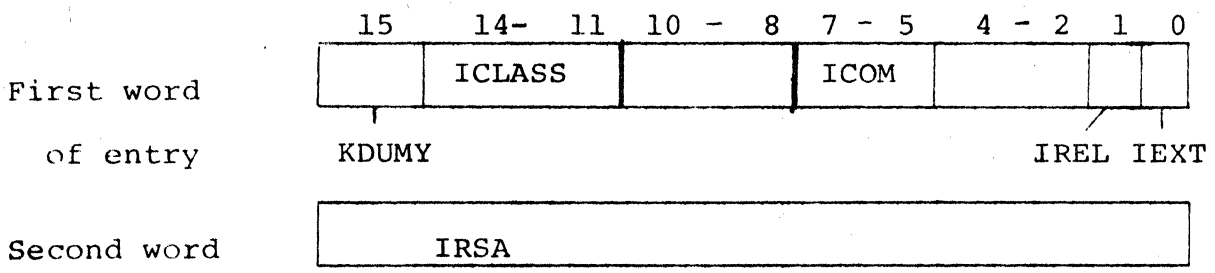
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

5.2 ASSEMBLER OUTPUT PHASE D/E

In phase D/E the pseudo-instructions output by phase C are converted to relocatable code. Address optimization and final assignment take place in this phase. The 'D/E' notation is used because for a given compilation either phase D or phase E is executed, but not both. Phase E is loaded and executed whenever the A or M options (assembly listing) are requested, otherwise phase D is used. The relocatable code output by phase D/E is punched on cards and/or stored on the load-and-go file according to the P and X options respectively.

5.2.0.1 Distinction between Phases D and E.

Phases D and E are virtually indentical with two major exceptions. First, subroutine SETPRT and all calls to SETPRT are omitted from Phase D. Second, in the 2.0A version of 1700 FORTRAN phase D uses a 'compressed' symbol table requiring two words per entry, as opposed to the five words required by the normal symbol table. The structure of this compressed symbol table is:



The meanings of the above variable names are found in the Chapter 7, where the normal symbol table is described. In the 2.0A version of Phase D, subrountine BEGINO packs the normal symbol table into the compressed symbol table. Subroutines GETSYM and SYMSCN are altered to reference the compressed table, and all routines except BEGINO and FINISH have access only to the compressed symbol table. In the 2.0A version of Phase D, subroutine AMOUT stores the IRSA entries of externals into INBUFF after the END record is processed, and subroutine FINISH retrieves these entries from INBUFF rather than from the normal symbol table.

5.2.0.2 Circular output buffer

Subroutine PHASE6 in phase D/E reads the output of phase C and stores it in the output buffer NOBUFF. For effective address optimization, NOBUFF must be filled at the 'bottom' as instructions at the 'top' are converted to relocatable code and output. To speed this process the storage assigned to NOBUFF is referenced by 'circular buffer pointers' that always increase. The NOBUFF subscript is calculated by reducing the given pointer modulo JNOB (NOBUFF size) using the function INDEX. In this way new information can be added to the output buffer without moving the unprocessed information in core, and the virtual size of NOBUFF is infinite. Subroutine PHASE6 refills NOBUFF from the phase C disk output file as necessary, and initializes and maintains the major circular buffer pointers IHEAD ('top' of buffer), NOUTOT ('bottom' of buffer), and NOCT (next available buffer location).

5.2.0.3 Summary of Phase D/E Flow

Records flagged ready for output are sent through the output routines starting with the first record in the buffer and continuing until a record is encountered that is not flagged ready for output. If this record is not sitting at the 'top' of the buffer, it is pushed to the 'top'. If no end record is present in the buffer, control is returned to PHASE6 to refill the buffer at the 'bottom.' On return, a forward scan of the buffer is initiated to find instructions that are one-word candidates. When a one-word candidate is encountered, subroutine ADMAX is called to decide whether or not to reduce the instruction to one word. The methods used to delete commands, maintain the program counter and make final assignment of operand addresses are fully detailed in the description of subroutines AMOUT and ADMAX. The scan of the output buffer is repeated once. At this point the record at the top of the buffer plus any following records flagged ready for output are forced out of the buffer. If no END record is present in the output buffer, control is returned to PHASE6 to fill the output buffer and the process is repeated.

5.2.1 Subroutine PHASE6

Subroutine PHASE6 is the main processor for phase D/E. It initializes the circular buffer pointers, calls subroutine BEGINO, fills the output buffer NOBUFF, and calls subroutine AMOUT. At every return from AMOUT, subroutine PHASE6 refills the output buffer at the 'bottom', overlaying the pseudo-instructions that have been processed by AMOUT. After AMOUT has processed the END record and returned, PHASE6 calls subroutine FINISH, punches the transfer card at the end of the relocatable output, and returns.

5.2.1.1 Flowchart of Subroutine PHASE6

5.2.2 Subroutine BEGINO

Subroutine BEGINO prints and punches the initial records of the final compiler output. It prints the length of labelled and blank common, converts the name of the program being compiled to ASCII code, prints (phase E only) and punches the NAM record and the DAT and COM list. In the 16K version of 1700 FORTRAN, subroutine BEGINO halves the symbol table page size and packs the full symbol table into the two-word entry compressed table used by phase D.

5.2.2.1 Flowchart of Subroutine BEGINO

5.2.3 Subroutine AMOUT

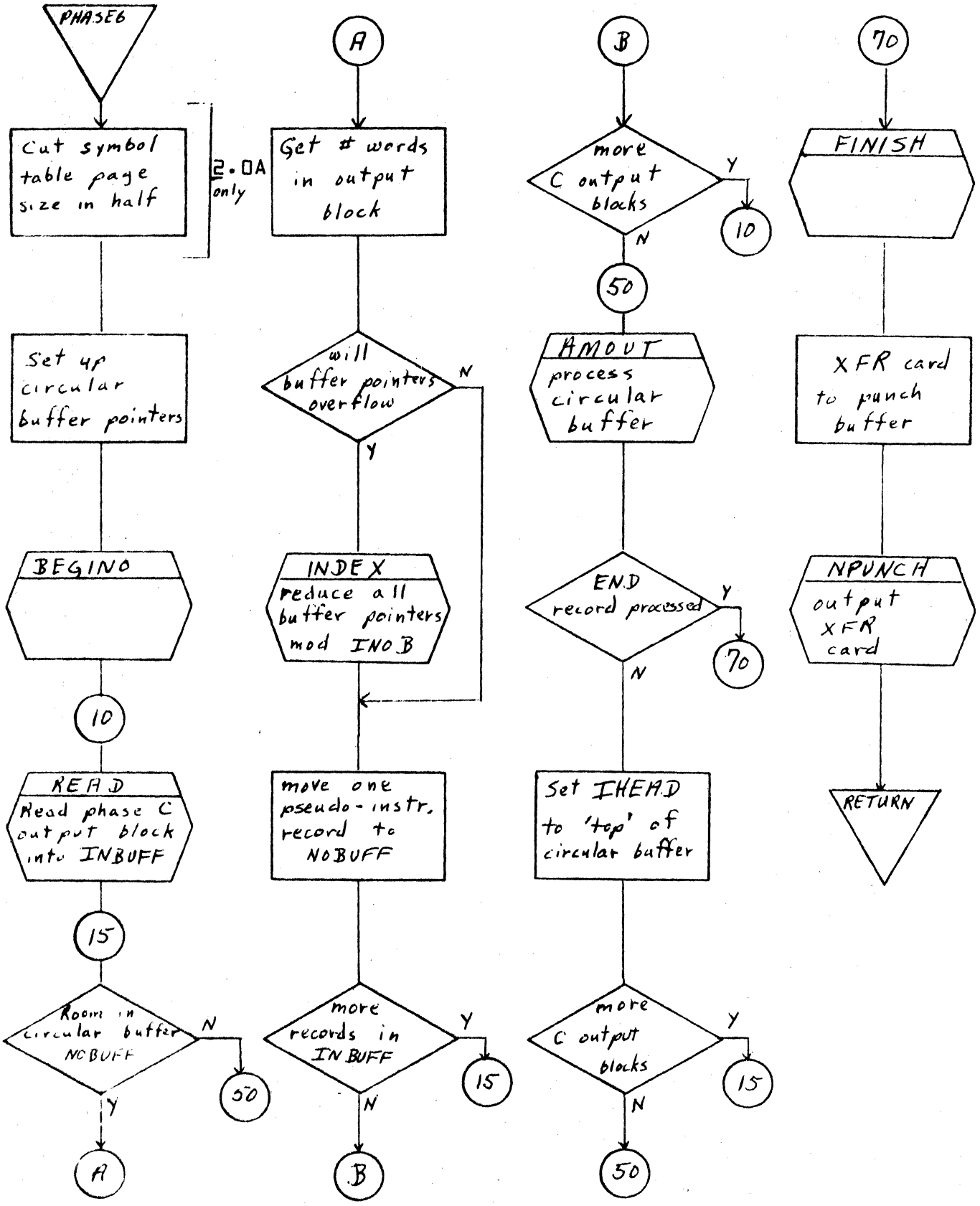
Subroutine AMOUT is the main processor for the output phase of the FORTRAN Assembler. Its function is to maximize use of one-word instructions, eliminate unneeded instructions, and send binary and ASCII output to the punch and list routines.

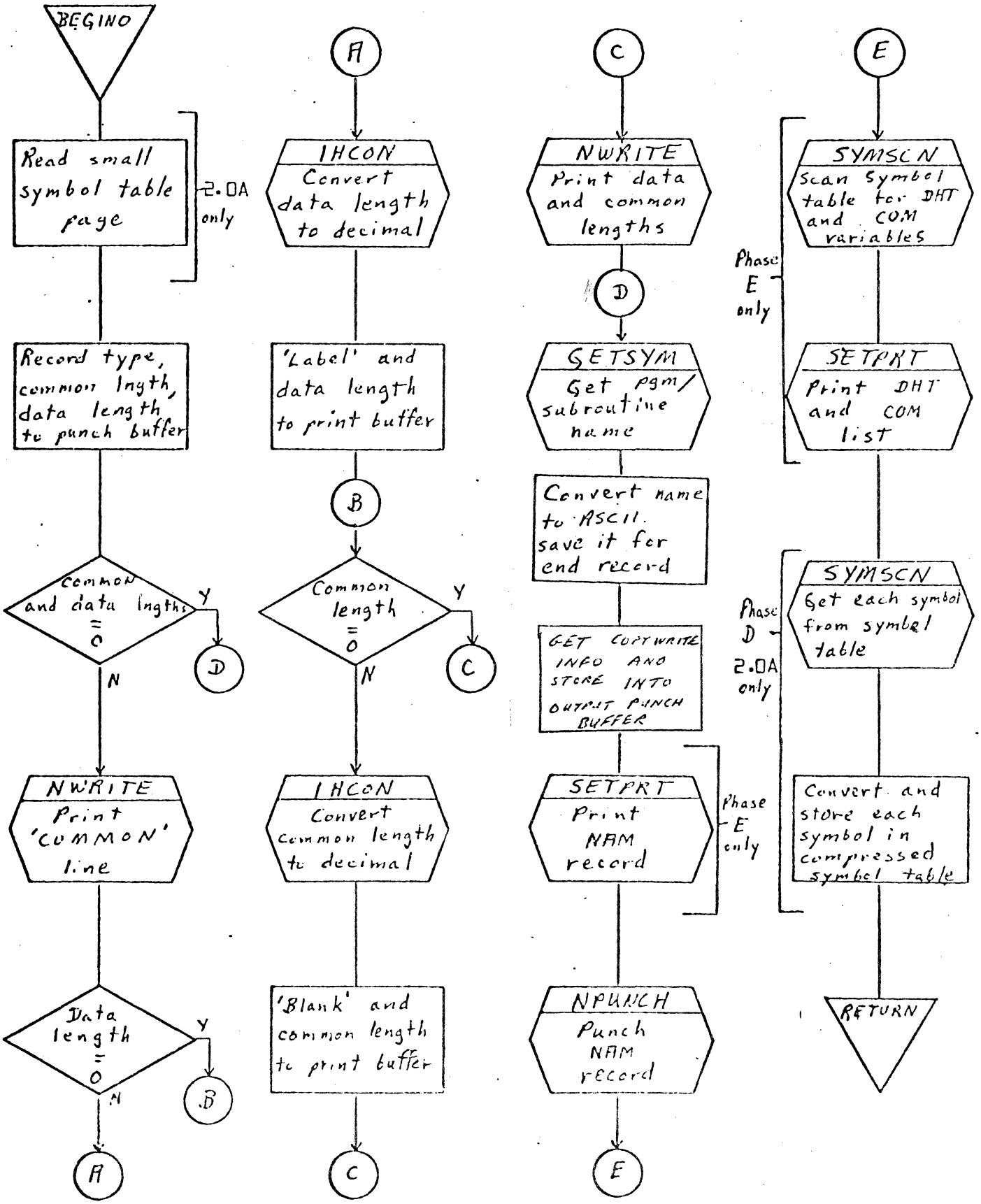
AMOUT makes a primary scan through the output buffer sending all instructions that are ready for output through processing and to the output routines until an instruction is encountered that is not ready for output. The output buffer is then moved up to eliminate the processed instructions and PHASE6 is reentered to fill the buffer.

If the instruction at the 'top' of the buffer is not ready for output, a secondary scan is initiated in which no output is generated. Every instruction that is a candidate for reduction to one word or elimina-

tion is sent to Subroutine ADMAX. ADMAX sets a change flag if any instruction is reduced or eliminated. The entire output buffer is scanned once. The program counter is initialized for each scan and kept current throughout the scan.

In the 2.0A version, AMT is used to call AMOUT.


PHASE6
Cut symbol table page size in half
2.0A only
Set up circular buffer pointers
BEGINO
10
READ
Read phase C output block into INBUFF
15
Room in circular buffer NOBUFF
N
Y
50
A
Get # words in output block
will buffer pointers overflow
N
Y
INDEX
reduce all buffer pointers mod INOB
move one pseudo-instr. record to NOBUFF
more records in INBUFF
Y
15
N
B
B
more C output blocks
Y
10
N
50
AMOUT
process circular buffer
END record processed
Y
70
N
Set IHEAD to 'top' of circular buffer
more C output blocks
Y
15
N
50
70
FINISH
XFR card to punch buffer
NPUNCH
output XFR card
RETURN


BEGIN0
Read small symbol table page
2.0A only
Record type, common lngth, data length to punch buffer
Common and data lngths = 0
Y
N
D
NWRITE
Print 'COMMON' line
Data length = 0
Y
N
B
A
A
IHCON
Convert data length to decimal
'Label' and data length to print buffer
B
Common length = 0
Y
N
C
IHCON
Convert common length to decimal
'Blank' and common length to print buffer
C
C
NWRITE
Print data and common lengths
D
GETSYM
Get pgm/ subroutine name
Convert name to ASCII. save it for end record
GET COPYWRITE INFO AND STORE INTO OUTPUT PUNCH BUFFER
SETPRT
Print NAM record
Phase E only
NPUNCH
Punch NAM record
E
E
SYMSCN
Scan Symbol table for DHT and COM variables
Phase E only
SETPRT
Print DHT and COM list
SYMSCN
Get each symbol from symbol table
Phase D 2.0A only
Convert and store each symbol in compressed symbol table
RETURN

After the optimization scan, the instruction at the top of the buffer must be output. If the address reference has not yet been identified or if the distance from the current location to the address reference is more than 1256_{10}, the instruction is stored in the future table. A binary output record to be read in at the current location at object time execution will be generated when the referenced location appears at the top of the output buffer.

If the distance from the current location to the address reference is equal to or less than 1256_{10}, the output buffer is fixed to the point of the forward reference. This is accomplished by storing the location of the forward reference in IFIX. The number or size of instructions to be output may not be changed as long as the location of the instruction or the location of its forward reference is less than or equal to IFIX.

Processing of the instruction at the top of the buffer is completed and the record is flagged ready for output. AMOUT then goes back to the primary scan.

When the END record appears in the output buffer, IFIX is set negative at the end of the secondary scan to force all instructions out of the buffer.

5.2.3.1 Flow Chart of Subroutine AMOUT

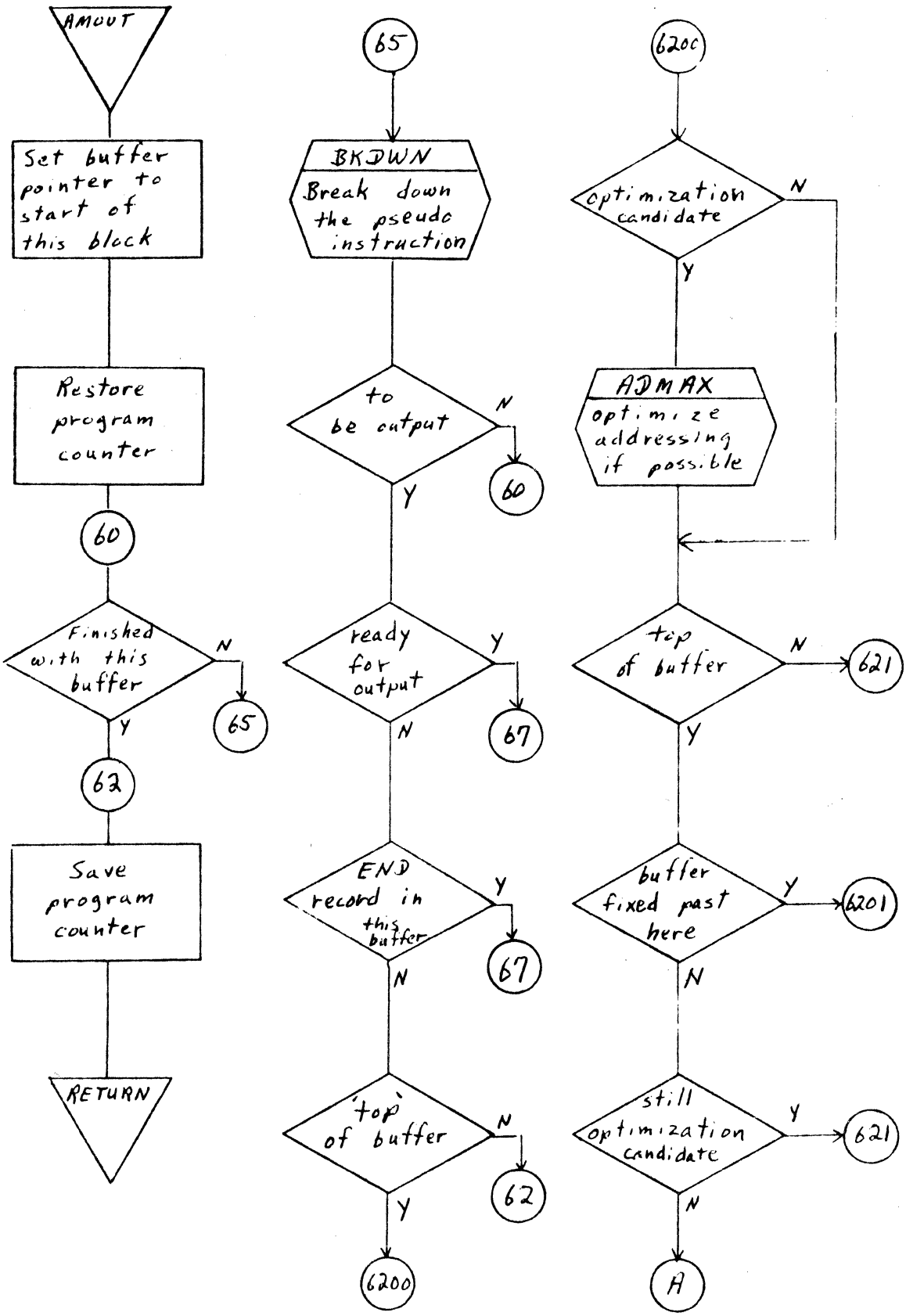
5.2.3.2 Format of the Future Table, NFTAB

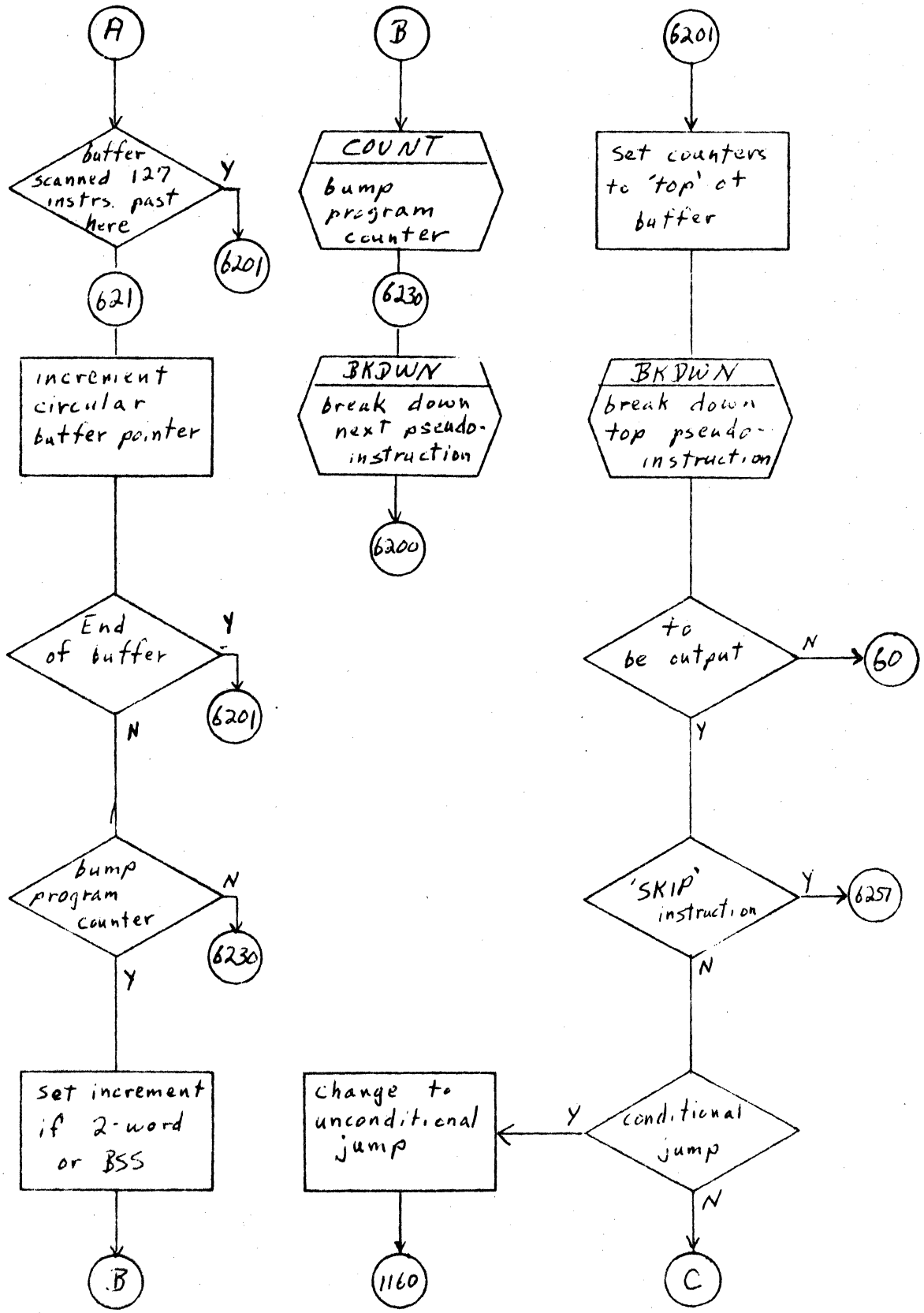
5.2.4 Subroutine ADMAX

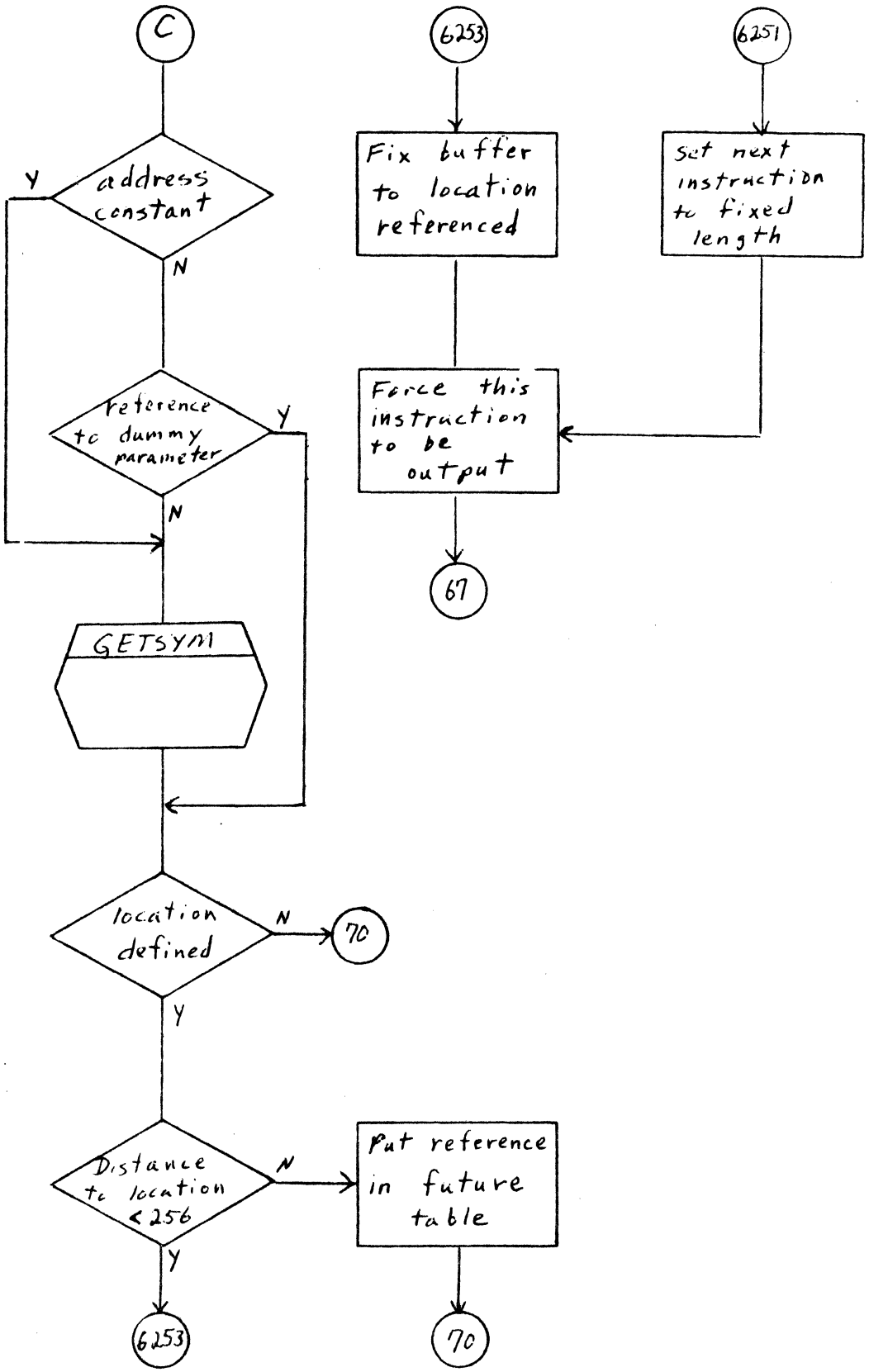
Subroutine ADMAX analyzes the instructions sent to it for possible reduction in size or elimination from the object program. It is called from subroutine AMOUT when a record is encountered that is not flagged as fixed length or ready for output.

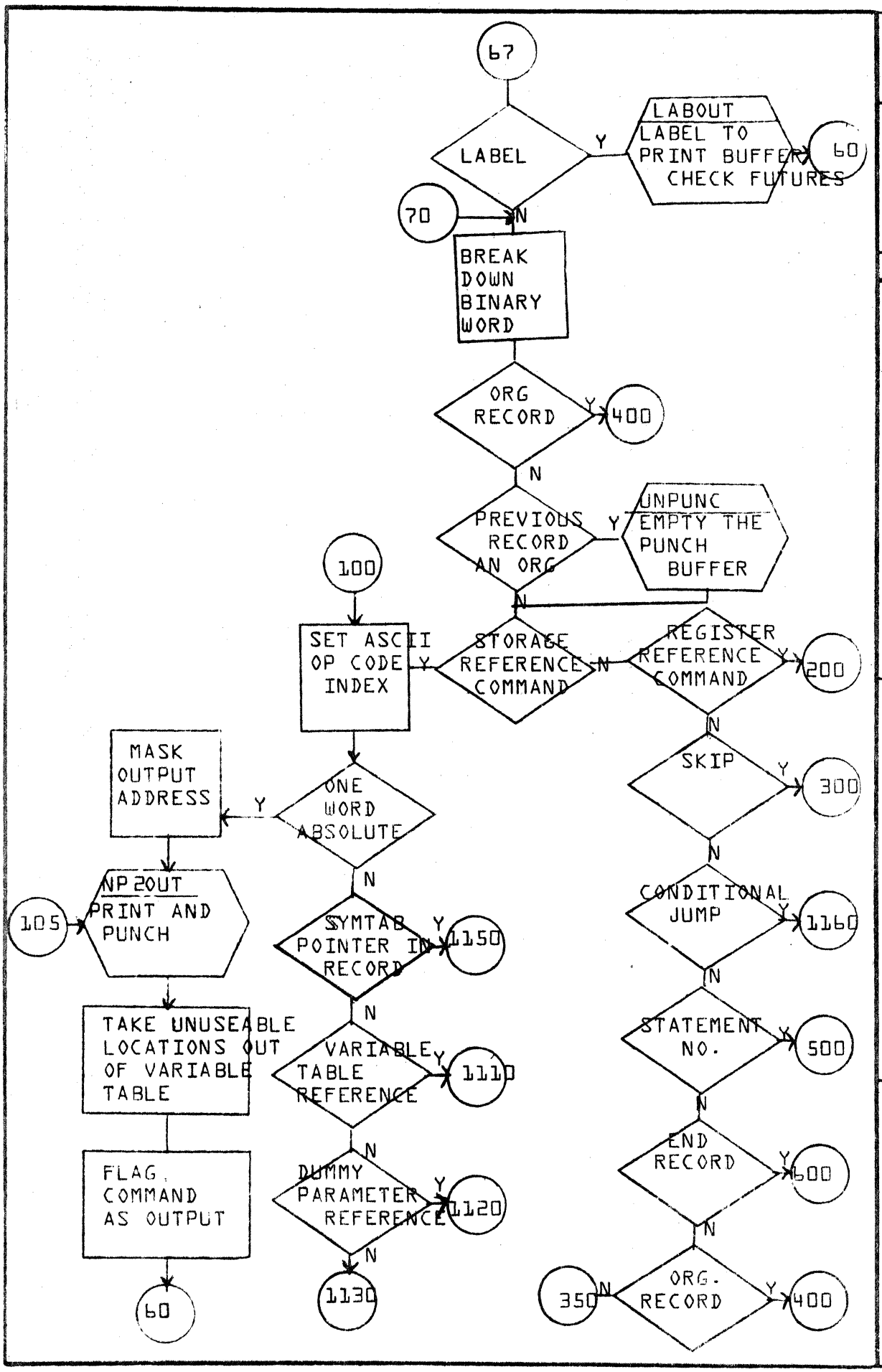
There are three situations which may produce reduction in size or elimination of an instruction:

1. The I register is not used in the object program. The compiler generated instructions to save and restore the I register and the I register holder are eliminated.
2. The program counter is not fixed and the distance to the address reference is less than 128_{10}. The type of the two-word instruction is changed to the corresponding one-word type; that is, two-word relative direct or indirect becomes one-word relative direct or indirect.


AMOUT
Set buffer pointer to start of this block
Restore program counter
60
Finished with this buffer
N
65
Y
62
Save program counter
RETURN
65
BKDWN
Break down the pseudo instruction
to be output
N
60
Y
ready for output
Y
67
N
END record in this buffer
Y
67
N
'top' of buffer
N
62
Y
6200
620C
optimization candidate
N
Y
ADMAX
optimize addressing if possible
top of buffer
N
621
Y
buffer fixed past here
Y
6201
N
still optimization candidate
Y
621
N
A


A
buffer scanned 127 instrs. past here
Y
6201
621
increment circular buffer pointer
End of buffer
Y
6201
N
bump program counter
N
6230
Y
set increment if 2-word or BSS
B
B
COUNT
bump program counter
6230
BKDWN
break down next pseudo-instruction
6200
6201
Set counters to 'top' of buffer
BKDWN
break down top pseudo-instruction
to be output
N
60
Y
'SKIP' instruction
Y
6257
N
conditional jump
Y
change to unconditional jump
1160
N
C


C
address constant
Y
N
reference to dummy parameter
Y
N
GETSYM
location defined
N
70
Y
Distance to location < 256
N
Y
6253
6253
Fix buffer to location referenced
6251
Set next instruction to fixed length
Force this instruction to be output
67
Put reference in future table
70



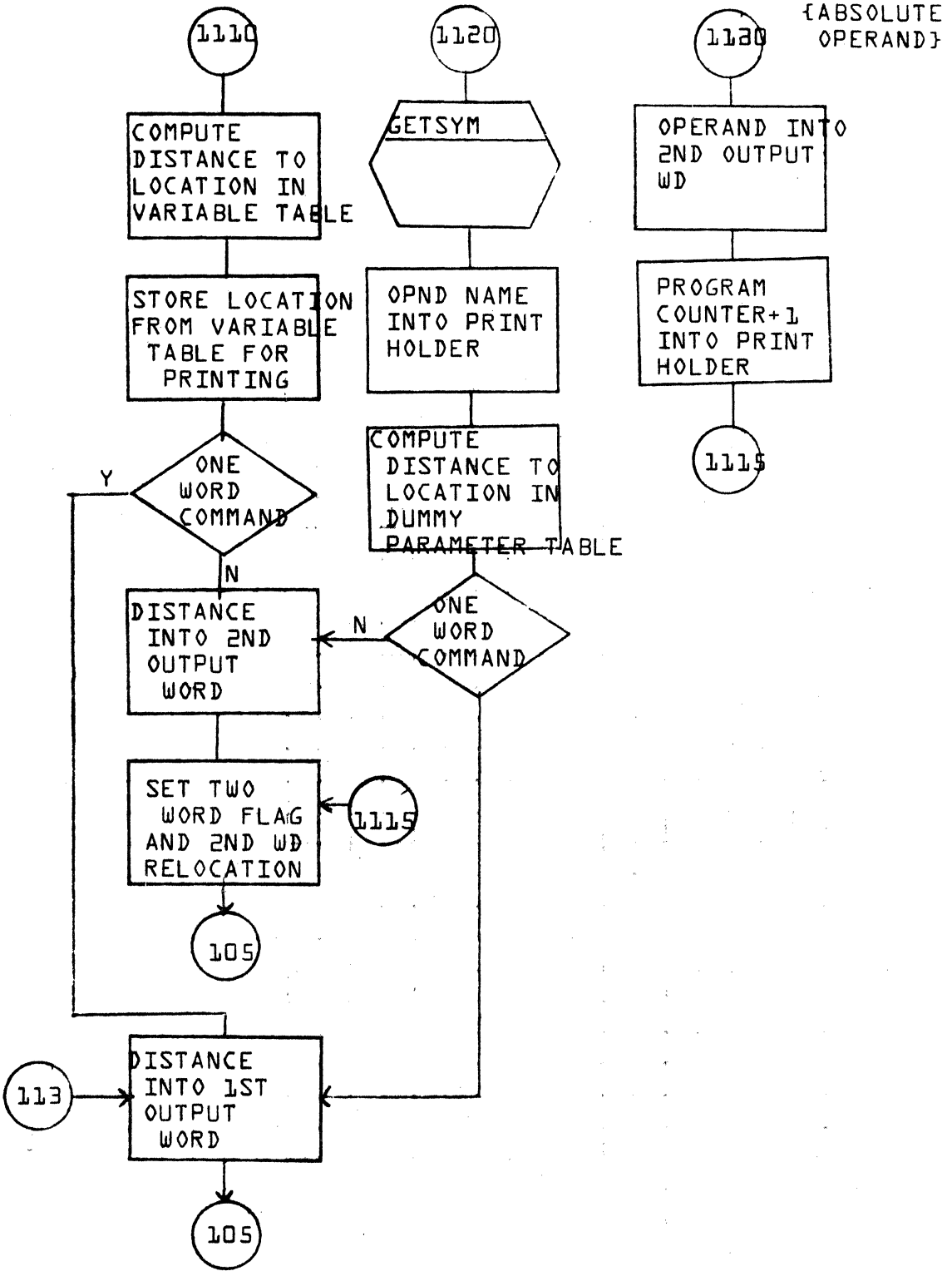
REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	
DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE AMOUNT
PAGE 3 OF 11	
ISSUE DATE	
NUMBER	
DRAWN BY	
DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

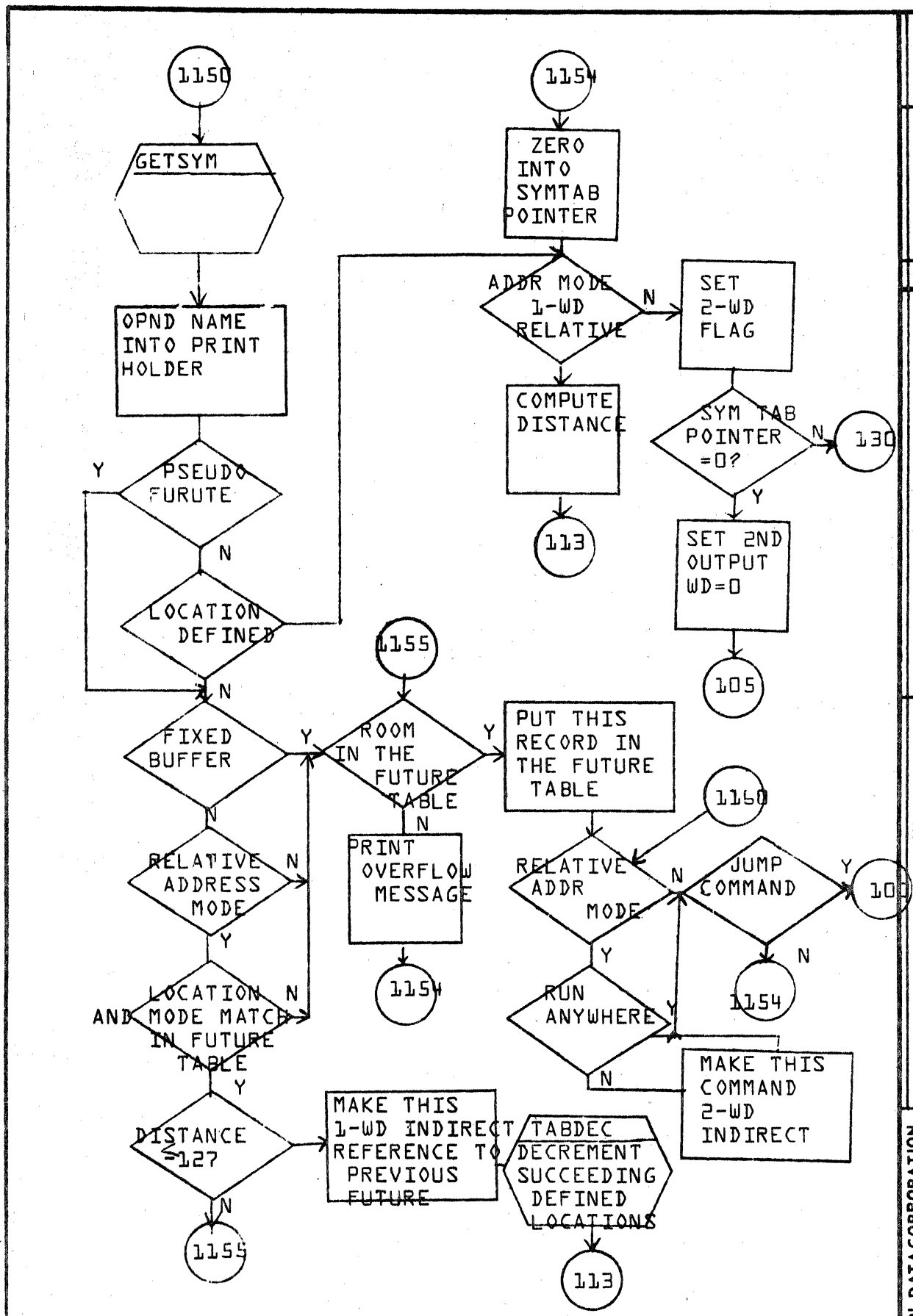
- SAMPLE CODE
- FLOWCHART [X]
- DECISION TABLE
- OTHER

A B C D



CONTROL DATA CORPORATION	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE: SUBROUTINE AMOUNT	PAGE 4 OF 11	PROJECT MGR.	APPROVED
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	NUMBER	ISSUE DATE	PROJECT NAME	REV
	DRAWN BY	DATE	TASK NO.	
			TASK NAME	

PRINTED IN U.S.A.



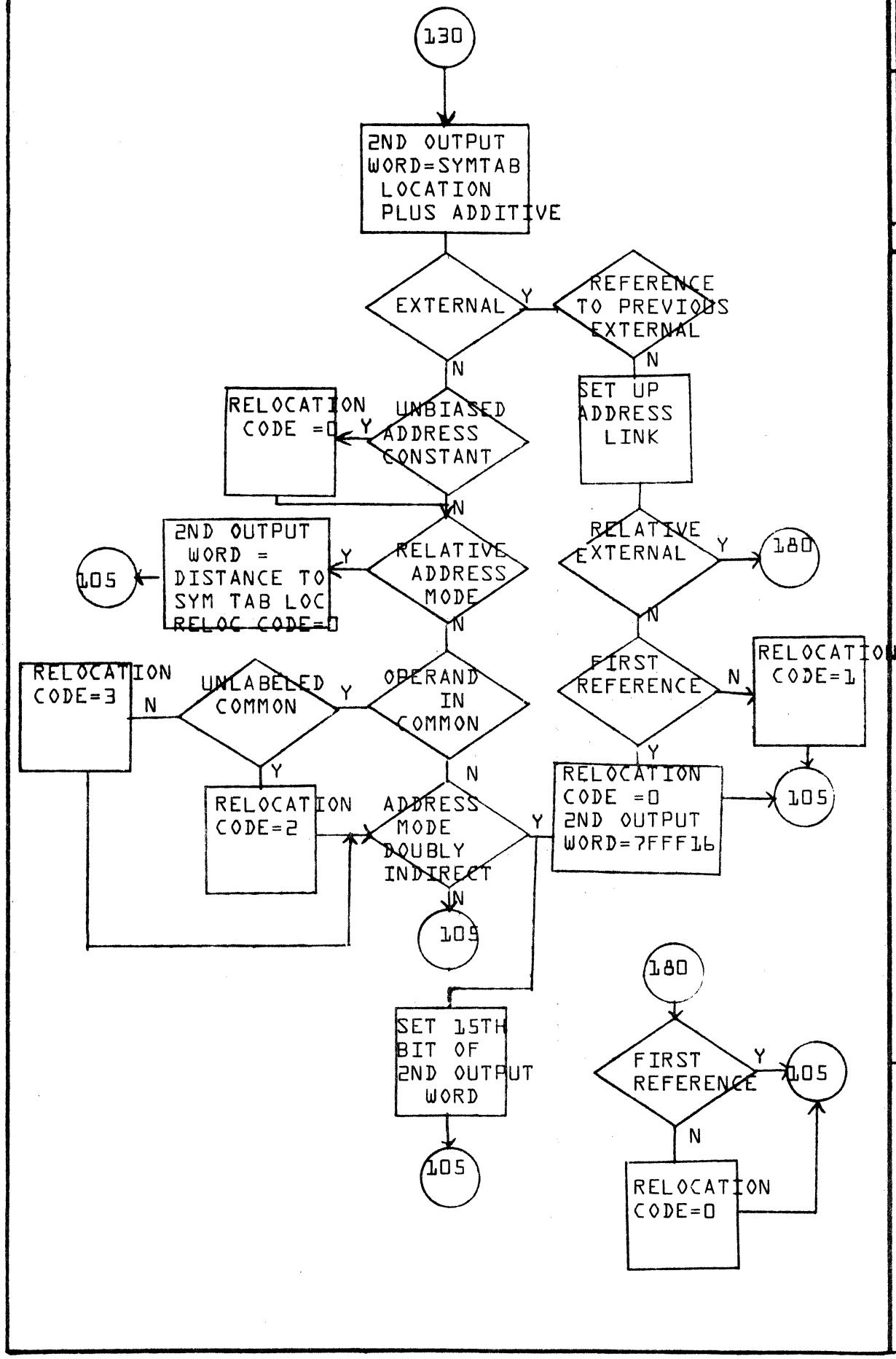
REV	APPROVED	DATE			
PROJECT NO.	PROJECT MGR.	PROJECT NAME			
TASK NO.		TASK NAME			
DOCUMENT CLASS: IMS	MACH. TYPE: 1700				
DOCUMENT TITLE: SUBROUTINE AMOUNT	PAGE 5 OF 11				
NUMBER	ISSUE DATE	DRAWN BY	DATE		

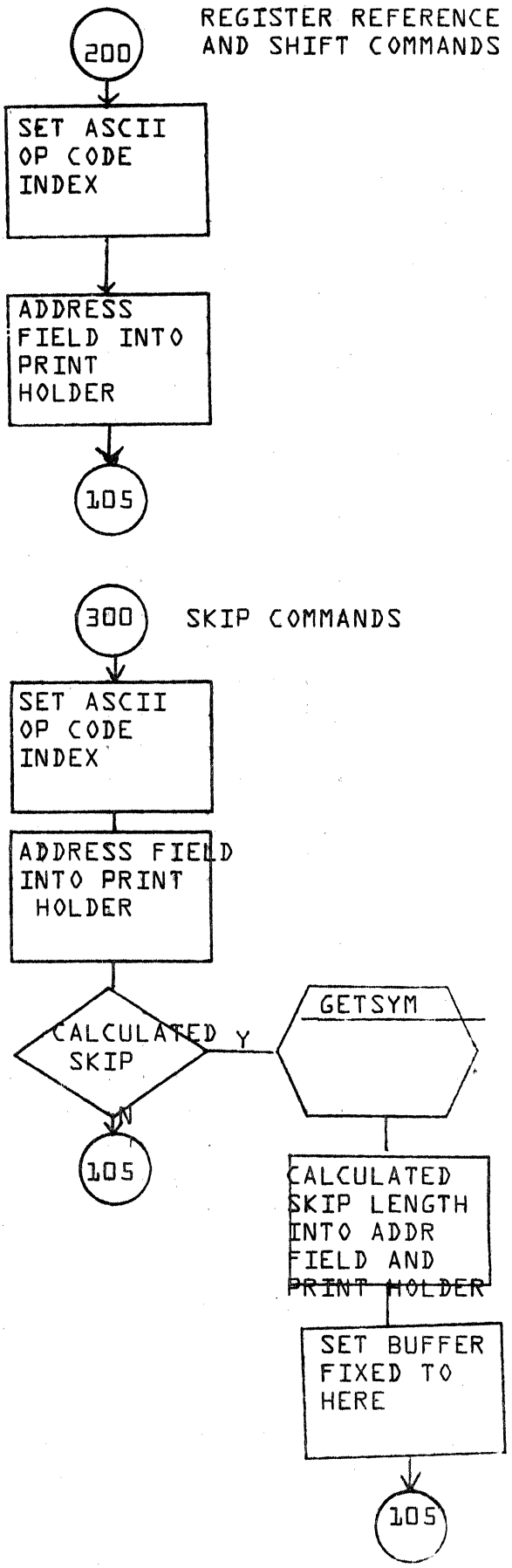
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☑
DECISION TABLE ☐
OTHER ☐

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)


130
2ND OUTPUT WORD=SYMTAB LOCATION PLUS ADDITIVE
EXTERNAL
Y
REFERENCE TO PREVIOUS EXTERNAL
N
N
SET UP ADDRESS LINK
RELOCATION CODE =0
Y
UNBIASED ADDRESS CONSTANT
N
2ND OUTPUT WORD = DISTANCE TO SYM TAB LOC RELOC CODE=0
105
Y
RELATIVE ADDRESS MODE
N
RELATIVE EXTERNAL
Y
180
N
RELOCATION CODE=3
N
UNLABELED COMMON
Y
OPERAND IN COMMON
N
FIRST REFERENCE
N
RELOCATION CODE=1
Y
Y
RELOCATION CODE=2
ADDRESS MODE DOUBLY INDIRECT
Y
N
RELOCATION CODE =0 2ND OUTPUT WORD=7FFF16
105
109
SET 15TH BIT OF 2ND OUTPUT WORD
105
180
FIRST REFERENCE
Y
105
N
RELOCATION CODE=0
DATE
APPROVED
REV
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
DOCUMENT CLASS
IMS
MACH. TYPE
1700
DOCUMENT TITLE
SUBROUTINE AMOUNT
PAGE 6 OF 11
NUMBER
ISSUE DATE
DRAWN BY
DATE
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

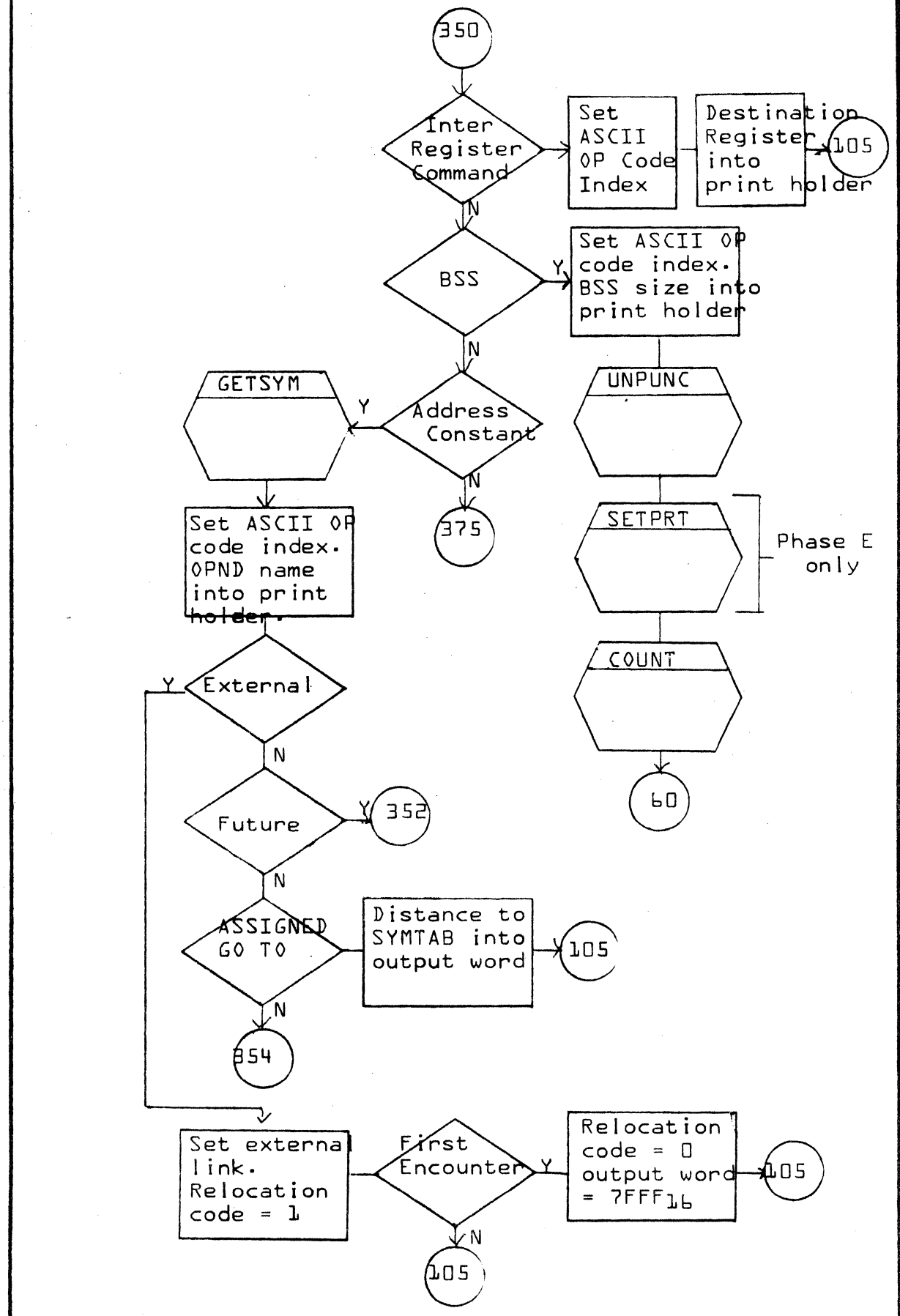
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: SUBROUTINE AMOUNT
PAGE 7 OF 11
ISSUE DATE
NUMBER
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

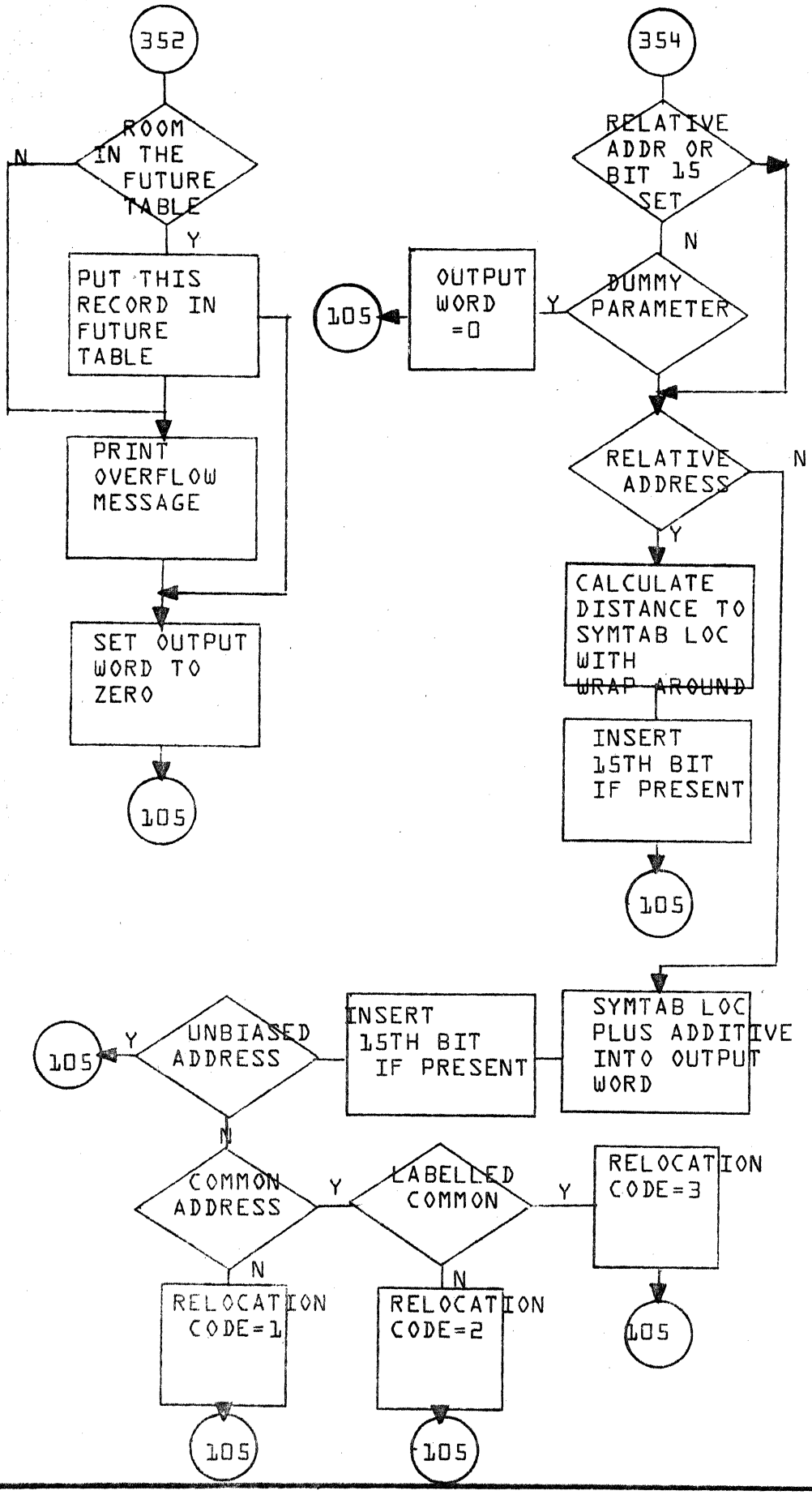
A B C D



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

REV	APPROVED	DATE

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: Subroutine AMOUT
PAGE 8 OF 11
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
NUMBER 5.2.1.1
ISSUE DATE
DRAWN BY N. Talbott
DATE 11/1/65



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

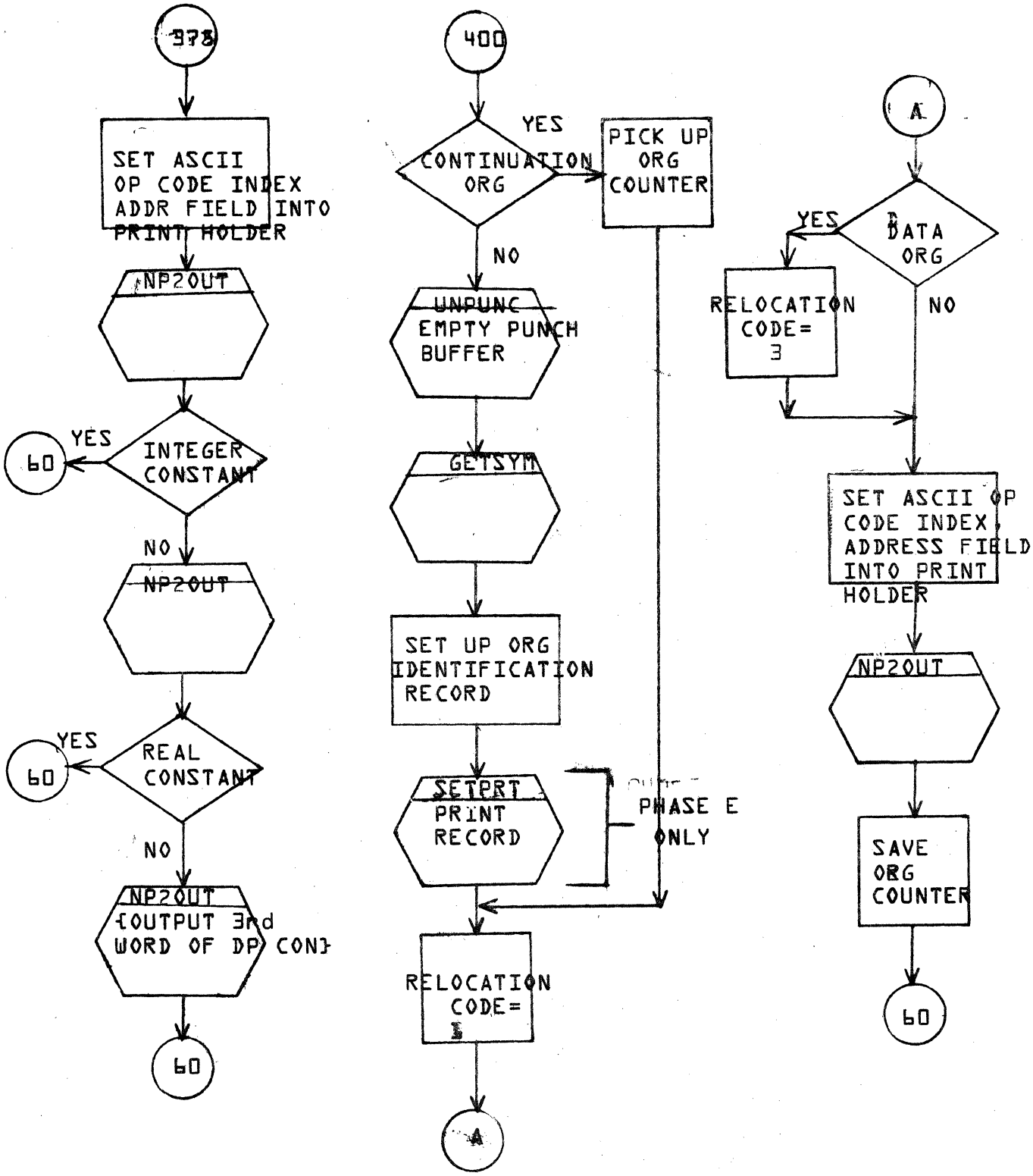
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: SUBROUTINE AMOUNT
PAGE 9 OF 11
ISSUE DATE
NUMBER
DATE
DRAWN BY

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

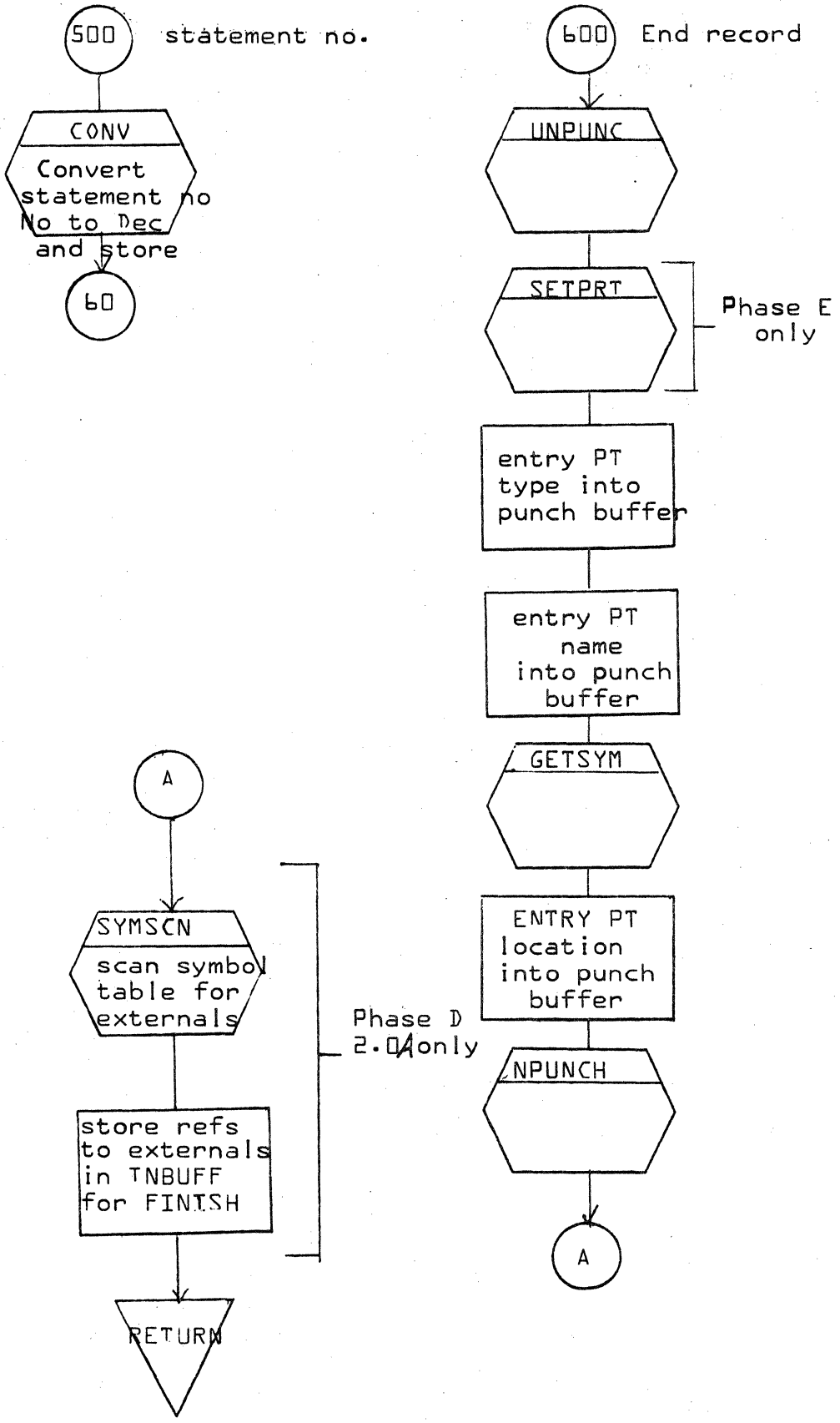
AA1385 (FORMERLY CA127-1)

A B C D



TITLE	SUBROUTINE AMOUT		DRG. NO.
			REVISION
DRAWN BY	PROJ.	DATE	SHEET 10 OF 11

CSD 209



REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐☐☐☐
FLOWCHART
DECISION TABLE
OTHER

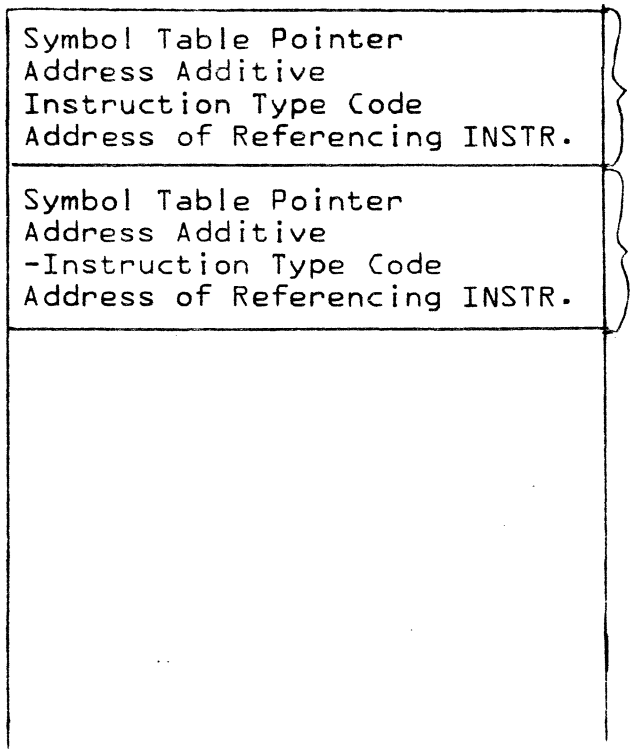
DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	Subroutine AMOUT
PAGE 11 OF 11	
NUMBER	5.2.1.1
ISSUE DATE	
DRAWN BY	N Talbott
DATE	11/26/66
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

PRINTED IN U.S.A.

CA1385 (FORMERLY CA127-1)

A B C D

FORMAT OF NFTAB



Pointer to Symbol Table Entry for this operand

This entry for an operand not yet defined

This entry for a defined operand with relative distance > 255_{10}

CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FORMAT OF NFTAB	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

3. The program counter is not fixed, the instruction is a conditional jump and the distance to the address reference is less than 16_{10}. The jump instruction is eliminated and the preceding skip or skips are changed to reflect the elimination of the jump.

   The fifteenth bit of the indicator is set in a record containing an instruction to be eliminated. This will cause subroutine AMOUT to ignore the record.

   A decrement count is set for instructions reduced in size or eliminated. Subroutine TABDEC is called to decrement all affected storage addresses.

5.2.4.1 Flow Chart of Subroutine ADMAX

5.2.5 Subroutine CONV

Subroutine CONV converts a binary word to decimal format. It is handed two parameters by the calling program. The first is the location of the word to be converted; the second is the location into which the converted word is to be stored.

5.2.6 Subroutine COUNT

Subroutine COUNT increments the program counter. The increment holder, ICT, is always set at one unless re-set by the calling program.

5.2.7 Subroutine FINISH

Subroutine FINISH lists the program length, referenced externals, and undefined symbols, if any. It also generates binary external records.

5.2.7.1 Flow Chart of Subroutine FINISH

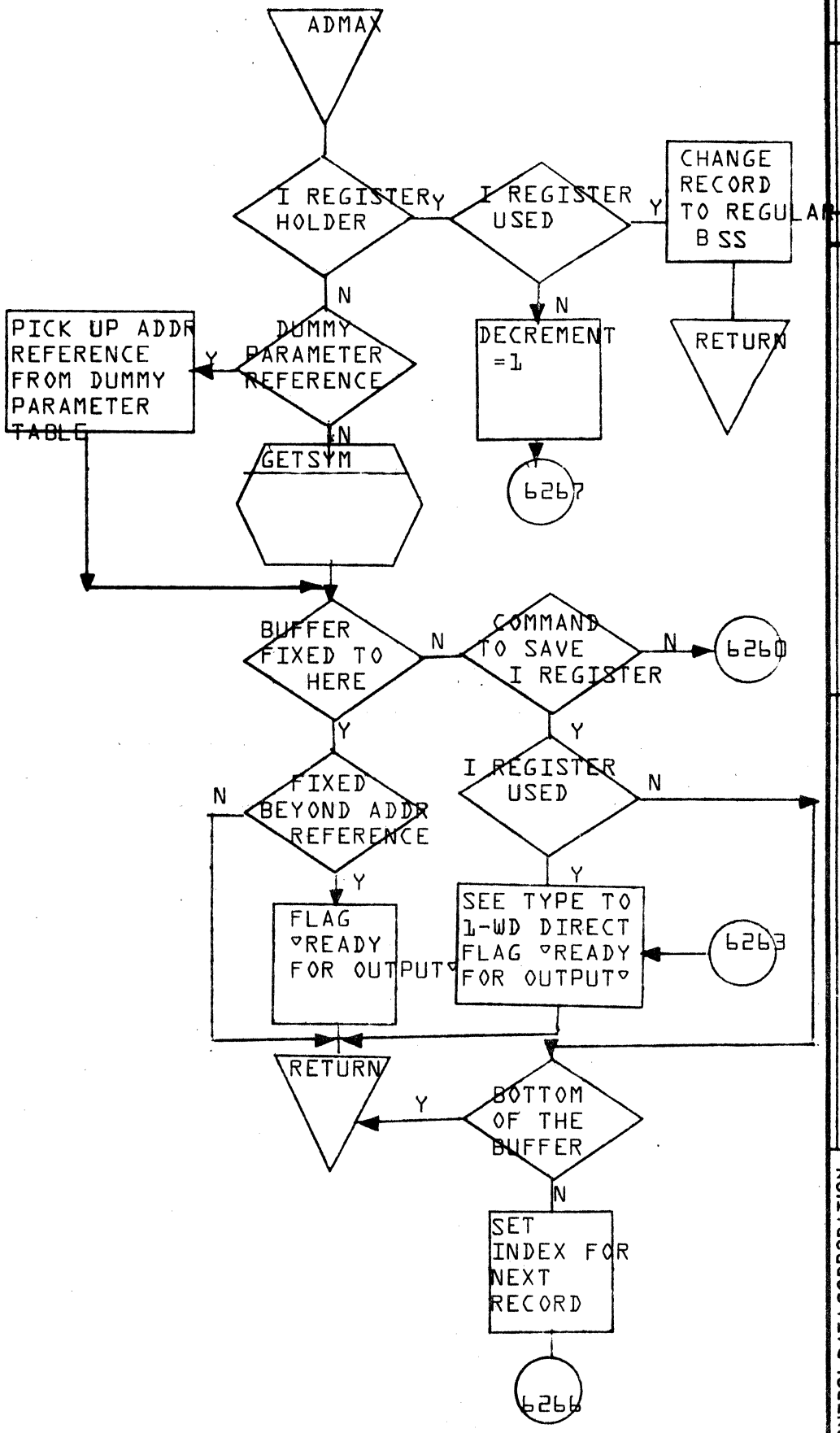
5.2.8 Subroutine IACON

Subroutine IACON converts a two-word symbol name (ISYM - packed by CNVT) to ASCII format and stores it in the print buffer.

5.2.9 Subroutine IHCON

Subroutine IHCON converts a binary word to hex format and stores it in the print buffer.

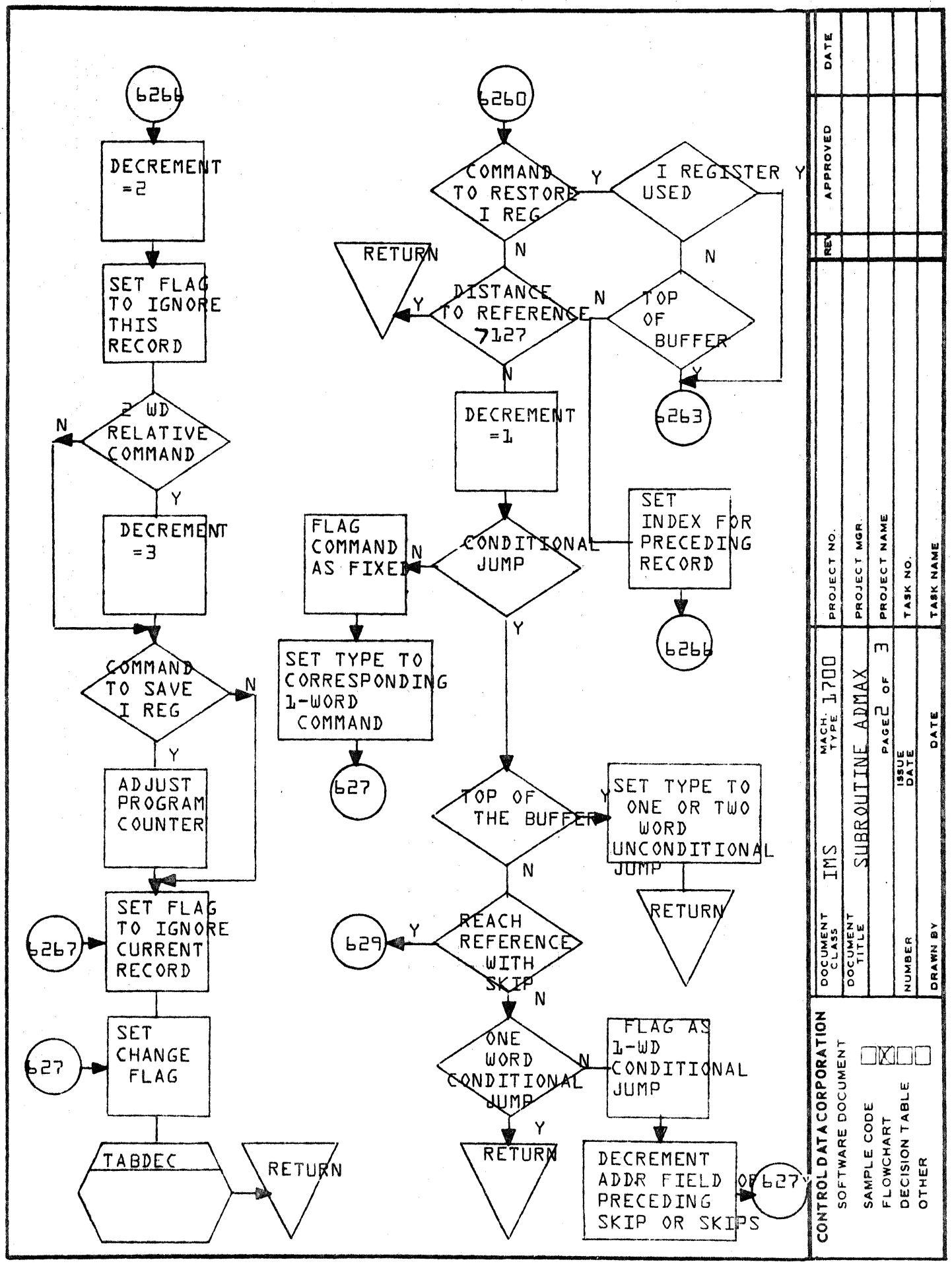
5.2.10 Function INDEX (INX) converts the circular buffer pointer INX (which always increases) to a subscript of NOBUFF. That is, INDEX (INX) is equal to INX modulo INOB (NOBUFF size).

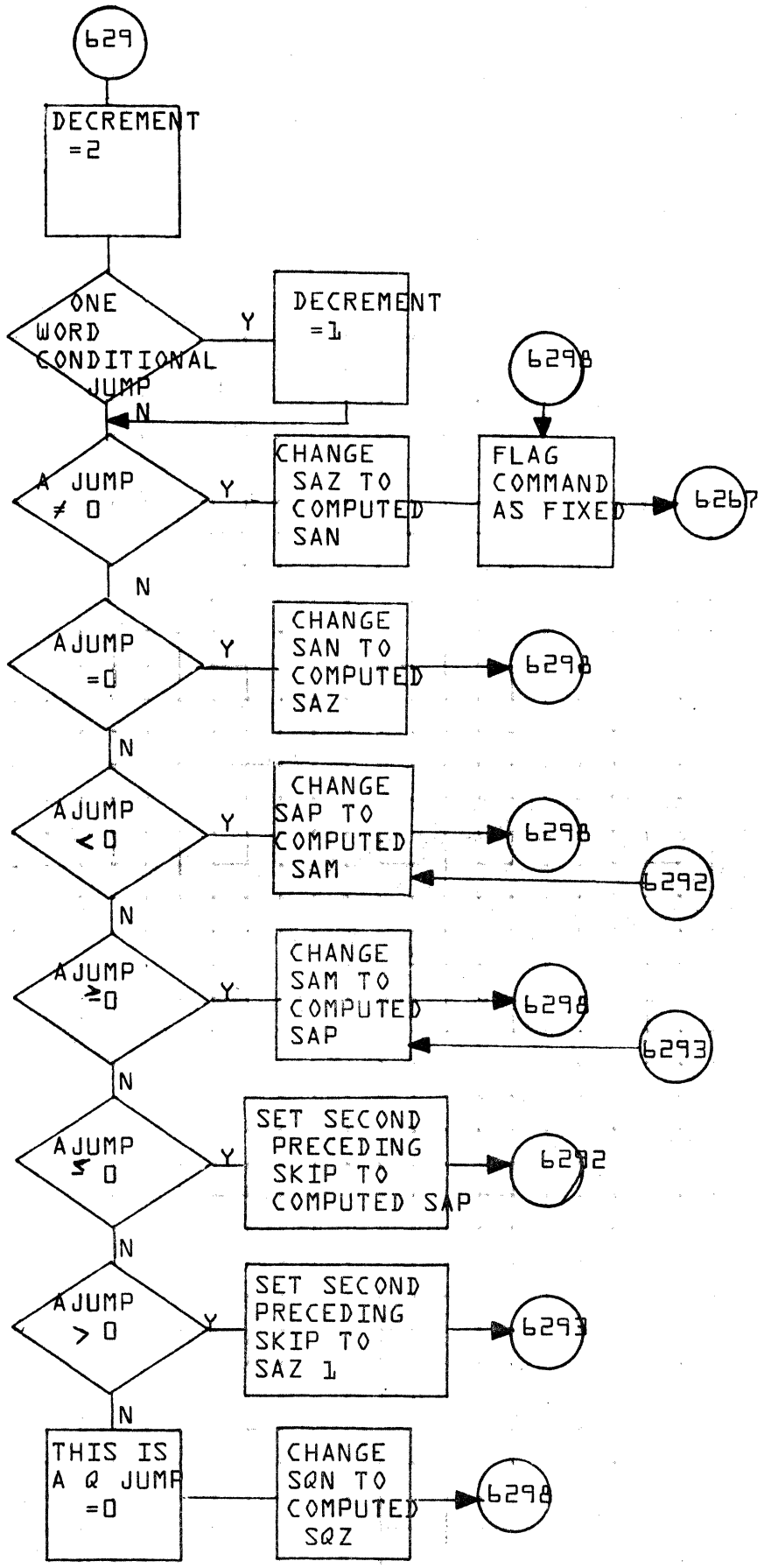


REV	APPROVED	DATE	
PROJECT NO.	PROJECT MGR.	PROJECT NAME	TASK NO.
TASK NAME			
DOCUMENT CLASS: IMS	MACH. TYPE: 1700		
DOCUMENT TITLE: SUBROUTINE ADMAX	PAGE 1 OF 3		
NUMBER	ISSUE DATE	DRAWN BY	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐


6266
DECREMENT =2
SET FLAG TO IGNORE THIS RECORD
2 WD RELATIVE COMMAND
N
Y
DECREMENT =3
COMMAND TO SAVE I REG
N
Y
ADJUST PROGRAM COUNTER
6267
SET FLAG TO IGNORE CURRENT RECORD
627
SET CHANGE FLAG
TABDEC
RETURN
6260
COMMAND TO RESTORE I REG
Y
I REGISTER USED
Y
N
N
RETURN
Y
DISTANCE TO REFERENCE 7127
N
TOP OF BUFFER
Y
N
6263
DECREMENT =1
SET INDEX FOR PRECEDING RECORD
FLAG COMMAND AS FIXED
N
CONDITIONAL JUMP
6266
SET TYPE TO CORRESPONDING 1-WORD COMMAND
627
Y
TOP OF THE BUFFER
Y
SET TYPE TO ONE OR TWO WORD UNCONDITIONAL JUMP
RETURN
N
629
Y
REACH REFERENCE WITH SKIP
N
ONE WORD CONDITIONAL JUMP
N
FLAG AS 1-WD CONDITIONAL JUMP
Y
RETURN
DECREMENT ADDR FIELD OF PRECEDING SKIP OR SKIPS
627
REV
APPROVED
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
DOCUMENT CLASS
IMS
MACH. TYPE
1700
DOCUMENT TITLE
SUBROUTINE ADMAX
PAGE 2 OF 3
ISSUE DATE
NUMBER
DATE
DRAWN BY
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER
PRINTED IN U.S.A.
AA1388 (FORMERLY CA127-1)



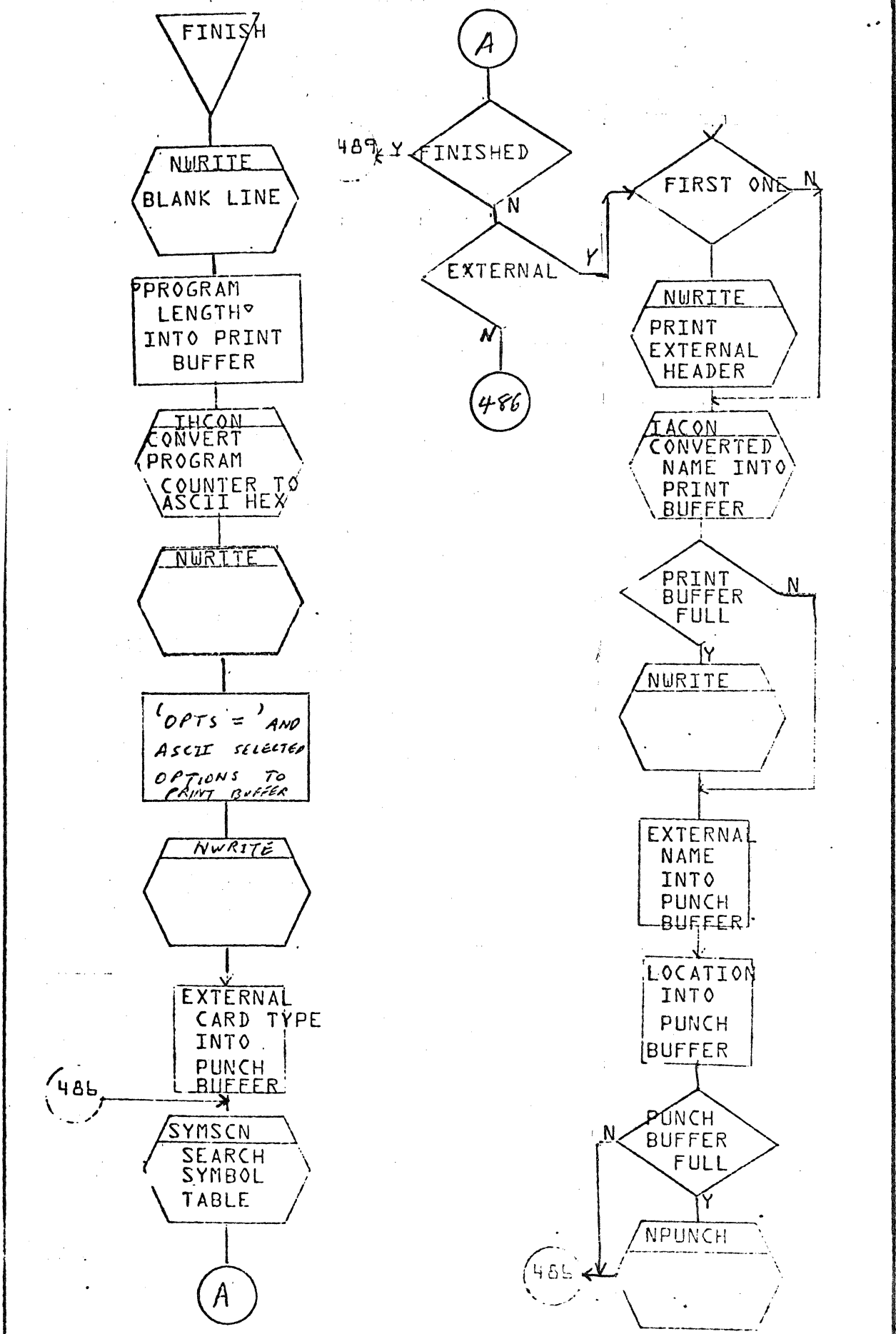
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE ADMAX	PAGE	3 OF 3
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

REV	APPROVED	DATE

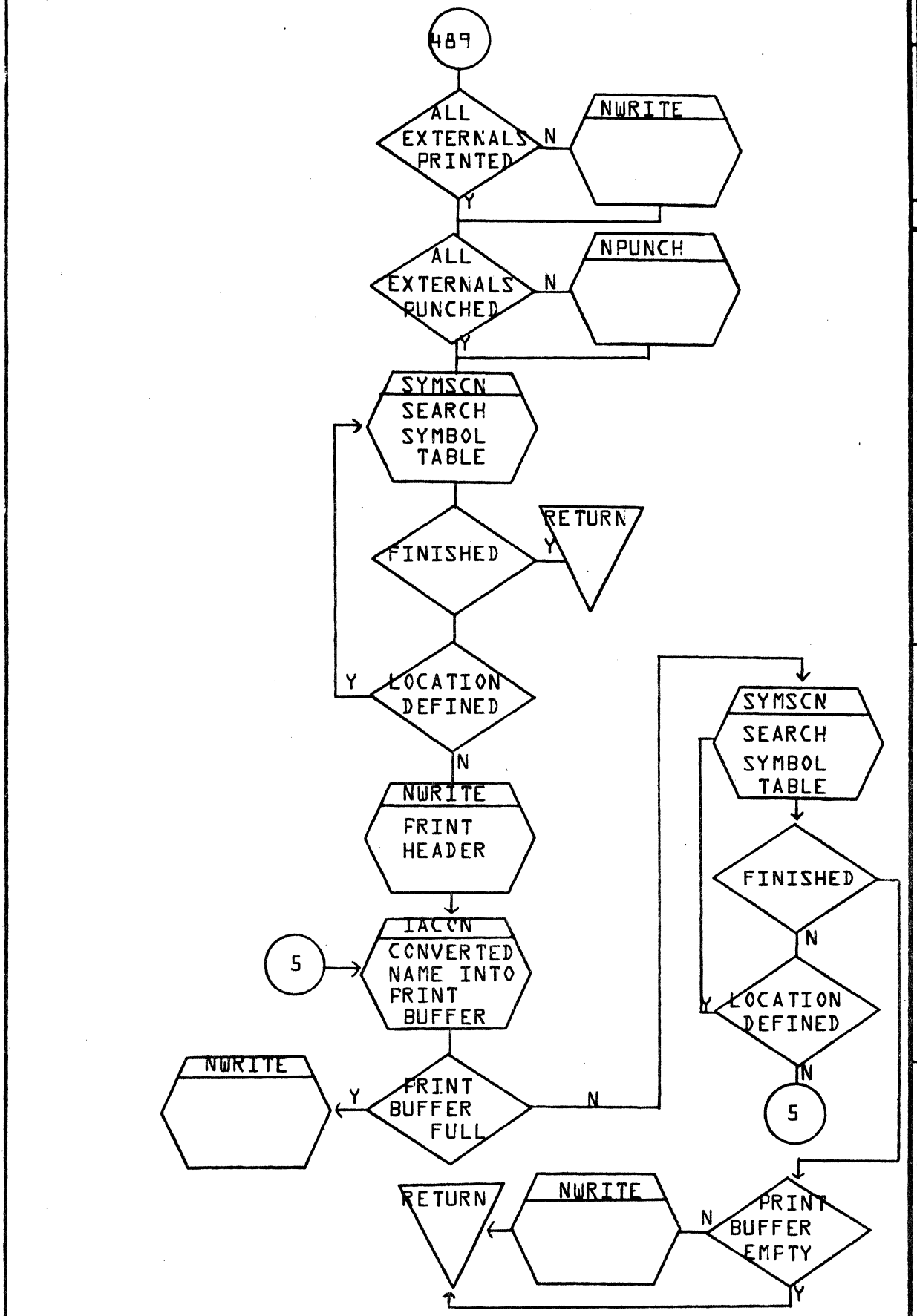
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

PRINTED IN U.S.A.



CONTROL DATA CORPORATION	DOCUMENT CLASS	IMS	MACH. TYPE	1700	REV
SOFTWARE DOCUMENT	DOCUMENT TITLE	SUBROUTINE FINISH	PAGE 1 OF 2		PROJECT NO.
SAMPLE CODE ☐ FLOWCHART ☑ DECISION TABLE ☐ OTHER ☐	NUMBER		ISSUE DATE		PROJECT MGR.
	DRAWN BY		DATE		PROJECT NAME
					TASK NO.
					TASK NAME
					APPROVED
					DATE

AA1345 (10-71) CA127-11



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE FINISH
PAGE	2 OF 2
NUMBER	5.2.5.1
ISSUE DATE	
DRAWN BY	
DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

5.2.11 Subroutine LABOUT

Subroutine LABOUT puts labels in the print buffer by calling subroutines IACON or IHCON as applicable. It also checks the future table, NFTAB, for references to this label. If any references are found, LABIN outputs a binary record origined at the location of the instruction referencing the label. The binary record contains the location of the label or the distance from the label to the reference depending on the addressing mode of the instruction.

5.2.11.1 Flow Chart of Subroutine LABOUT

5.2.12 Subroutine NPUNCH

Subroutine NPUNCH outputs binary records from the punch buffer and clears the buffer.

5.2.13 Subroutine NP2OUT

Subroutine NP2OUT sends one or two output images to the binary output routine and to the list output routine (Phase E only) according to the size of the instruction. The program counter is bumped for each image that is output.

5.2.13.1 Flow Chart of Subroutine NP2OUT

5.2.14 Subroutine NWRITE

Subroutine NWRITE outputs one list record from the print buffer and sets the buffer to ASCII blanks.

5.2.15 Subroutine PACK

Subroutine PACK is called from subroutine NWRITE to pack the output image in the print buffer.

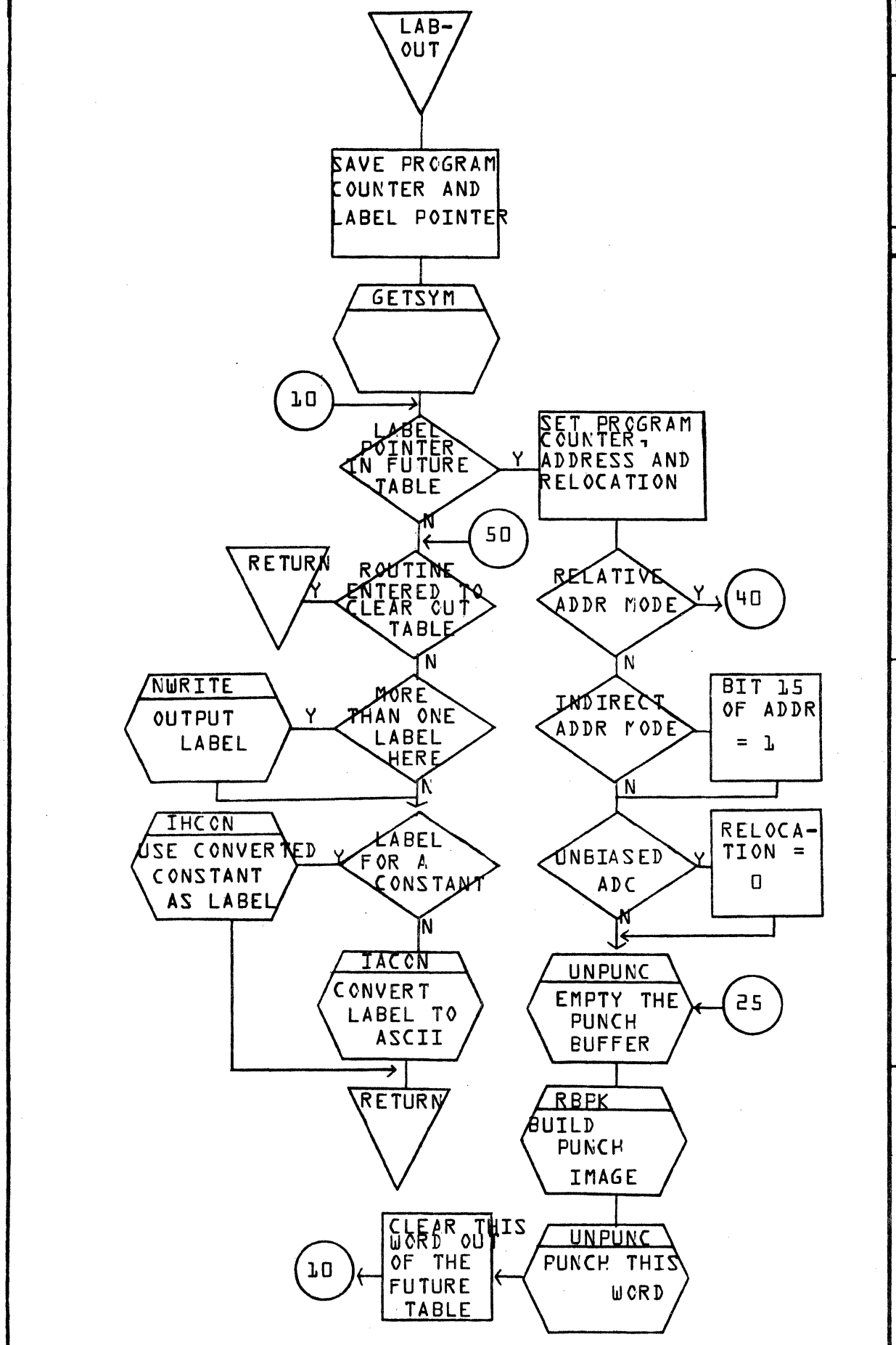
5.2.16 Subroutine RBDX

Subroutine RBDX is called from subroutine RBPK to compute the location and position of a binary word and its associated relocation byte in the binary output record. RBDX is called from subroutine UNPUNC to compute the location and position of the "last word flag" in the binary output record.

5.2.16.1 Flow Chart of Subroutine RBDX

5.2.17 Subroutine RBPK

Subroutine RBPK packs one binary word into the punch buffer. When the buffer is full, subroutine UNPUNC is called to empty the buffer.



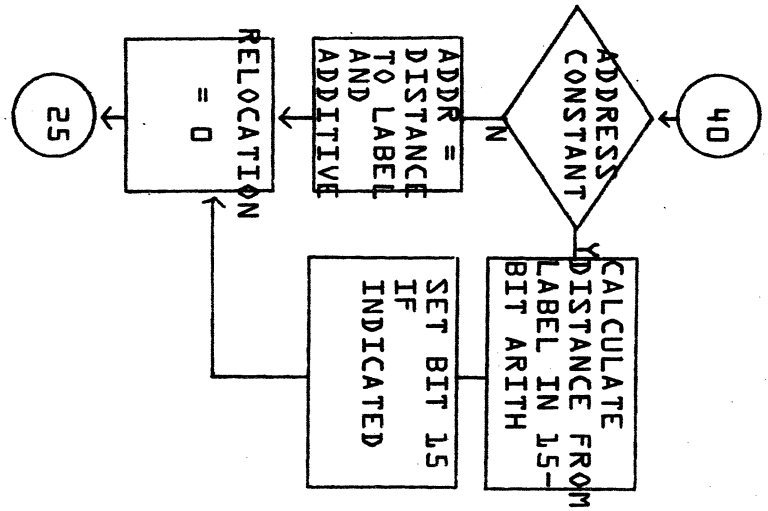
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

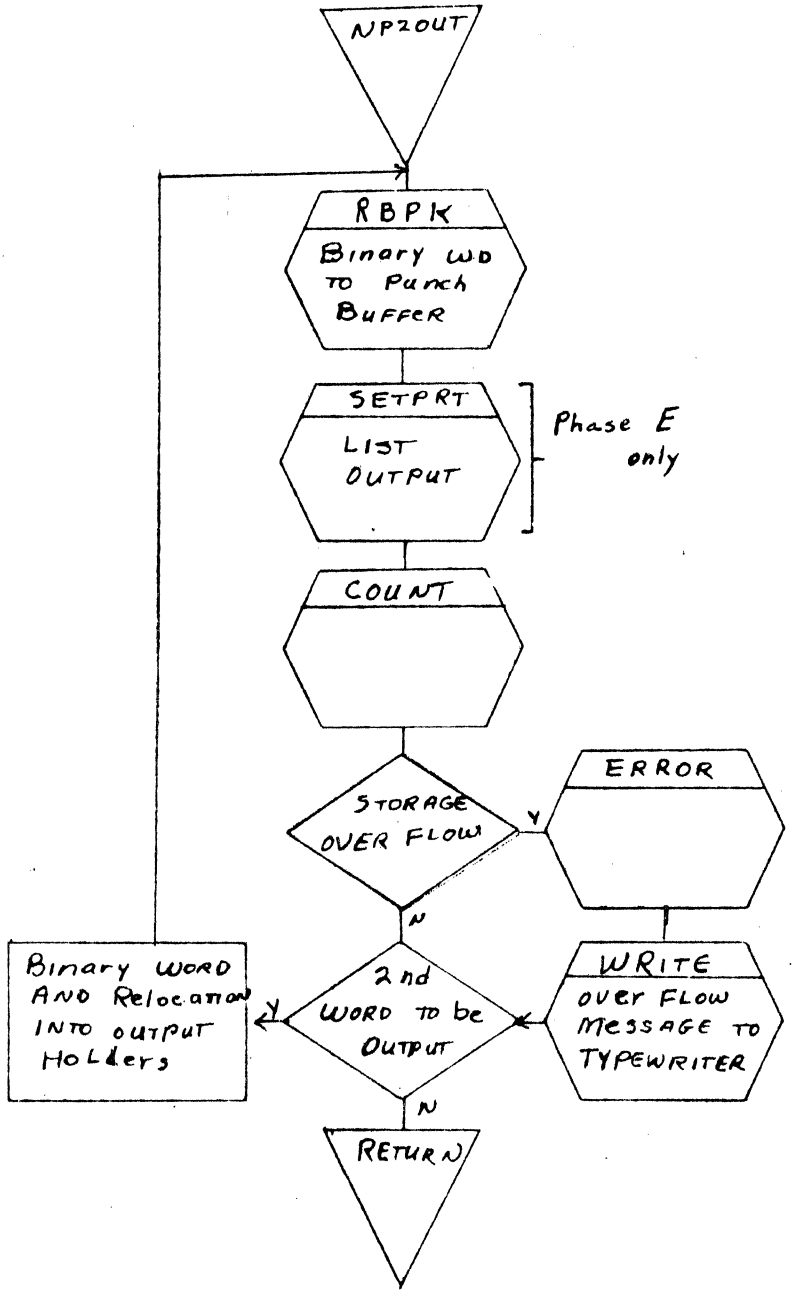
DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE LABOUT
PAGE	1 OF 2
NUMBER	5.2.8.1
ISSUE DATE	
DRAWN BY	
DATE	

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBROUTINE LABOUT	PROJECT MGR.			
	PAGE 2 OF 2	PROJECT NAME			
	NUMBER 5.2.8.1 ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



REV	DATE	APPROVED

PROJECT NO.

PROJECT MGR

PROJECT NAME

TASK NO

DOCUMENT CLASS IMS

MACH. TYPE 1700

DOCUMENT TITLE SUBROUTINE NP2OUT

PAGE 1 OF 1

NUMBER 5.2.10.1

ISSUE DATE

CONTROL DATA CORPORATION

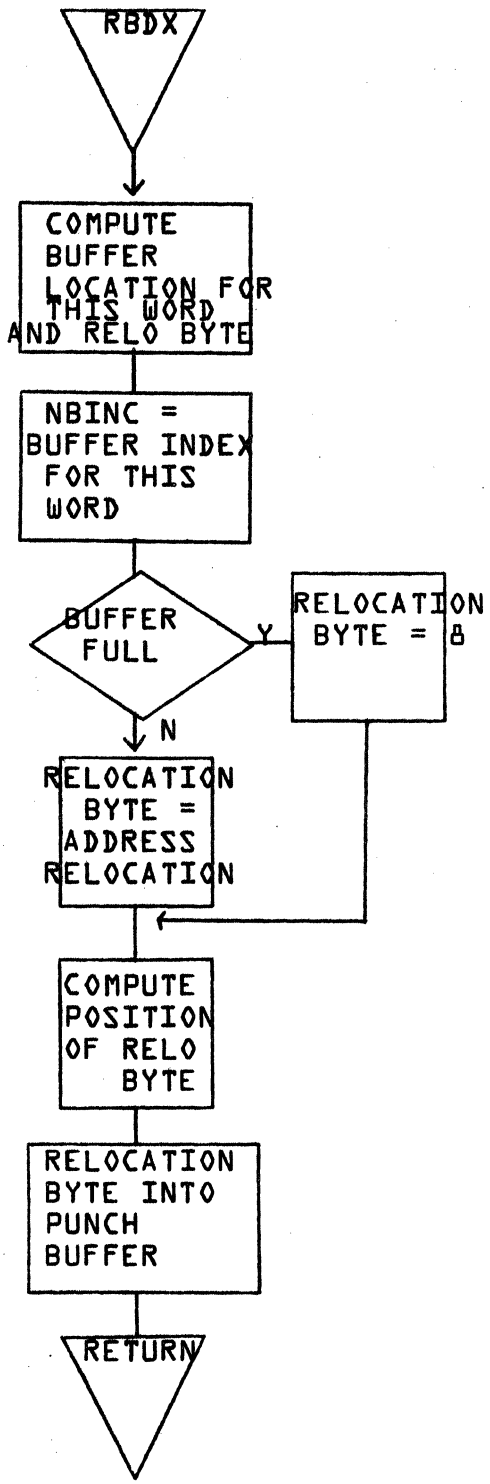
SOFTWARE DOCUMENT

SAMPLE CODE ☐

FLOWCHART ☐

DECISION TABLE ☐

OTHER ☐



REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE RBDX		
		PAGE 1 OF 1	
NUMBER	5.2.13.1	ISSUE DATE	
DRAWN BY		DATE	

PROJECT NO.	PROJECT MGR.	PROJECT NAME	TASK NO.	TASK NAME

AA1385 (FORMERLY CA127-1)

A B C D

5.2.17.1 Flow Chart of Subroutine RBPK

5.2.18 Subroutine SETPRT

Subroutine SETPRT constructs the opcode and address fields of a list output record and calls Subroutine NWRITE to output the record. SETPRT is in Phase E only.

5.2.18.1 Flow Chart of Subroutine SETPRT

5.2.19 Subroutine TABDEC

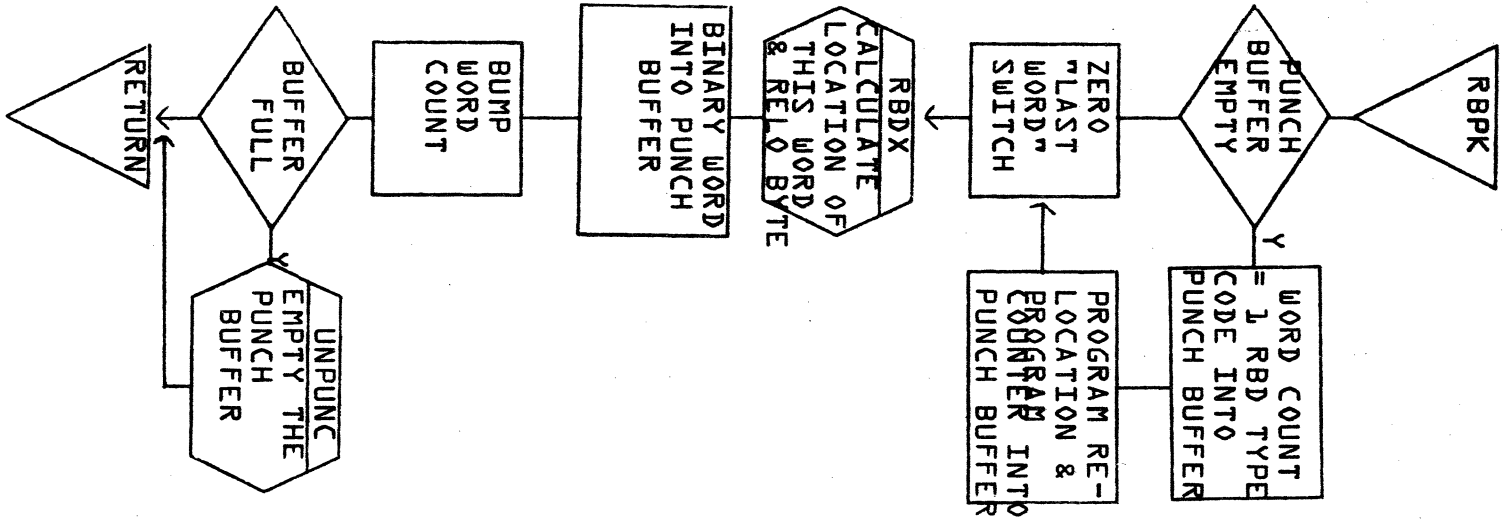
Subroutine TABDEC is called from subroutine ADMAX to decrement addresses that are affected by a change in the output buffer made in ADMAX. The symbol table, and the two tables of address holders, IDVTAB and NDSTAB, are searched for program addresses that are greater than the current program count. All such addresses found are decremented by the amount handed from ADMAX to TABDEC in the decrement holder, IDECR.

5.2.19.1 Flow Chart of Subroutine TABDEC

5.2.20 Subroutine UNPUNC

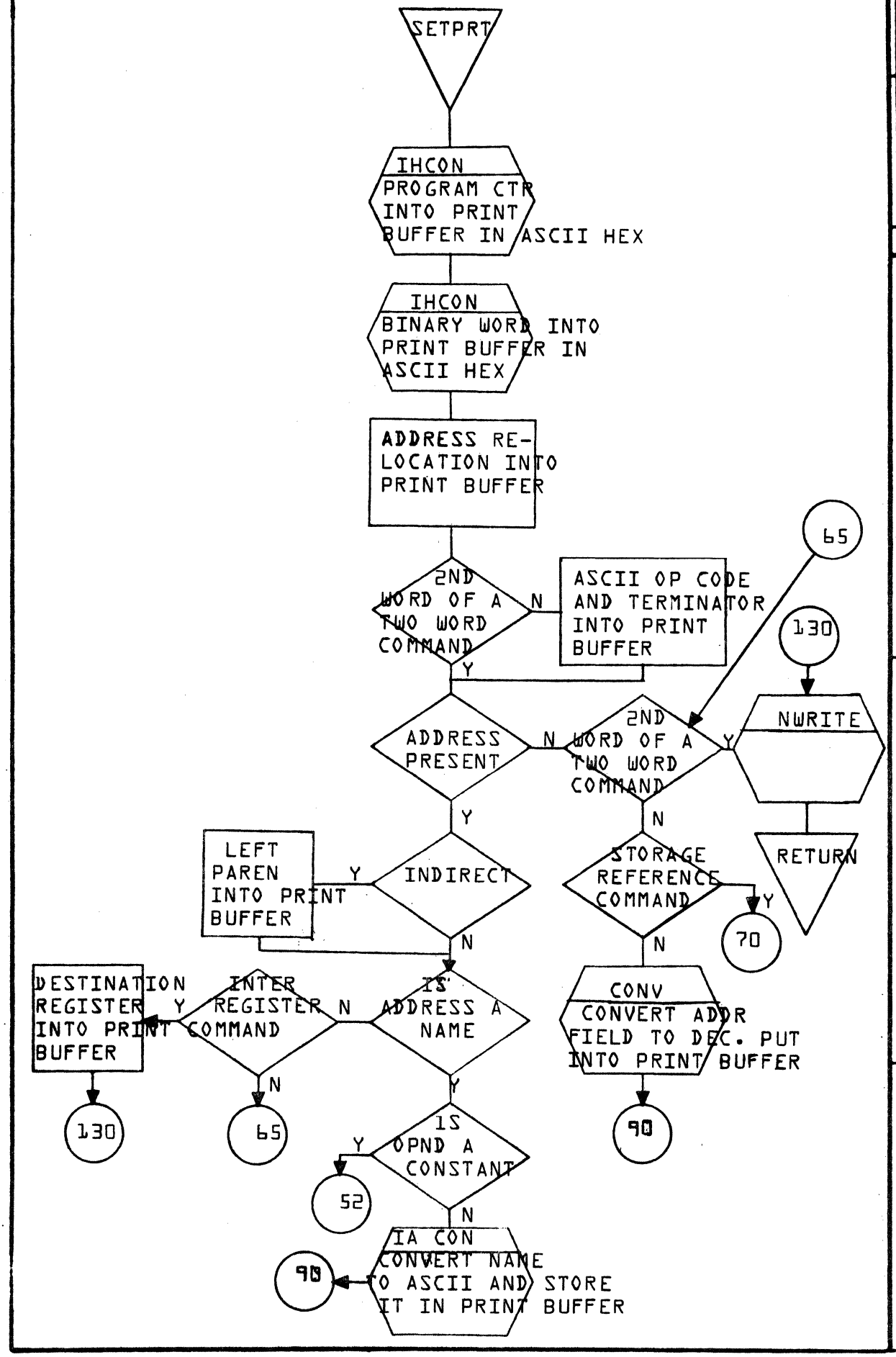
Subroutine UNPUNC calls subroutines RBDX and NPUNCH to output the contents of the punch buffer.

5.2.20.1 Flow Chart of Subroutine UNPUNC



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SUBROUTINE RBPK	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER 5.2.14.1 ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FOR [illegible] Y CA127-1)

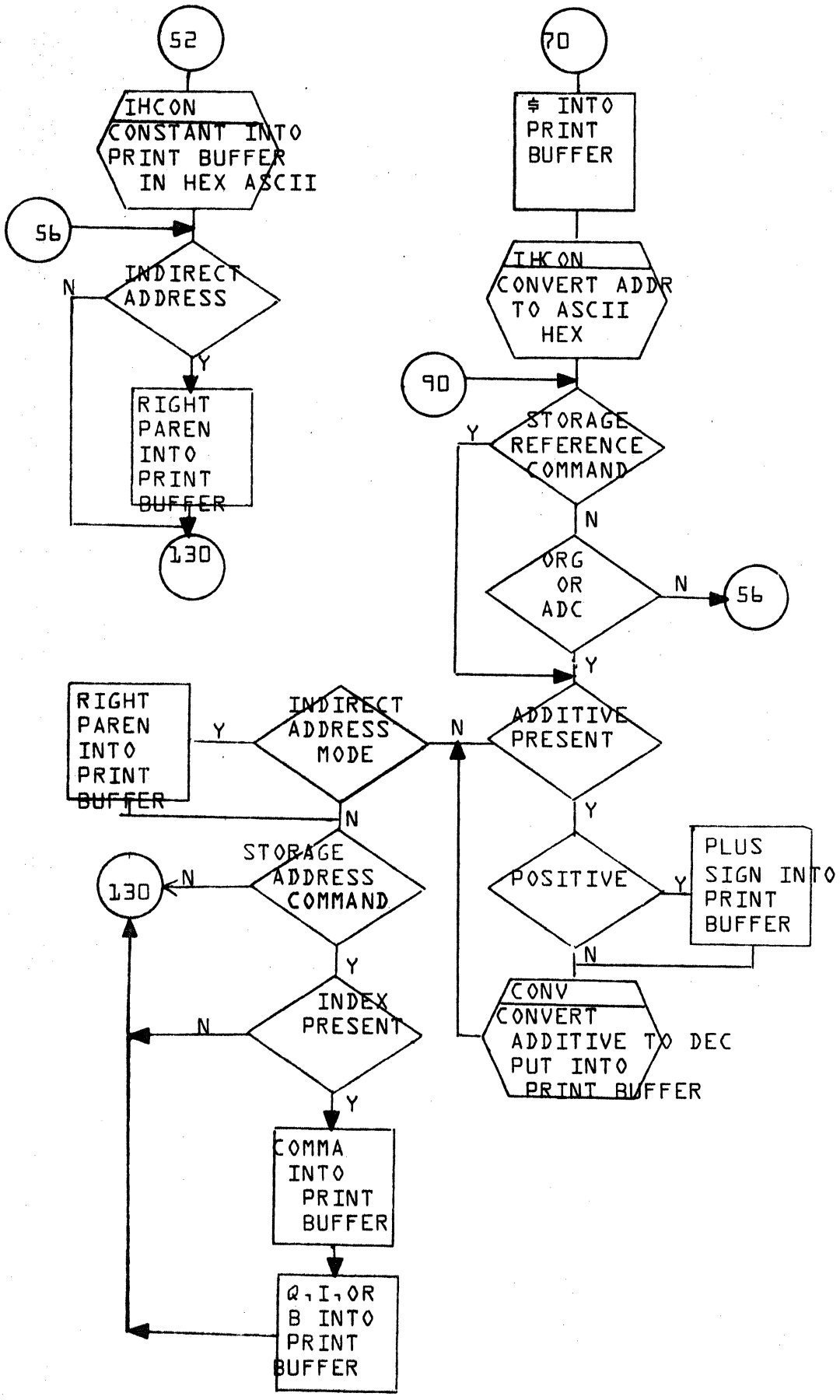


REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: SUBROUTINE SETPRT
PAGE 1 OF 2
NUMBER
ISSUE DATE
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

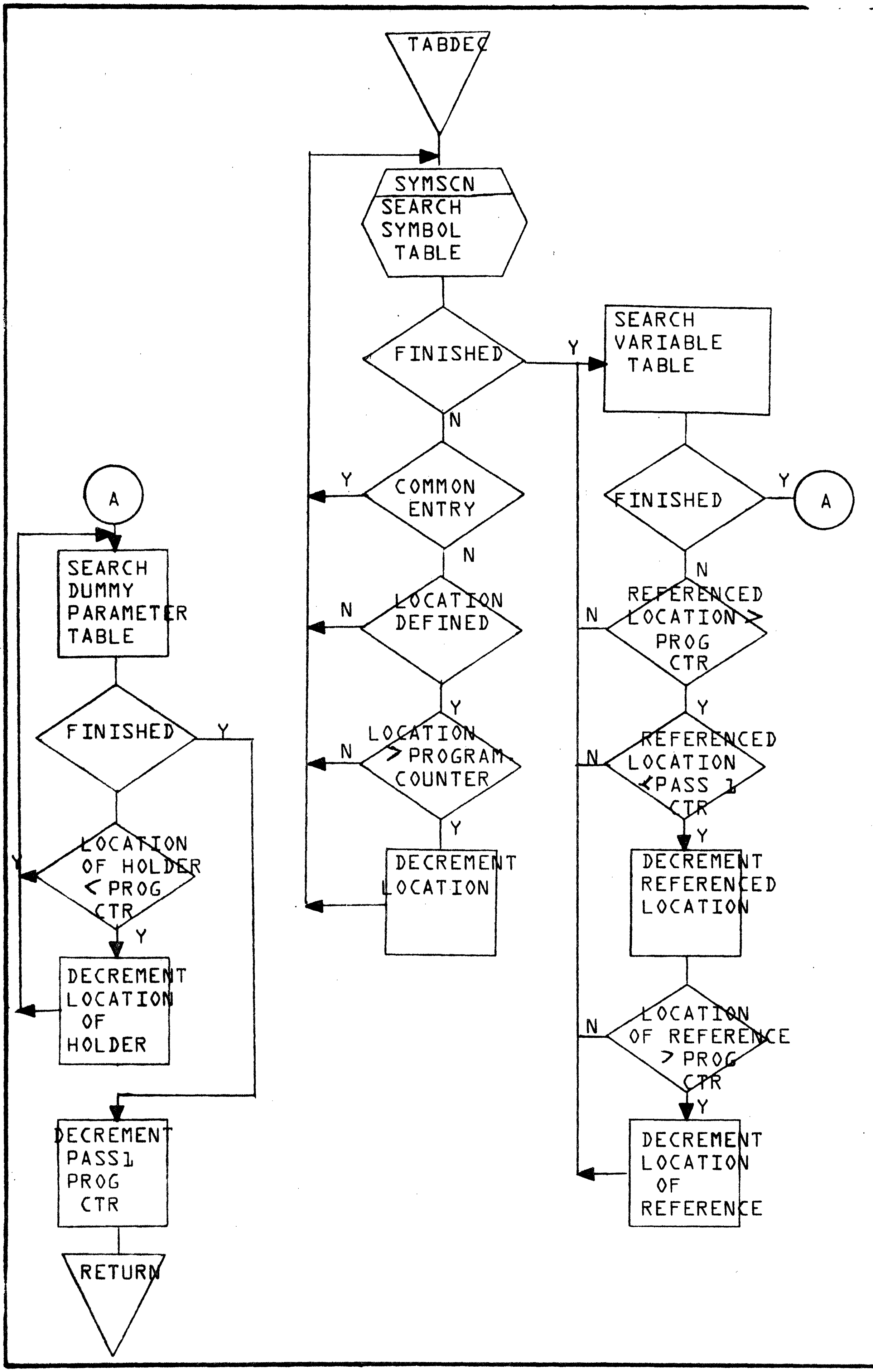
DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	SUBROUTINE SETPRT
PAGE	2 OF 2
ISSUE DATE	
NUMBER	
DRAWN BY	
DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

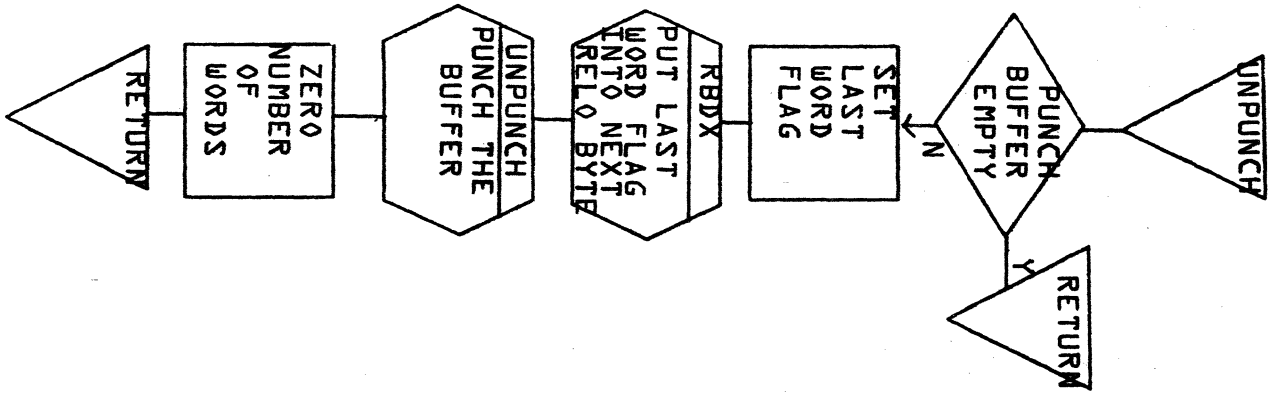
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

AA1385 (FORMERLY CA127-1)

A B C D



CONTROL DATA CORPORATION	SOFTWARE DOCUMENT		
SAMPLE CODE ☐	FLOWCHART ☒	DECISION TABLE ☐	OTHER ☐
DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	
DOCUMENT TITLE: SUBROUTINE TABDEC		PROJECT MGR.	
	PAGE 1 OF 1	PROJECT NAME	
NUMBER	ISSUE DATE	TASK NO.	
DRAWN BY	DATE	TASK NAME	
REV	APPROVED	DATE	



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE SUBROUTINE UNPUNC		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER 5.2.17.1	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

CHAPTER 6

T A B L E O F C O N T E N T S

6.0 OBJECT TIME I/O ROUTINES

6.1 GENERAL DESCRIPTION

I/O Requests of the following forms are handled by the I/O routines:

READ {u,f}L	WRITE {u,f}L	READ {k{n},f}L	WRITE {k{n},f}L
READ {u,f}	WRITE {u,f}	READ {k{n}f}	WRITE {k{n},f}
READ {u}L	WRITE {u}L	READ {k{n}}L	WRITE {k{n}}L
READ {u}	REWIND u	READ {k{n}}	WRITE {k{n}}
BACKSPACE u	ENDFILE u	OPEN {k,i,j,u,x}	
EOF {u}	IOCK {u}	IFALT {m}	

where u is the logical unit number of an input/output device, f is the number of a format statement, L is a list of parameters to be input or output, k is a file number, n is a sector number, i is the number of sectors per record, j is the number of records per file, x is a starting sector number, and m is the type of floating point fault.

6.2 CALLING SEQUENCE FOR I/O REQUESTS

6.2.1 Three Request I/O

I/O requests with parameter lists generate calls to three routines: the initialize routine, the transmission routine, and the terminate routine.

Initialize Routine

The initialize routine is named Q8QINI and has the following calling sequence:

```
RTJ  Q8QINI
```

1. flag word
2. I/O request number
3. unit number or address of unit number
4. format statement address or address of format statement address.

for disk files, words 3 and 4 are:

3. file number or address of file number
4. sector number or address of sector number
5. format information as in 4 for non-disk

The bits of the flag word are designated to represent the following information:

BIT	SETTING
15	Not Used

BIT	SETTING
14	1 formatted 0 unformatted
13	1 unit number 0 address of unit number
12	1 format statement address 0 address of format statement address
11	1 list 0 no list
10	1 read 0 not read
9	1 write 0 not write
8	1 disk I/O 0 no disk I/O
7	1 sector number address 0 address of sector number address
6-0	not used

Transmission Routine

The transmission routine is named Q8QX and has the following calling sequence:

```
RTJ     Q8QX

        1.  address of list element
```

Terminate Routine

The terminate routine is named Q8QEND and has the following calling sequence:

```
RTJ     Q8QEND
```

6.2.2 One Request I/O

I/O requests without parameter lists generate only a call to the initialize routine. The calling sequence is the same as described for Q8QINI above.

6.2.3 Auxiliary Input/Output Statements Calling Sequence

```
REWIND u
     RTJ    Q8QWND
            1. address of u

BACKSPACE u
     RTJ    Q8QBCK
            1. address of u

END FILE u
     RTJ    Q8QFLE
            1. address of u

OPEN - allocate disk addresses and open disk file
     RTJ    Q8QDFNF
            1. I/O request number
            2. address of file number
            3. address of number of sectors per word
            4. address of number of records per file
            5. address of logical unit number
            6. address of starting sector number
```

6.2.4 Basic External Functions

End of file check EOF

```
     RTJ    EOF

            1. address of u
```

I/O error check IOCK

```
     RTJ    IOCK

            1. address of u
```

Floating point error condition check IFALT

```
     RTJ    IFALT

            1. number 0, 1, 2 in binary
```

6.2.5 Special Calls

Pause and stop statements

PAUSE

```
     RTJ    Q8QPSE
```

PAUSE n

```
RTJ    Q8QPSEN
```

1. octal number n in binary

STOP

```
RTJ    Q8STP
```

STOP n

```
RTJ    Q8STPN
```

1. octal number n in binary

6.3 DISCUSSION BY STATEMENT TYPE

6.3.1 READ and WRITE

I/O requests of the form READ and WRITE are handled by the same I/O routines. The initialize routine, Q8QINI, interprets the information in the calling sequence and initializes the required routines. The transmission routine transmits the address of the element to the processing routine. If no FORMAT statement is involved, the processing routine in Q8BINB; otherwise Q8IFRM is the processing routine. When all the elements have been processed, Q8QEND is called to terminate the input or output.

READ and WRITE requests without a list are handled by the initialize routine, Q8QINI. After the calling sequence has been interpreted, Q8QINI calls the required processing routine. When the request has been processed, Q8QINI calls Q8QEND to terminate the input or output.

6.3.2 Auxiliary Input/Output Statements

A REWIND u statement is handled by the routine Q8QWND. A status request is made to determine if the unit may be rewound. If the statement is illegal, error message 6 is typed and exit is made to the Utility System. If the statement is legal, a rewind request is made.

A BACKSPACE u statement is handled by the routine Q8QBCK. A status request is made to determine if the unit may be backspaced. If the unit may not be backspaced, error message 6 is typed out and exit is made to the Operating System.

An END FILE u statement is processed by the routine Q8QFLE. A status request is made to determine if the unit is capable of writing an end of file (EOF). If an EOF may not be written, error message 6 is typed and exit is made to the Operating System.

Q8QWND, Q8QBCK, and Q8QFLE are grouped together in the program names TAPCON. They make use of common subroutines.

An OPEN d, i, j, u, x request is handled by the routine Q8DFNF which is in the program Q8DFIO. A table entry containing the file number, logical unit, and starting sector number is set up and the file space used is allocated. Several errors can result if the parameters are incorrect. The table entry is used by subsequent READ or WRITE requests for that file.

6.3.3 Basic External Functions

EOF(u) function is serviced by the routine EOF. If bit 12 of the unit history table entry for u is set, an end of file has been written by the previous WRITE request. The A register will contain one upon exit from EOF. Bit 12 not set results in the A register containing two upon exit from EOF.

EOF is grouped with Q8QFLE, Q8QWND, and Q8QBCK in the program TAPCON.

IOCK(u) function is serviced by the routine IOCK. If bit 2 of the unit history table entry for u is set, an error occurred during a previous I/O operation. The A register contains one upon exit from IOCK. Bit 2 not set results in the A register containing two upon exit from IOCK.

IFALT(I) function is serviced by the routine IFALT. The input parameter I may take on the following meanings:

I = 0	Check for overflow
I = 1	Check for divide fault
I = 2	Check for underflow

The values that the parameter may return are:

I = 1	Condition occurred
I = 2	Condition did not occur

6.3.4 Special Calls

A PAUSE statement is handled by the routine Q8PSE. The message PAUSE is typed out and a typewriter read is executed. The program continues when a carriage return is typed in. PAUSE n is handled by the routine Q8PSEN which is the same as Q8PSE except PAUSE n, where n is a 5 digit octal number, is typed out.

A STOP statement is handled by the routine Q8STP. The message STOP is typed out and an exit request is made to the Operation System. STOP n is handled by Q8STPN which is identical to Q8STP except STOP n, where n is a 5 digit octal number, is typed out.

Q8PSE, Q8PSEN, Q8STP, and Q8STPN are grouped together in the program named PSSTOP.

6.4 DISCUSSION BY ROUTINE

6.4.1 Q8QINI Routine

The purpose of this routine is to interpret the calling sequence and initialize the appropriate routines. The routine is written in 1700 assembly language.

Three entry points are declared: Q8QINI, Q8UNIT, and Q8SKIP. The calling sequence to Q8QINI is described in 6.2.1. Q8UNIT contains the unit number of the I/O request. Q8SKIP is called whenever the next record for formatted I/O is to be input or output.

The following externals are declared:

Q8ERRM in Q8ERRM
Q8INTB in Q8RWBU
Q8EREM in Q8ERRM
Q8TOM in Q8QX
Q8CMP0 in Q8CMP
Q8MAGT in Q8MAGT
Q8QEND in Q8QEND
Q8IFRM in Q8IFRM
Q8IGP in Q8FGET
Q8CMP1 in Q8CMP
Q8QUN2 in Q8QUNI

6.4.2 Q8QX Routine

The purpose of this routine is to transmit the address of the element and call the appropriate processing routine (binary or format). The routine is written in 1700 Assembly Language.

The following entry points are declared: Q8QTOM, Q8QX{Q8QTRM}, Q8QY, Q8QZ, and Q8MOVE.

Q8QTOM calling sequence:

```
LDA     flag word
RTJ     Q8QTOM
```

Q8QX {Q8QTRM} is described in 6.2.1.

Q8MOVE calling sequence:

```
RTJ     Q8MOVE
ADC     value
ADC     switch
```

For Binary Transmission the following rules apply:

A. RTJ Q8QX {one word transmission}

1. address of integer list element

B. RTJ Q8QY {two word transmission}

1. address of single precision floating point element

C. RTJ Q8QZ {three word transmission}

1. address of double precision floating point element

For read, value is moved to the address of the list element plus switch. For write, the list element plus switch is moved to the address of value.

The following externals are declared:

Q8IFRM in Q8IFRM
Q8BINB in Q8RWBU
Q8QUN1 in Q8QUN1
Q8UNIT in Q8QINI

6.4.3 Q8QUNI Routine

This routine is composed of three subroutines to insert data into and fetch data out of the unit history table. The routine is written in 1700 assembly language. The following entry points are declared: Q8QUN1, Q8QUN2, and Q8QUN3.

Q8QUN1 calling sequence:

```
LDQ  unit
RTJ  Q8QUN1
```

Return with A register containing unit history table entry

Q8QUN2 calling sequence:

```
LDA  value
LDQ  unit
RTJ  Q8QUN2
```

Value is logically ORed into the unit history table entry

Q8QUN3 calling sequence:

```
LDA  value
LDQ  unit
RTJ  Q8QUN3
```

Value is stored in the unit history table entry

The following external is declared:

Q8EREM in Q8ERRM

6.4.4 Q8CMP Routine

The routine is written in 1700 assembly language. Q8CMP has 2 entry

points, Q8CMP0 and Q8CMP1. Q8CMP0 initializes the input/output request. Q8CMP1 implements the input/output request retaining a composite error history which is left in the A register upon return, when entry is from Q8QEND.

Q8CMP0 calling sequence:

```
LDA  flag word
LDQ  unit
RTJ  Q8CMP0
```

Q8CMP1 calling sequence:

```
LDA  indicator (non zero if entry from Q8QEND, otherwise 0)
RTJ  Q8CMP1
```

Return with A = composite error history (if indicator non zero)

The following externals are declared:

```
Q8EREM in Q8ERRM
Q8BEGB in Q8RWBU
Q8LOCB in Q8RWBU
Q8CLRB in Q8RWBU
Q8RINT in Q8RWBU
```

6.4.5 Q8RWBU Routine

This routine is written in 1700 assembly language. Q8RWBU has seven entry points: Q8IBUF, Q8INTB, Q8LOCB, Q8BEGB, Q8CLRB, Q8RWBU, and Q8BINB. Q8IBUF is the first word in the physical record buffer. As such Q8IBUF is zero for all physical records except for the last physical record when Q8IBUF is equal to the number of physical records in the logical record.

Q8INTB initializes buffer counters and, if the request is a READ, inputs the first record from the I/O device. Q8LOCB loads the address of the current buffer location into the A register. Q8BEGB loads the address of the first buffer location into the A register. Q8CLRB backgrounds the buffer with zeros for binary I/O and ASCII blanks for formatted I/O. Q8RINT reinitializes the buffer counters.

Q8RWBU loads a character from the conversion routine into the buffer for WRITE requests and from the buffer to the conversion routine for READ requests.

Q8BINB loads a binary word from the list into the buffer for WRITE requests and from the buffer into the list for READ requests.

Q8INTB calling sequence:

```
LDA  flag word
RTJ  Q8INTB
```

Q8LOCB calling sequence:

```
RTJ  Q8LOCB
```

Return with A = address of current buffer location

Q8BEGB calling sequence:

```
RTJ  Q8BEGB
```

Return with A = address of beginning buffer location

Q8CLRB calling sequence:

```
RTJ  Q8CLRB
```

Q8RINT calling sequence:

```
RTJ  Q8RINT
```

Q8RWBU calling sequence:

```
CALL  Q8RWBU {CHAR}   or   RTJ  Q8RWBU
                           ADC  CHAR
```

CHAR is the location of the character to be stuffed into the buffer {WRITE} or the location to stuff the next buffer character into {READ}.

In a statement of the form

```
CALL   Q8RWBU {CHAR}
```

CHAR may be a decimal number whose corresponding hexadecimal number gives the ASCII symbol to be stuffed into the buffer {WRITE}.

Q8BINB calling sequence:

```
RTJ  Q8BINB
```

The following externals are declared:

```
Q8CMP1 in Q8CMP
Q8EREM in Q8ERRM
```

6.4.6 Q8FGET Routine

This routine is written in 1700 assembly language. Q8FGET consists of four entry points: Q8L0CF, Q8IGP, Q8FGET, and Q8FPUT. Q8L0CF loads the character counter into the A register. Q8IGP clears the

character counter and stores the FORMAT statement address. Q8FGET fetches a character from the FORMAT statement. Q8FPUT stores a character into the FORMAT statement.

Q8LOCF calling sequence:

```
RTJ  Q8LOCF
```

Return with A = character counter

Q8IGP calling sequence:

```
LDA  format statement address
RTJ  Q8IGP
```

Q8FGET calling sequence:

```
CALL Q8FGET (ADR, SAVE, REP) or  RTJ  Q8FGET
                                 ADC  ADR, SAVE, REP
```

Where ADR is the address where the FORMAT statement character is to be stored; SAVE is a flag - 0 means ignore, 1 means save the current character count less 1 in FR, 2 means save the current character count less 1 in GR; REP is a flag - 0 means ignore, 1 means set the character count to FR, 2 means set the character count to GR.

Q8FPUT calling sequence:

```
CALL Q8FPUT (ADR)  or  RTJ  Q8FPUT
                       ADC  ADR
```

Where ADR is the address of the character to be stored into the FORMAT statement.

There are no externals declared by Q8FGET.

6.4.7 Q8ERRM Routine

The routine is written in 1700 assembly language. The purpose of this routine is to process error messages. Three entries are declared: Q8ERRM, Q8EREM, and Q8FERM. Q8ERRM converts the I/O request number and unit number from binary to ASCII decimal. Q8EREM processes error messages from 1700 assembly language routines. Q8FERM processes error messages from 1700 FORTRAN language routines.

Q8ERRM calling sequence:

```
LDA  I/O request number
LDQ  unit
RTJ  Q8ERRM
```

Q8EREM calling sequence:

```
LDA  error type
RTJ  Q8EREM
```

Q8FERM calling sequence:

```
CALL Q8FERM (I)  or  RTJ  Q8FERM
                     ADC  I
```

Where I is the error type number.

The following externals are declared:

```
Q8LOCF in Q8FGET
Q8LOCB in Q8RWBU
```

6.4.8 Q8MAGT Routine

The purpose of this routine is to check the status of the magnetic tape unit specified for the READ/WRITE request. The routine is written in 1700 assembly language. There is one entry: Q8MAGT.

Q8MAGT calling sequence:

```
LDQ  flag word
LDA  unit
RTJ  Q8MAGT
```

The following externals are declared:

```
Q8EREM in Q8ERRM
Q8QUN2 in Q8QUNI
Q8COMI in PSSTOP
Q8QWND in TAPCON
```

6.4.9 Q8IFRM Routine

The purpose of this routine is to initialize certain cells and call either the format scanner (Q8FS) or the conversion and transmission routine (Q8TRAN) based on the setting of the input parameter IENTY. It is coded in FORTRAN and is defined as follows:

SUBROUTINE Q8IFRM (IENTY, IRW, ILIST)

The input parameters have the following meaning:

IENTY - Switch which determines the phasing of the call. 0 = I/O list just beginning to be processed (entered from Q8QINI). 1 = I/O list being processed (entered from Q8QX).

IRW - Read/Write Switch. 0 = Read, 1 = Write.

ILIST - Any list associated with this I/O. 1 = list, 0 = no list.

6.4.10 Q8FS Routine

The purpose of this routine is to scan the FORMAT statement, interpret individual subfields, and pass the resulting field specification parameters to the conversion and transmission routine, Q8TRAN. It is coded in FORTRAN and is defined as follows:

SUBROUTINE Q8FS (IRW, IFW, IND, ITYPE, IRF, IRG, IBCT, ISWR, ILIST, IANYL)

The input parameters have the following meaning:

IRW - Read/Write switch. 0 = Read, 1 = Write.
IFW - Number of characters in I/O field.
IND - Number of characters to right of decimal point.
ITYPE - Type of conversion to be done
1 = A
2 = R
3 = E
4 = F
5 = $
6 = I
7 = D
IRF - Number indicating the number of times the current field is to be repeated.
IRG - Number indicating the number of times the current group is to be repeated.
IBCT - Number indicating the parenthesis count. +1 for all (and -1 for all).
ISWR - When this switch = 2 a new I/O is performed, otherwise not. It is a delayed read/write switch.
ILIST - 0 = no list; 1 = list associated with this I/O statement.
IANYL - 0 no conversion specification exists in FORMAT statement. Non-zero = at least 1 conversion specification exists.

6.4.11 Q8TRAN Routine

The purpose of this routine is to convert the next subfield for I/O based on the appropriate FORMAT specification and transmit the subfield to or from core. It controls all I/O done under FORMAT control.

It is coded in FORTRAN and is defined as follows:

```
SUBROUTINE Q8TRAN (IRW, IFW, IND, ITYPE, IRF, IRG, IBCT, ISWR,
ILIST, IANYL)
```

The input parameters have the same meaning as described in section 6.4.10 for Q8FS.

6.4.12 Q8QEND Routine

The purpose of Q8QEND is to terminate the READ/WRITE request and store the cumulative error history into the unit history table. The routine is written in 1700 assembly language. The only entry is Q8QEND.

Q8QEND calling sequence:

```
RTJ  Q8QEND
```

The following externals are declared:

```
Q8CMP1 in Q8CMP
Q8QUN1 in Q8QUNI
Q8QUN2 in Q8QUNI
Q8UNIT in Q8QINI
```

6.4.13 TAPCON Routine

TAPCON consists of magnetic tape function routines. The routine is written in 1700 assembly language. Entries are Q8QBCK, Q8QFLE, Q8QWND, and EOF. Calling sequences and functions are described in 6.2.3, 6.2.4, 6.3.2, and 6.3.3.

The following externals are declared:

```
Q8EREM in Q8ERRM
Q8CMP0 in Q8CMP
Q8CMP1 in Q8CMP
Q8QUN1 in Q8QUNI
Q8QUN2 in Q8QUNI
Q8QUN3 in Q8QUNI
Q8IBUF in Q8QINI
```

6.4.14 IOCK Routine

The routine is written in 1700 assembly language. IOCK is a routine to check for any I/O error. It is described in 6.2.4 and 6.3.3. It has one entry point: IOCK.

The following externals are declared:

Q8QUN1 in Q8QUNI
Q8QUN3 in Q8QUNI

6.4.15 PSSTOP Routine

PSSTOP consists of the special calls Q8PSE, Q8PSEN, Q8STP, Q8STPN, described in 6.2.5 and 6.3.4 plus the entry Q8COMI. The routine is written in 1700 assembly language. Q8COMI reads the teletypewriter for one character.

Q8COMI Calling Sequence

```
RTJ  Q8GOMI
```

One external is declared: Q8PAND

6.4.16 Q8PAND Routine

The purpose of Q8PAND is to type PAUSE or STOP followed by 5 octal digits if specified. The routine is written in 1700 assembly language. It has one entry point, Q8PAND.

Q8PAND calling sequence:

```
LDA  NUM
LDQ  ADR
RTJ  Q8PAND
```

Where NUM is a binary number to be converted to 5 digit ASCII octal, or, if NUM 0, to be ignored. ADR is the address of PAUSE or STOP in ASCII.

There are no externals.

6.4.17 Subroutine Q8EXP9

Routine converts a binary integer and decimal exponent into a floating point number. The integer is assumed to be greater than 0. Exponents may be positive, negative, or zero.

Input parameters: P address of 2 word binary integer
P contains most significant 16 bits
P + 1 contains least significant 12 bits, right justified
P1 address of decimal exponent

Output parameters: P, P+1 contains floating point number (0 if underflow has occurred, $7FFF, $FFFF if overflow has occurred)
P1 error indicator
0 implies no error
≠0 implies error

Routines Used: FLOT

Used by: Q8TRAN (FORTRAN object time conversion routine)

Language: 1700 Assembly Language

6.4.18 Subroutine Q8EXP1

Converts a floating point number to an octal fraction with decimal exponent.

Input parameter: P address of 4 word table P+1, P+2 contain floating point number to convert

Output parameters: P+1, P+2 contain octal fraction (lower 12 bits of each word)
P+3 value of decimal exponent

Used by: Q8TRAN

Routines used: FLOT

Other externals: Q8EXPT (in Q8EXP9)

Language: 1700 Assembly Language

6.4.19 Q8DFIO Routine

Q8DFIO which is written in 1700 assembly language, handles the disk I/O necessary for OPEN statement. This routine may be split into two separate parts.

The first part, which has the entry point Q8DFIN, is called by Q8QINI to initialize the disk I/O.

Q8DFIN calling sequence:

```
RTJ   Q8DFIN
```

where

A contains the file number
Q contains the flag word shifted 7 left

This routine finds the corresponding file table entry, computes the starting sector number and places it in Q8DFAD (in Q8CMP) and returns with the logical unit in A.

The second part, which has the entry point Q8DFNF, is entered by a compiler generated call from the user's program when executing an OPEN statement.

Q8DFNF calling sequence:

```
RTJ   Q8DFNF
      1.  address of file number
      2.  address of number of sectors per record
      3.  address of number of records per file
      4.  address of disk logical unit number
      5.  address of starting sector number
```

The disk file table starts in the low address of unused core and extends upwards. Three words are created for each file. The format is as follows:

```
WORD 0:  file number
WORD 1:  logical unit number
WORD 2:  starting sector address (x)
```

The following externals are declared in Q8DFIO:

```
Q8ERRM  in Q8ERRM
Q8EREM  in Q8ERRM
Q8QINI  in Q8QINI
Q8DFAD  in Q8CMP
Q8QENS  in Q8CMP
```

6.4.20 IFALT Routine

This routine checks for a floating point error condition.

The input parameter I may take on the following values:

I =	0	Check for overflow
	1	Check for divide fault
	2	Check for underflow

The parameter I may return the following values:

I =	1	Condition occurred
	2	Condition did not occur

This routine, which is written in 1700 Assembly Language, uses the contents of location $C8 in checking for the above conditions. The routines FLOAT and DFLOT store their error condition information in this low core location.

There are no externals.

6.4.21 Subroutine Q8DXP9

This routine converts a binary integer and decimal exponent into a double precision (3 word) floating point number. The integer is assumed to be greater than zero. Exponents may be positive, negative, or zero.

Input parameters:

P	address of 4 word binary integer
P	contains the most significant 4 bits
P+1	contains the next intermediate significant 12 bits right justified
P+2	contains the next intermediate significant 12 bits right justified
P+3	contains the least significant 12 bits right justified
P1	address of decimal exponent

Output parameters:

P, P+1, P+2	contains the double precision point number (0 if underflow has occurred $7FFF, $FFFF, $FFFF if overflow has occurred.
0	implies no error
≠0	implies error

Routines Used: DFLOT

Used by: Q8TRAN {FORTRAN object time conversion routine}

Language: 1700 Assembly Language

6.4.22 Subroutine Q8DXP1

This routine obtains a double precision floating point number and converts it to an intermediate form of a decimal integer and a decimal exponent.

Input parameter:
P address of 6 word table
P+1, P+2, P+3 } contain the double precision floating point number to be converted

Output parameter:
P+1, P+2, P+3, P+4 } contain the intermediate binary fraction {Lower 12 bits of each word} {only the upper 4 bits of the lower 12 bits are used}
P+5 contains the decimal exponent

Used by: Q8TRAN

Routines Used: DFLOT

Other Externals: Q8DXPT and Q8DXP2 {in Q8DXP9}

Language: 1700 Assembly Language

6.5 I/O Restrictions

The number of logical units which may be used in one program is limited to 30. This number is introduced to limit the number of cells necessary for storage of the unit history table. The unit history table must be available to give information on the EOF and IOCK requests.

Each time a new logical unit is used in an I/O request, an entry will be made in the table. When use of more than 30 units is attempted, an error message will by typed. The transmission information on the I/O request is placed in the table upon completion of the request.

6.5.1 Structure of Unit History Table Entry

BIT	
15	Not used
14	1 Formatted 0 Unformatted
13	1 Rewind 0 Non rewind
12	1 EOF Read 0 No EOF Read
11	1 Backspace 0 Not backspace
10	1 Read 0 Not read
9	1 Write 0 Not write
8-3	Not used
2	1 Error occurred 0 Error free request
1-0	Not used

6.6 ERROR MESSAGE

6.6.1 General Discussion

Error messages will be output on the standard comment medium. The message form will be one of the following three types:

Type 1:
```
N
I/O RQST n
ffff
```

Type 2:
```
N
I/O RQST n
ffff
gggg
```

Type 3:
```
N
I/O RQST n
XX
```

1. N is an error number whidh corresponds to an error message.
2. n is the I/O request statement number.
3. ffff is the current value of the format statement pointer. This value is in decimal.
4. gggg is the current value of the input field pointer. This value is in decimal.
5. xx is the decimal unit number of an improperly used device.

6.6.2 An Error in the FORMAT Statement

MESSAGE: 1
I/O RQST n
ffff

Illegal character in format statement

Action: program termination

6.6.3 Illegal Character in the Input Field

MESSAGE: 2
I/O RQST n
ffff
gggg

Action: program termination

6.6.4 Input Data Exceeds Limits of 1700 Word

MESSAGE: 3
I/O RQST n
ffff
gggg

Action: program termination

6.6.5 Improper Use of Unit

MESSAGE: N
I/O RQST n
unit number

N	Description of Improper Use	Action
4	Attempt to read on a write unit or write on a read unit	Program termination
5	Read or write request after an end of file has been written without first doing an EOF check.	Program termination
6	Attempt to write an EOF, rewind, or backspace any unit other than a magnetic tape	Program termination
7	Write attempted on magnetic tape with no write enable	Type (cr) to continue
8	Attempt to use logical unit number greater than 30	Program termination
9	Backspace at load point	Program termination
10	End of magnetic tape sensed	Rewind tape. Type (cr) to continue

6.6.7 Illegal Formatted Input

More elements are given than are contained in an input record.

```
MESSAGE:    12
            I/O RQST n
            ffff
Action:  program termination
```

6.6.8 Illegal List

A list is given but there are no conversion codes in the format statement.

```
MESSAGE:    13
            I/O RQST n
            ffff
Action:  program termination
```

6.6.9 File defined twice

More than one OPEN request given for the same file.

MESSAGE: 14
I/O RQST n
file number

ACTION: Program is terminated.

6.6.10 Parameter negative or zero

One of the parameters in an OPEN statement is negative or zero.

MESSAGE: 15
I/O RQST n
file number

ACTTON: Program is terminated

6.6.11 Sector address too large

The starting sector address or ending address exceeds $2^{15}-1$.

MESSAGE: 16
I/O RQST n
file number

ACTTON: Program is terminated

6.6.12 File not defined

A READ or WRITE request was given for a file which was not defined by an OPEN statement.

MESSAGE: 17
I/O RQST n
file number

ACTION: Program is terminated

6.6.13 Logical unit not a mass storage device

MESSAGE: 18
I/O RQST n
file number

ACTION: Program is terminated

6.6.14 Record number in READ or WRITE request incorrect. Resulting sector address is out of the range of the file or is zero.

MESSAGE: 19
. I/O RQST n
file number

ACTION: Program is terminated

6.7 RECORDS

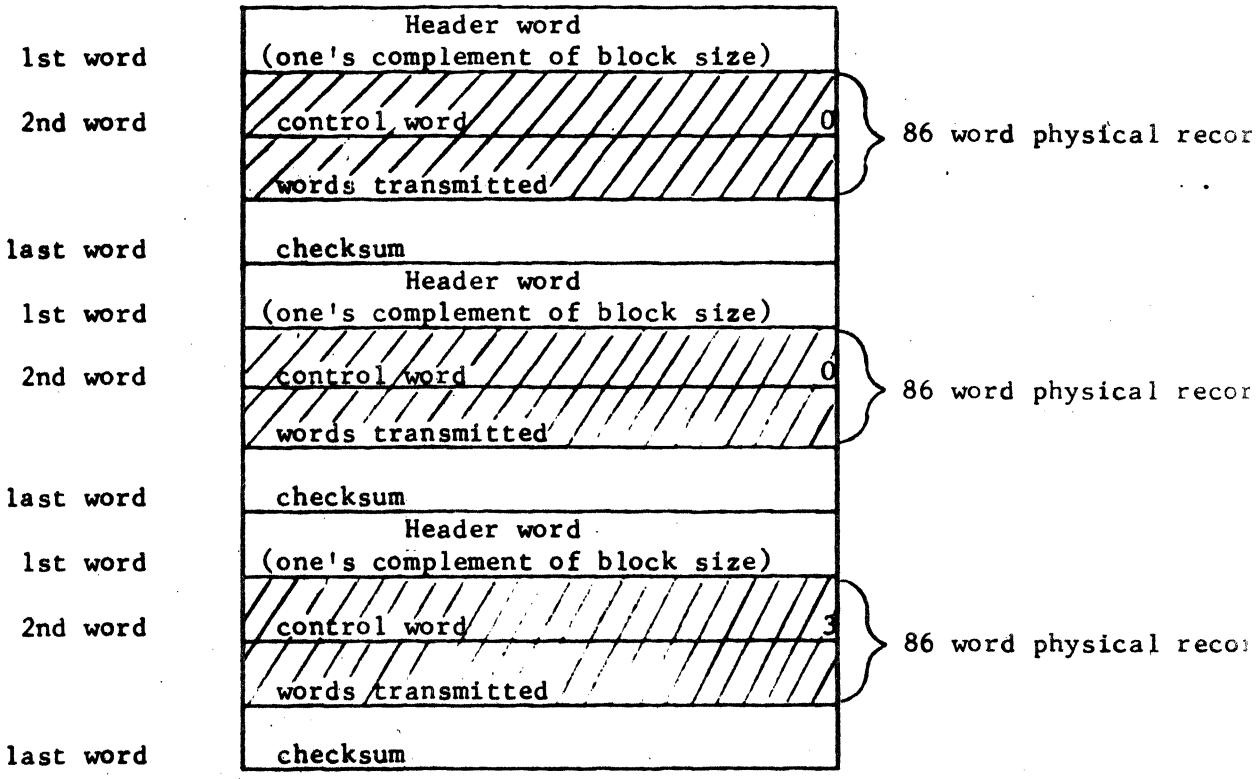
6.7.1 Binary READ-WRITE Statements

A binary write produces a logical record which may be composed of any number of physical records.

A READ (u) statement spaces the tape one logical record.

A WRITE (u) statement is illegal.

The first word will be interpreted as a control word and does not represent data. A WRITE (u)L will produce on paper tape a logical record as follows:



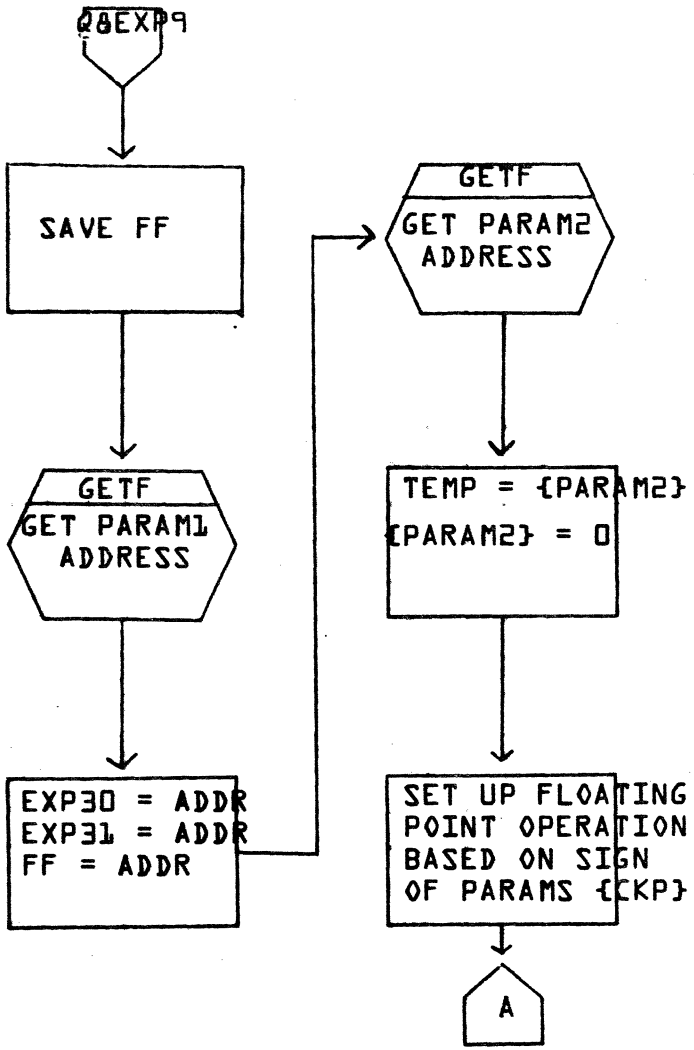
The mode is binary. The checksum when added to the sum of all the data words and the header word will result in zero. Overflow is ignored in the computation of the checksum.

A READ {u}L will read one logical record with the above format. The header word and checksum are not read into the buffer.

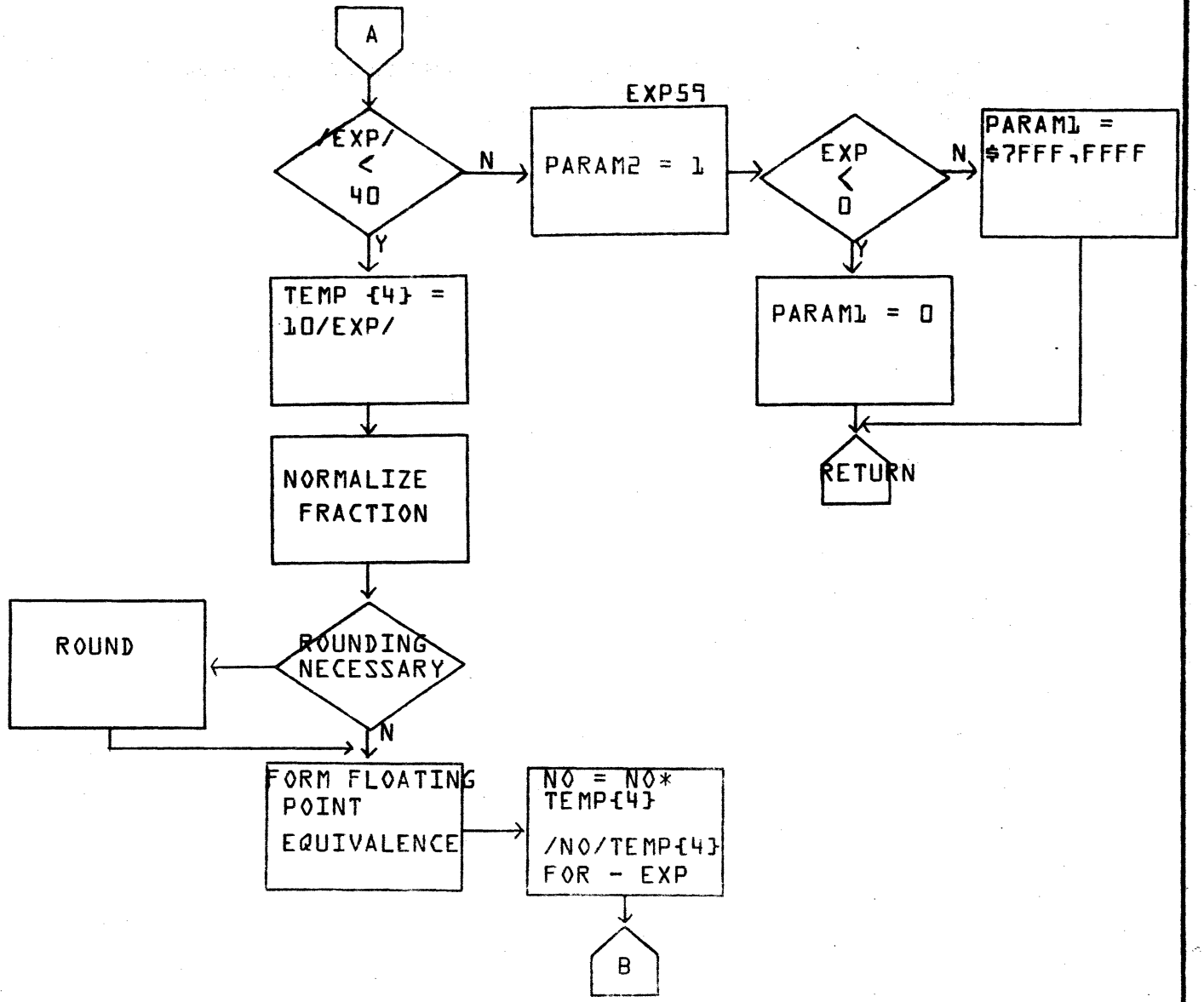
A WRITE {u} will produce on magnetic tape a logical record which has the same form as the paper tape record except a record gap replaces the checksum and header word.

6.7.2 Formatted READ-WRITE Statements

All formatted read-write statements produce a record of 60 or less words, 2 ASCII characters per word. Attempts to read more than 60 words {120 characters} in a record result in an error message and program termination. Attempts to write more than 60 words {136 characters} in a record results in truncation of the record at the 136th character.

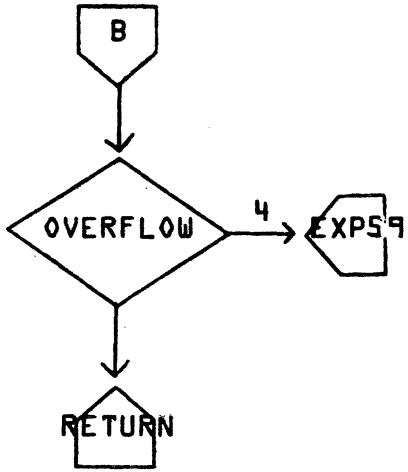


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE Q8EXP9		PROJECT MGR.		
		PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

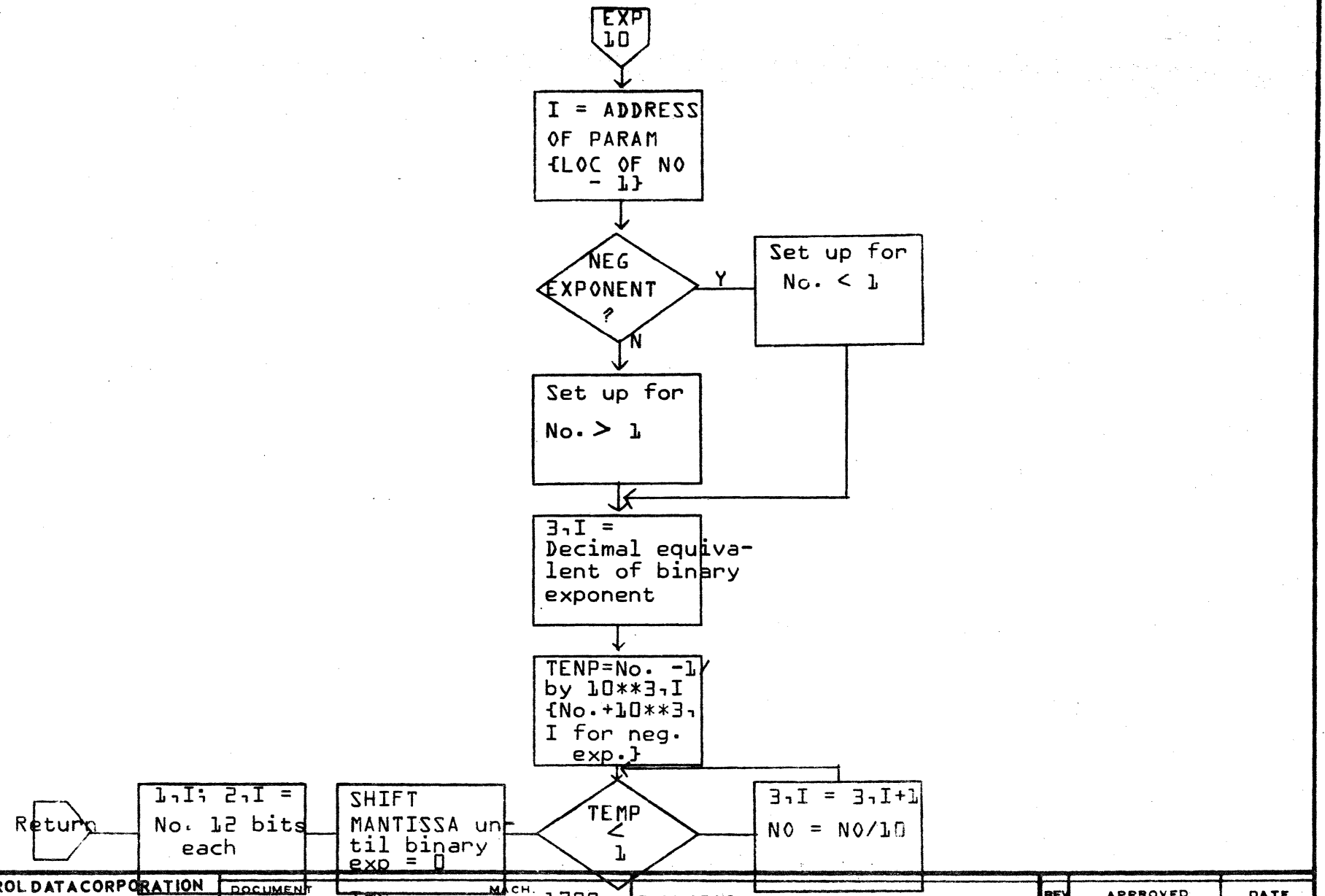


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE Q8EXP9		PROJECT MGR.		
		PAGE 2 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

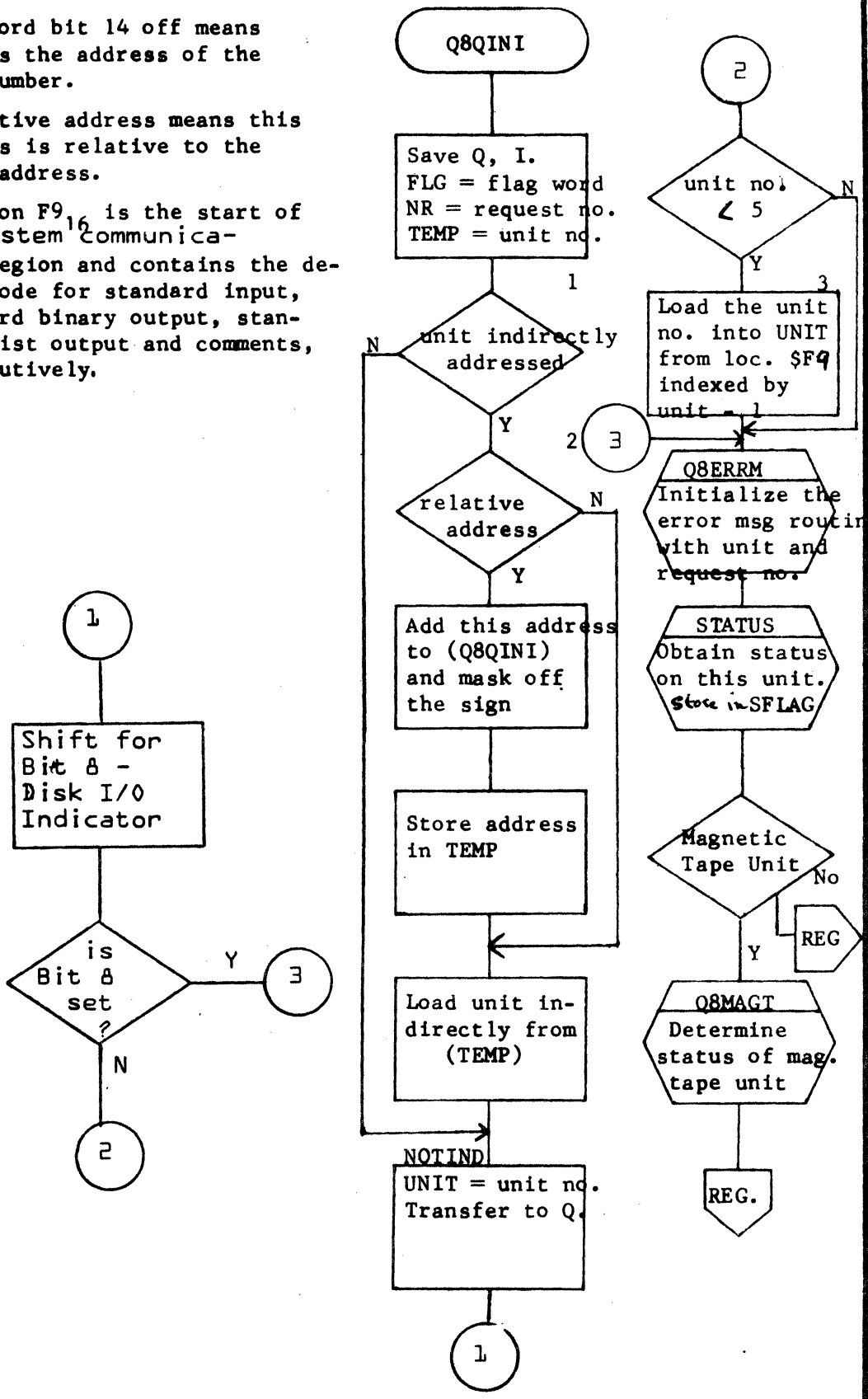


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE QAEXP9	PROJECT MGR.			
	PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE QSEXP1		PROJECT MGR.			
SAMPLE CODE ☐		PAGE 1 OF 1	PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

1. Flag word bit 14 off means this is the address of the unit number.
2. A negative address means this address is relative to the given address.
3. Location $F9_{16}$ is the start of the System communication region and contains the device code for standard input, standard binary output, standart list output and comments, consecutively.

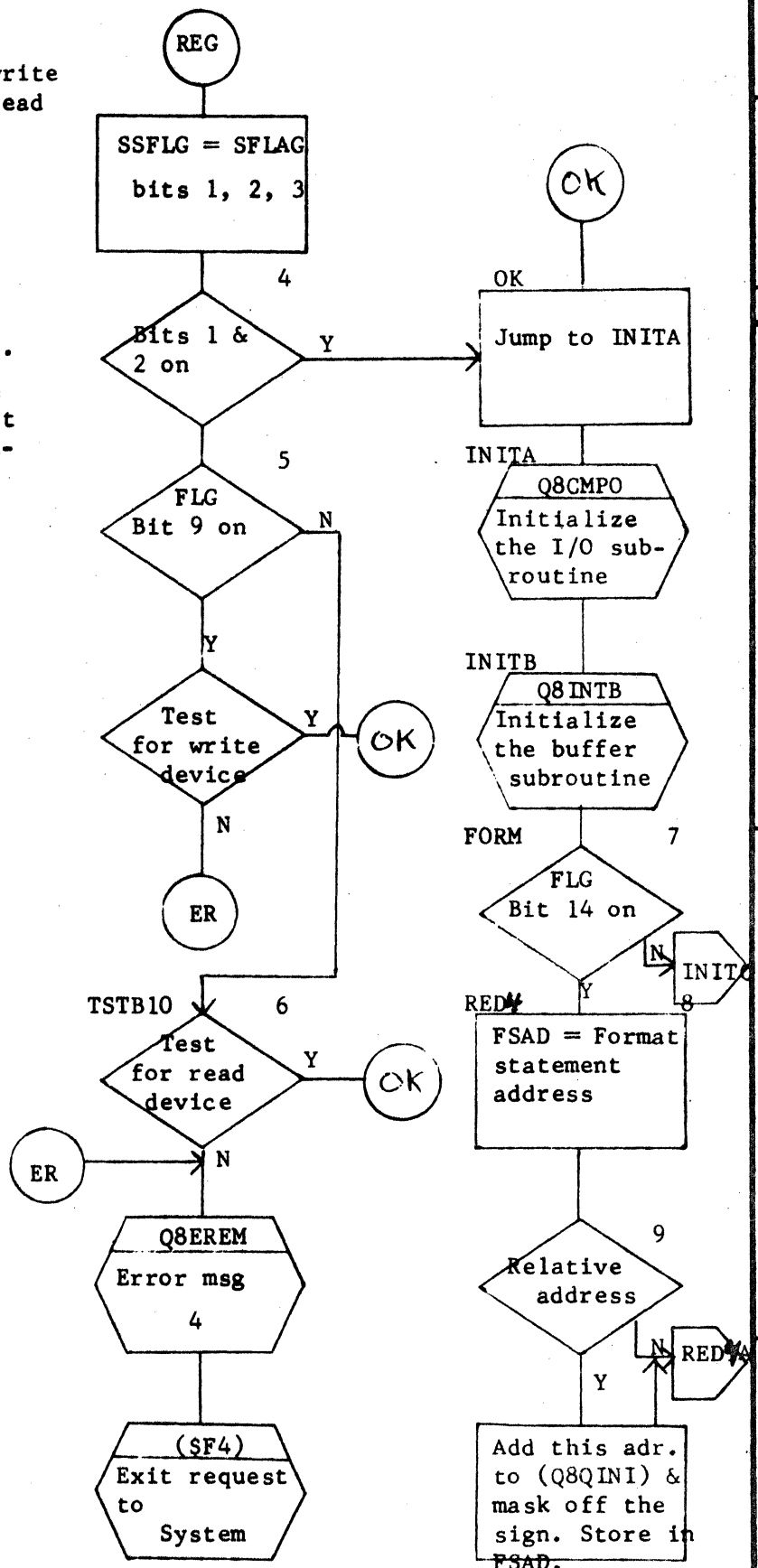


REV	APPROVED	DATE			
PROJECT NO.	PROJECT MGR.	PROJECT NAME	TASK NO.	TASK NAME	
DOCUMENT CLASS	DOCUMENT TITLE	NUMBER	ISSUE DATE	DRAWN BY	
MACH. TYPE 1700		Q8QINI	PAGE 1 OF 89	DATE 3-67	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

4. Bit 1 on for read unit; bit 2 on for write unit; bit 3 on for read or write.
5. Flag word bit 9 on means write request.
6. Flag word bit 10 on means read request.
7. Flag word bit 14 on means format request.
8. FSAD can contain the address of the format statement direct, indirect or relative.
9. A negative address means this address is relative to the given address.



REV	APPROVED	DATE

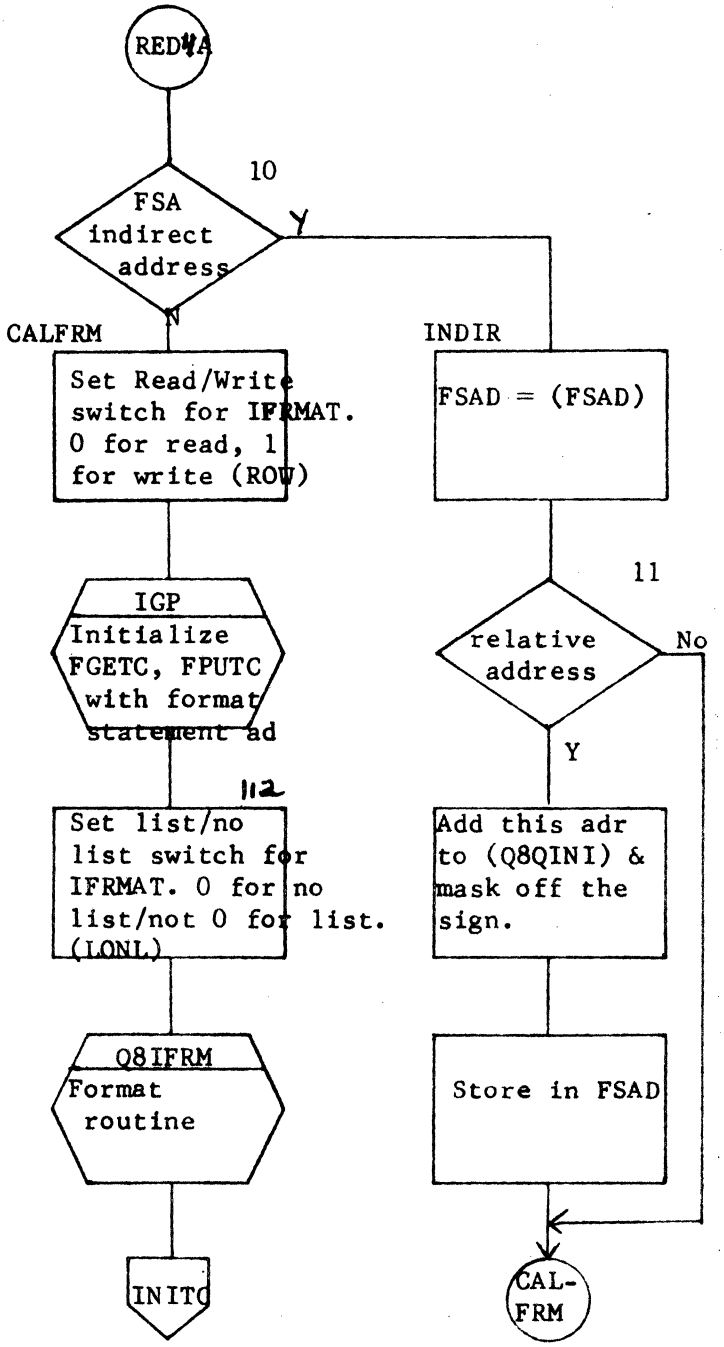
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
MACH. TYPE 1700
PAGE 2 OF 89
ISSUE DATE
DATE 7-67
DOCUMENT CLASS
DOCUMENT TITLE Q8QINI
NUMBER
DRAWN BY

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

10. Flag word bit 12 off means this is the address of the format statement address.

11. A negative address means this address is relative to the given address.

112. This box belongs on page 2 at label FORM.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

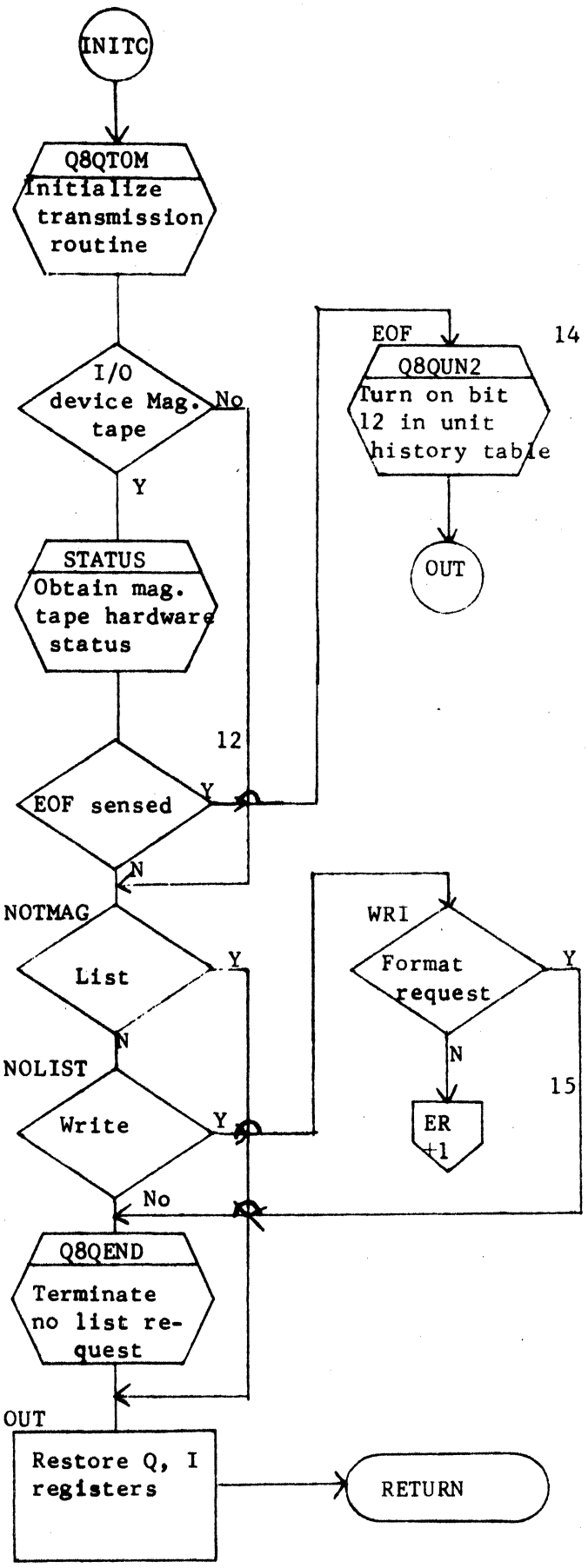
DOCUMENT CLASS: ZMC
MACH. TYPE: 1700
DOCUMENT TITLE: Q8QINI
PAGE 3 OF 89
NUMBER
ISSUE DATE
DRAWN BY
DATE 3-1-67

12. Bit 11 is on if an end of file was sensed by the magnetic tape status check.

13. LONL non-zero means this request has a list. Flag word bit 11 means list.

14. Bit 12 in the unit history table is used to flag end of file status on the device.

15. A formatted write request without list is a fatal error. Error message 11 is typed out.



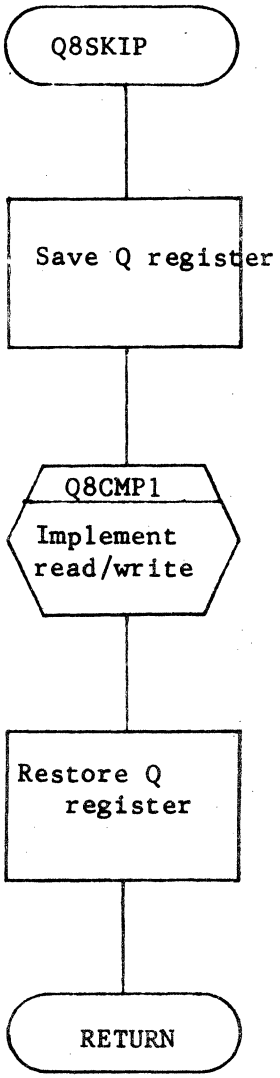
REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER: Q8QINI
MACH. TYPE: 1700
PAGE 4 OF 89
ISSUE DATE
DRAWN BY
DATE: 3-1-71

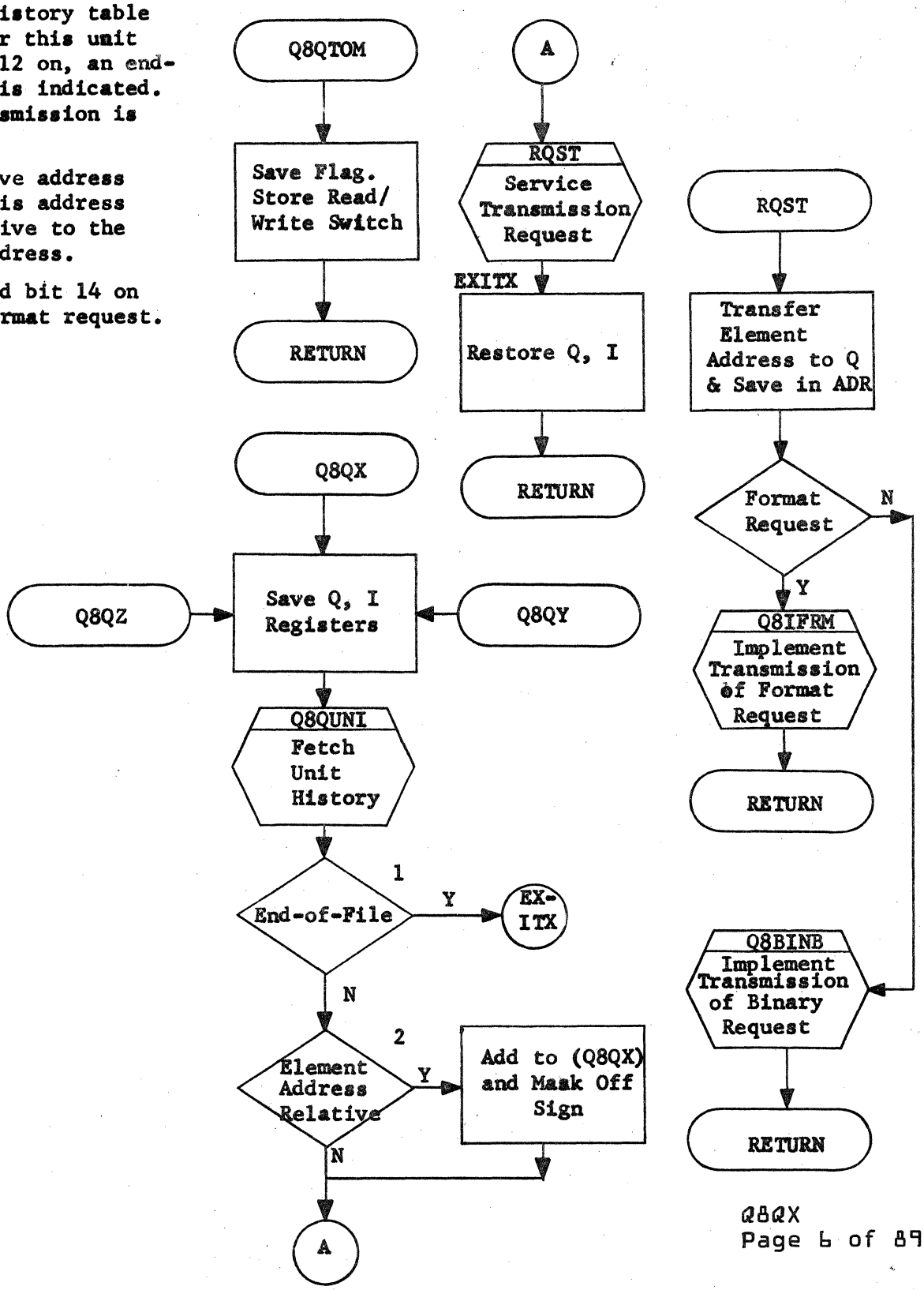
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [Y]
DECISION TABLE
OTHER

1. Q8SKIP is an entry point in Q8QINI but is an independent subroutine called by the format routines to skip to the next record. It is called whenever a slash occurs in the format statement or deamed necessary by the format routines.

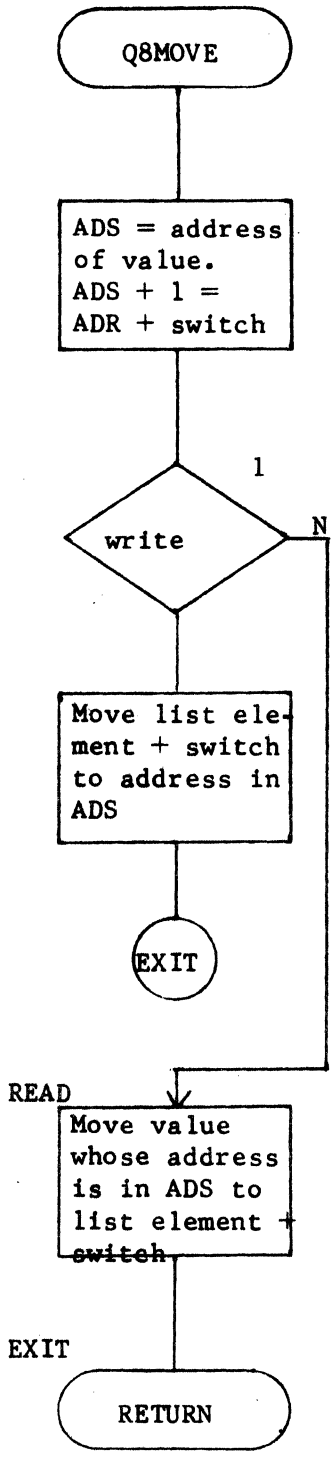


CONTROL DATA CORPORATION	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE		PROJECT MGR.			
SAMPLE CODE ☐	Q8SKIP	PAGE 5 OF 89	PROJECT NAME			
FLOWCHART ☒	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

1. If the history table entry for this unit has bit 12 on, an end-of-file is indicated. All transmission is ignored.
2. A negative address means this address is relative to the given address.
3. Flag word bit 14 on means format request.



1. Flag word bit 9 on means write.
2. Q8MOVE is an entry point in Q8QX.



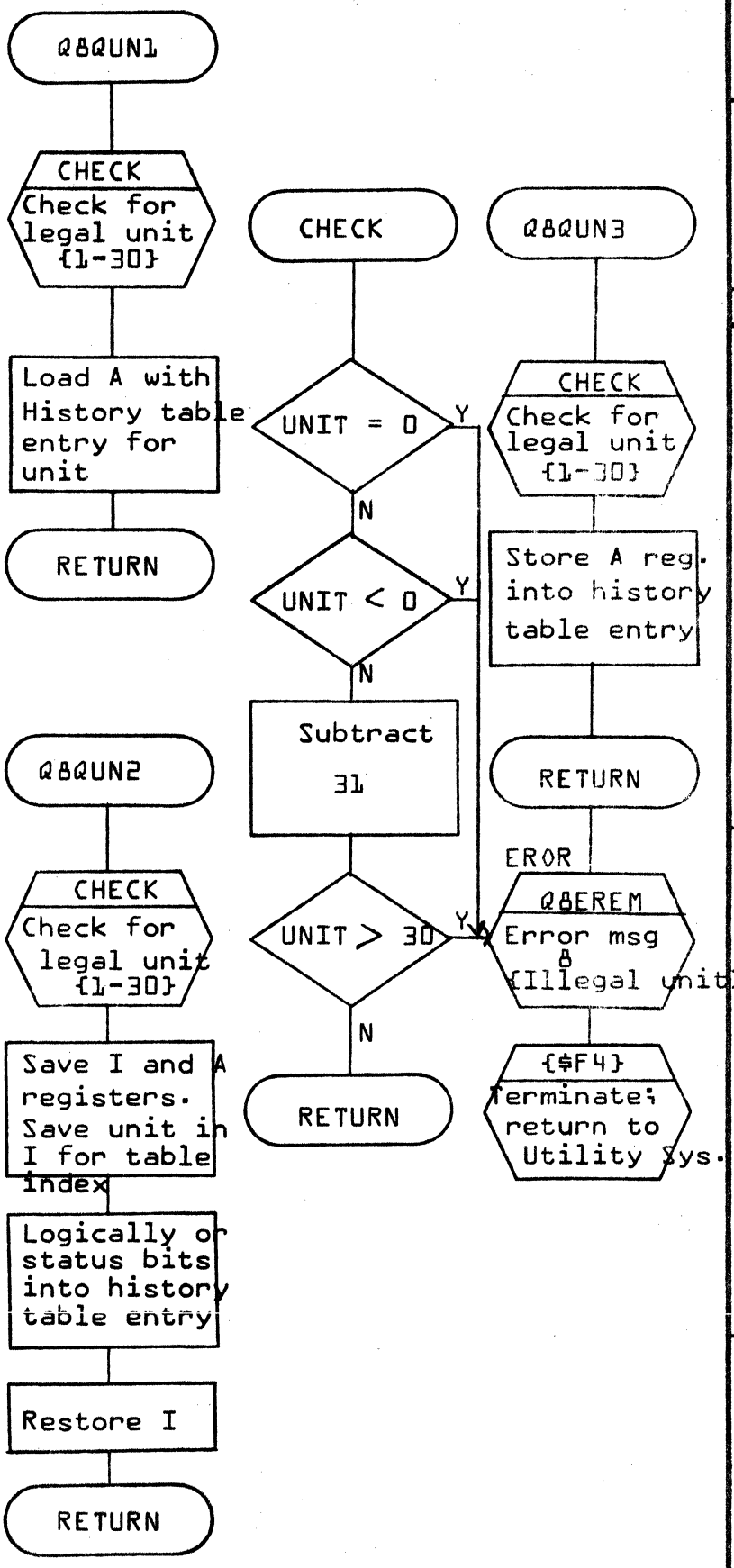
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER Q8QX
DRAWN BY
MACH. TYPE 1700
PAGE 7 OF 89
ISSUE DATE
DATE 3-67

AA1388 (FORMERLY CA127-1)



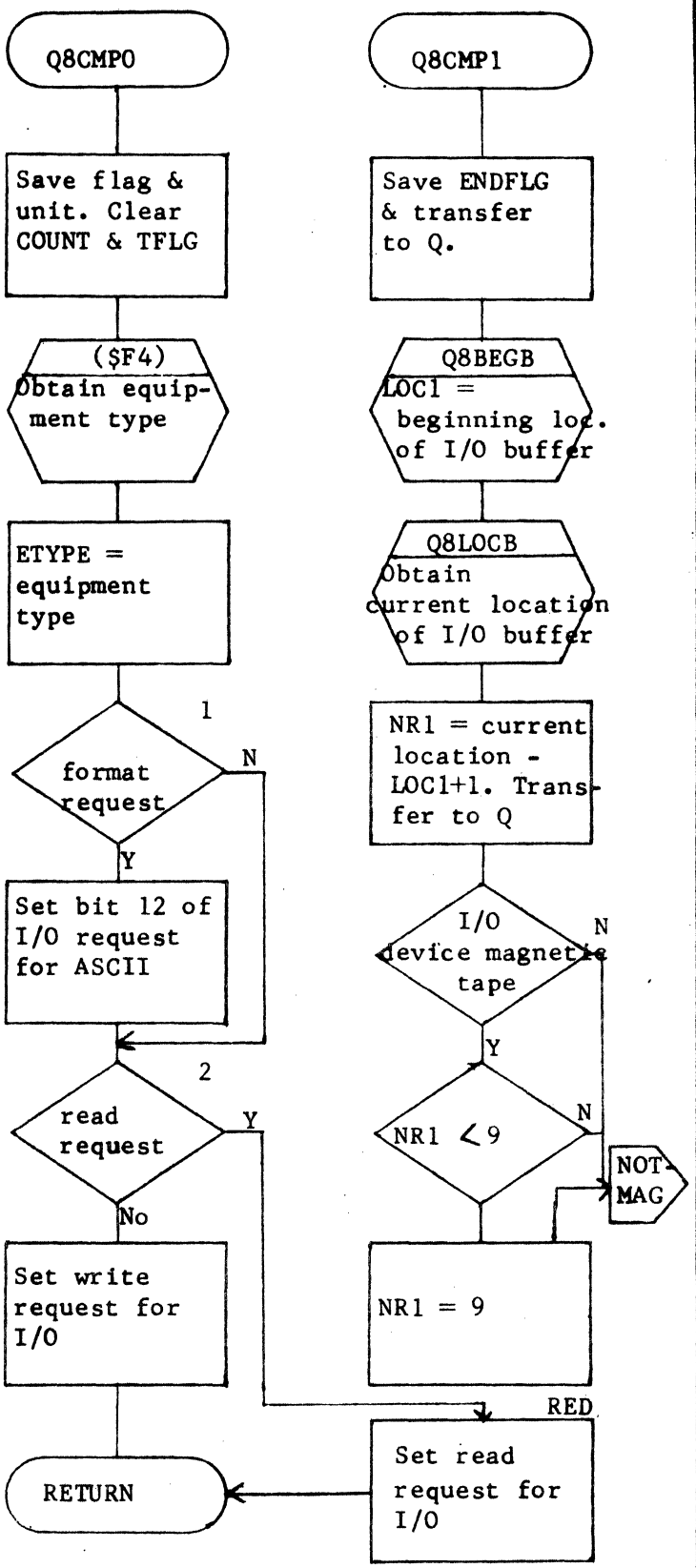
REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS	MACH. TYPE	1700	PROJECT NO.	
DOCUMENT TITLE	Q8QUN1		PAGE 8 OF 89	PROJECT MGR.	
NUMBER		ISSUE DATE		PROJECT NAME	
DRAWN BY		DATE		TASK NO.	
				TASK NAME	

AA1385 (FORMERLY CA127-1)

1. Flag word bit 14 on means formatted I/O.
2. Flag word bit 10 on means read request.
3. Magnetic tape transfers must be at least 9 words in length due to hardware constraints.



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

MACH. TYPE 1700
PAGE 9 OF 89
ISSUE DATE
DATE 3-67

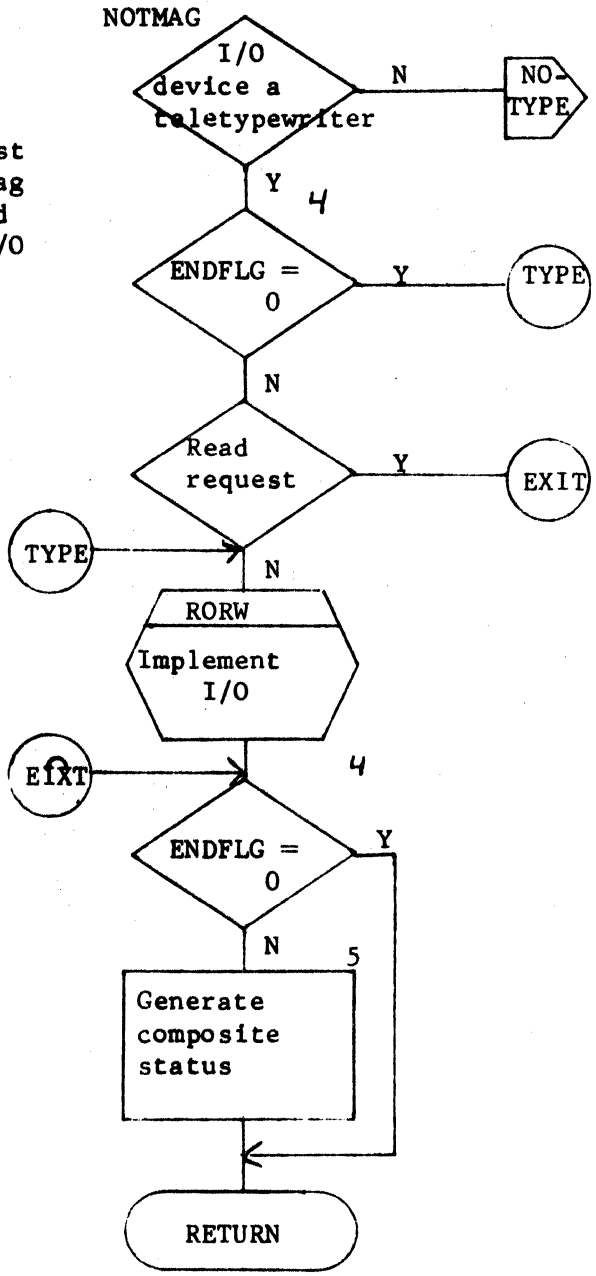
DOCUMENT CLASS ZMS
DOCUMENT TITLE
Q8CMP0
NUMBER
DRAWN BY

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

4. ENDFLG indicates entry from Q8QEND if non-zero.

5. Composite status consist of bits 9-10 of the flag word for read/write and bits 13-14-15 of the I/O request status.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER Q8CMP0
DRAWN BY
MACH. TYPE
PAGE 10 OF 89
ISSUE DATE
DATE 3-69

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV. | APPROVED | DATE

AA1385 (FORMERLY CA127-1)

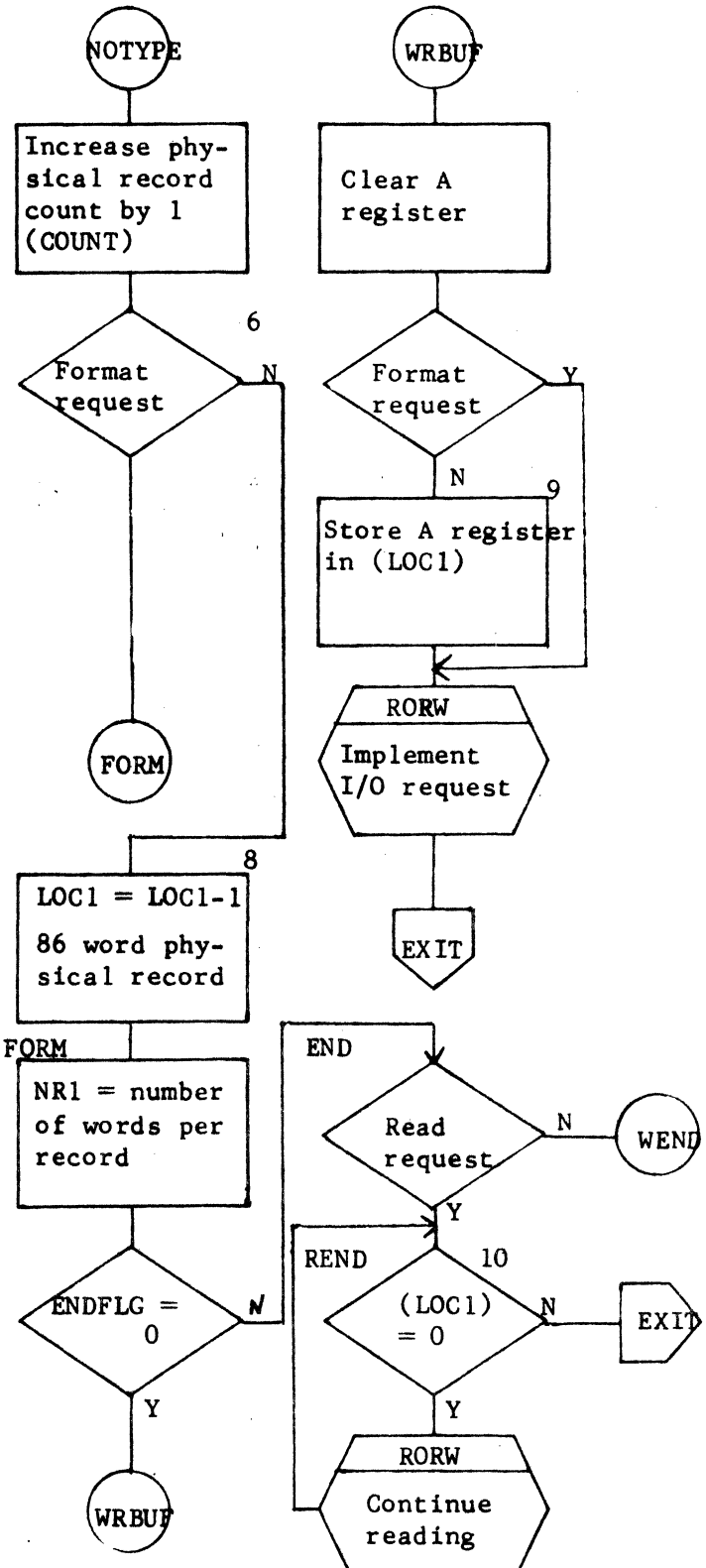
6. Flag word bit 14 on means format request.

8. All binary I/O requests are in logical records of 86 words each.

   The first word of each 86 word record is zero for all records except the last which contains the number of physical records per logical record (COUNT).

9. Clear 1st word of binary physical record.

10. 1st word zero means more physical records to follow. Q8QEND requires input of all physical records until entire logical record has been read.



REV	APPROVED	DATE

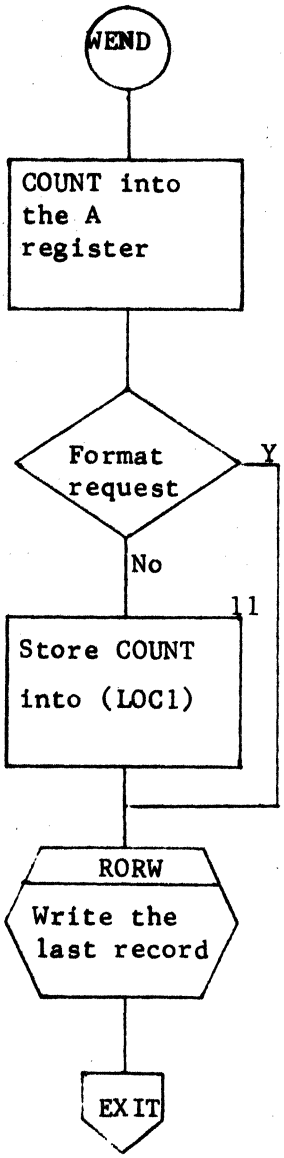
PROJECT NO. | PROJECT MGR. | PROJECT NAME | TASK NO. | TASK NAME

MACH TYPE 1700 | PAGE 11 OF 89 | DATE 3-67

DOCUMENT CLASS | DOCUMENT TITLE | ISSUE DATE | NUMBER Q8CMP0 | DRAWN BY

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

11. Q8QEND means terminate the write request. For binary this means the physical record count is stored in the first word of this, the last, record. For format, output the record.

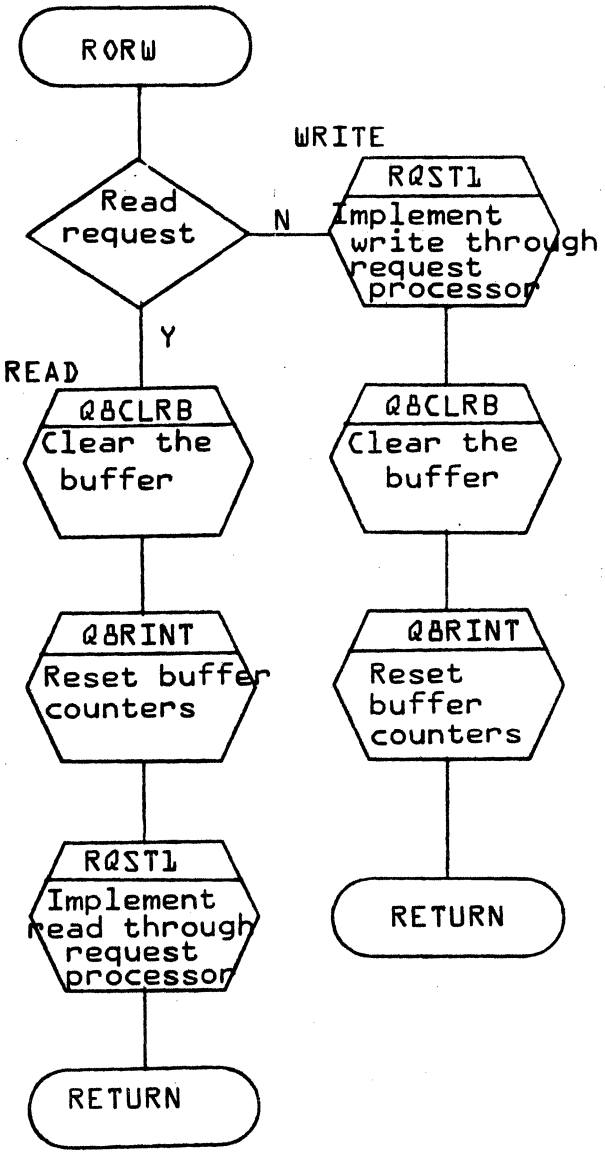


REV	APPROVED	DATE

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER Q8CMPQ
ISSUE DATE
DRAWN BY
MACH. TYPE
PAGE 12 OF 89
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

AA1385 (FORMERLY CA127-1)



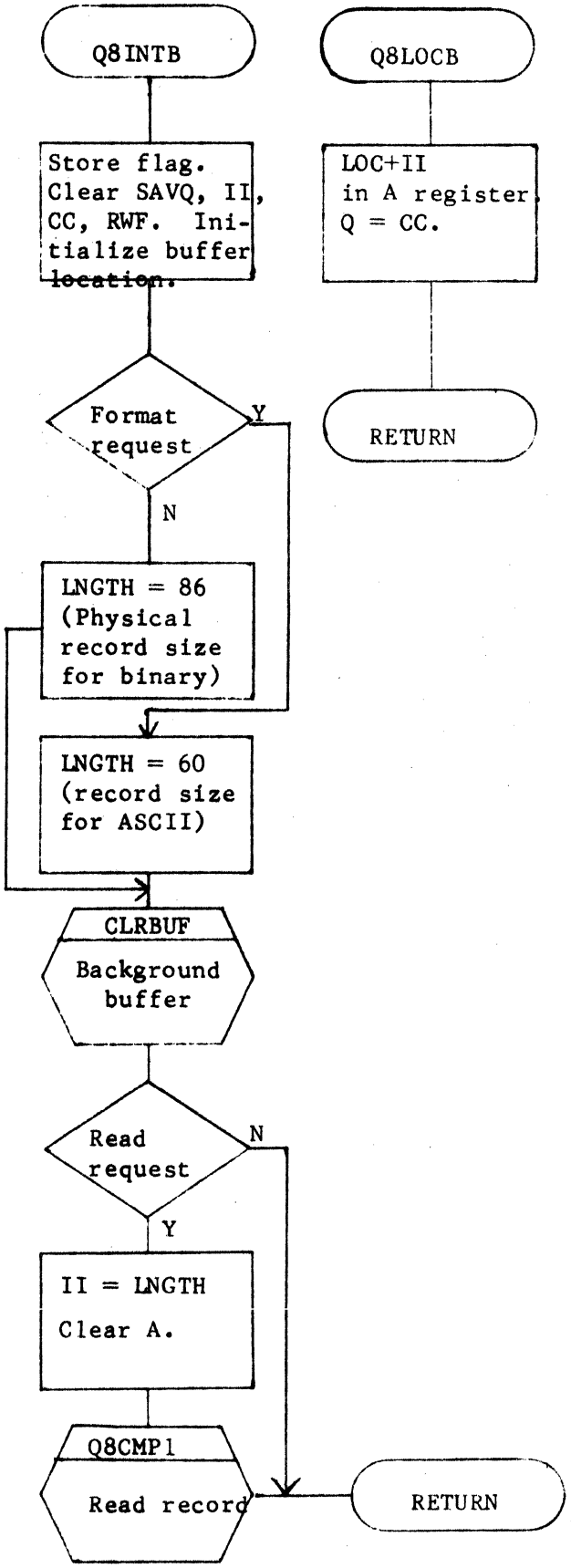
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS	MACH. TYPE	1700	
DOCUMENT TITLE	Q8CMP0	PAGE 13 OF 89		
NUMBER		ISSUE DATE		
DRAWN BY		DATE		

REV	APPROVED	DATE
PROJECT NO.		
PROJECT MGR.		
PROJECT NAME		
TASK NO.		
TASK NAME		

1. LOC+II is the current location of the input/ output buffer.



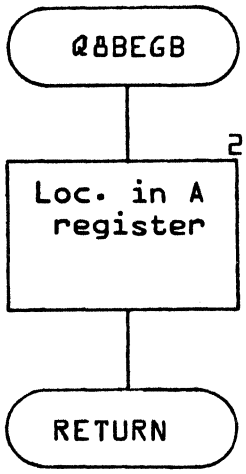
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER Q8RWRU
MACH TYPE
PAGE 14 OF 89
ISSUE DATE
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV | APPROVED | DATE

2. LOC is the beginning location of the input/ output buffer.



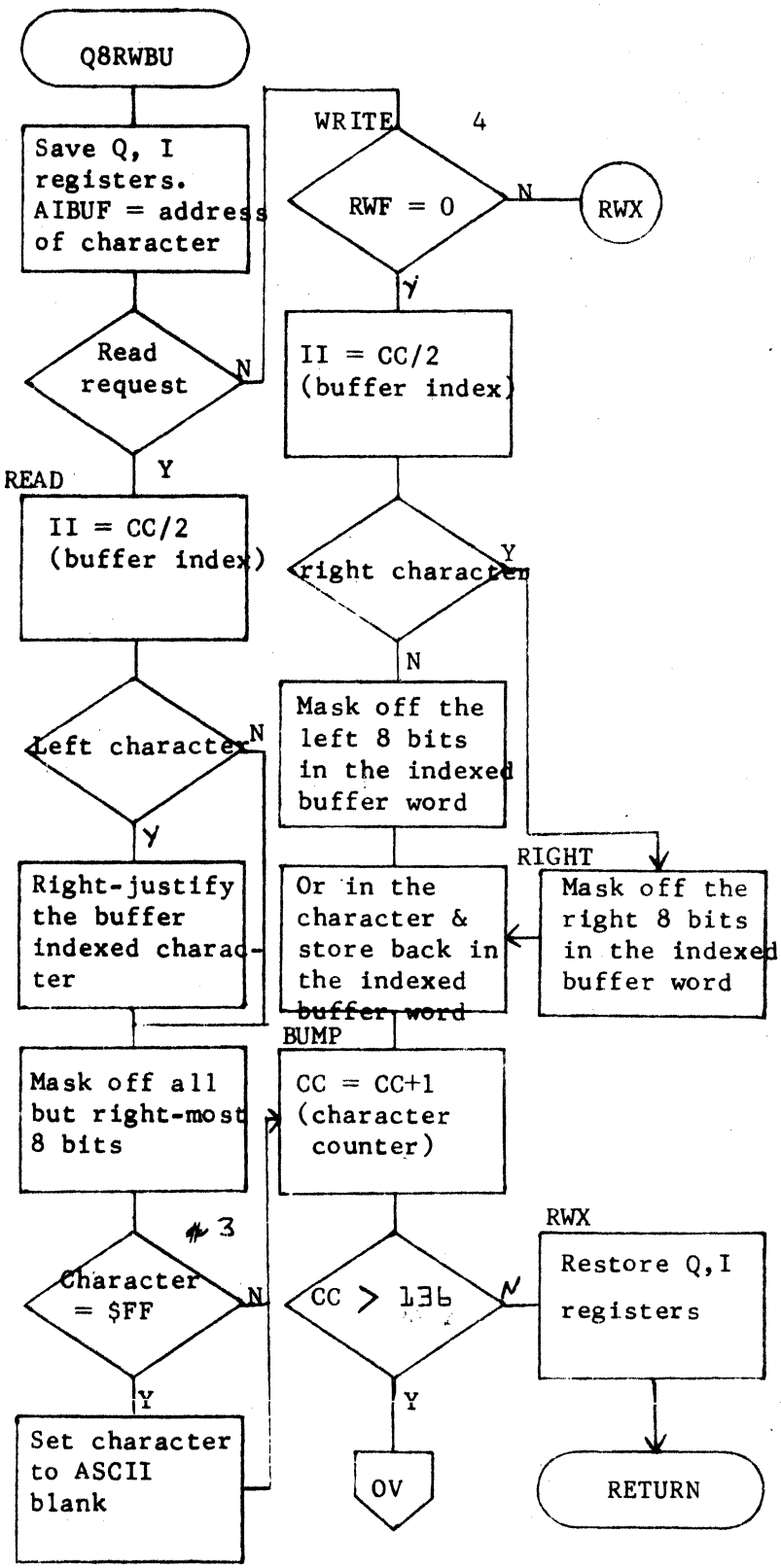
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	Q8RWBU	PAGE	15 OF 89
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

PROJECT NO.			
PROJECT MGR.			
PROJECT NAME			
TASK NO.			
TASK NAME			

REV	APPROVED	DATE

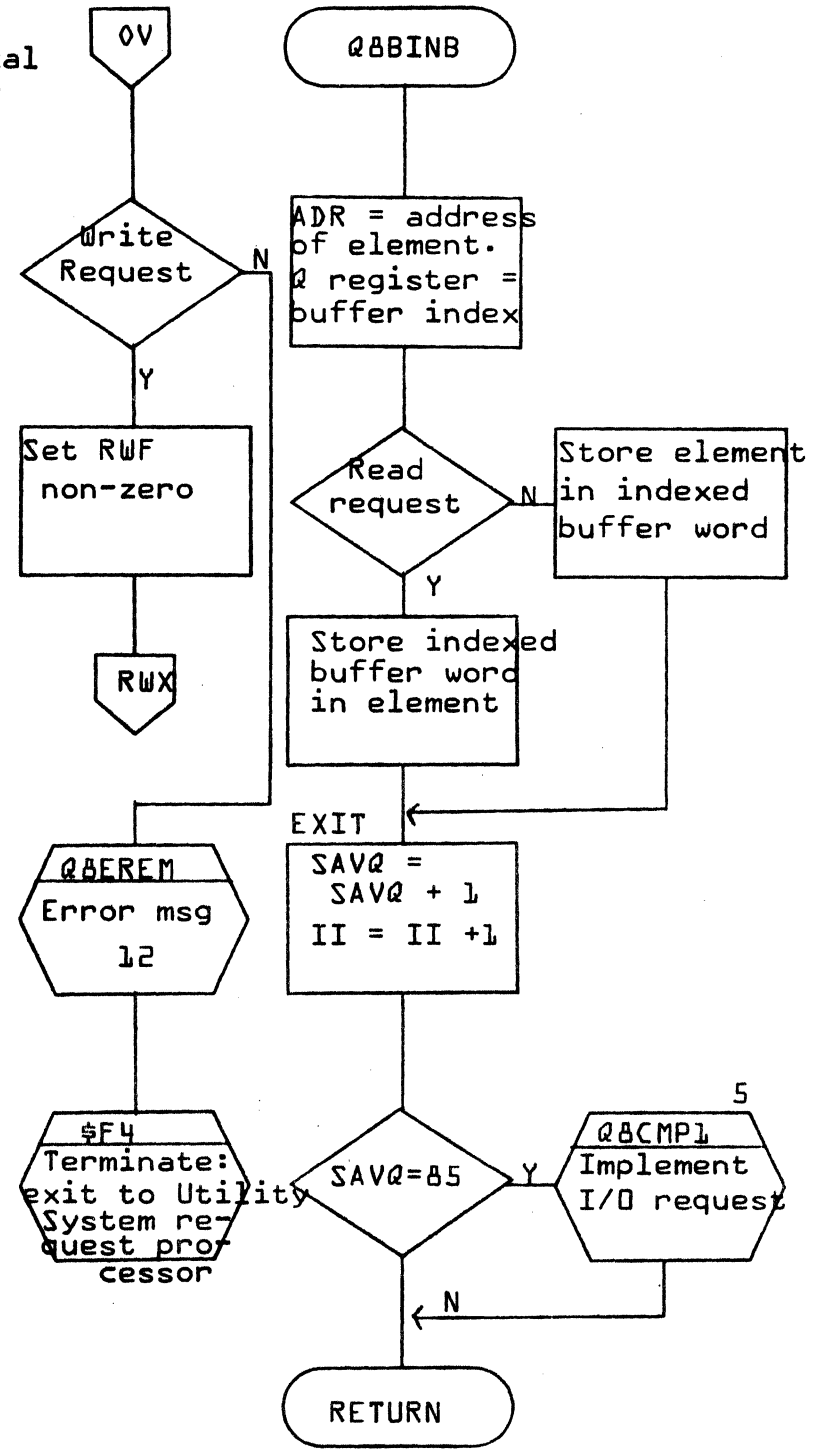
3. An FF_{16} will be encountered when a read was terminated at an odd numbered character. The I/O drivers fill the word with FF_{16}.
4. RWF flags more than 120 characters transferred. On a write request truncation occurs at 136 characters.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER Q8RWBU
ISSUE DATE
DRAWN BY
MACH. TYPE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
DATE
PAGE 16 of 89
REV
APPROVED
DATE

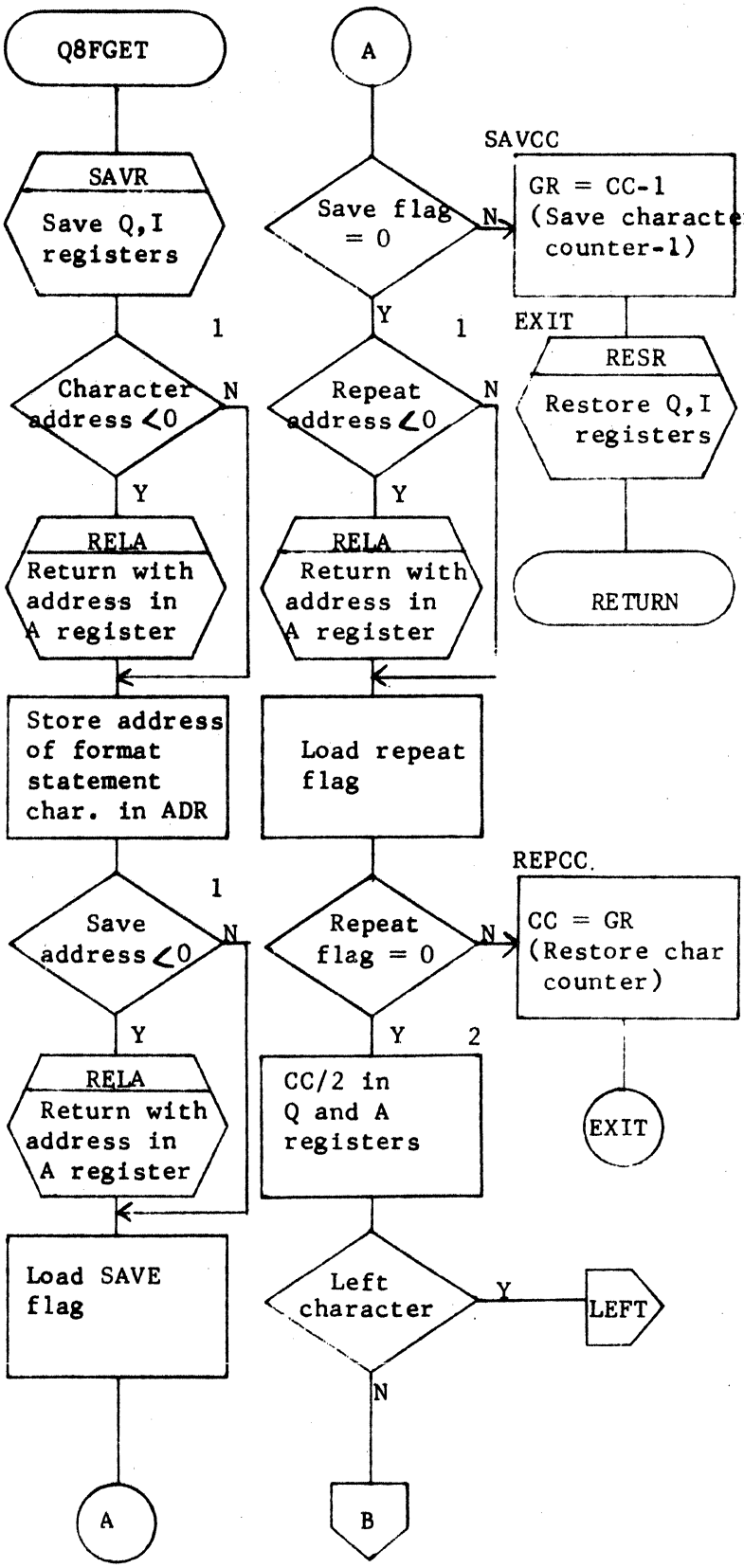
5. When the binary buffer is full, the I/O request is implemented. A physical record is read or written.



DATE	
APPROVED	
REV	
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	
DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	QBRWBU
PAGE 17 OF 89	
NUMBER	
ISSUE DATE	
DRAWN BY	
DATE	

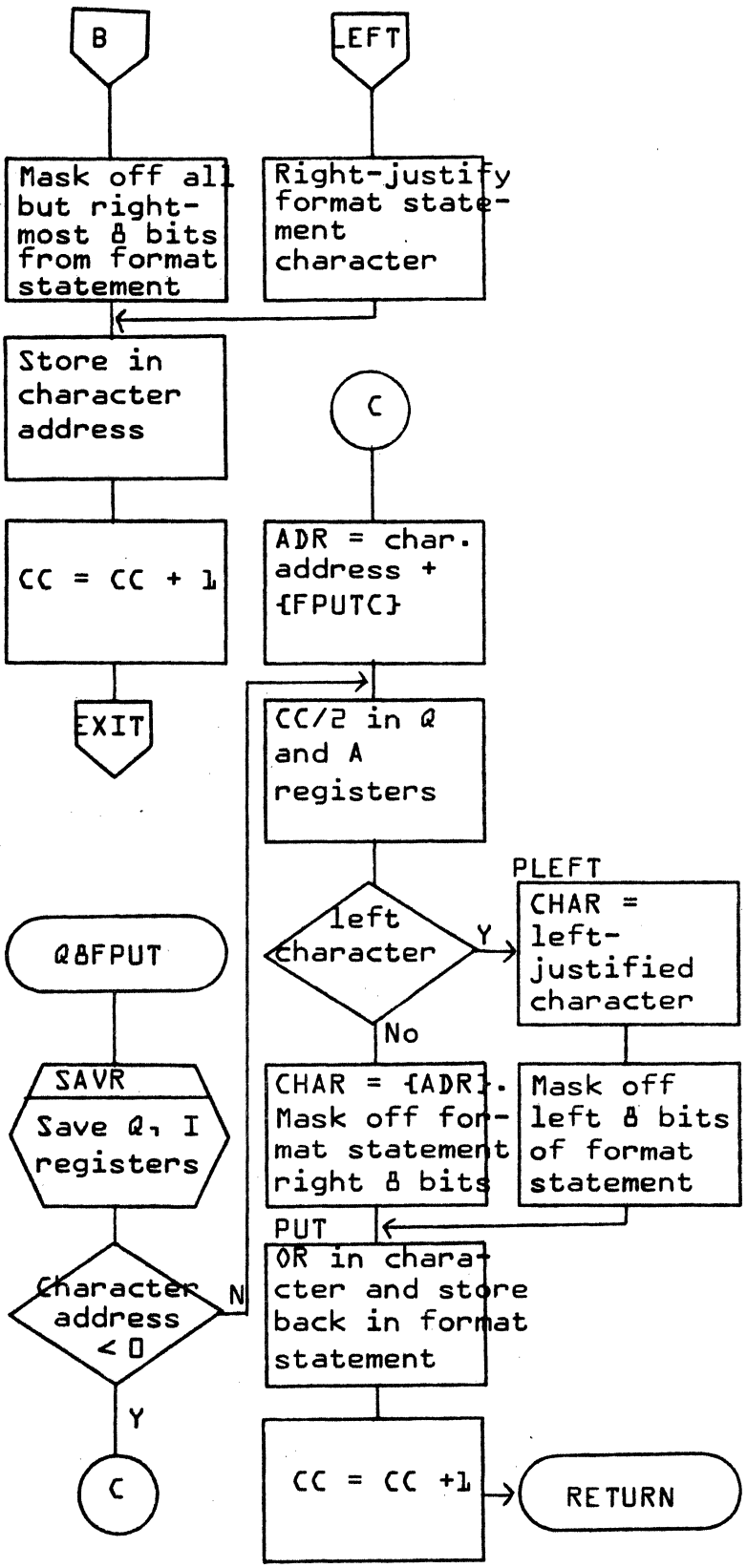
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

1. A negative address means this address is relative to the given address.
2. Loading the Q register with CC and shifting right one place has the effect of loading Q with the format statement word index and leaving a pointer to the right character. CC, the character counter, is odd for right characters, even for left characters.



DATE					
APPROVED					
REV					
PROJECT NO.	PROJECT MGR.	PROJECT NAME	TASK NO.	TASK NAME	
DOCUMENT CLASS	DOCUMENT TITLE	Q8FGET	NUMBER	DRAWN BY	
MACH. TYPE		PAGE 18 OF 89	ISSUE DATE	DATE 5-67	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐



REV	APPROVED	DATE

PROJECT NO.			
PROJECT MGR.			
PROJECT NAME			
TASK NO.			
TASK NAME			

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	Q8FGET	PAGE 19 OF 89	
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

CONTROL DATA CORPORATION

SOFTWARE DOCUMENT

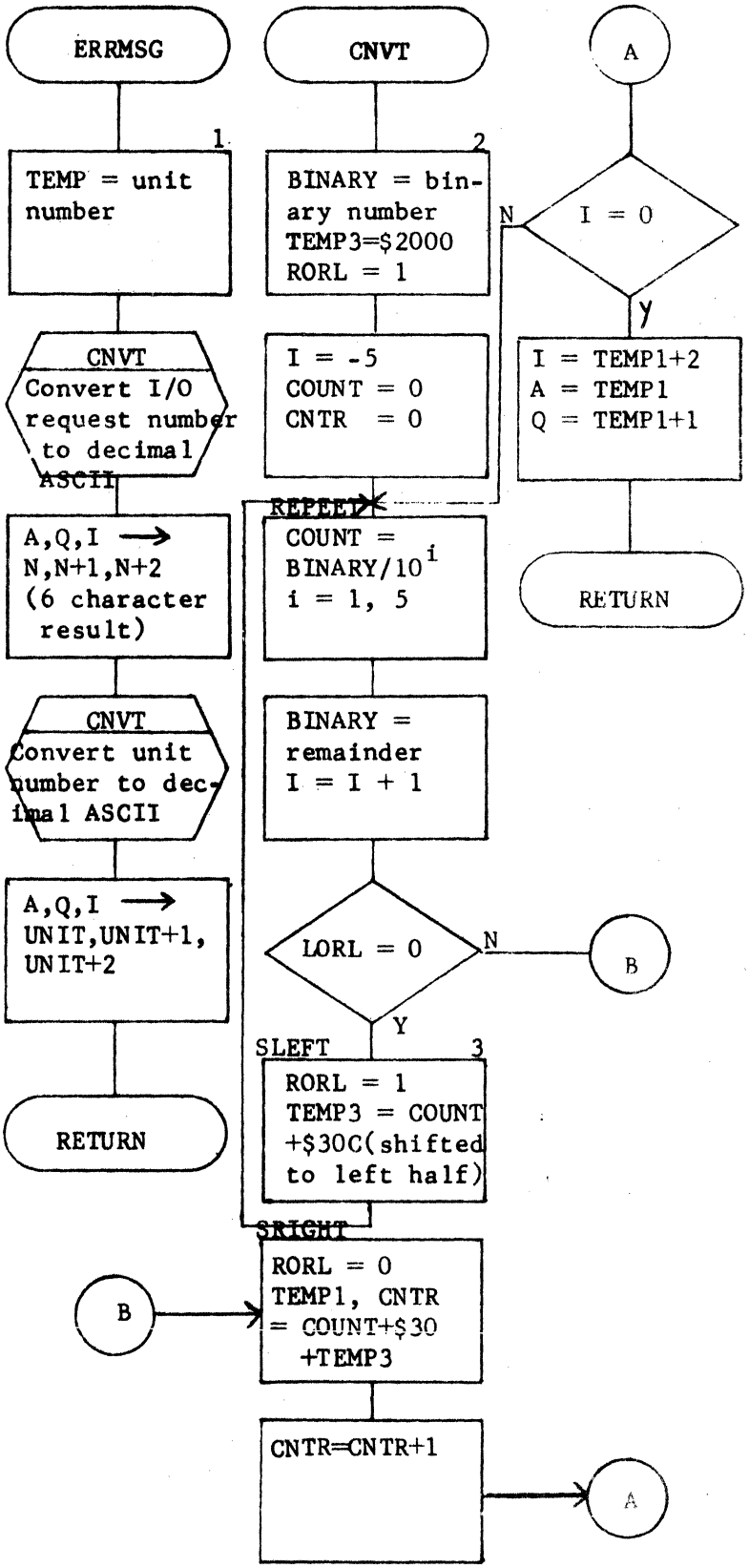
SAMPLE CODE ☐☐☐☐

FLOWCHART

DECISION TABLE

OTHER

1. Enter with I/O request number in A and unit number in Q.
2. LORL is a flag; 1 for right half store; 0 for left half store.
3. $30 is ASCII zero.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

REV	APPROVED	DATE

PROJECT NO. | PROJECT MGR. | PROJECT NAME | TASK NO. | TASK NAME

DOCUMENT CLASS | DOCUMENT TITLE | MACH. TYPE 1700 | NUMBER Q8ERRM | ISSUE DATE | DRAWN BY | DATE 3-67

PAGE 20 OF 89

AA1365 (FORM) (CA127-1)

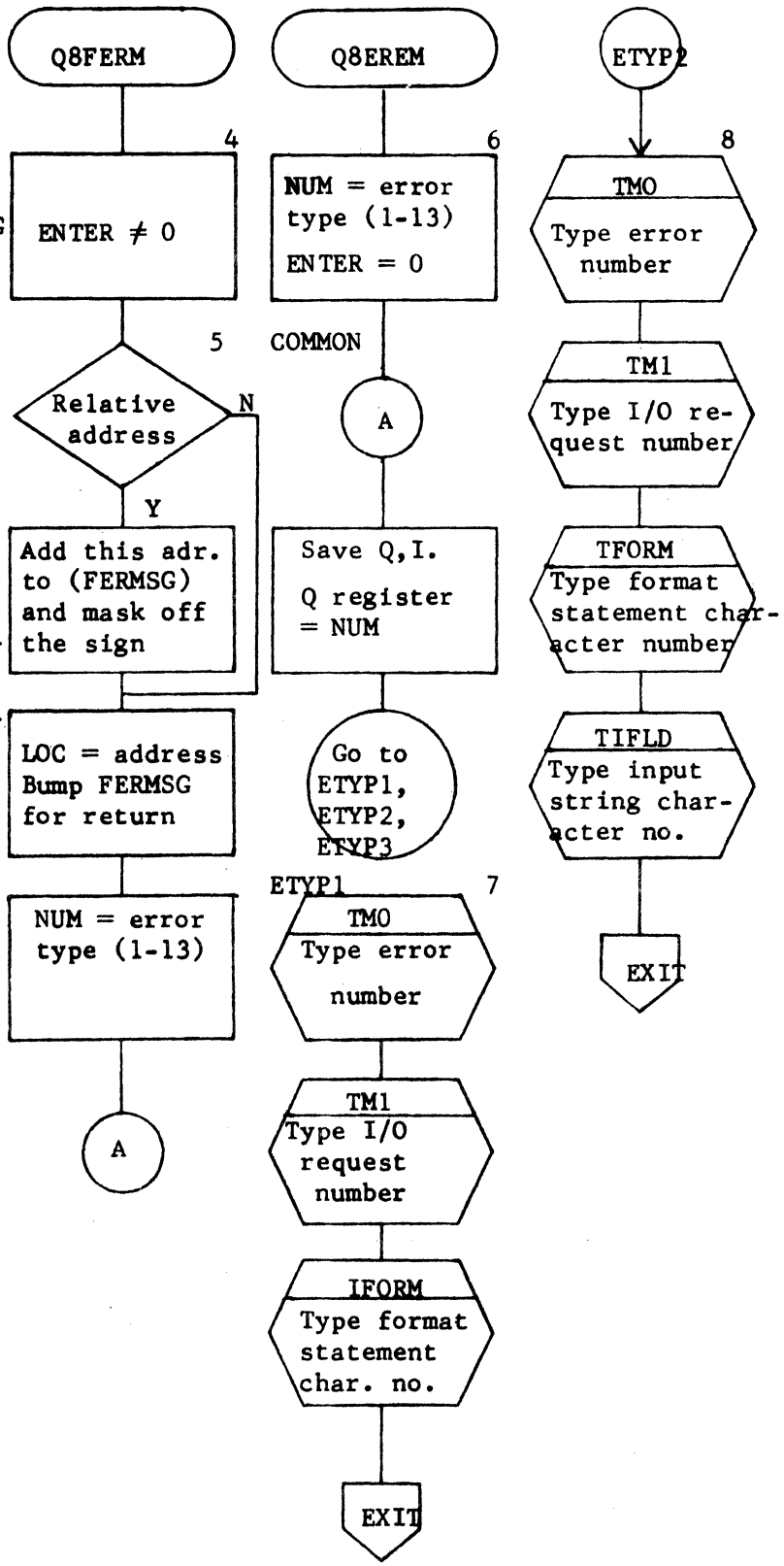
4. FERMSG is entered from the format package which is coded in FORTRAN. The flag, ENTER, is set non-zero to indicate FERMSG was entered.

5. A negative address means this address is relative to the given address.

6. ERRMSG is entered from the 1700 machine language routines of the I/O package.

7. ETYP1 is the error types 1, 12, 13.

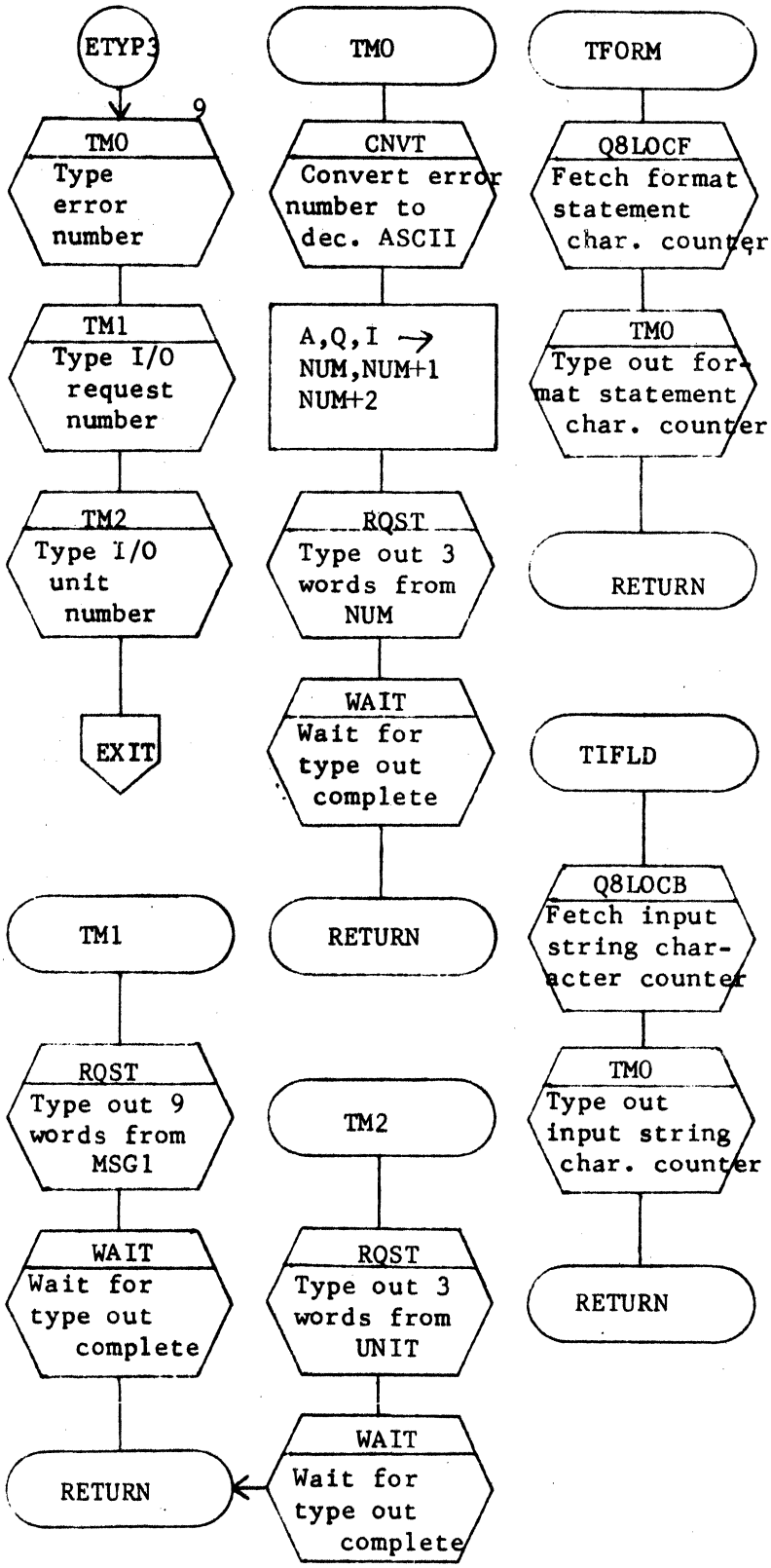
8. ETYP2 is the error format for error types 2 and 3.



REV	APPROVED	DATE			
PROJECT NO.	PROJECT MGR.	PROJECT NAME	TASK NO.	TASK NAME	
DOCUMENT CLASS	DOCUMENT TITLE	MACH. TYPE	PAGE 21 OF 89	ISSUE DATE	
NUMBER	Q8ERRM	DRAWN BY	DATE 3-6-67		

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

9. ETYP3 is the error format for error types 4, 5, 6, 7, 8, 9, 10, and 11.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

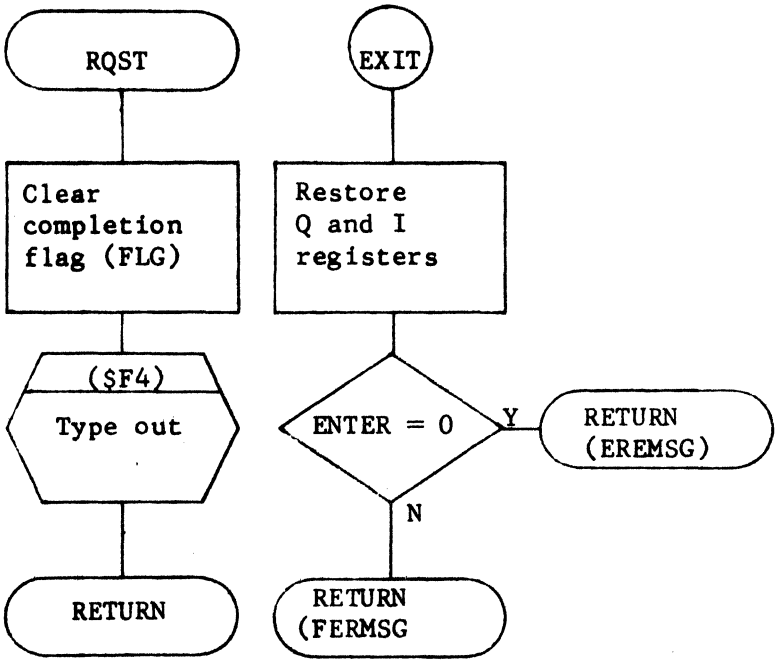
DOCUMENT CLASS
DOCUMENT TITLE
NUMBER Q8ERRM
MACH. TYPE
PAGE 22 OF 89
ISSUE DATE
DRAWN BY
DATE

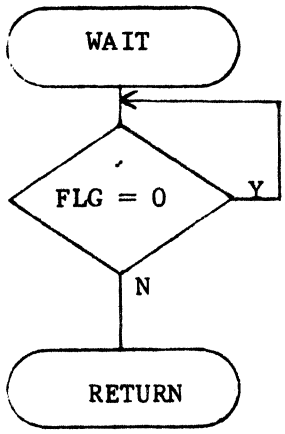
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

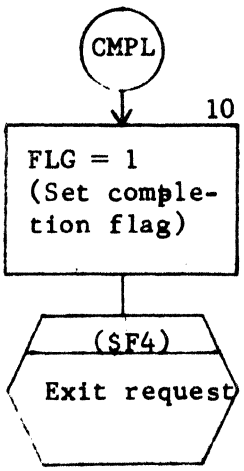
REV | APPROVED | DATE

AA 1385 (FORMERLY CA 127-1)

10. CMPL is the completion address entry point from the System request.





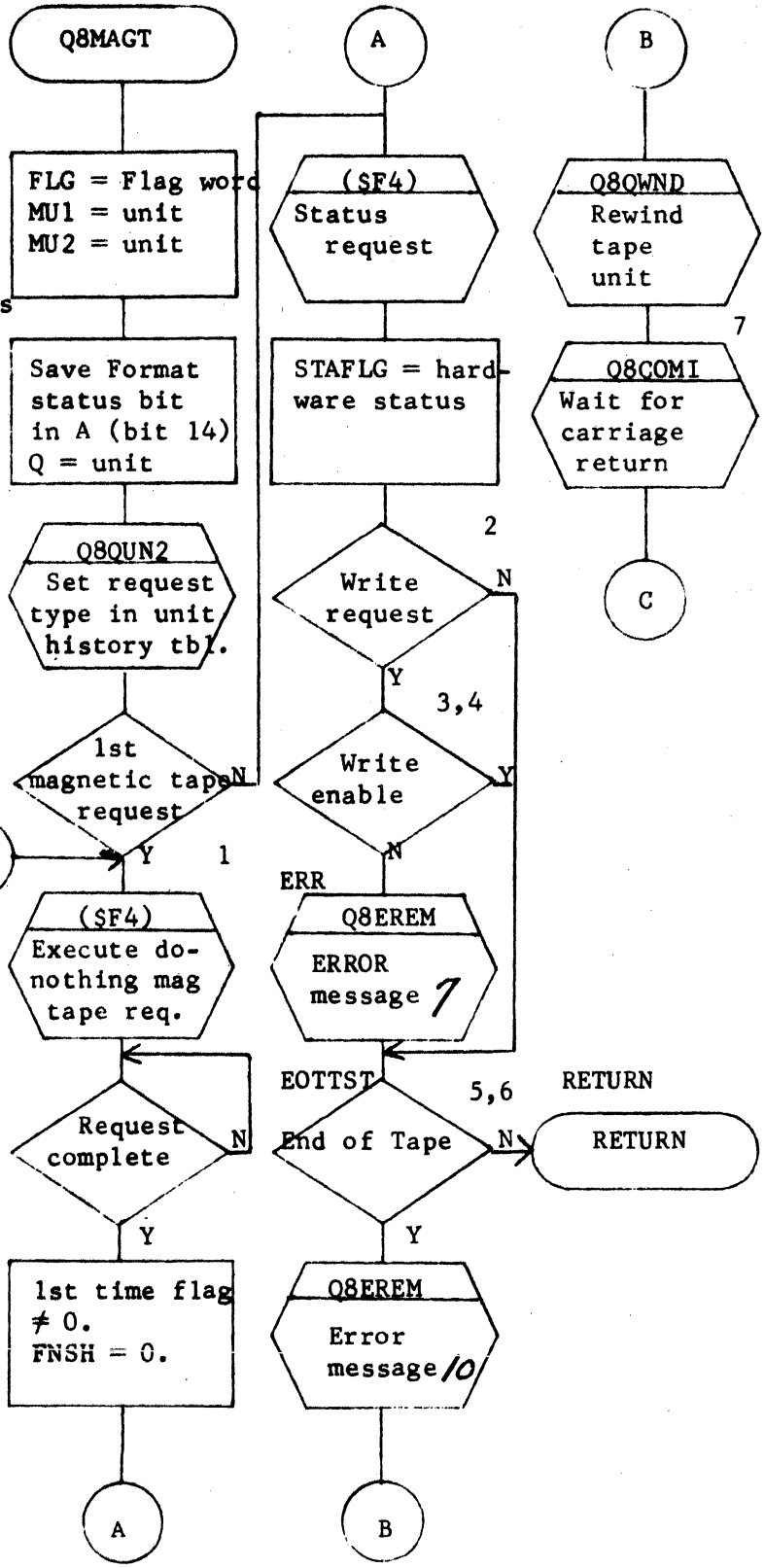


REV	APPROVED	DATE			
PROJECT NO.	PROJECT MGR.	PROJECT NAME	TASK NO.	TASK NAME	
DOCUMENT CLASS	DOCUMENT TITLE	NUMBER	ISSUE DATE	DRAWN BY	
MACH. TYPE		Q8ERRM	PAGE 29 OF 89	DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

AA1385 (FORMERLY CA127-11)

1. If no magnetic tape operation has occurred, a do-nothing request must be executed to obtain magnetic tape status.
2. Bit 9 of the flag word set means a write request.
3. Bit 15 of hardware status word set means write is enabled.
4. No write enable results in type-out of error message 7.
5. Bit 9 of hardware status word set means end of tape.
6. End of tape results in type-out of error message 10.
7. Wait for operator to mount a new tape.



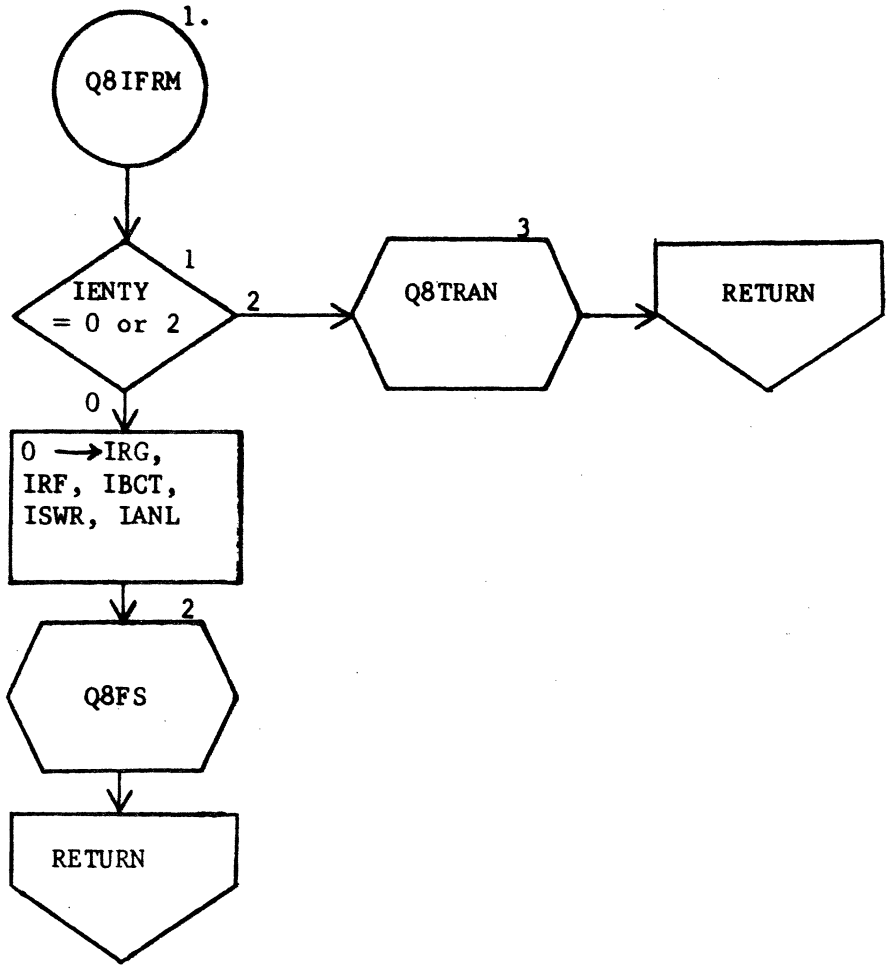
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

REV	DATE	APPROVED

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS
MACH. TYPE
DOCUMENT TITLE
Q8MAGT
NUMBER
PAGE 24 OF 89
ISSUE DATE
DRAWN BY
DATE

1. This routine is called from Q8QINI and Q8QX. The calling sequence to this routine has two parameters. They are:
   IENTY - 0 if routine called from Q8QINI
   IENTY - 1 if routine called from Q8QX.
2. Call format scanner. This routine interprets FORMAT statement parameters and sets them up in a form that the transmission routine can use.
3. Call transmission routine. This routine converts the next I/O field according to the format specifications.

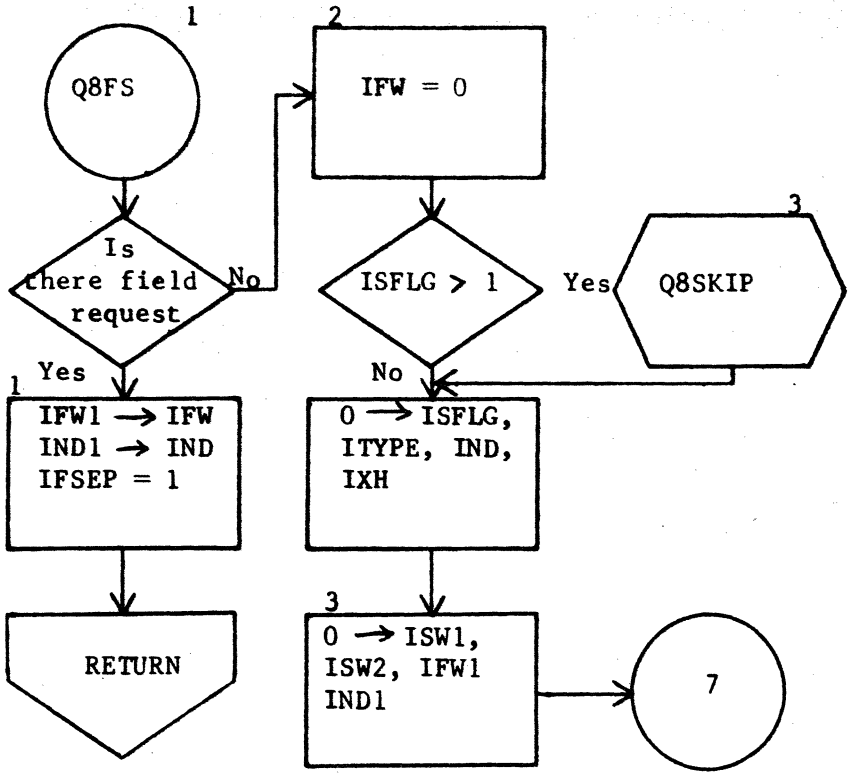


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1A1	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	Q8IFRM	PAGE 25 OF 89	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 5-6-69	TASK NAME		

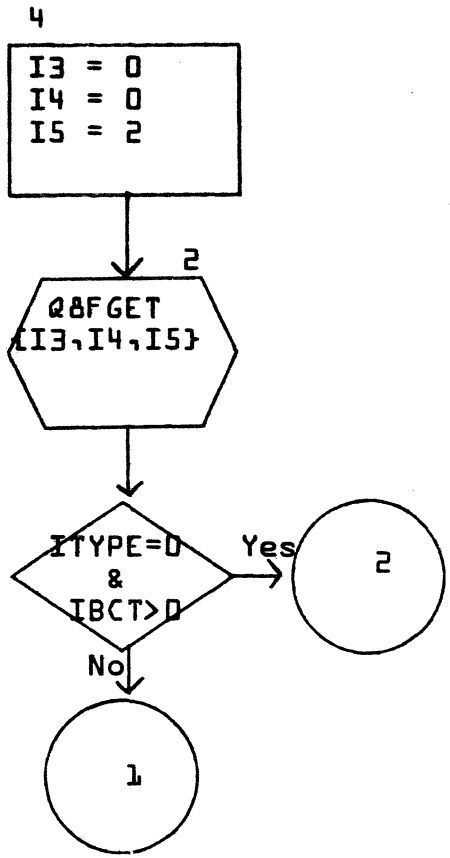
1. This routine is called from Q8IFRM and Q8TRAN. Its function is to interpret the FORMAT statement and put subfield parameters into boxes accessible by the transmit routine Q8TRAN. The parameters passed are as follows:
   - IFW = field width
   - IND = no. decimal places
   - ITYPE = type of conversion
   - IRF = field repeat count
   - IRG = grout repeat count
   - IBCT = bracket count
   - ISWR = I/O switch
   - ILIST = FORMAT statement accompanied by I/O list
   - IANYL = switch to denote existance of some conversion specification.

2. Gets next character from read buffer or puts next character in output buffer. Its 3 parameters are:
   - I3 = next character loc.
   - I4 = save char. position/ or No.
   - I5 = reposition/or no

3. Routine initiates I/O.

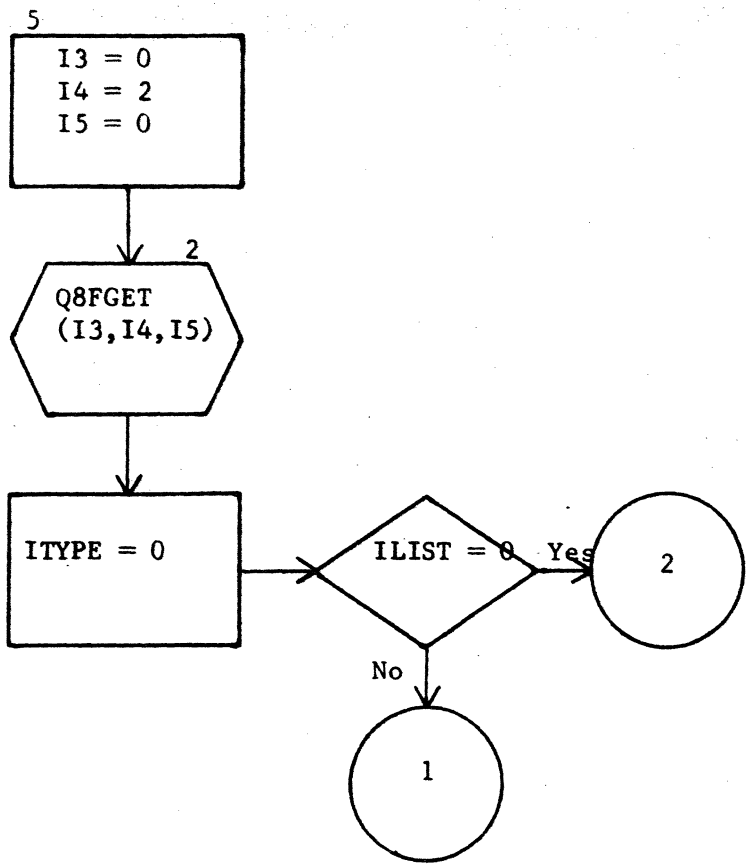


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	Q8FS	PAGE 26 OF 89	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 5-64	TASK NAME		

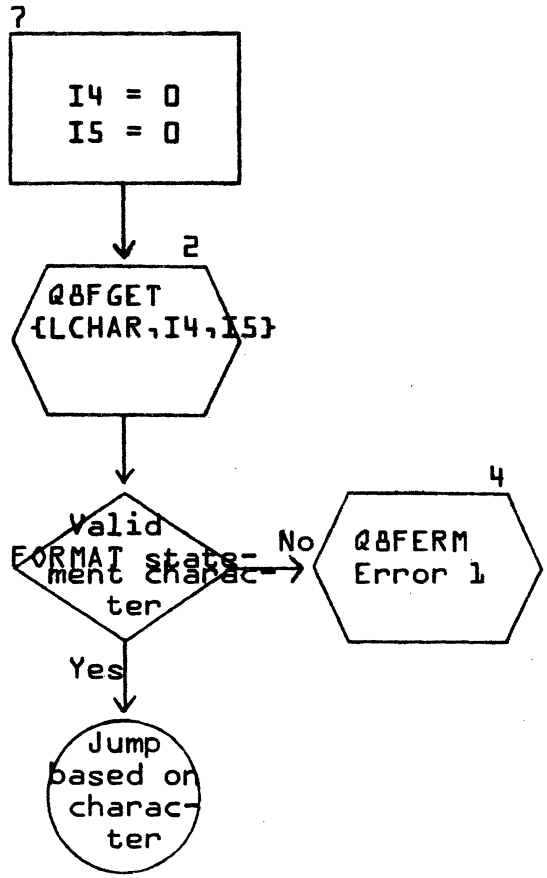


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8FS PAGE 27 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

4. Error 1 - Invalid character in FORMAT statement.

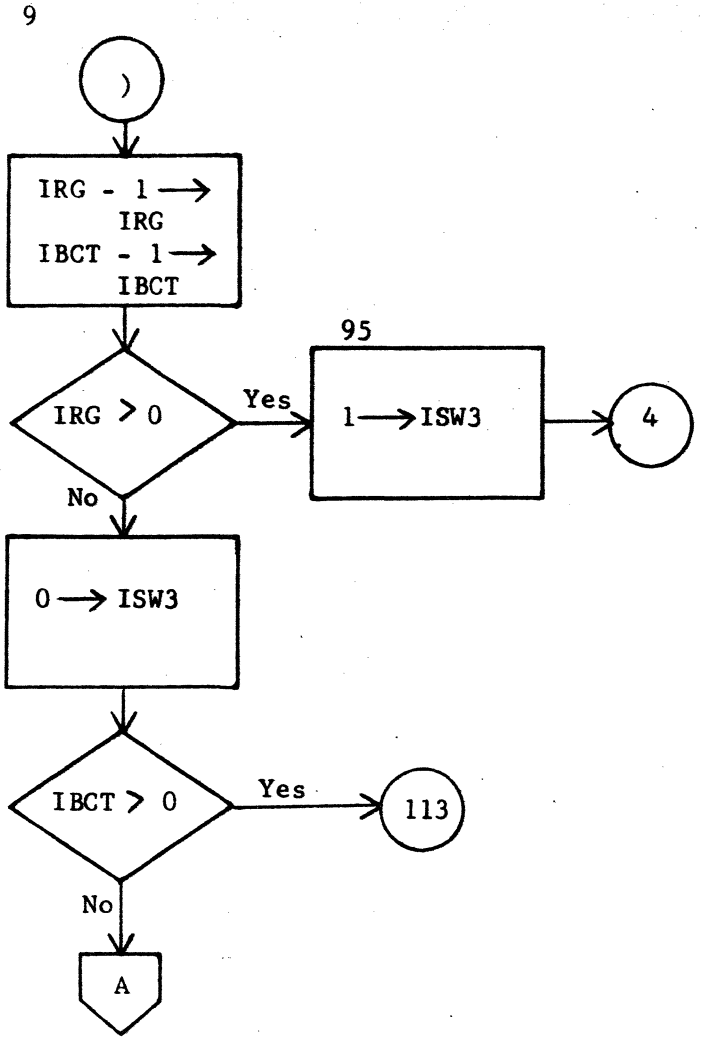


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☒	Q8FS	PAGE 28 OF 89	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE 3-67	TASK NAME		

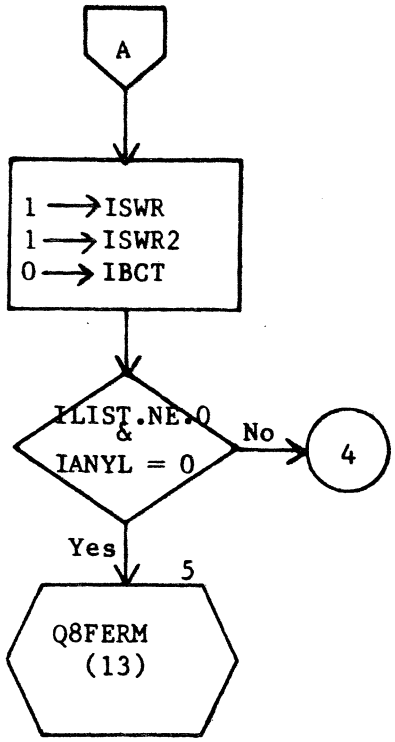


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	Q8FS	PAGE 29 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

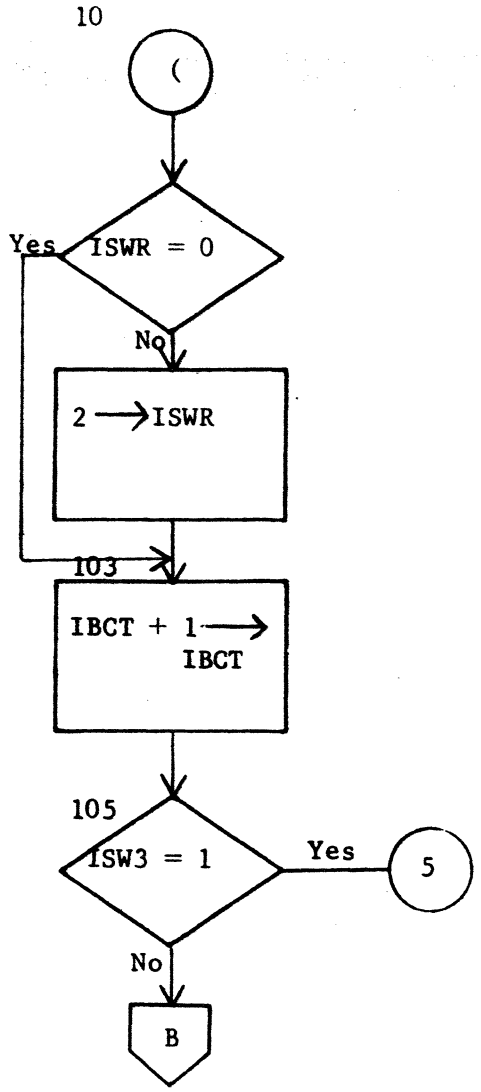
5. Error 13 - I/O list with READ, WRITE statement but no conversion specification in FORMAT statement.



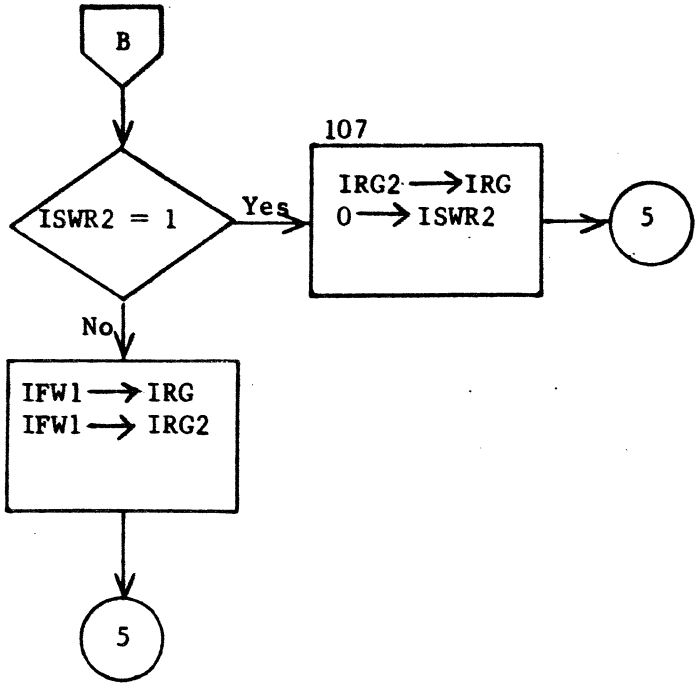
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS LMS. MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	08FS PAGE 30 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IM	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒	Q8FS	PAGE 31 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 3-67	TASK NAME			

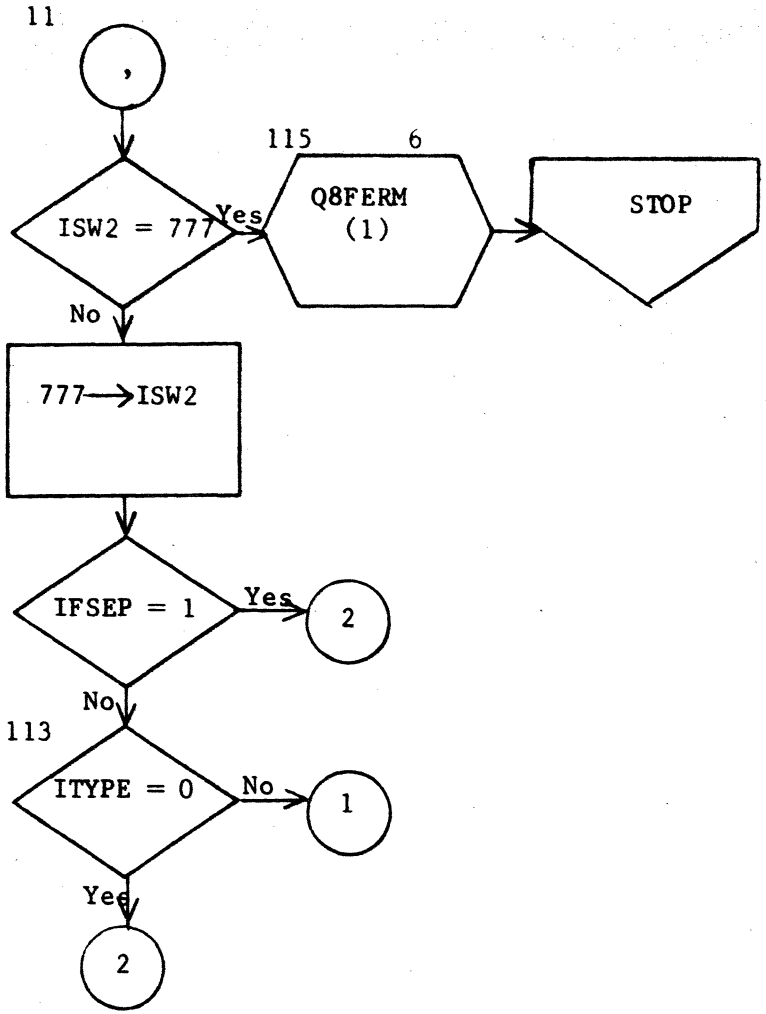


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒	Q8FS	PAGE 32 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 3-67	TASK NAME			

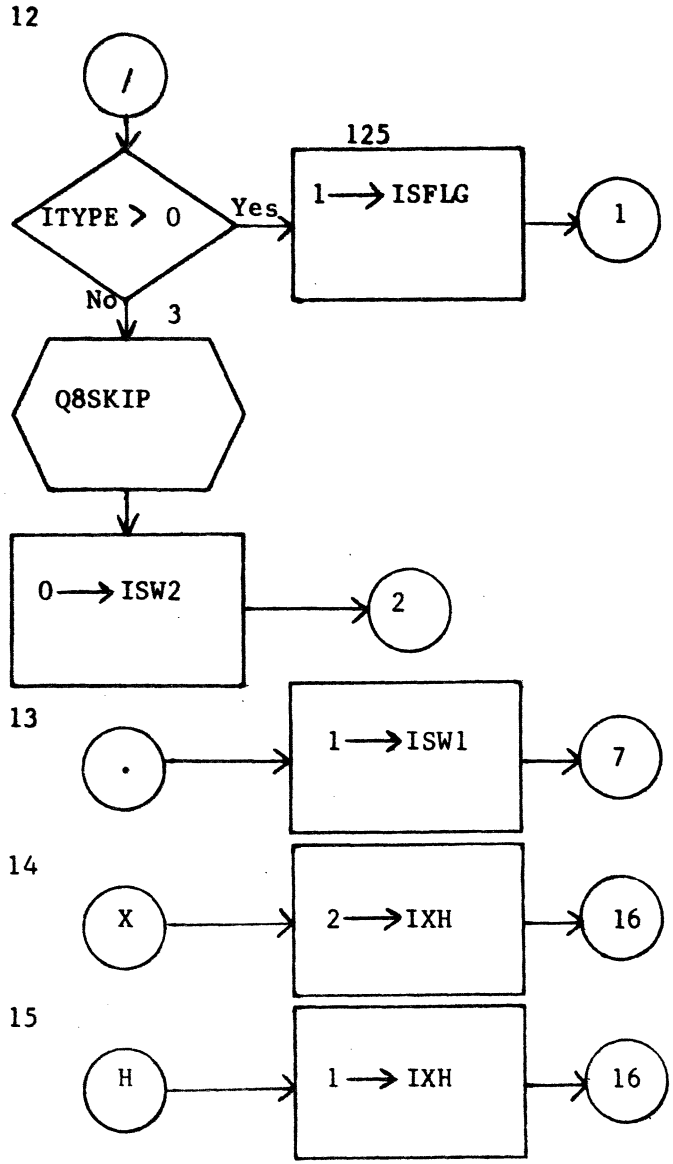


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 7165	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	08FS	PAGE 33 OF 89	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 5.67	TASK NAME		

6. Error 1 - two consecutive commas in FORMAT specification is not allowed.



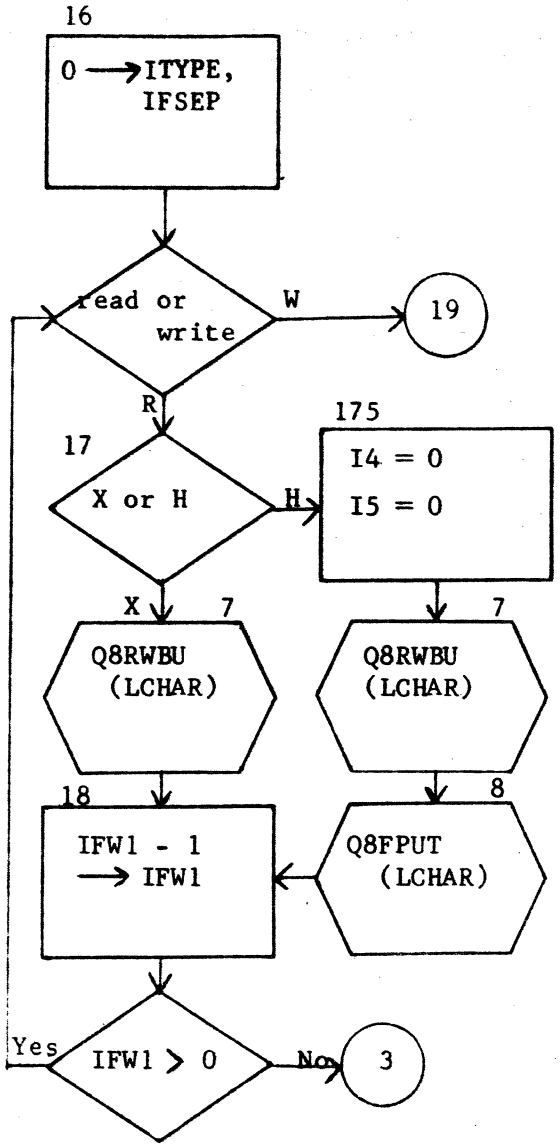
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 7CC	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8FS PAGE 34 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-6-7	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IM2 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8FS PAGE 35 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-67	TASK NAME			

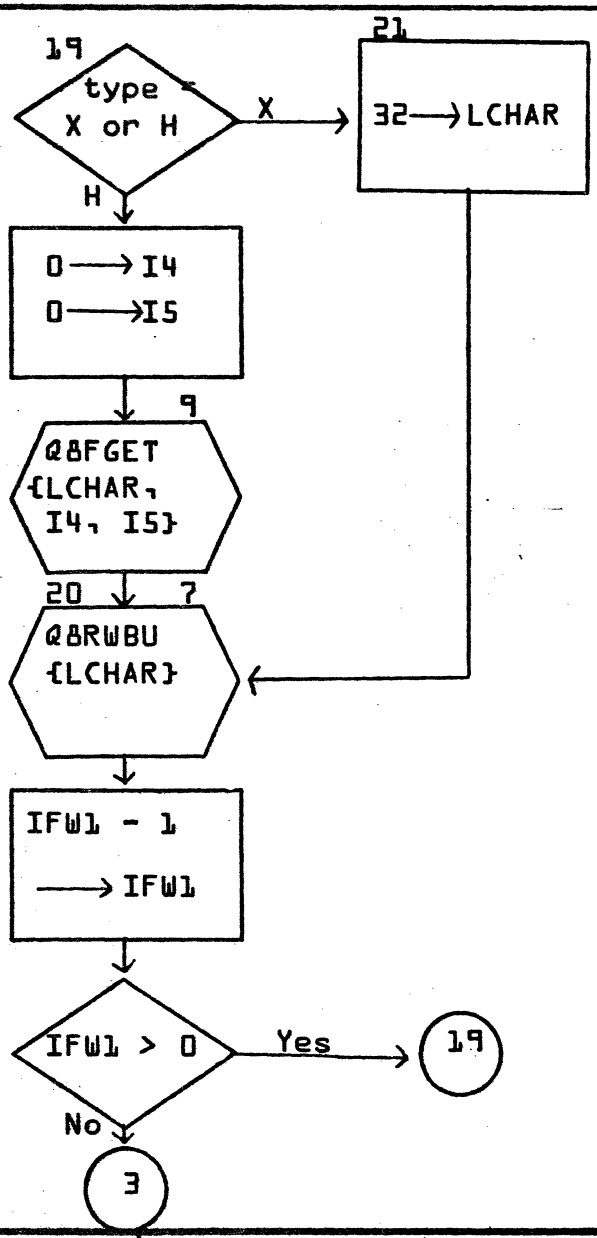
7. This routine gets the next character from the input buffer or puts the next character into the output buffer.

8. This routine puts a character into the next position of the FORMAT statement.

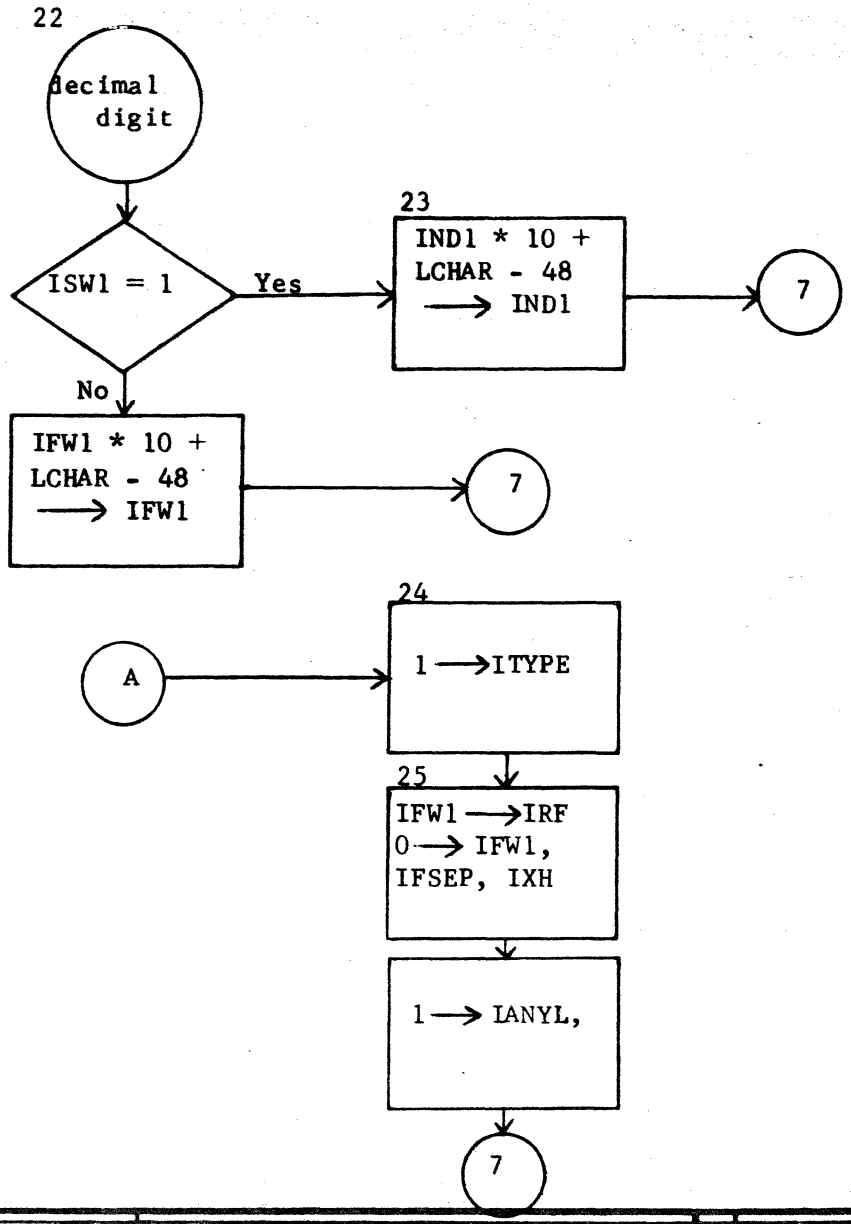


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IM	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8FS	PAGE 36 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 5-67	TASK NAME			

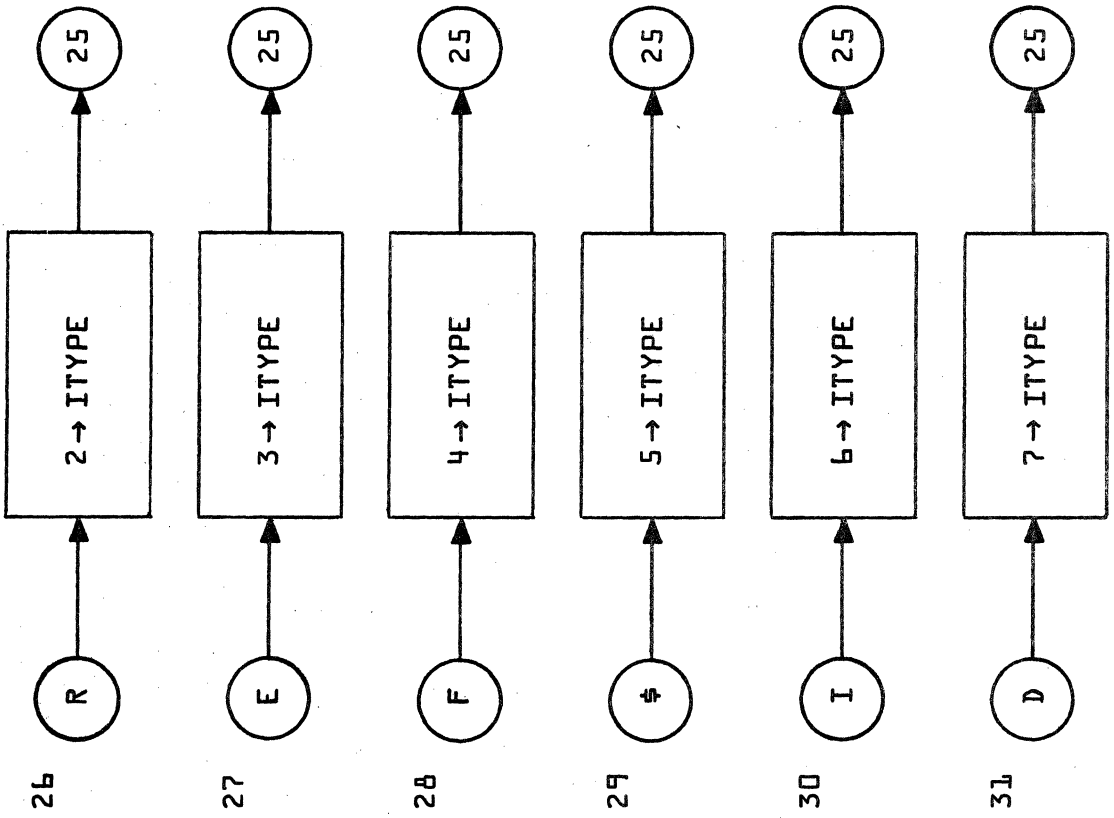
9. This routine gets the next character from the FORMAT statement.

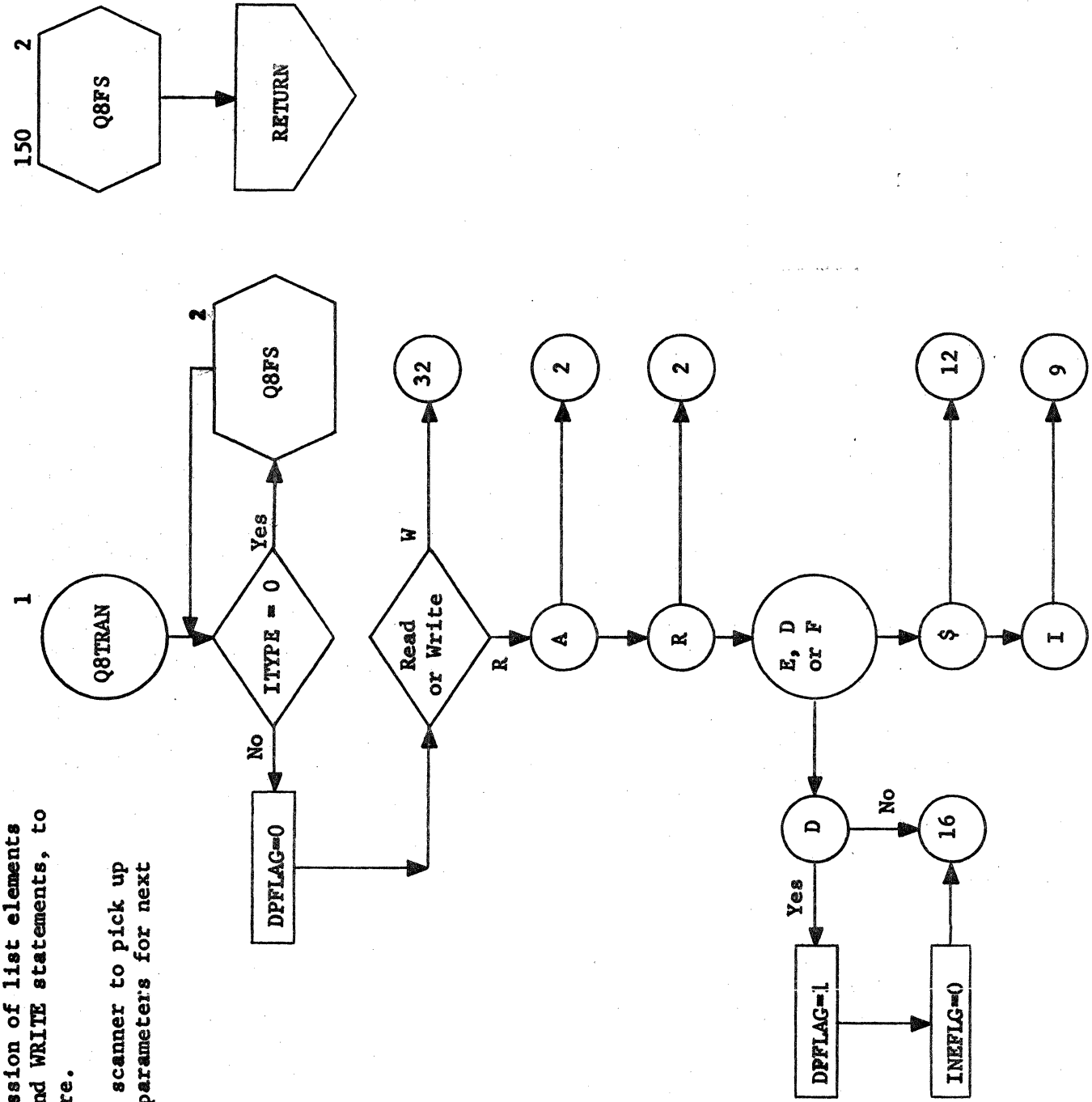


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8FS	PAGE 37 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS .1MS	MACH. TYPE FTN	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	Q8FS	PAGE 38 OF 89	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 5-67	TASK NAME		


25
25
25
25
25
25
2→ ITYPE
3→ ITYPE
4→ ITYPE
5→ ITYPE
6→ ITYPE
7→ ITYPE
R
E
F
$
I
D
26
27
28
29
30
31



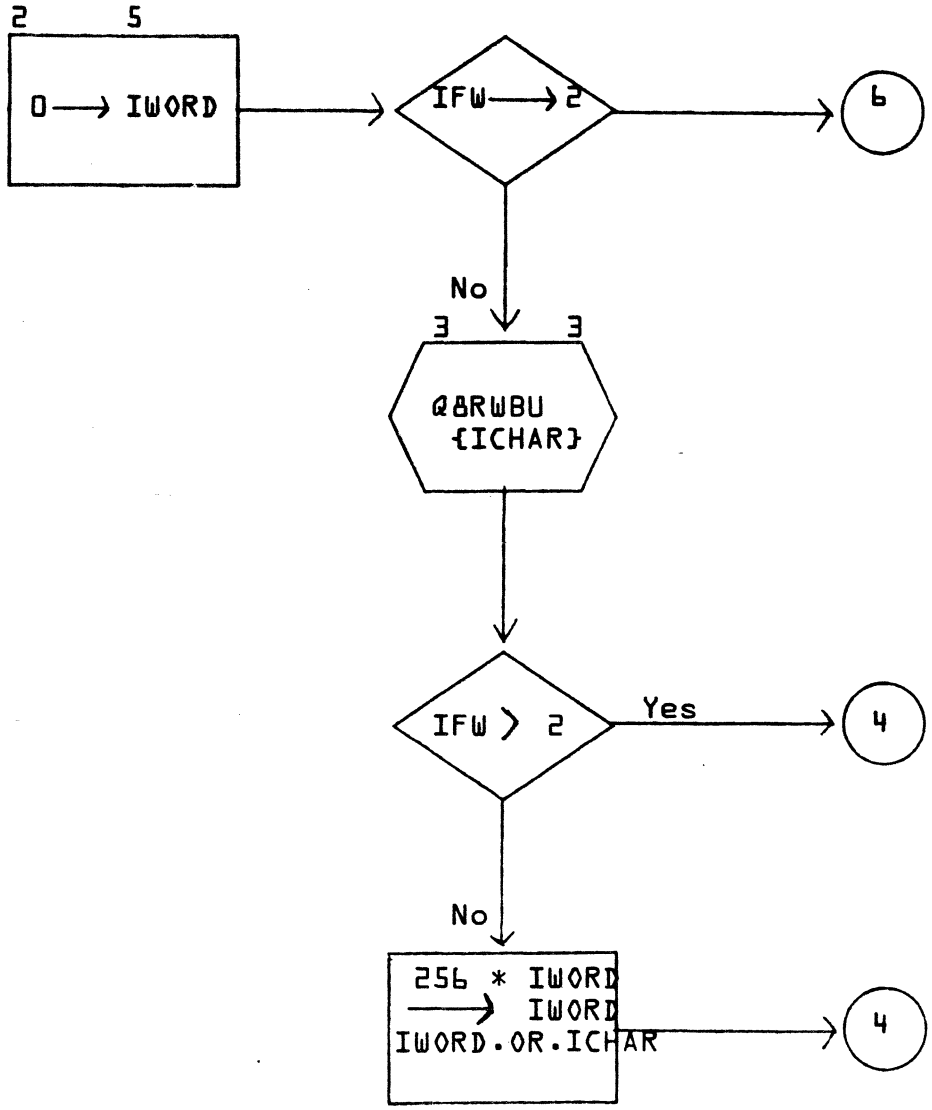
1. This routine controls the conversion and transmission of list elements from READ and WRITE statements, to and from core.

2. Call format scanner to pick up conversion parameters for next subfield.

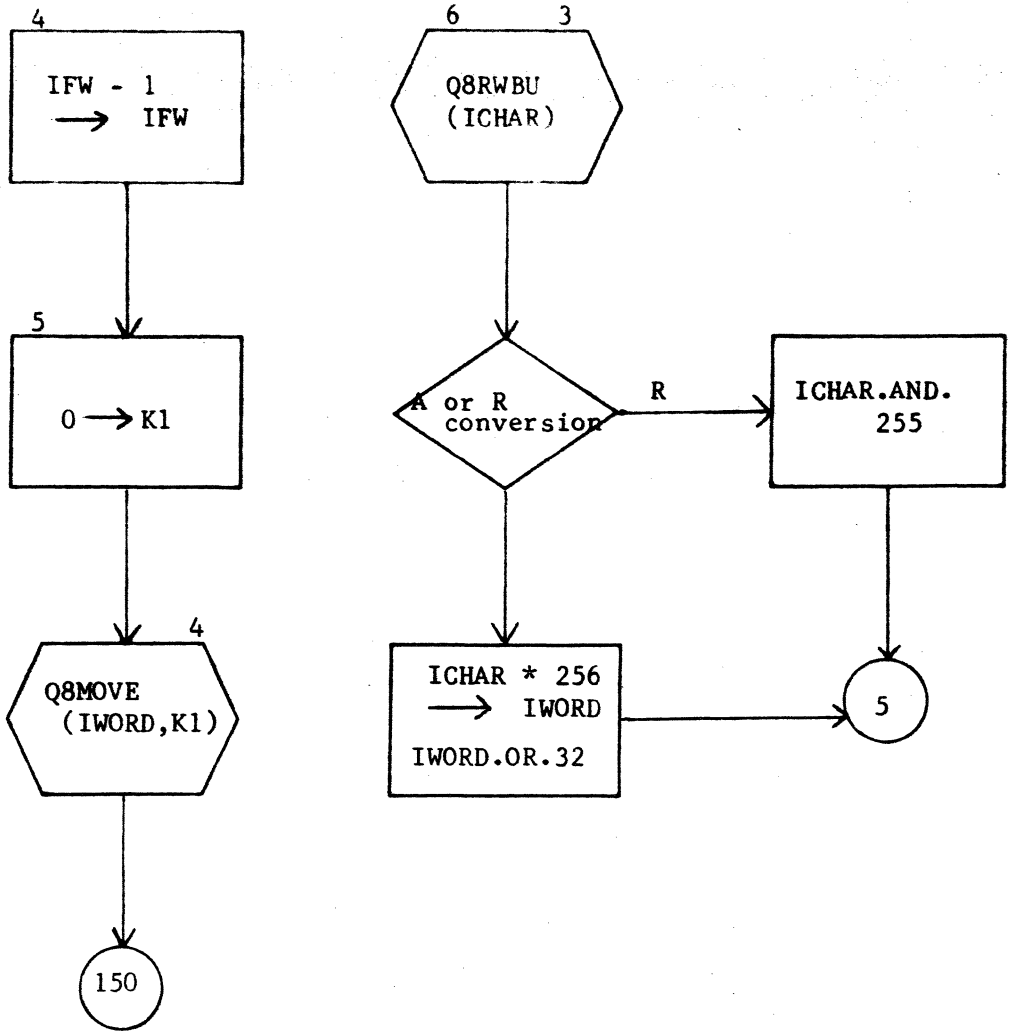
3. Pick up next character from input string or on output, place next character into output buffer.

4. This routine moves one word to or from core based on nature of I/O currently being executed. The first parameter is the word to be moved, and the second parameter is the increment to be added to the base location to determine the actual core location.

5. A and R input core - versions comes here.

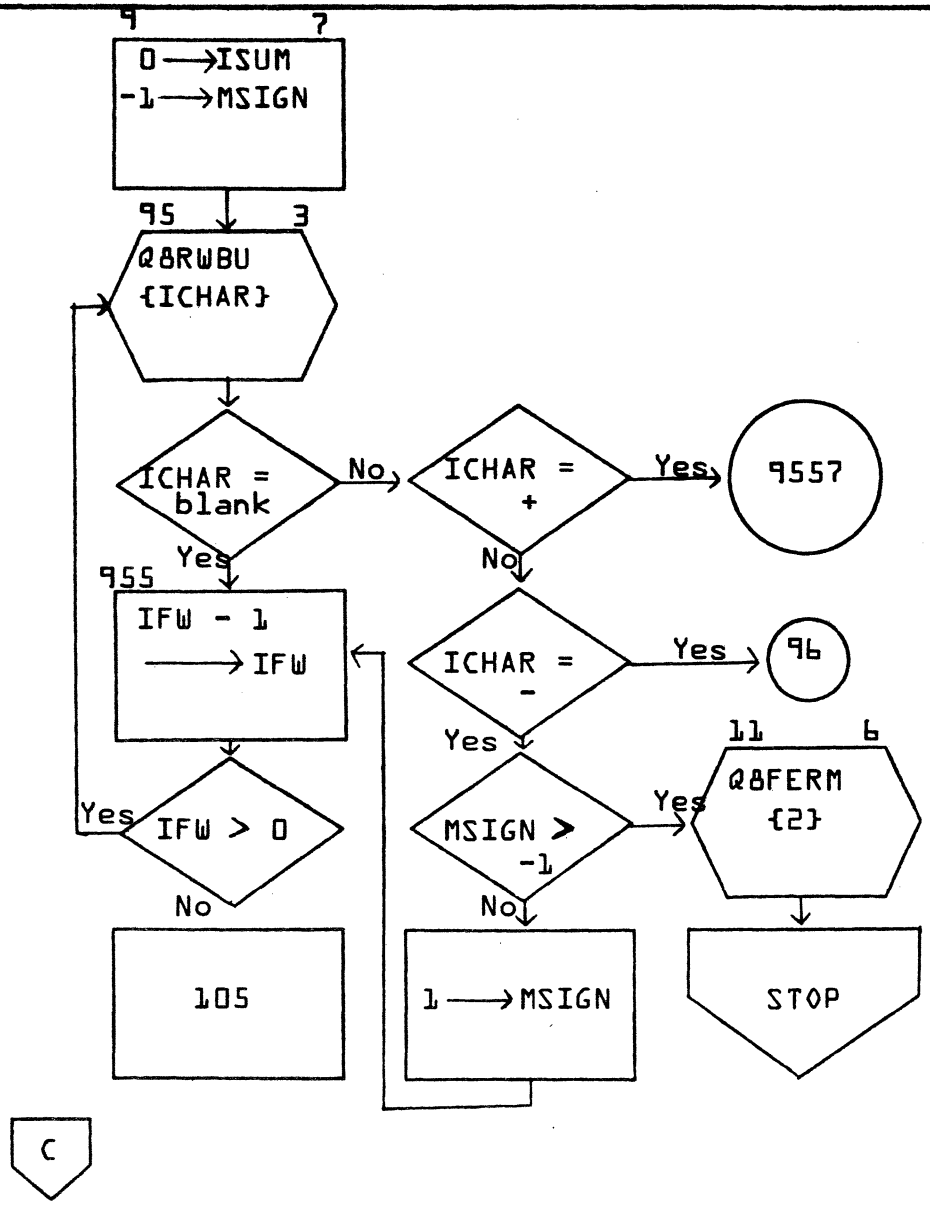


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8TRAN	PAGE 41 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

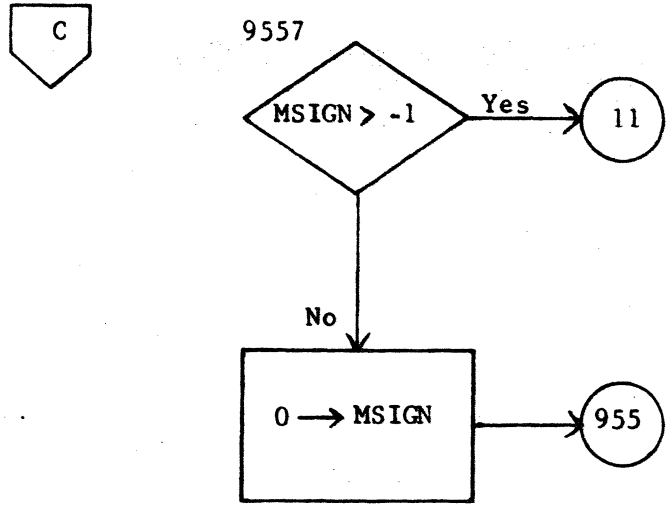


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS LM	MACH. TYPE 7600	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☑	Q8TRAN	PAGE 42 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 3-1-7	TASK NAME			

6. Error 2. Bad character in input string.
7. Integer input conversion comes here.

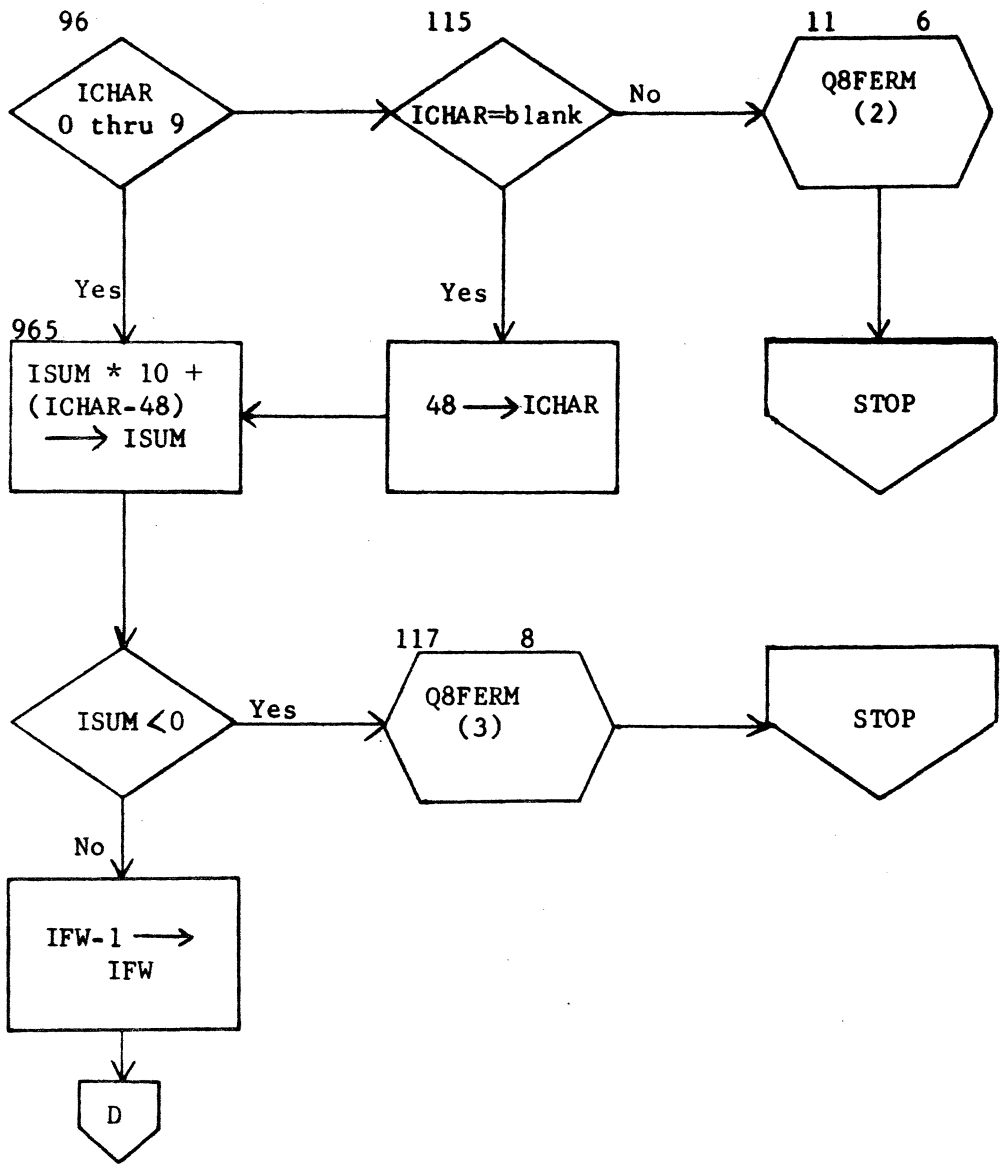


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	Q8TRAN	PAGE 43 OF 89	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



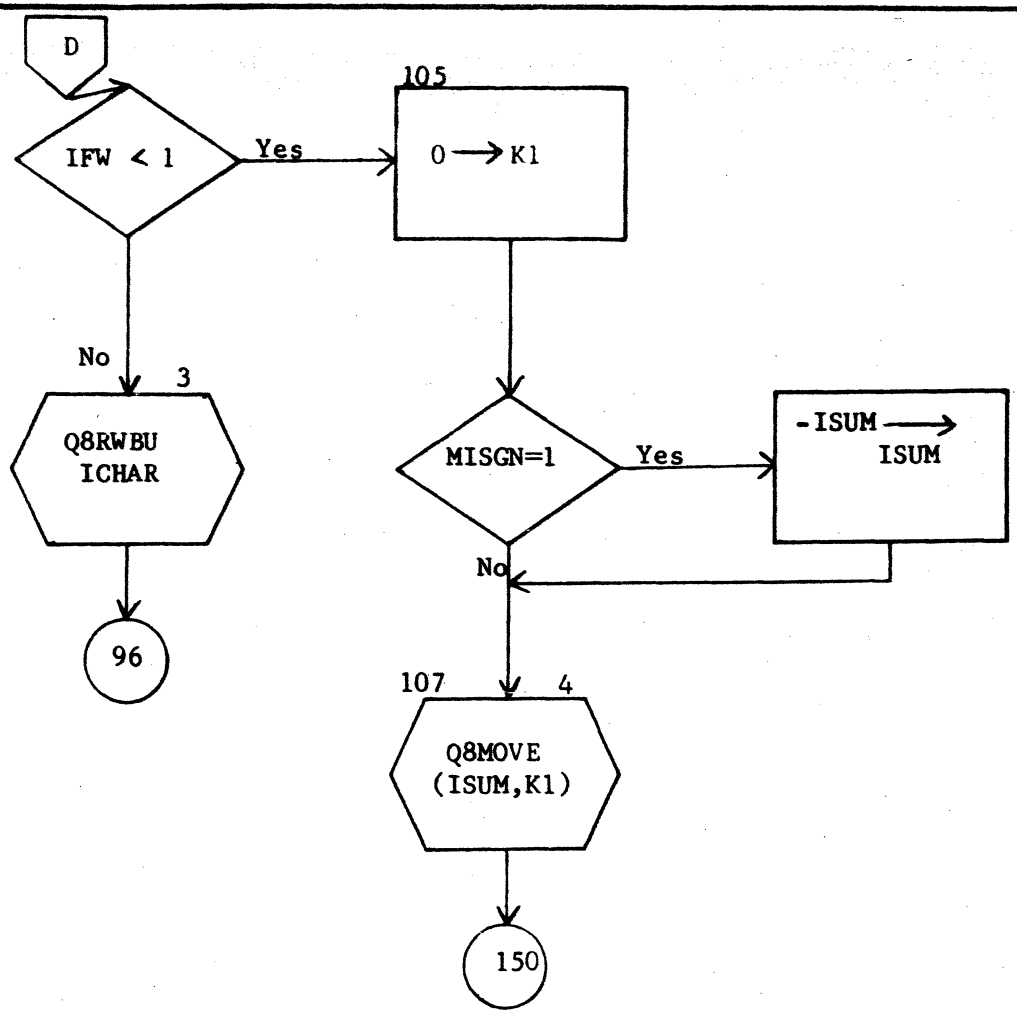
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8TRAN PAGE 44 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 5-1-7	TASK NAME			

8. Error 3. Magnitude of number being processed exceeds the full specification.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS JMC	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	Q8TRAN	PAGE 45 OF 49	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 5-6-69	TASK NAME			

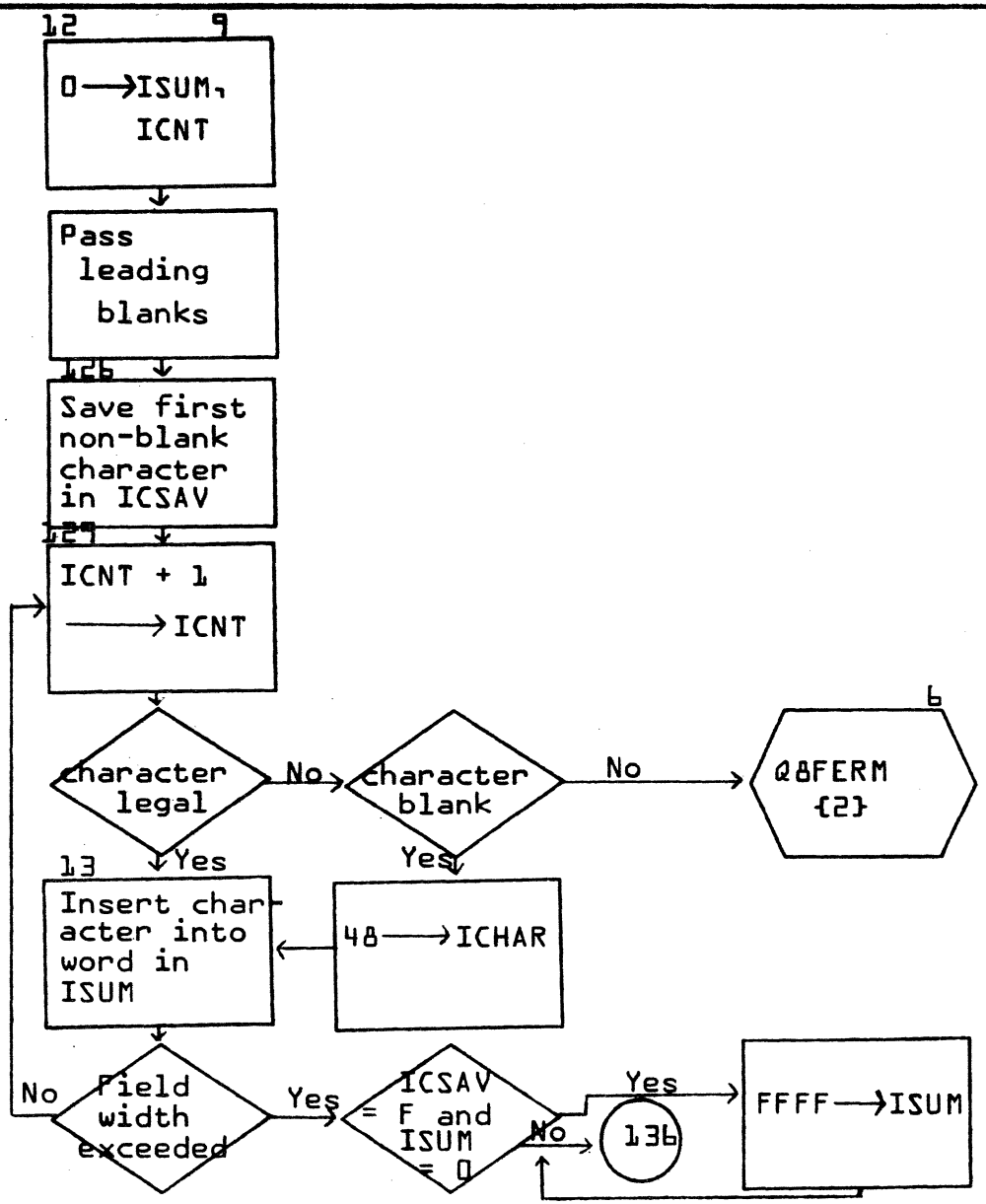
1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	Q8TRAN	PAGE 46 OF 84	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 3-6-7	TASK NAME		

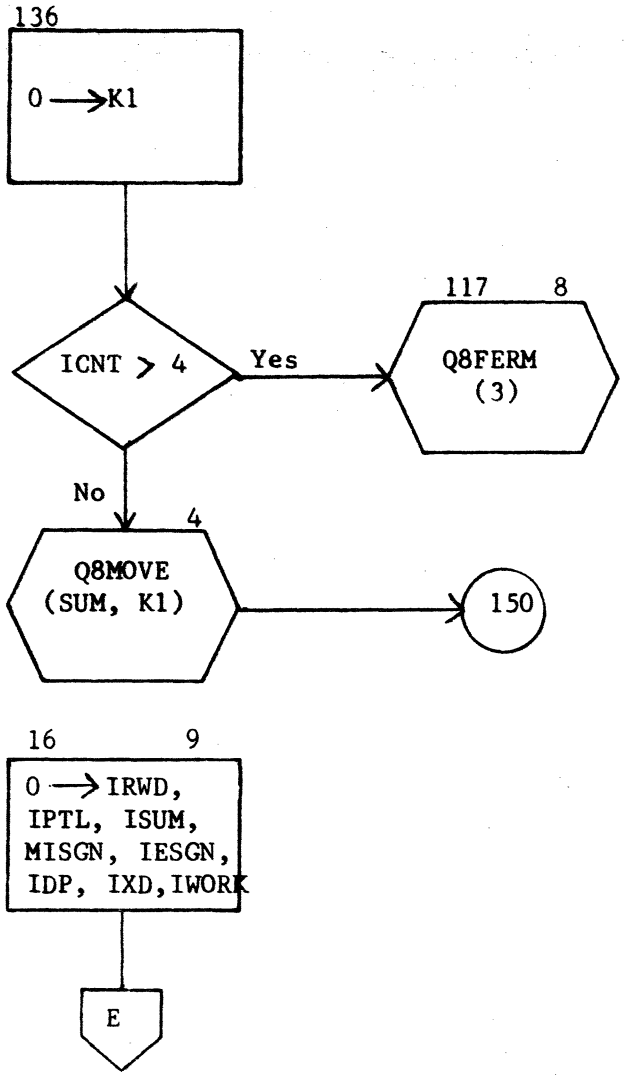
AA1385 (FORME , A127-1)

9. Hexadecimal input conversion comes here.

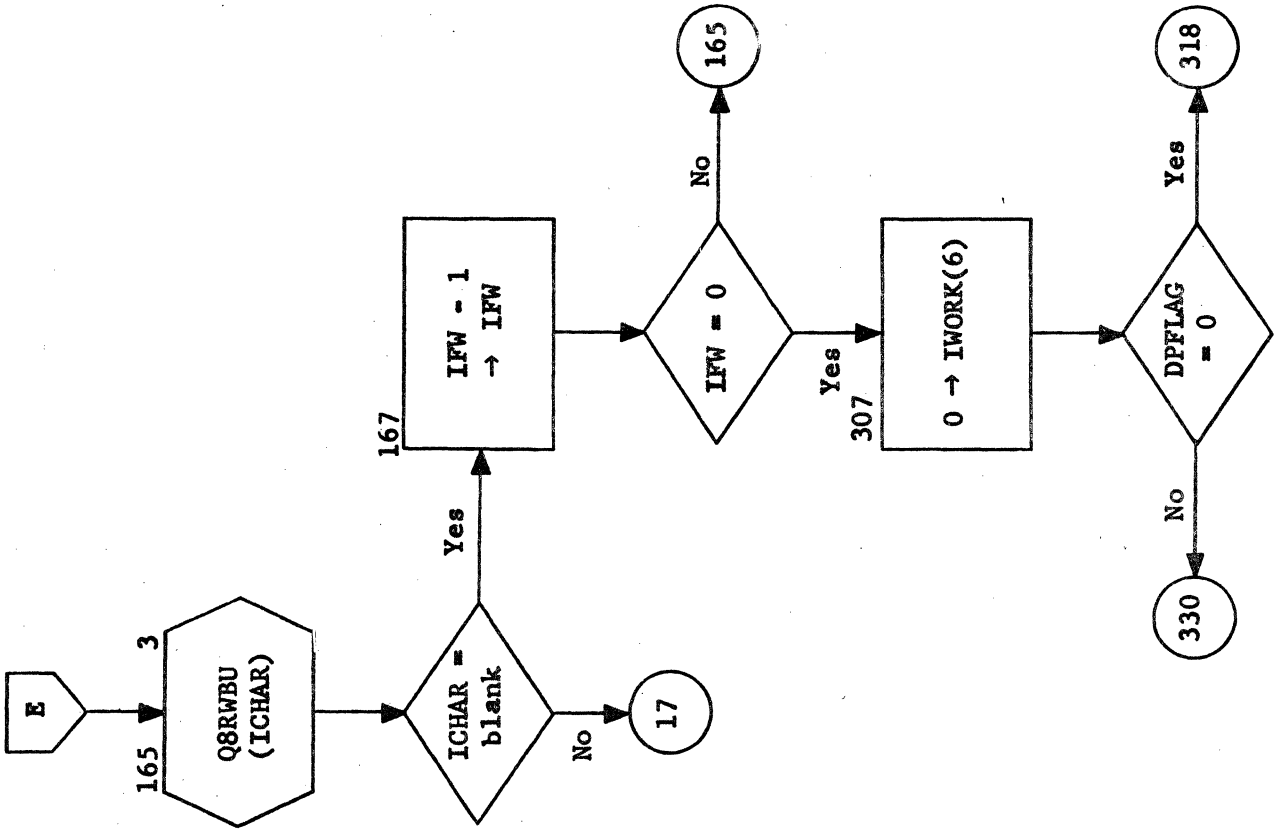


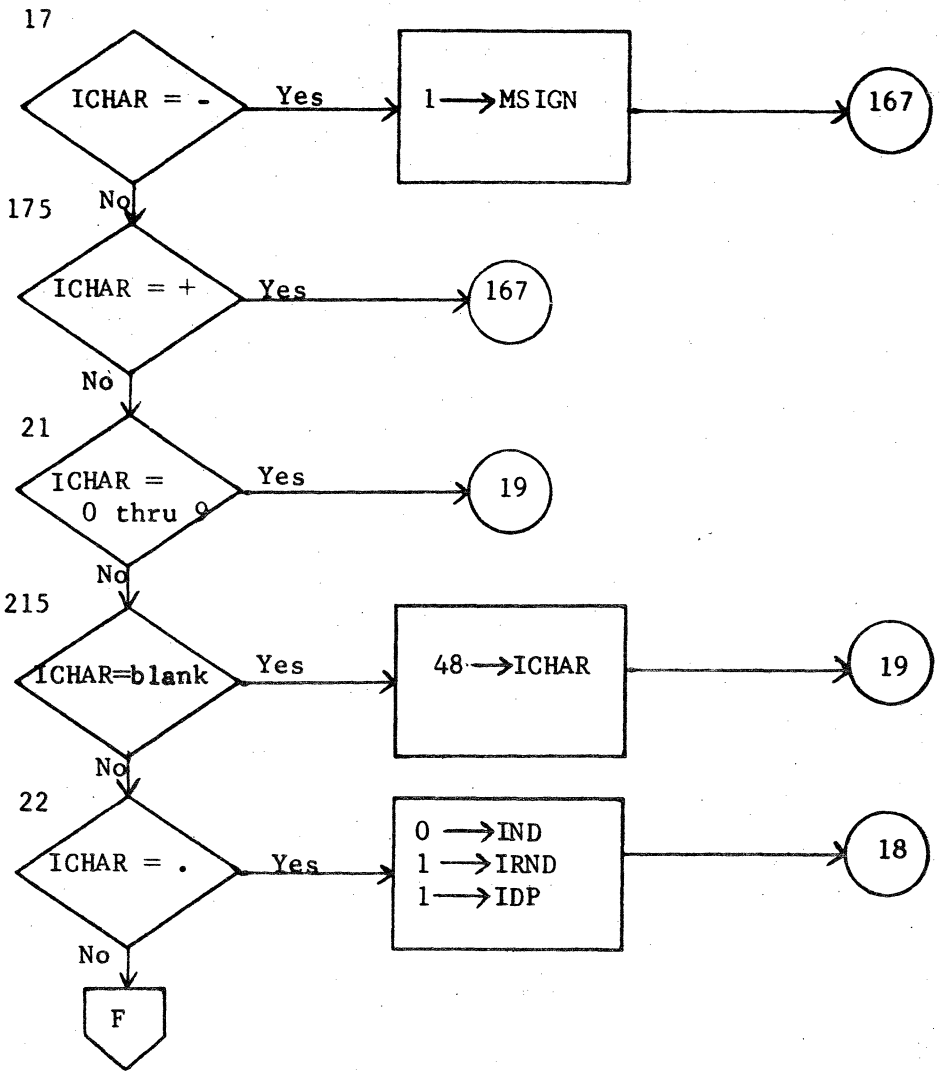
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8TRAN	PAGE 47 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

9. E and F input conversions come here.



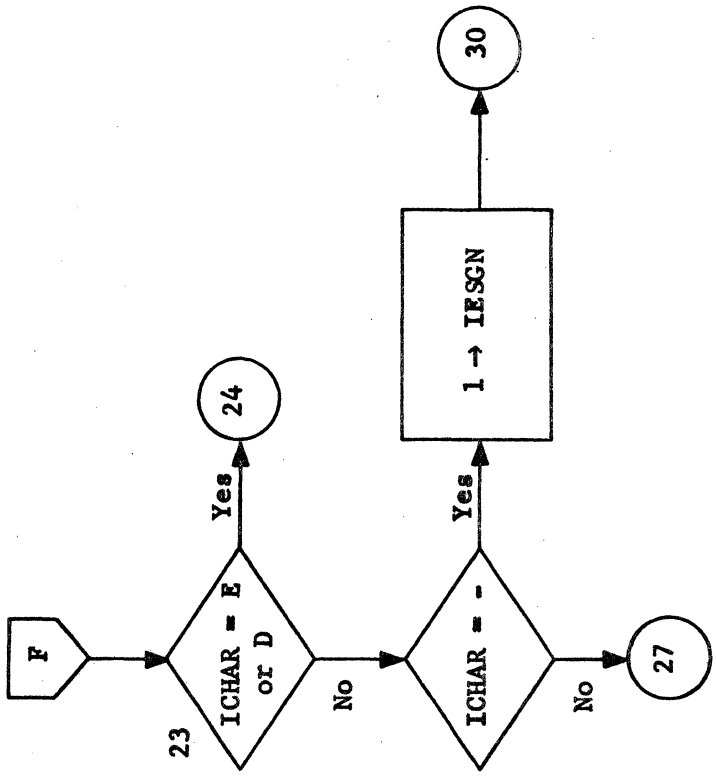
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	Q8TRAN PAGE 48 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE 3-1-71	TASK NAME			

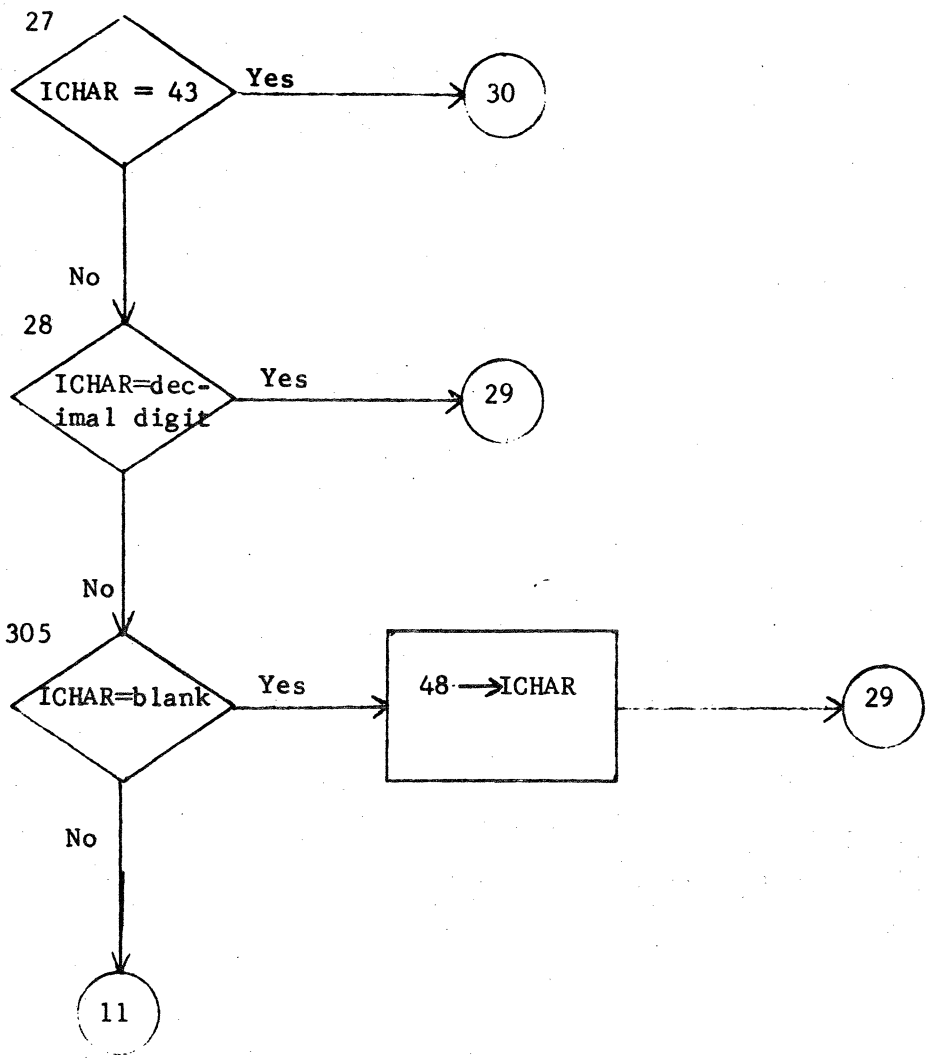

E
3
165
Q8RWBU (ICHAR)
ICHAR = blank
No
17
Yes
167
IFW = 1 → IFW
IFW = 0
No
165
Yes
307
0 → IWORK(6)
DPFLAG = 0
Yes
318
No
330



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE [ ] FLOWCHART [X] DECISION TABLE [ ] OTHER [ ]	DOCUMENT CLASS Misc. MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8TRAN PAGE 50 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 3-6-64	TASK NAME			

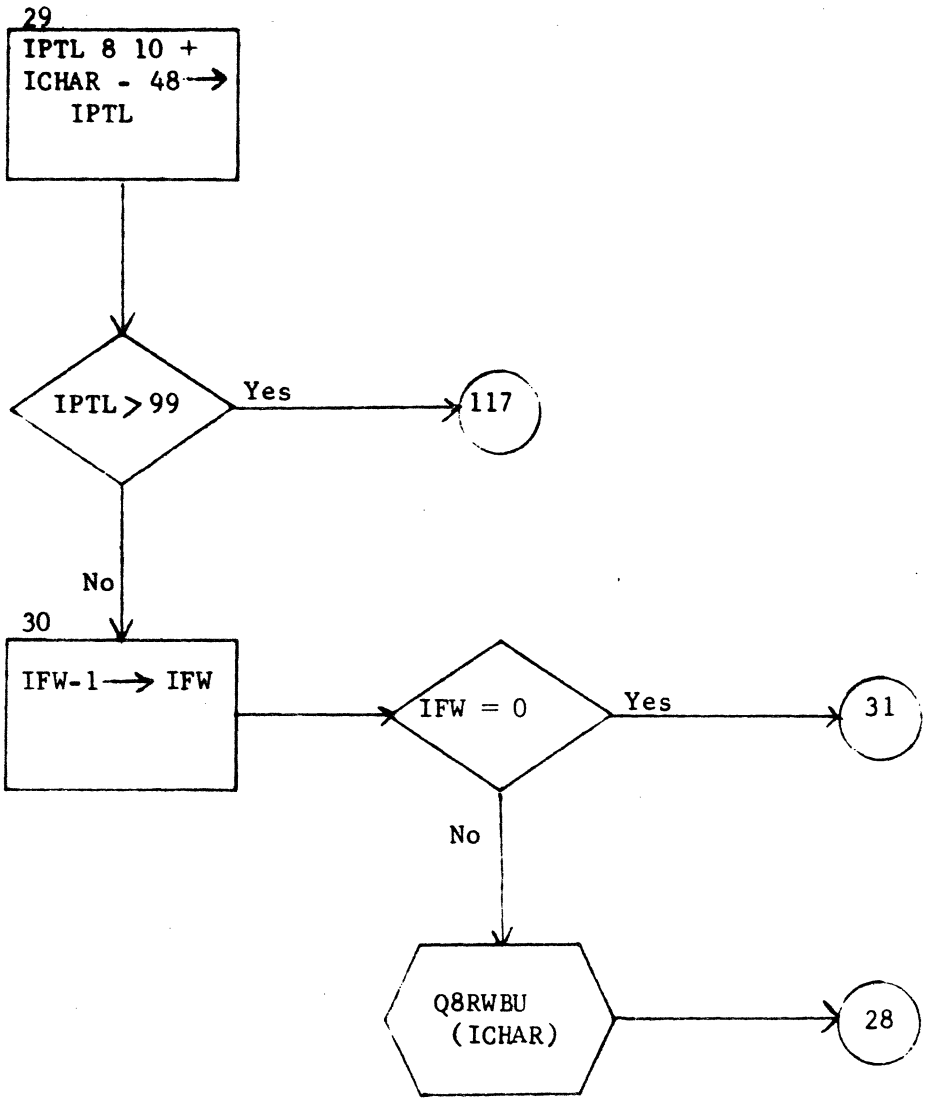
AA1385 (FORM) CA127-1)


F
23
ICHAR = E or D
Yes
24
No
ICHAR = -
Yes
1 → IESGN
30
No
27

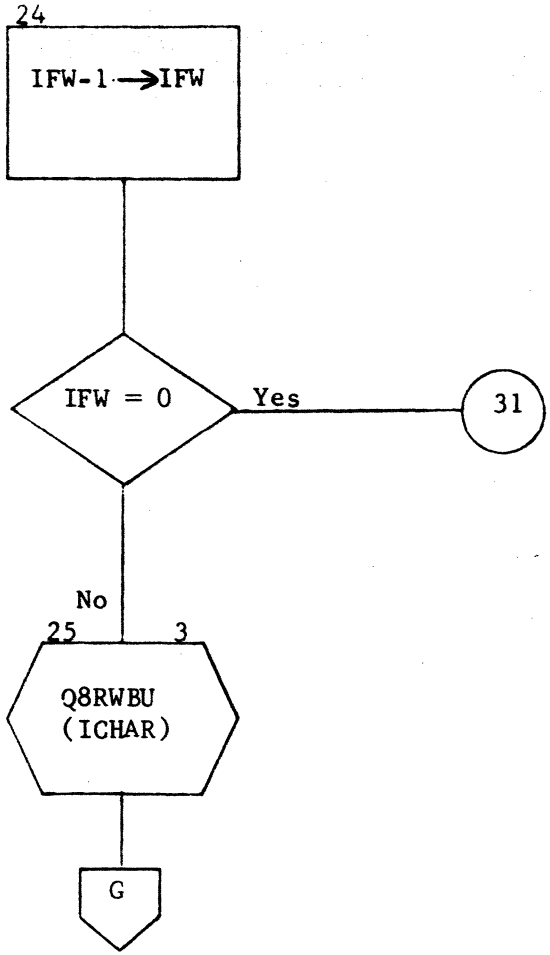


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS JM	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☑	Q8TRAN	PAGE 52 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 3/1/7	TASK NAME			

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS Ext. MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8TRAN PAGE 53 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 5-67	TASK NAME			



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS Util.	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒	Q8TRAN	PAGE 54 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 3/6/1	TASK NAME			

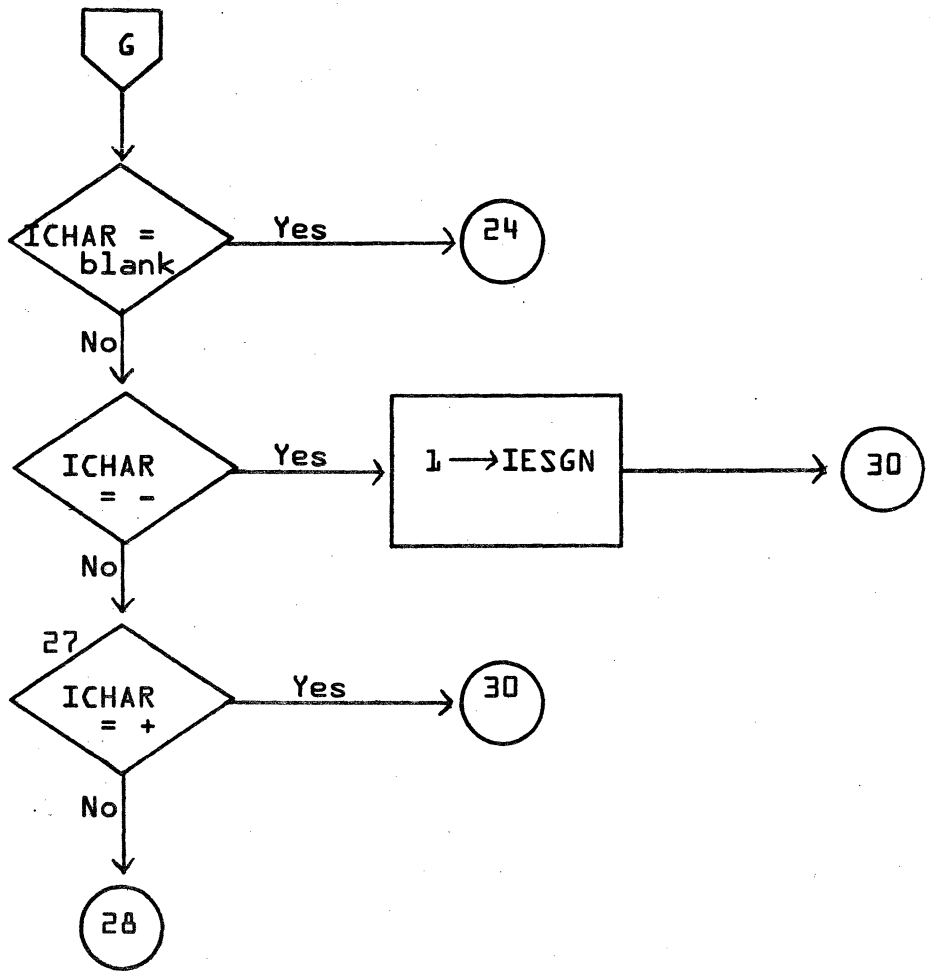
AA1385 (FORMERLY CA127-1)

A

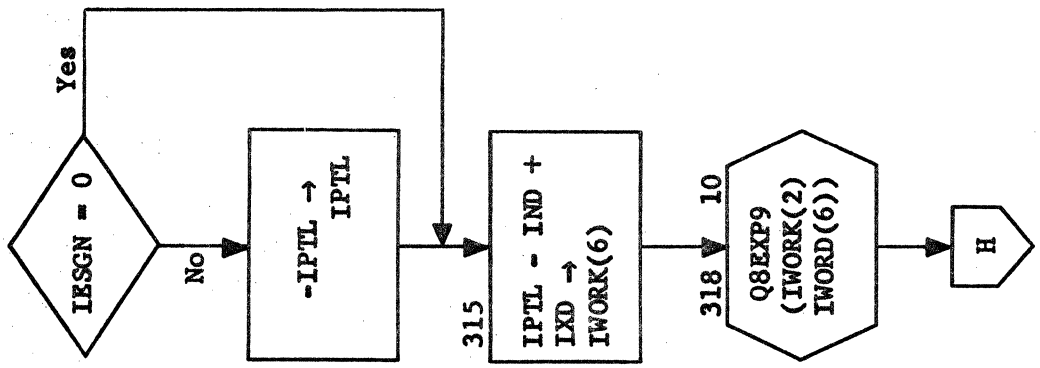
B

C

D

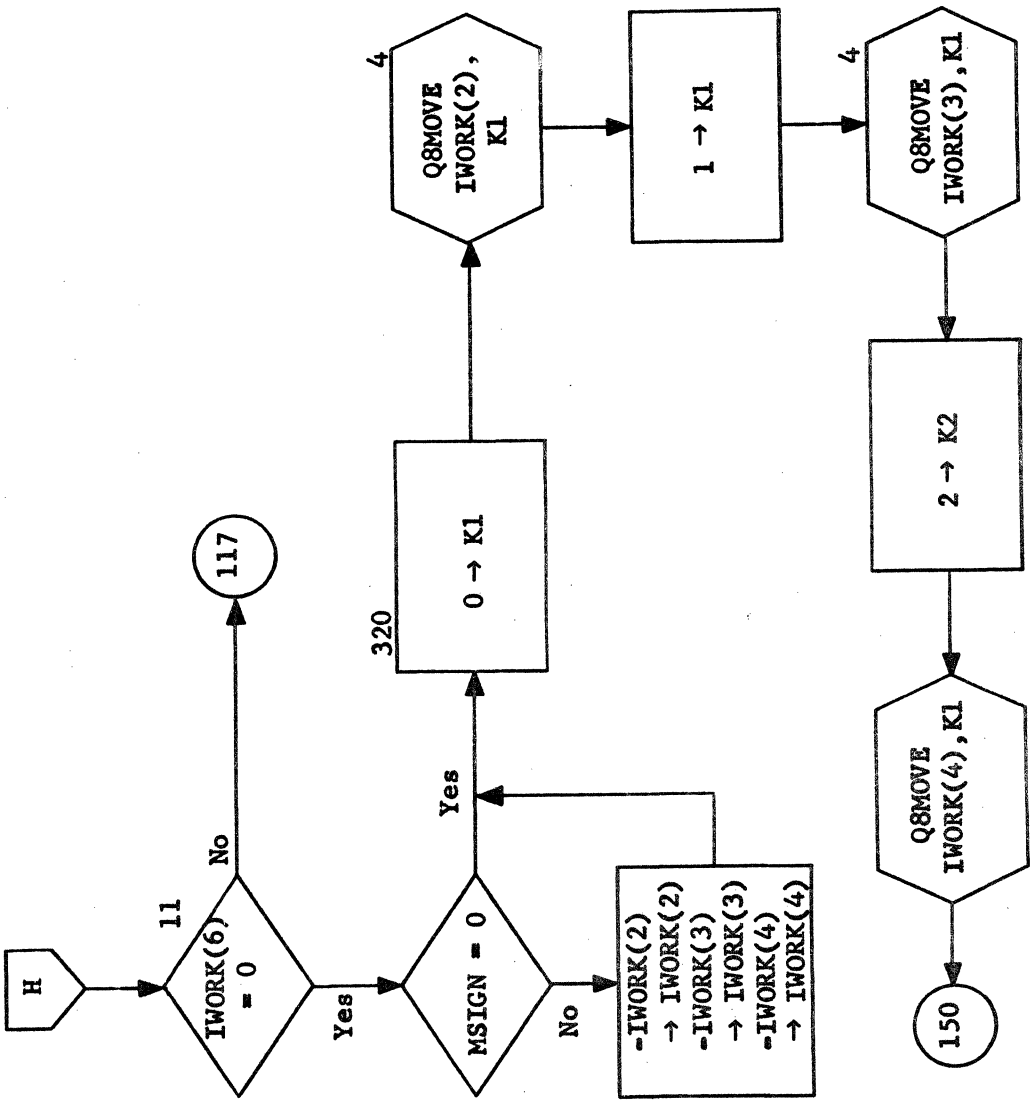


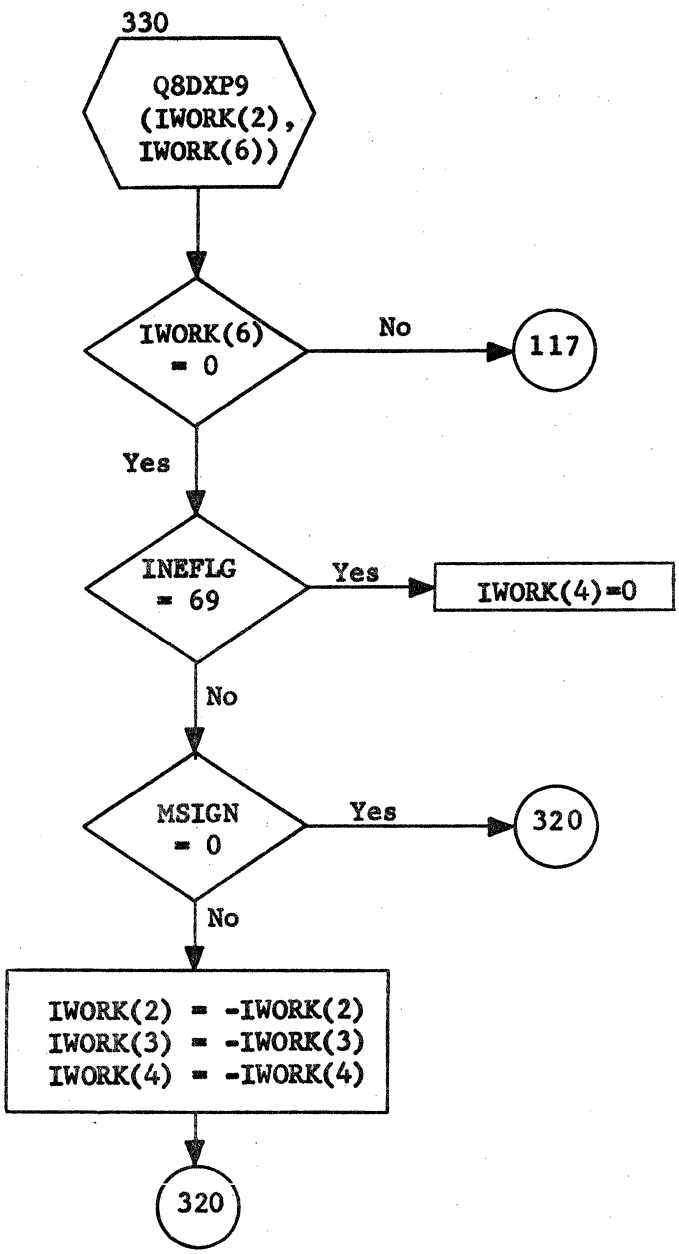
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	QBTRAN	PAGE 55 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



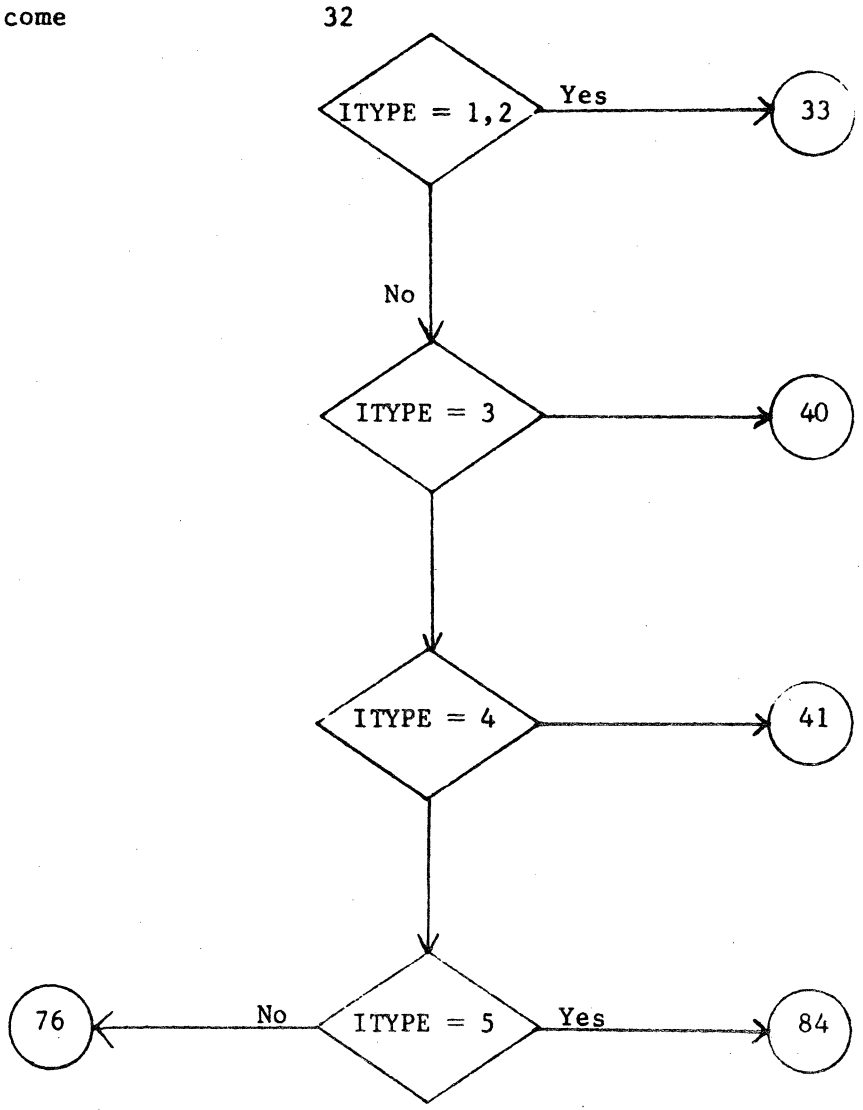
10. This routine picks up a decimal integer at IWORD(2), IWORK(3), IWORK(4), and IWORK(5) and the corresponding decimal exponent at IWORK(6) and converts these values into the binary equivalent in the real number form required by the floating point package.

11. If magnitude of number to be converted is excessive, IWORK(6) is set to non-zero.

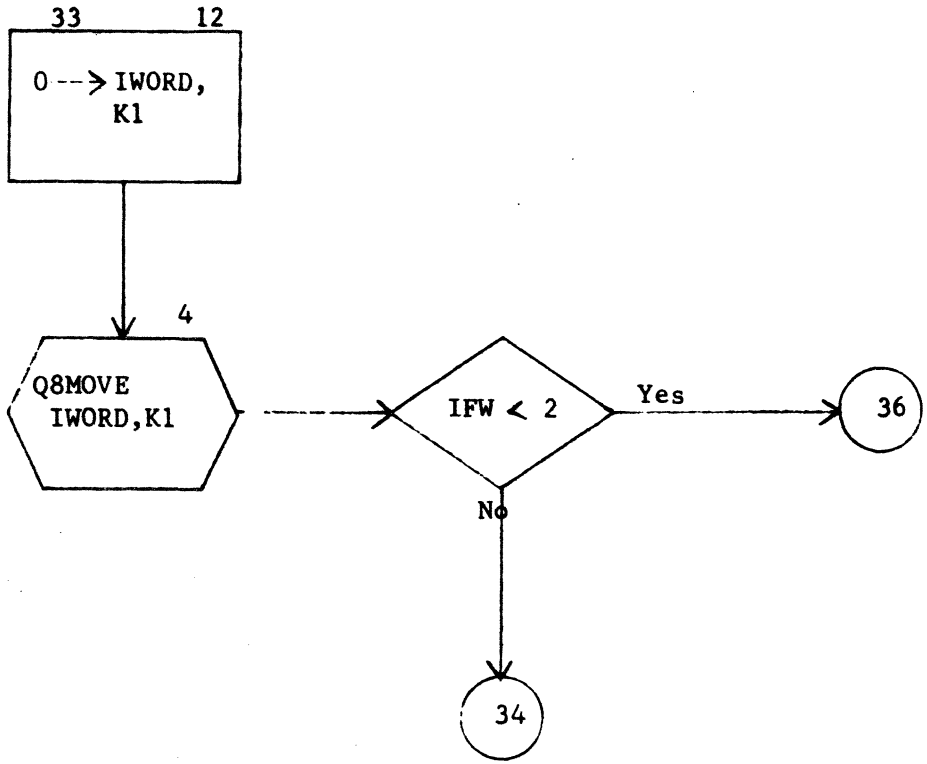

H
11
IWORK(6) = 0
No
117
Yes
MSIGN = 0
Yes
No
320
0 → K1
-IWORK(2) → IWORK(2)
-IWORK(3) → IWORK(3)
-IWORK(4) → IWORK(4)
4
Q8MOVE IWORK(2), K1
1 → K1
4
Q8MOVE IWORK(3), K1
2 → K2
Q8MOVE IWORK(4), K1
150


330
Q8DXP9 (IWORK(2), IWORK(6))
IWORK(6) = 0
No
117
Yes
INEFLG = 69
Yes
IWORK(4)=0
No
MSIGN = 0
Yes
320
No
IWORK(2) = -IWORK(2)
IWORK(3) = -IWORK(3)
IWORK(4) = -IWORK(4)
320

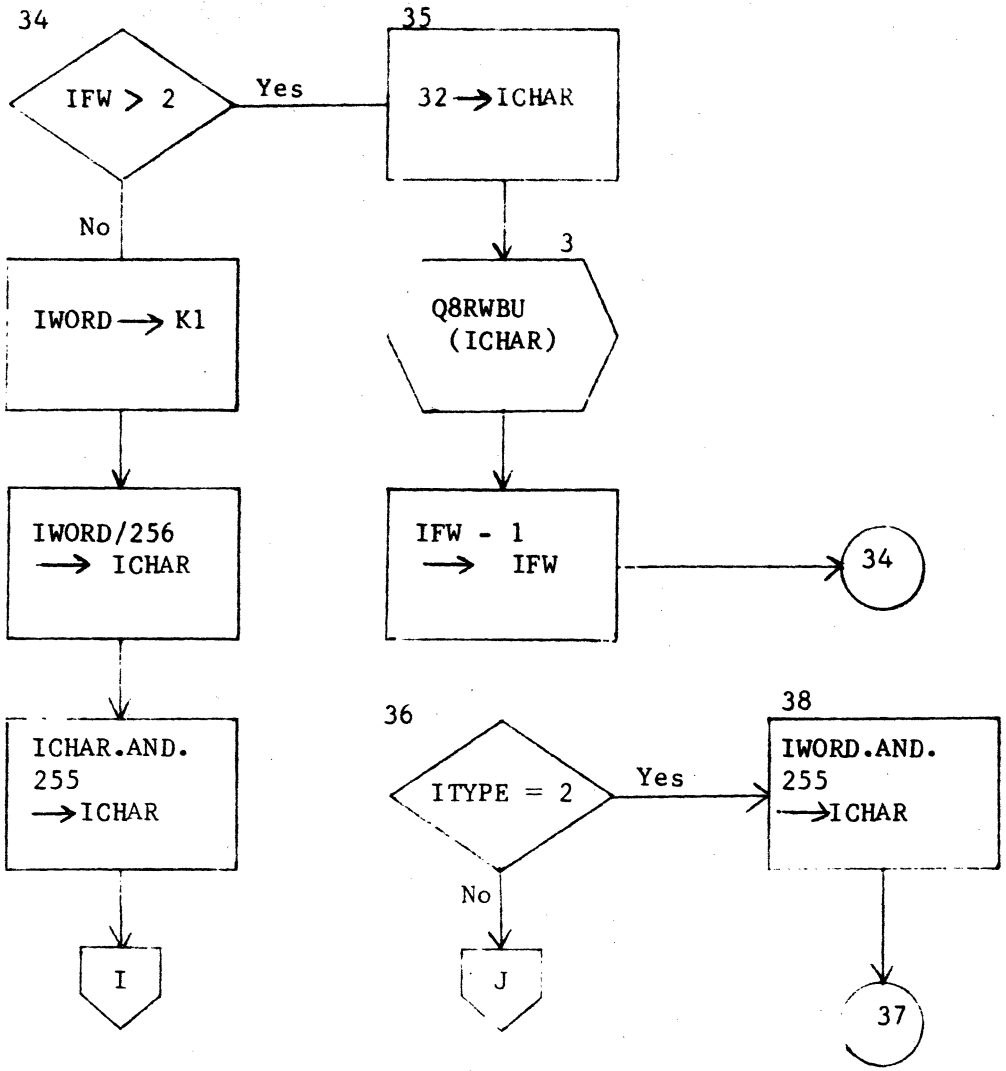
A and R output conversions come here.



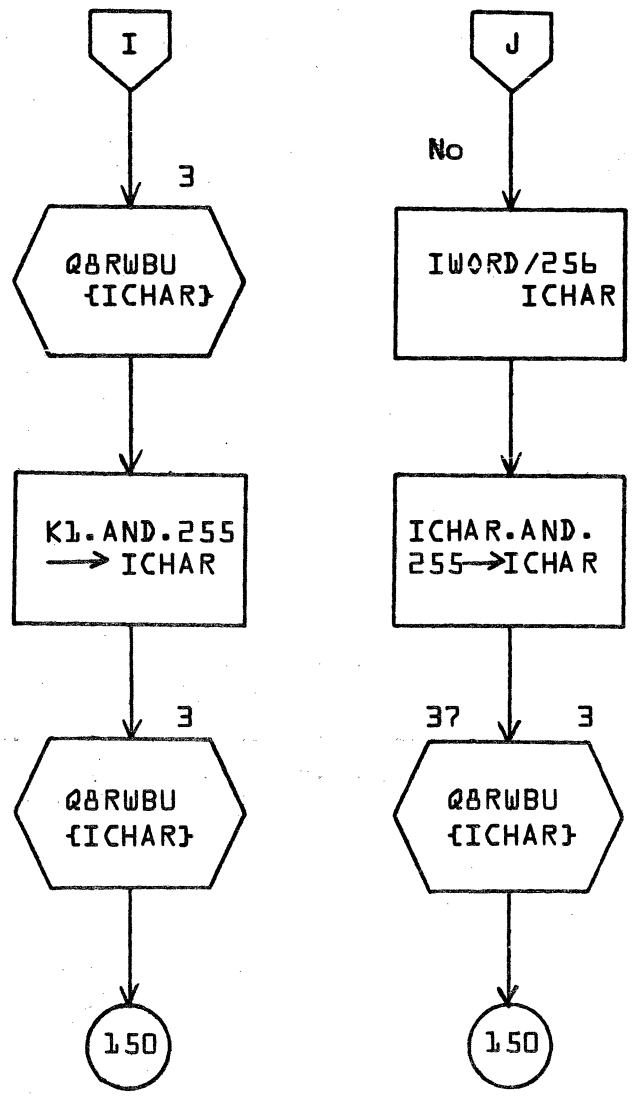
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS Lang	MACH. TYPE Cyber	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒	Q8TRAN	PAGE 58 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE 7-11	TASK NAME			



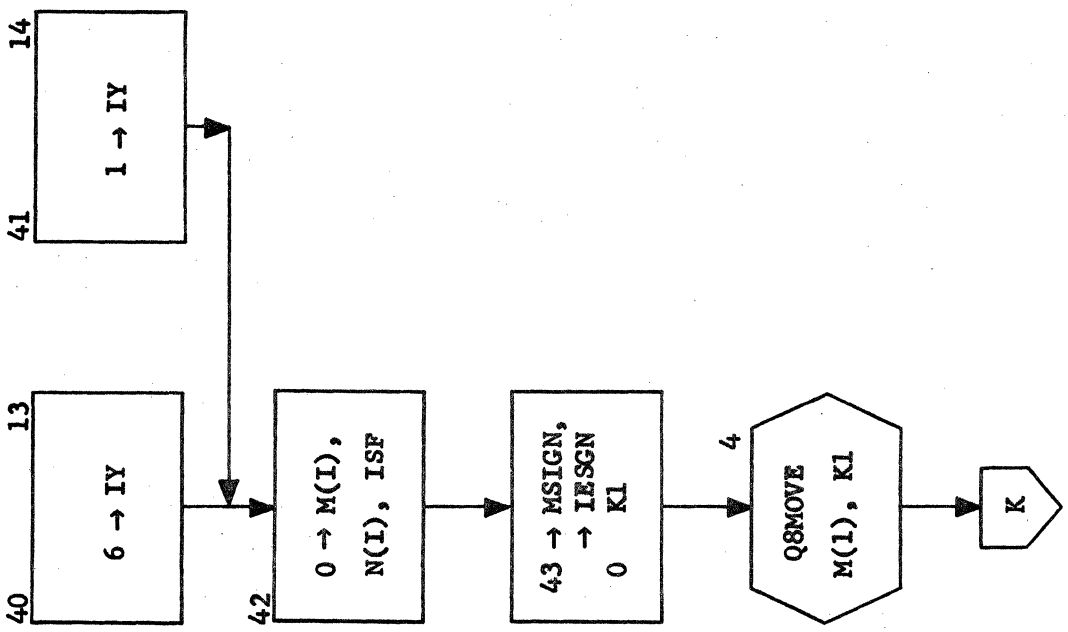
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS 1MC MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	Q8TRAN PAGE 59 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE 3-6-7	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS LM MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8TRAN PAGE 60 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 5-67	TASK NAME			

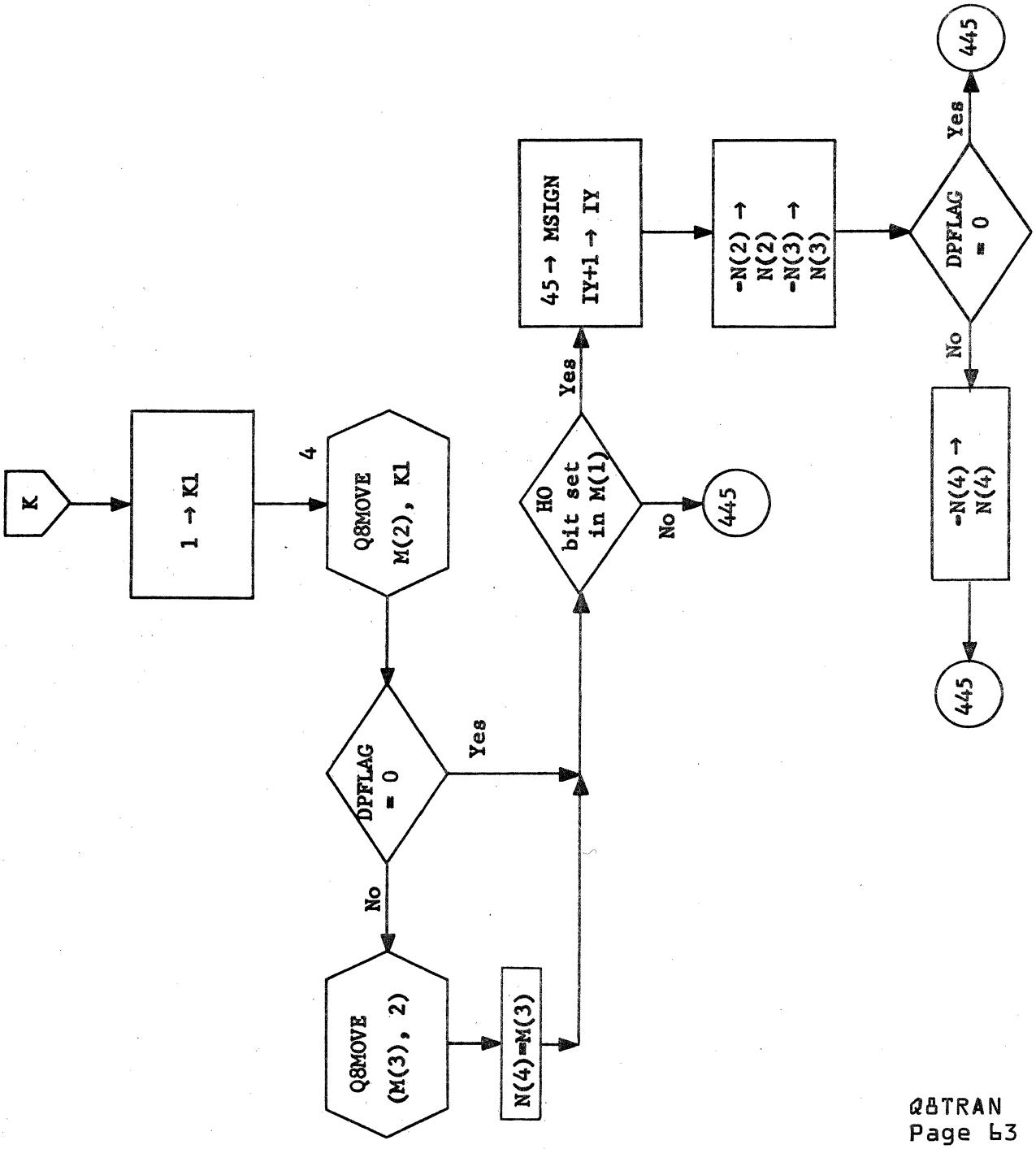


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒	Q8TRAN	PAGE 61 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



13. E and D conversions come here.

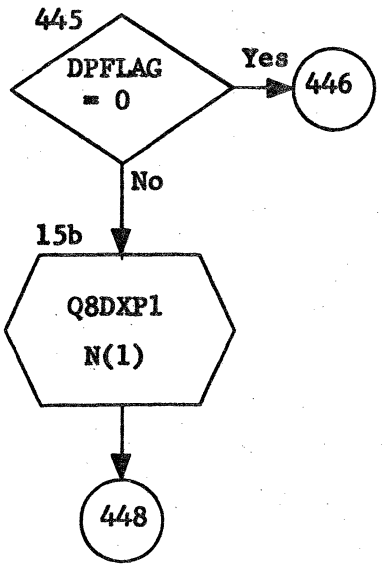
14. F conversion comes here.

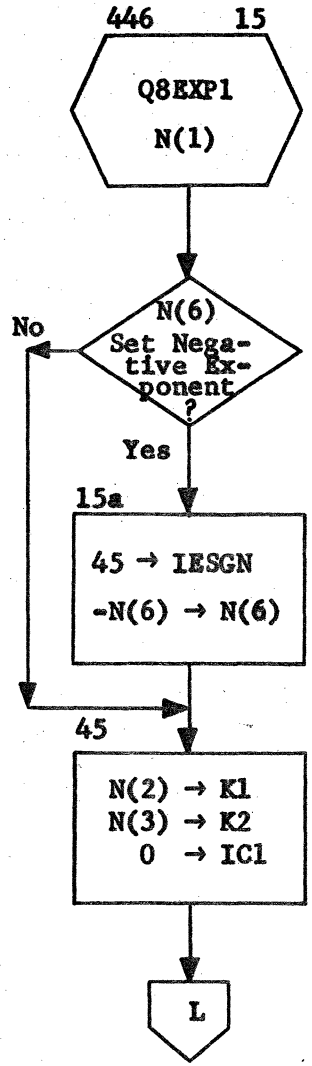

K
1 → K1
4
Q8MOVE M(2), K1
HO bit set in M(1)
Yes
No
445
45 → MSIGN
IY+1 → IY
-N(2) → N(2)
-N(3) → N(3)
DPFLAG = 0
Yes
445
No
-N(4) → N(4)
445
DPFLAG = 0
Yes
No
Q8MOVE (M(3), 2)
N(4)=M(3)

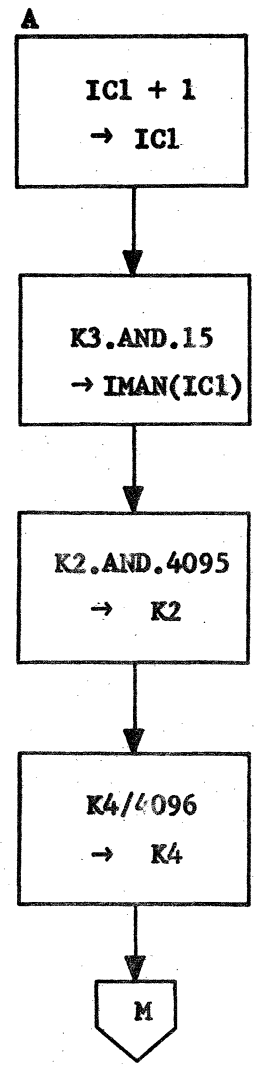
15. This routine takes floating point number in the form required by the floating point package and converts it to an intermediate form of a decimal integer representing the mantissa in N(2), N(3) a decimal exponent in N(6).

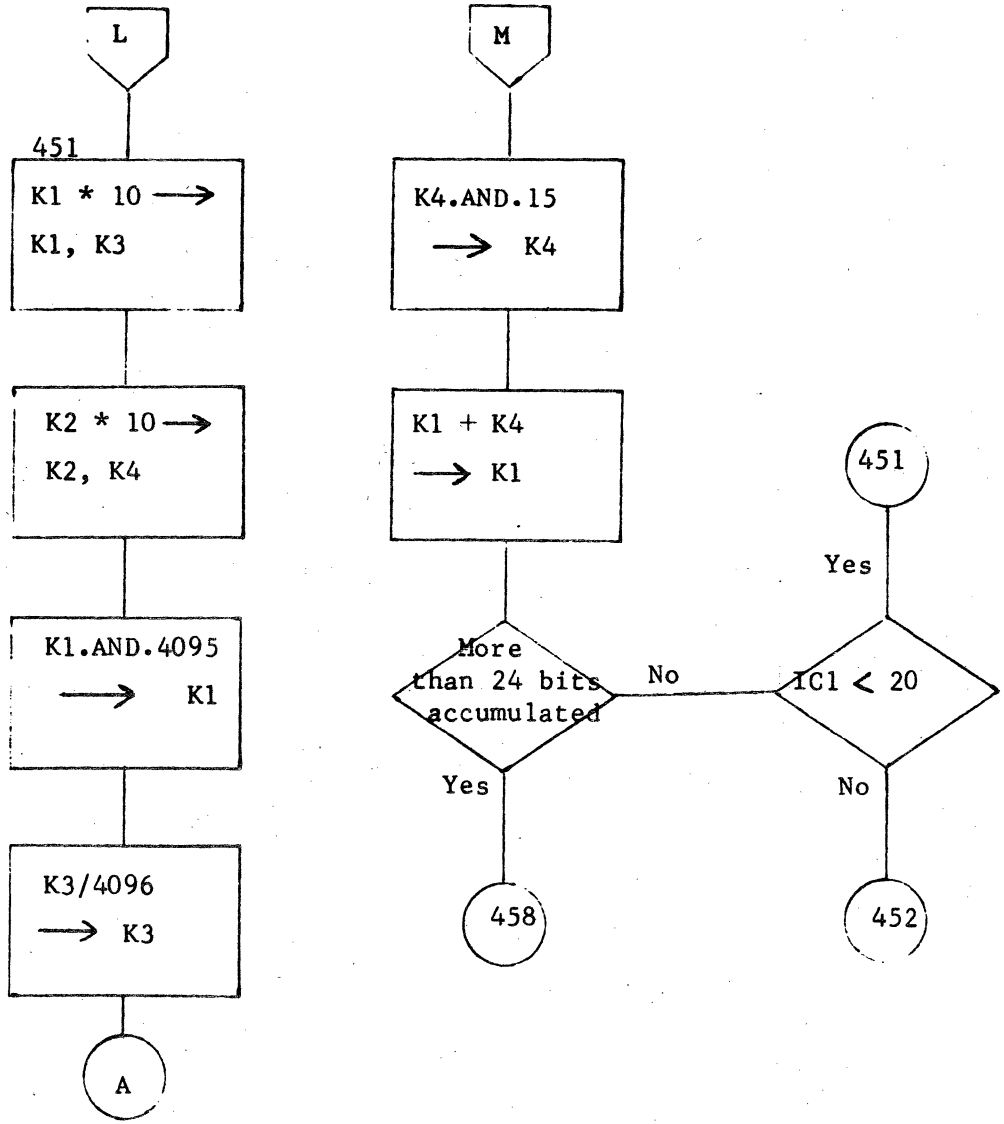
15a. IESGN = 45 means negative exponent
IESGN = 43 means positive exponent

15b. This routine takes a double precision floating point number in the form required by the floating point package and converts it to an intermediate form of a decimal integer representing the mantissa in N(2), N(3), N(4), and N(5) and the decimal exponent in N(6).

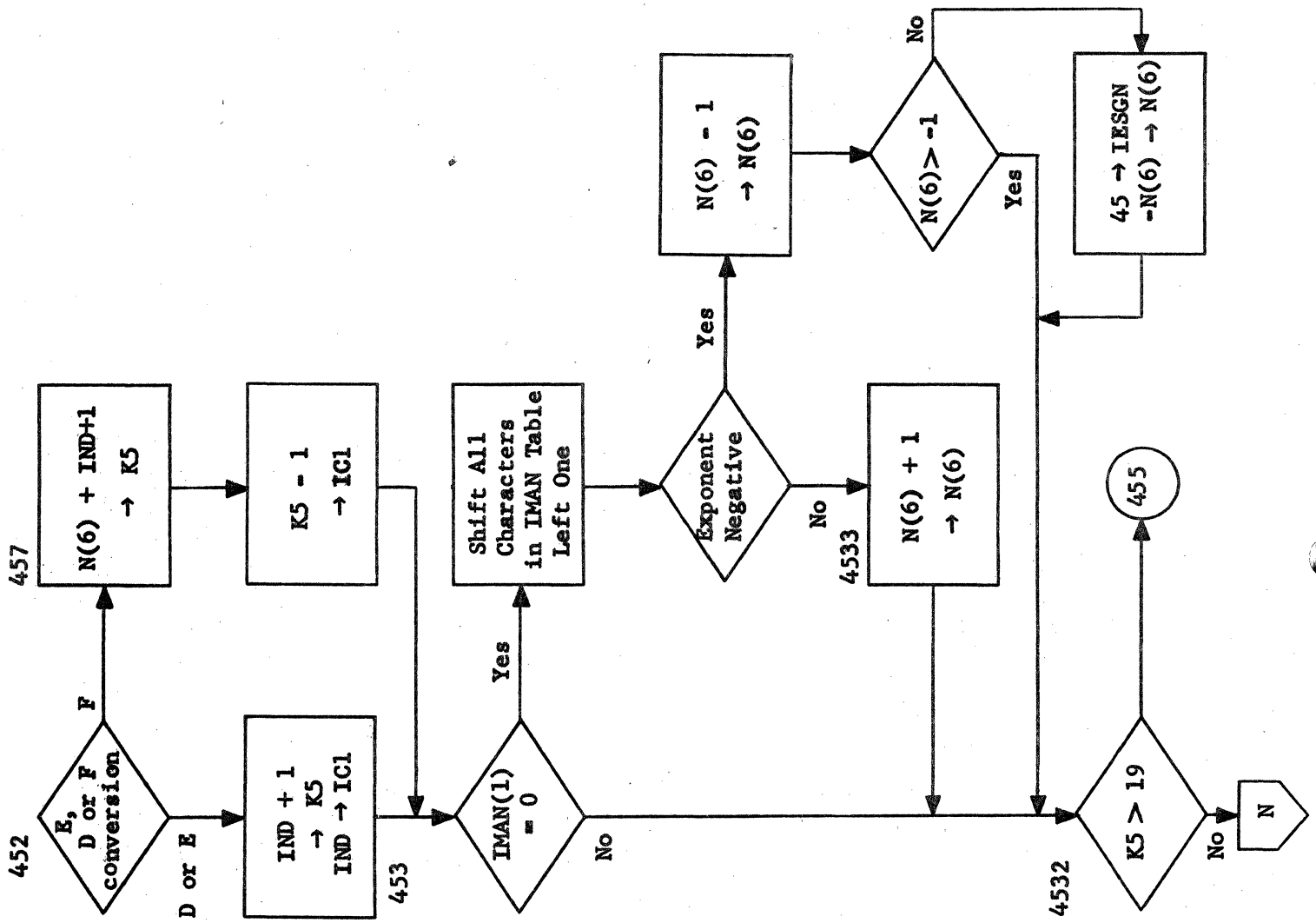


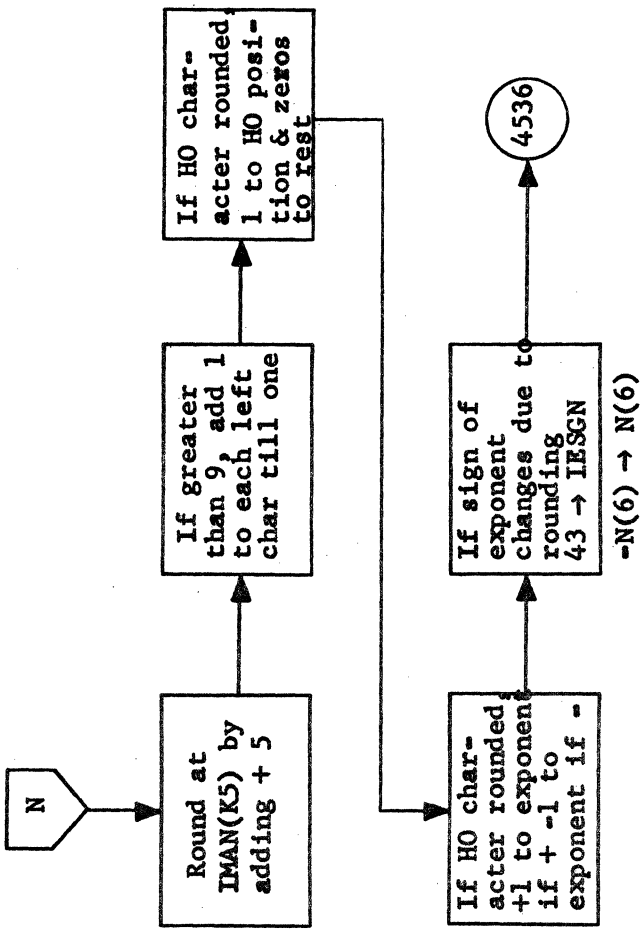


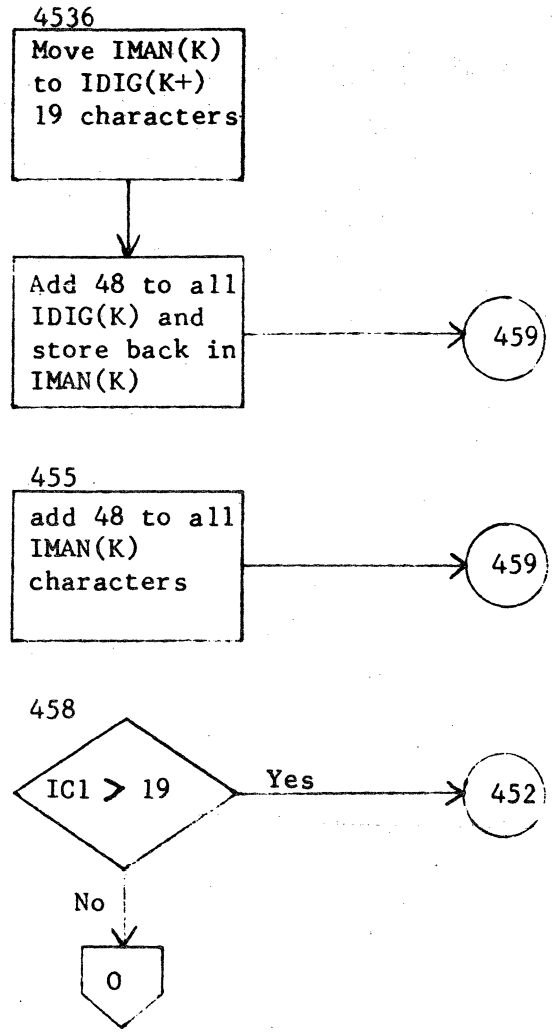




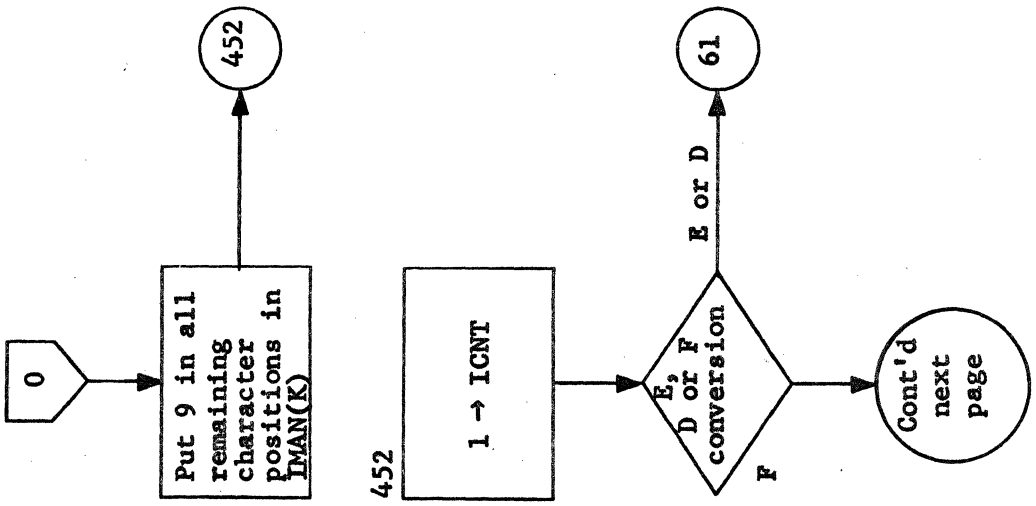
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TM · MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8TRAN · PAGE 65 OF 89	PROJECT NAME			
	NUMBER · ISSUE DATE	TASK NO.			
	DRAWN BY · DATE 7-1-69	TASK NAME			

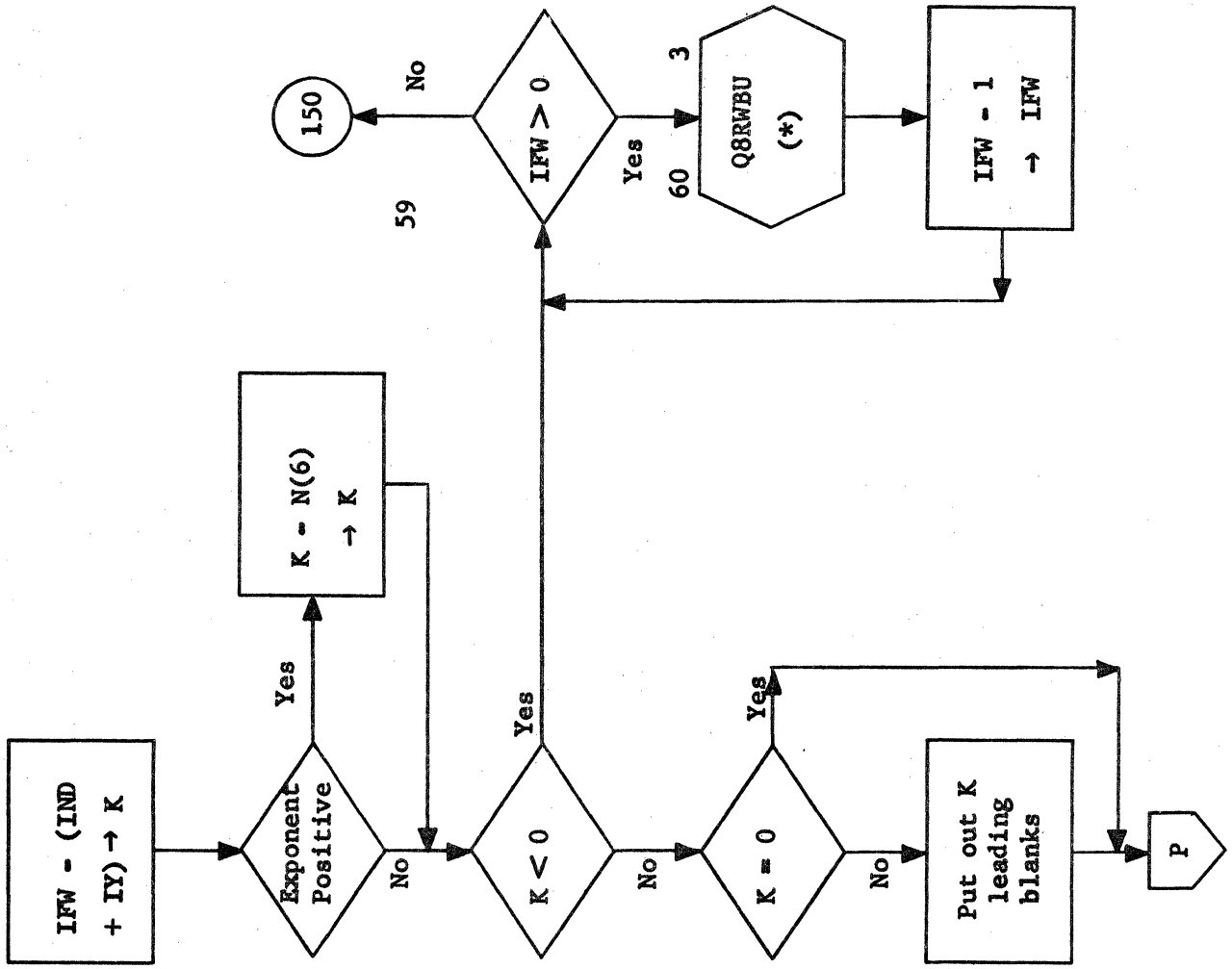

452
E, D or F conversion
F
D or E
457
N(6) + IND+1 → K5
IND + 1 → K5
IND → IC1
K5 - 1 → IC1
453
Shift All Characters in IMAN Table Left One
Yes
IMAN(1) = 0
No
Exponent Negative
Yes
4533
No
N(6) + 1 → N(6)
N(6) - 1 → N(6)
No
N(6)>-1
Yes
45 → IESGN
-N(6) → N(6)
455
4532
K5 > 19
No
N

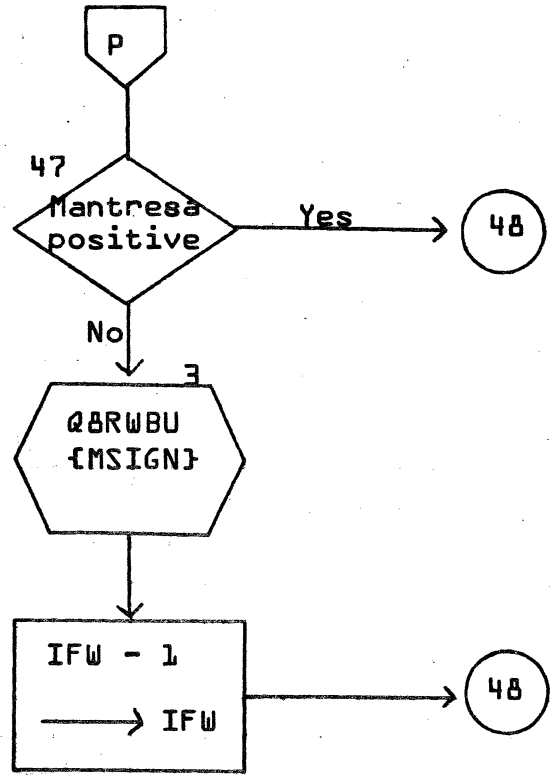

N
Round at IMAN(K5) by adding + 5
If greater than 9, add 1 to each left char till one
If HO character rounded, 1 to HO position & zeros to rest
If HO character rounded +1 to exponent if + -1 to exponent if -
If sign of exponent changes due to rounding 43 → IESGN
-N(6) → N(6)
4536



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☑	Q8TRAN PAGE 68 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE [illegible]	TASK NAME			

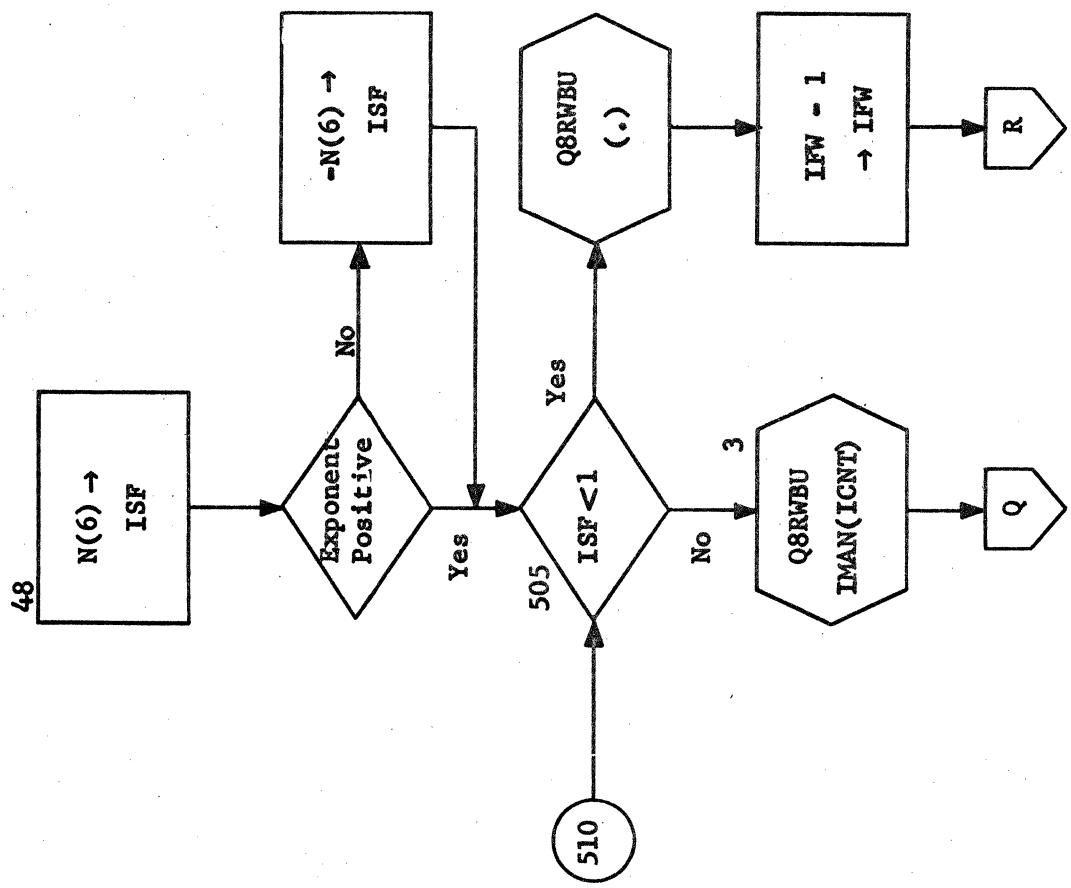

452
61
E or D
0
Put 9 in all remaining character positions in IMAN(K)
452
1 → ICNT
E, D or F conversion
F
Cont'd next page

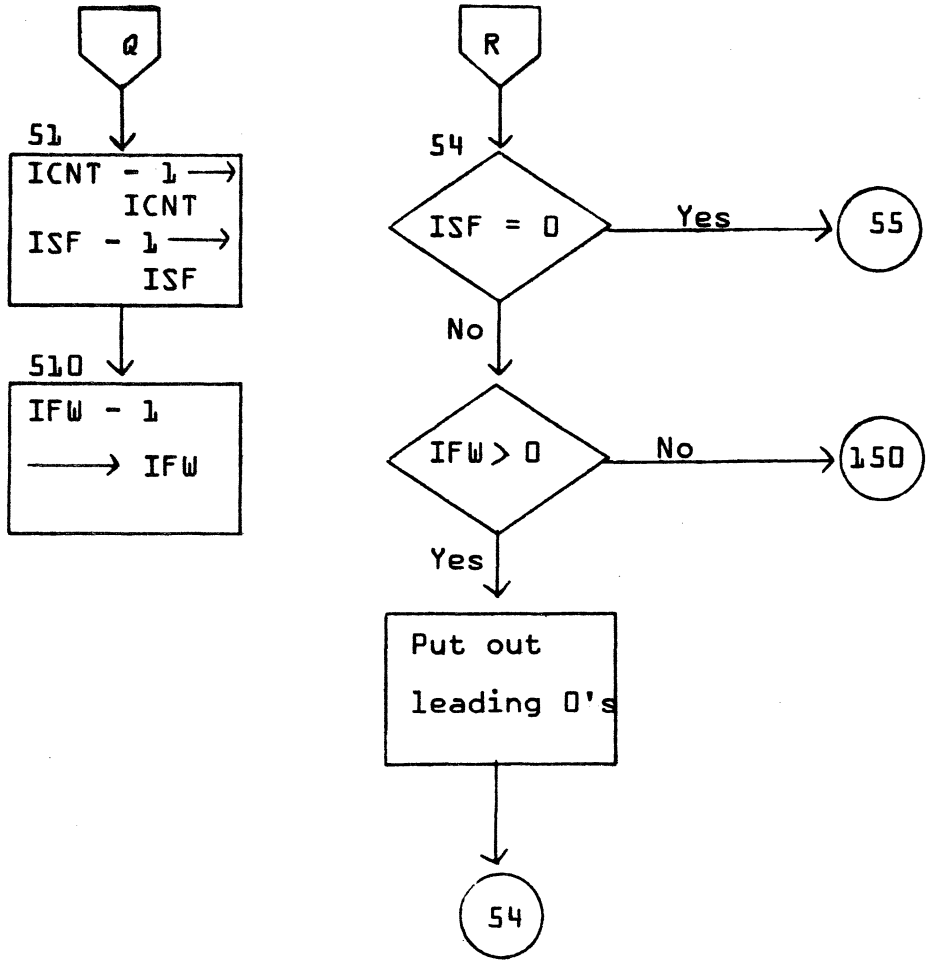

IFW = (IND + IY) → K
Exponent Positive
Yes
No
K = N(6) → K
K < 0
Yes
No
K = 0
Yes
No
Put out K leading blanks
P
IFW > 0
No
150
59
Yes
3
60
Q8RWBU (*)
IFW - 1 → IFW



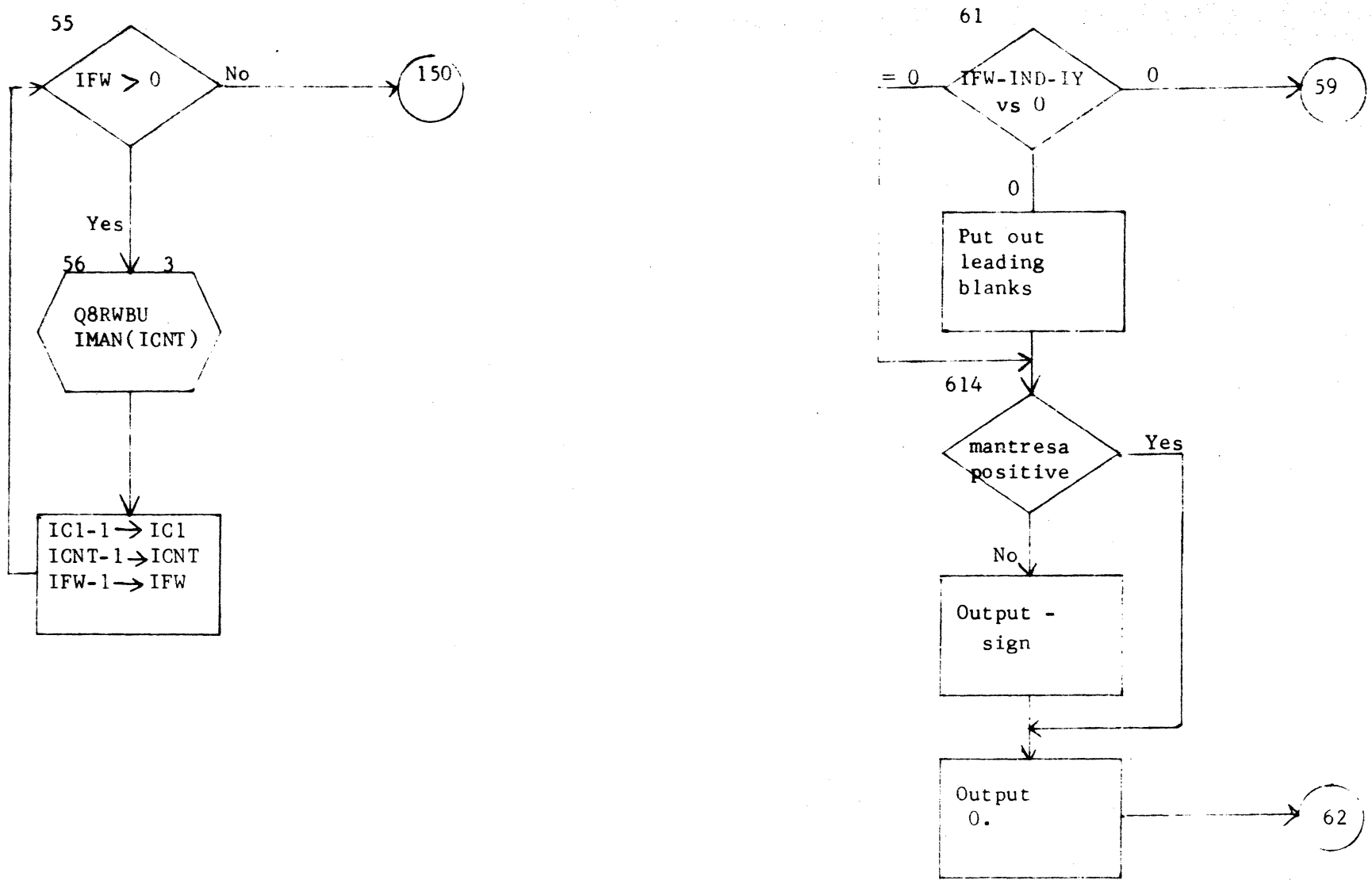
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	Q8TRAN	PAGE 71 OF 89	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

60-103


48
N(6) → ISF
Exponent Positive
No
Yes
-N(6) → ISF
505
ISF<1
Yes
No
Q8RWBU (.)
IFW - 1 → IFW
R
3
Q8RWBU IMAN(ICNT)
Q
510

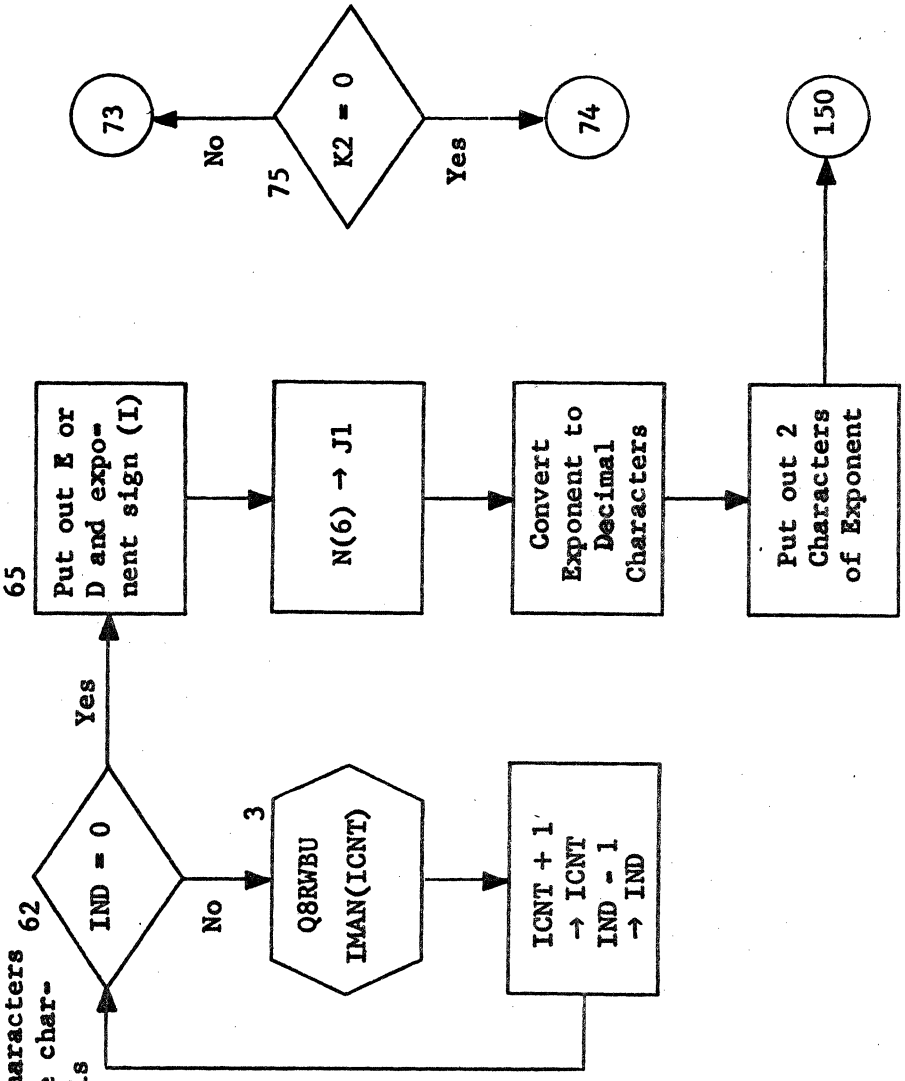


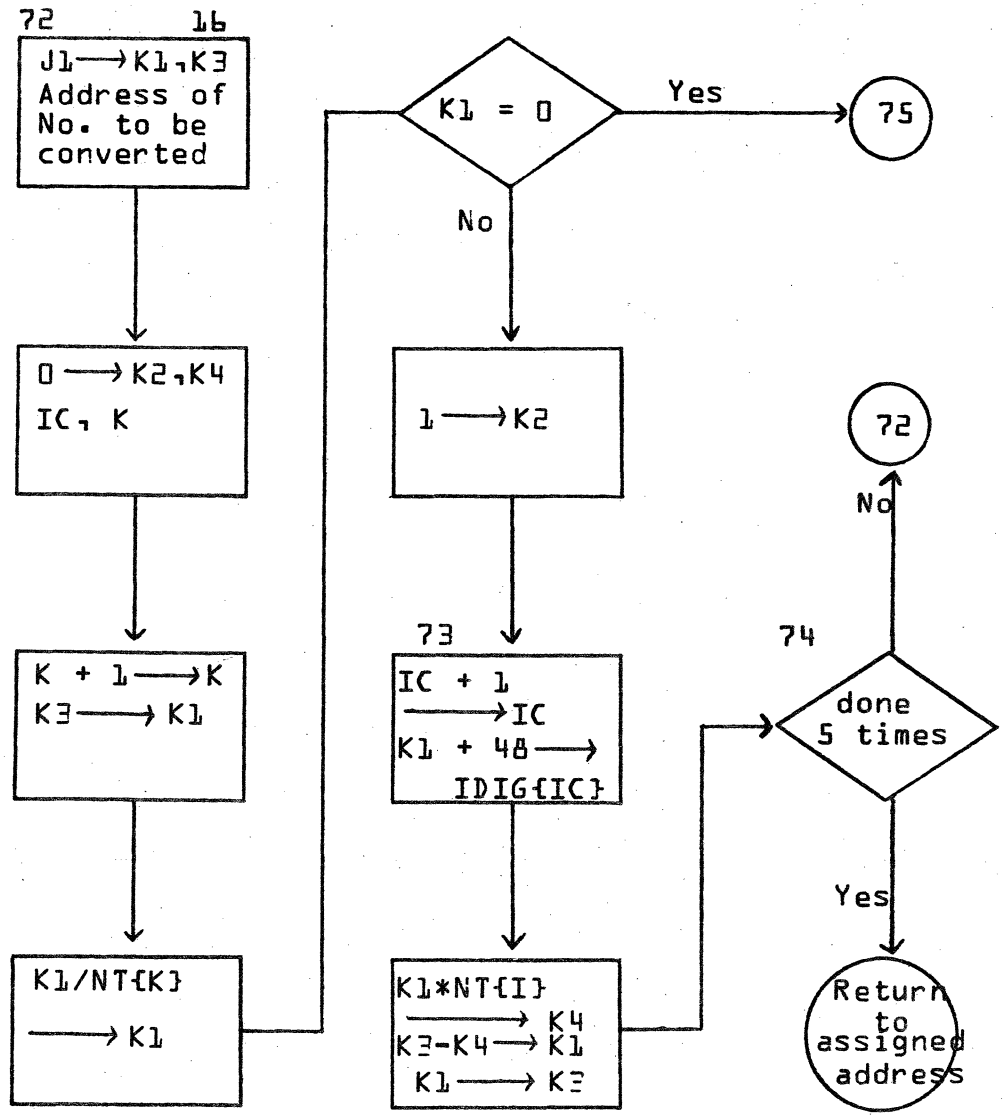
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	QBTRAN	PAGE 73 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS // MACH TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8TRAN PAGE 74 OF 89	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

16. This section of code converts a binary integer placed in J1 into the equivalent characters in IDIG(I). IC is the character count. Return is made to IAD.



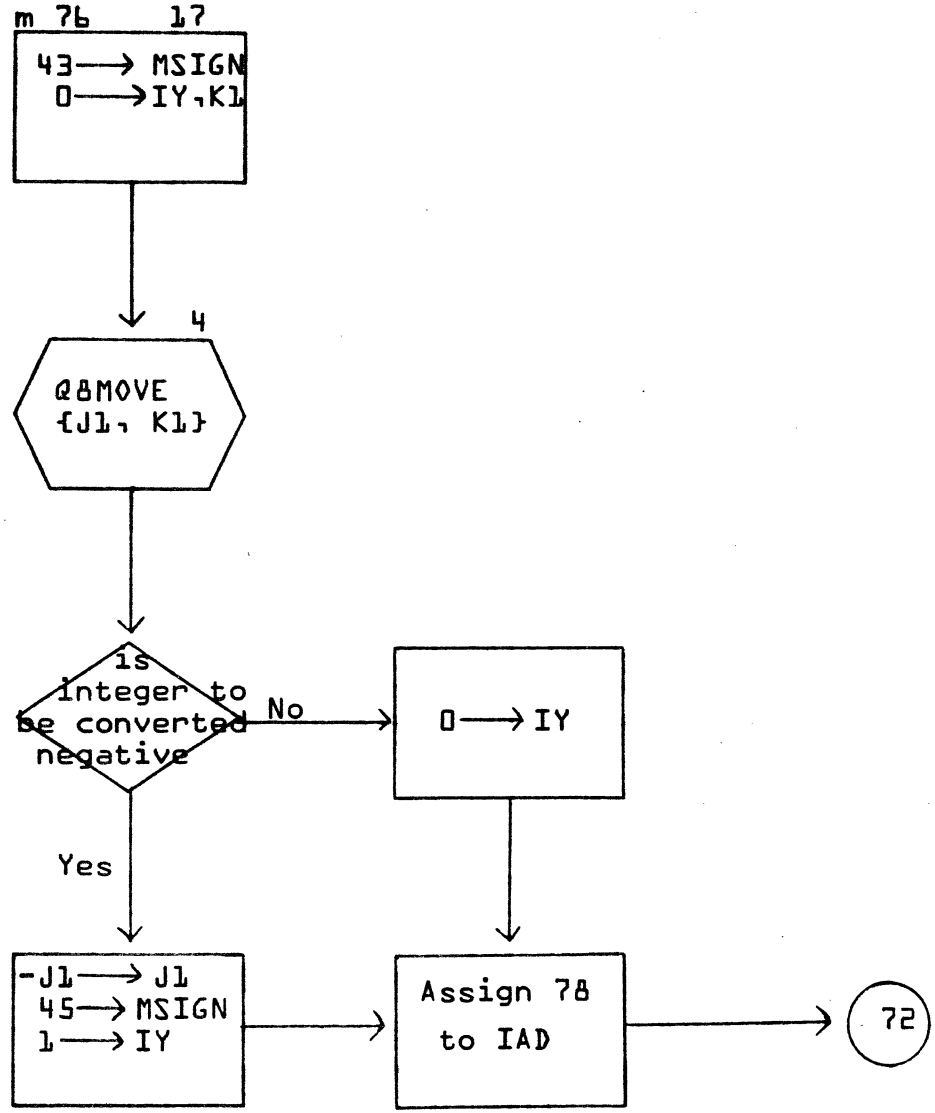


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	QBTRAN	PAGE 76 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

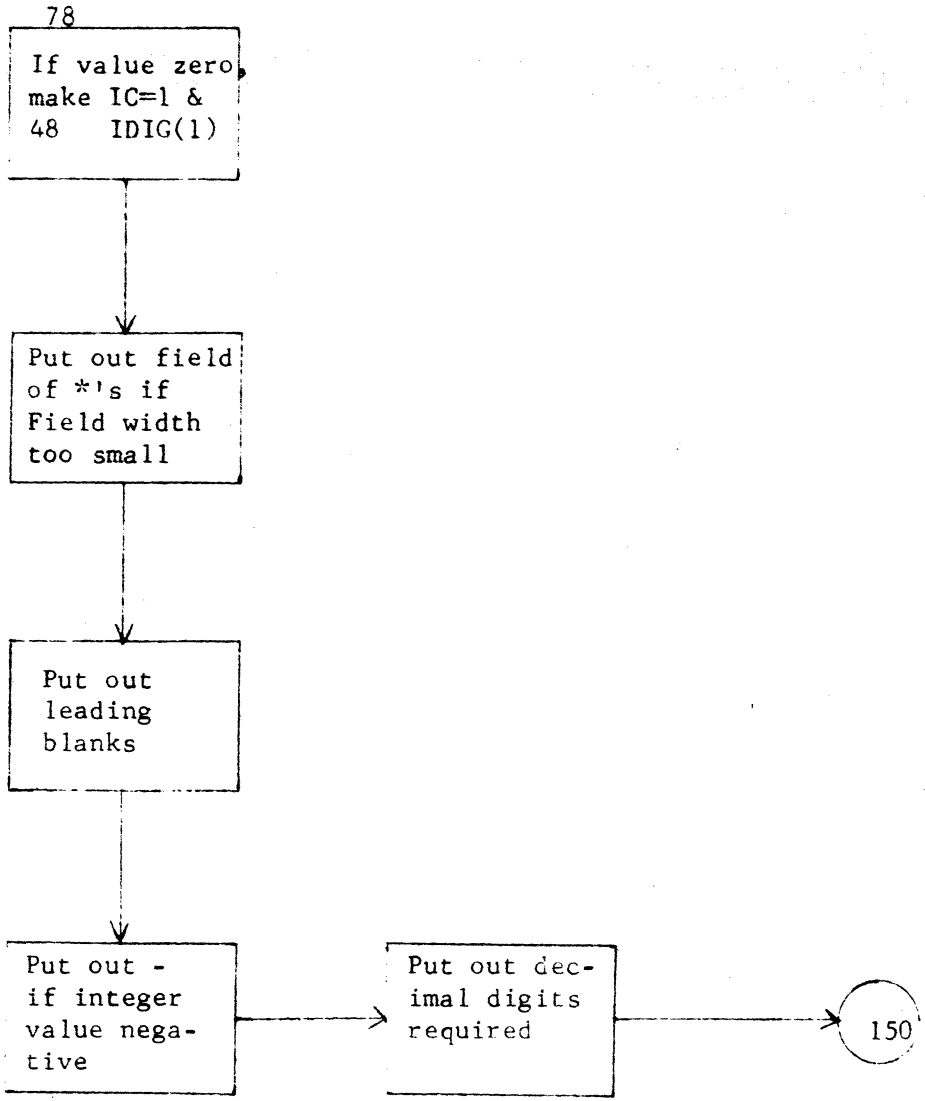
6-10-8

AA1385 (FORMERLY CA127-1)

17. Integer output conversion comes here.

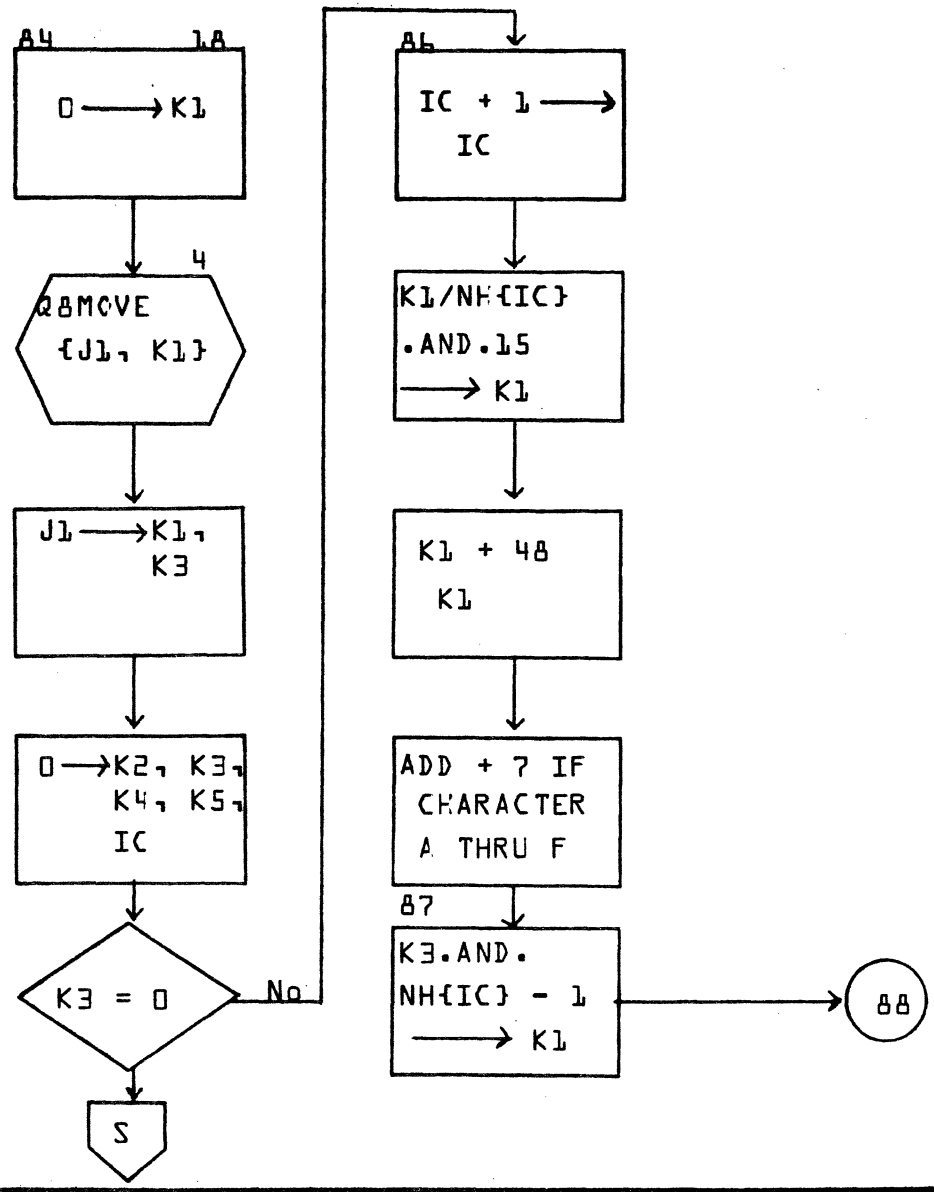


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8TRAN	PAGE 77 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

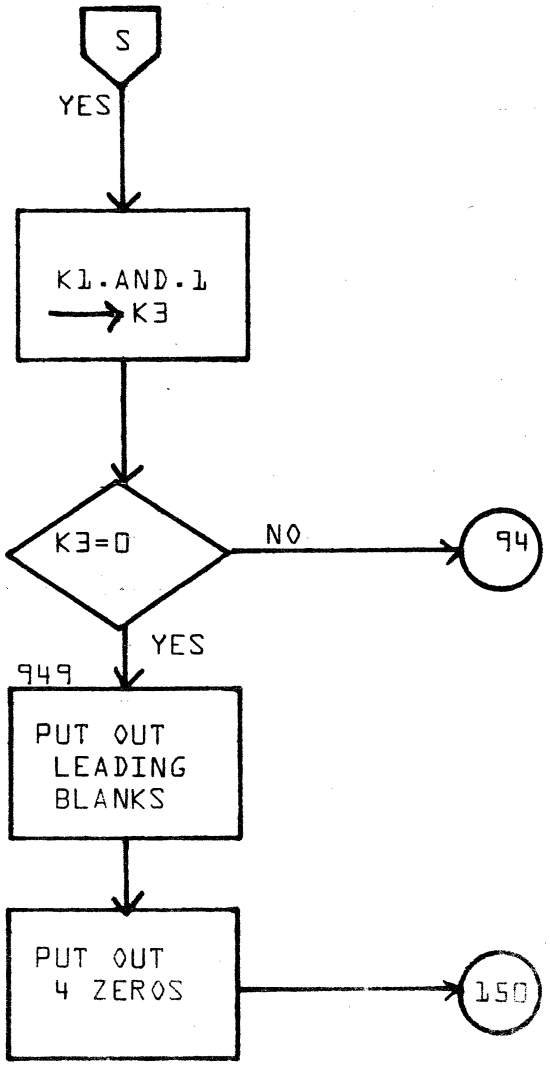


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☒	Q8TRAN	PAGE 78 OF 89	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

18. Hexadecimal output conversion comes here.



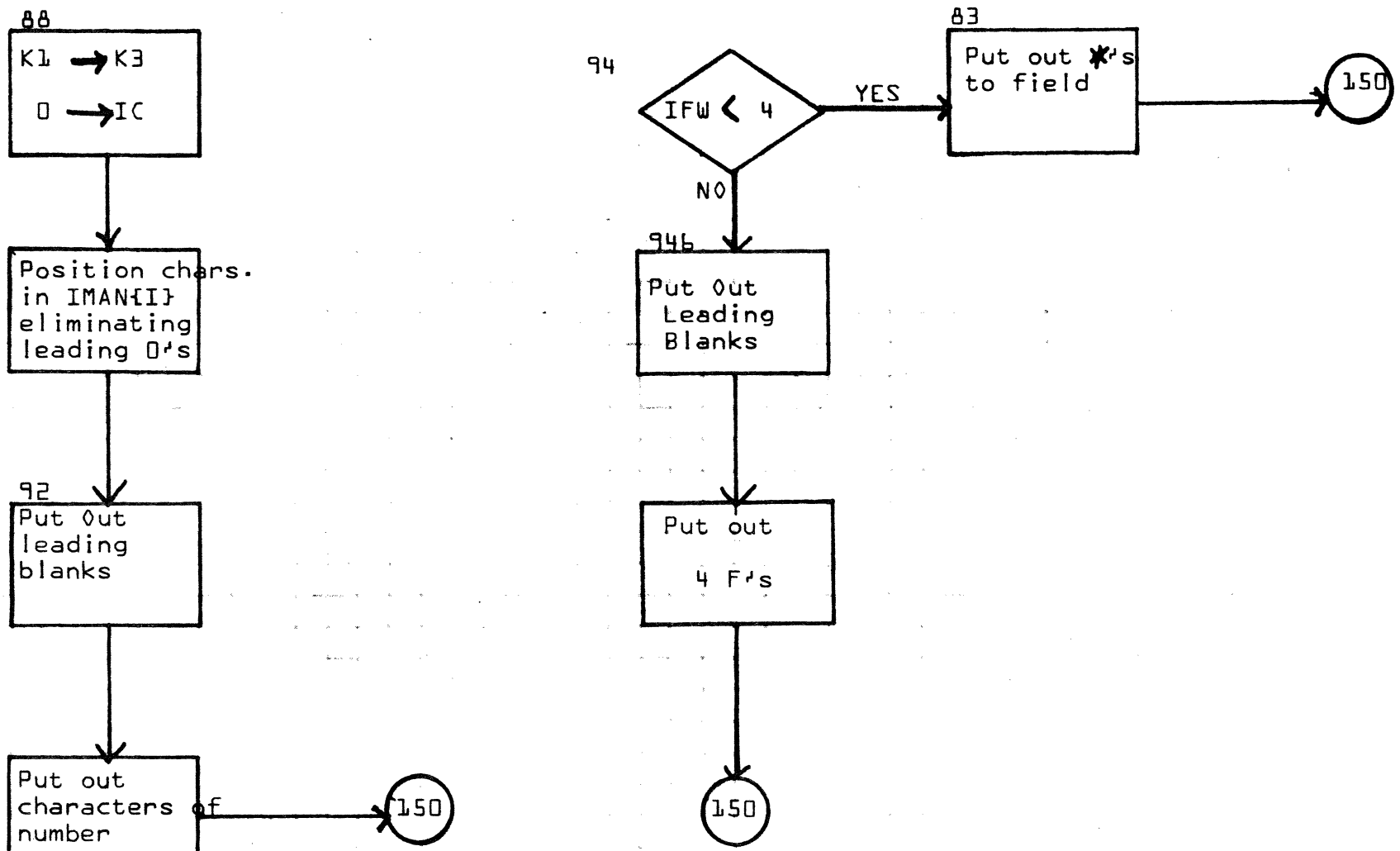
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8TRAN	PAGE 79 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒	8TRAN	PAGE 80 OF 89	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

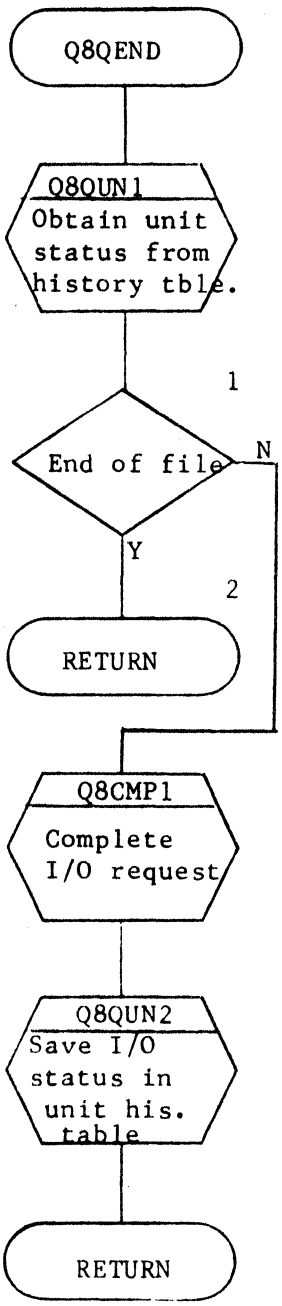
AA1388 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	QSTRAN	PAGE 81 OF 89	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

1. Bit 12 in unit history table entry set means an end of file was detected during this I/O request.
2. Q8QEND entry when an end of file has been read is to be ignored.



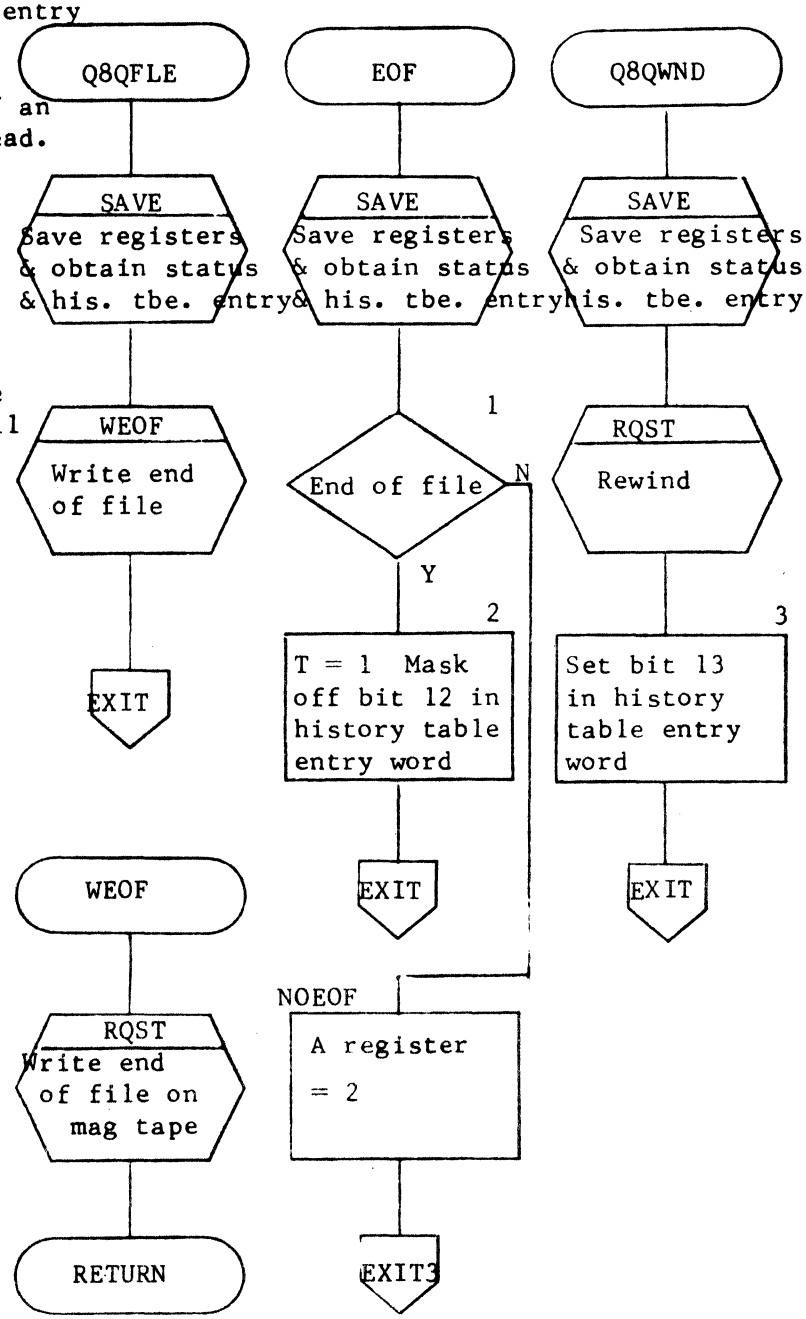
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER Q8QEND
MACH TYPE
PAGE 82 OF 89
ISSUE DATE
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR
PROJECT NAME
TASK NO.
TASK NAME

REV | APPROVED | DATE

1. The history table entry word for the indicated unit will have bit 12 set if an end of file was read.
2. The A register contains T when EOF returns to the calling program.
3. The history table entry word for the indicated unit will have bit 13 set after a rewind is implemented.



REV	APPROVED	DATE

PROJECT NO. | PROJECT MGR | PROJECT NAME | TASK NO. | TASK NAME

DOCUMENT CLASS | MACH. TYPE | PAGE 83 OF 89

DOCUMENT TITLE | ISSUE DATE

NUMBER | DRAWN BY | DATE

TAPCON

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE [ ]
FLOWCHART [X]
DECISION TABLE [ ]
OTHER [ ]

4. Bit 10 in the magnetic tape hardware status word (STAFLG) set means the tape is at load point.

5. The previous read/write request type is contained in bit 14 of the history table entry for the indicated unit. Bit 14 set means format request (1 physical record).

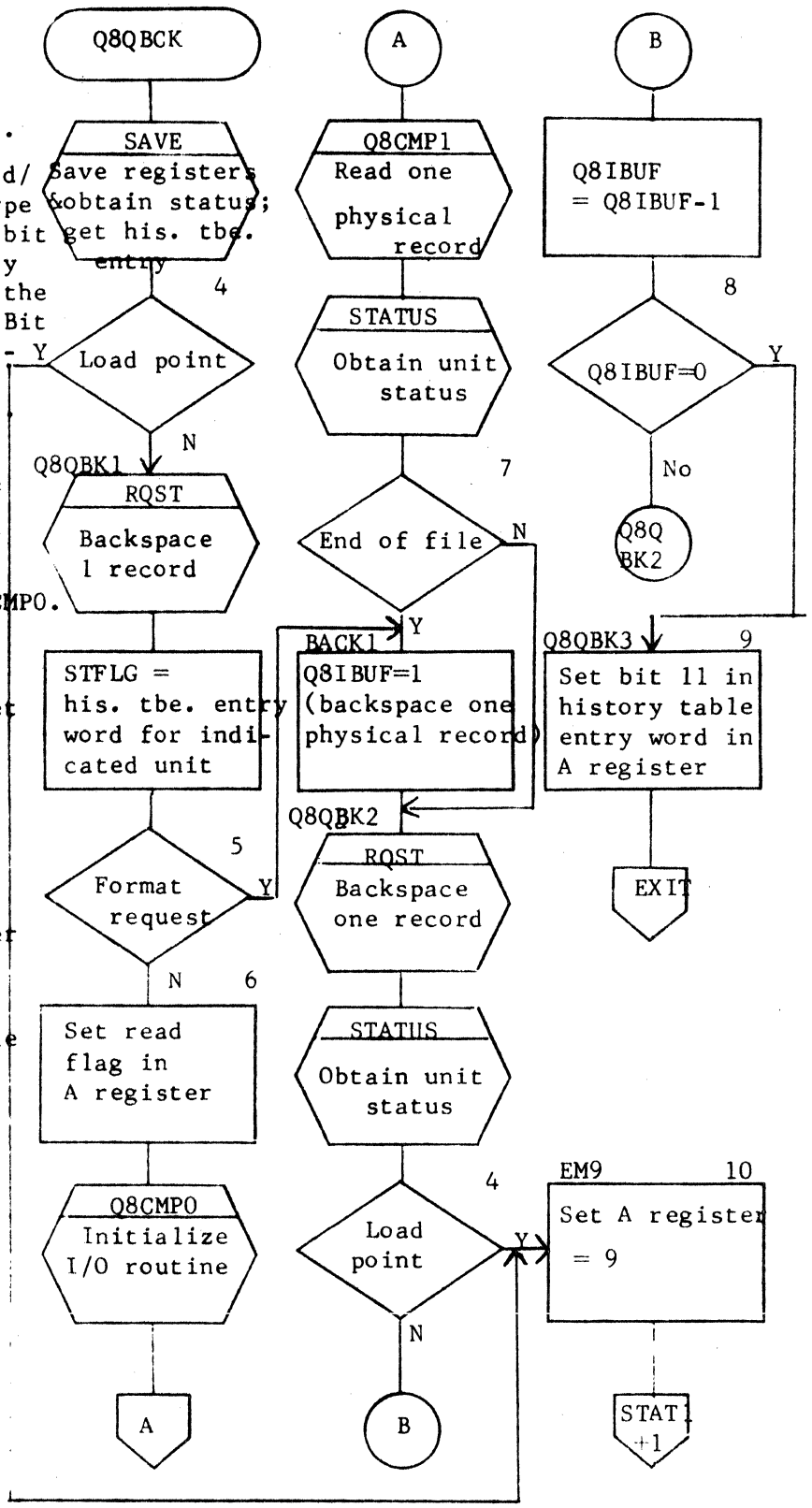
6. The A register now contains the history table entry with bit 10 set to indicate read to Q8CMP0.

7. Bit 11 of the hardware status word (STAFLG) set means an end of file was read.

8. For binary records Q8IBUF contains the number of physical records per logical records.

9. Bit 11 set in the history table entry indicates backspace.

10. Backspace at load point results in error message 9 and program termination.

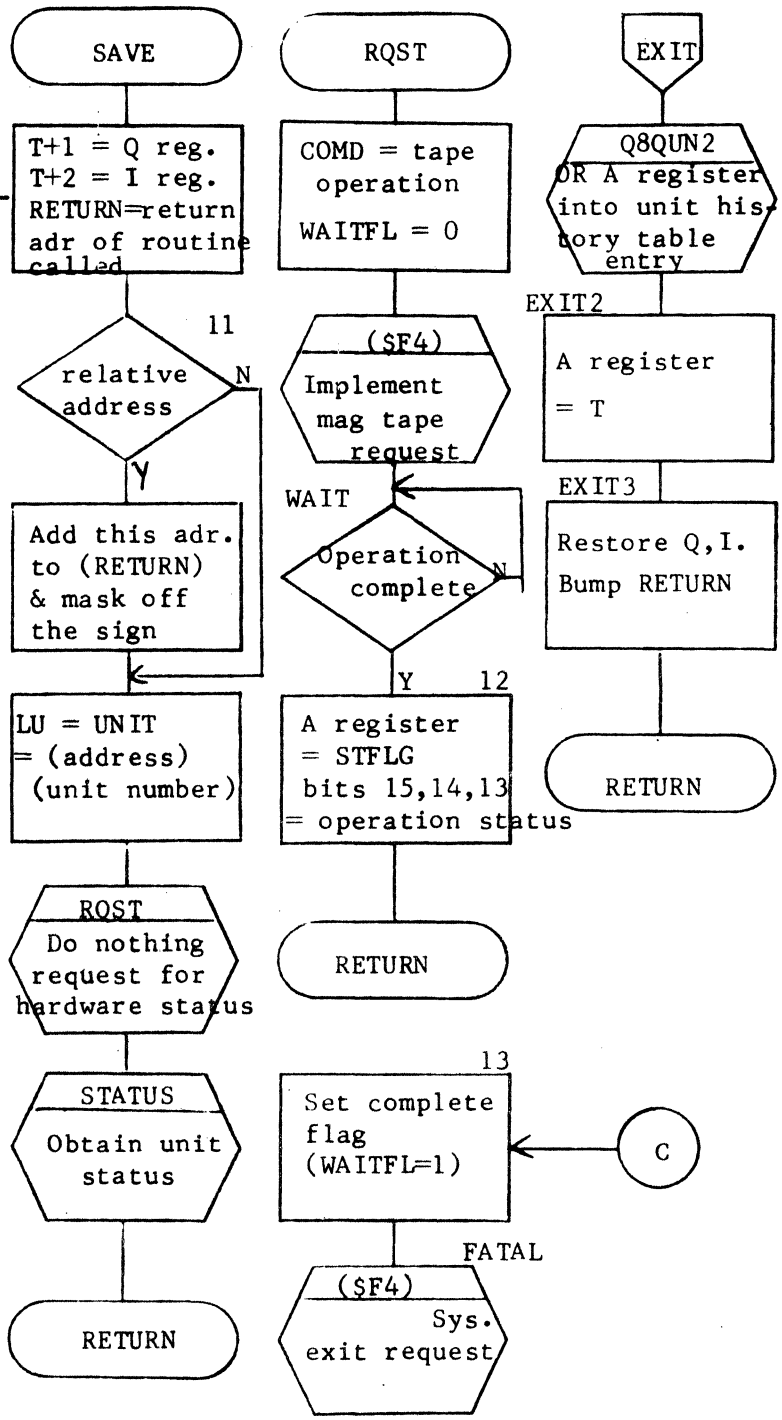


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS
DOCUMENT TITLE
MACH TYPE
PAGE 89 OF
TAPCON
NUMBER
ISSUE DATE
DRAWN BY
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
REV
APPROVED
DATE

AA1385 (FORMERLY CA127-1)

11. A negative address means this address is relative to the given address.
12. Bits 15, 14, 13 contain the following: Bit 15 set if error occurred; bit 14 set if too few words transferred; bit 13 set if device failed.
13. Entered from completion routine after magnetic tape operation implemented.

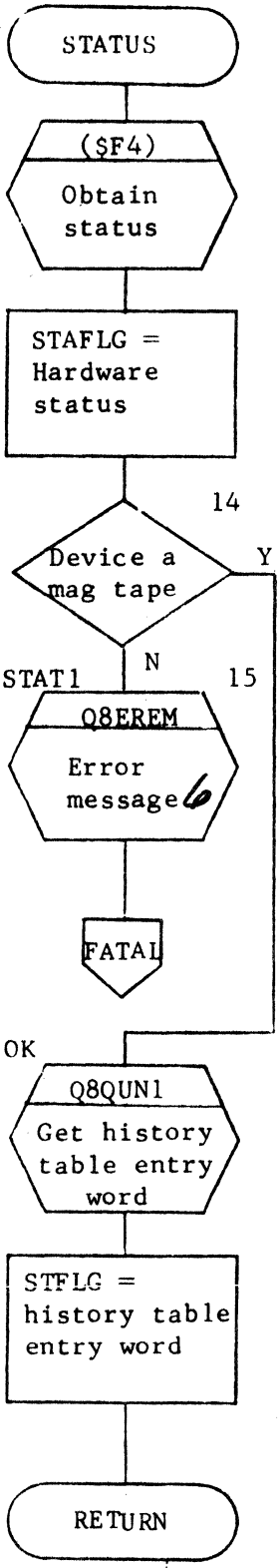


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE [ ]
FLOWCHART [X]
DECISION TABLE [ ]
OTHER [ ]

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER TAPCON
MACH. TYPE
ISSUE DATE
DRAWN BY
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
REV | APPROVED | DATE
PAGE 85F 89
DATE

14. System Status request results in the Q register containing the device code in bits 4-7. Mag tape device code is 9.

15. Device not a mag tape results in error message 6 and program termination.

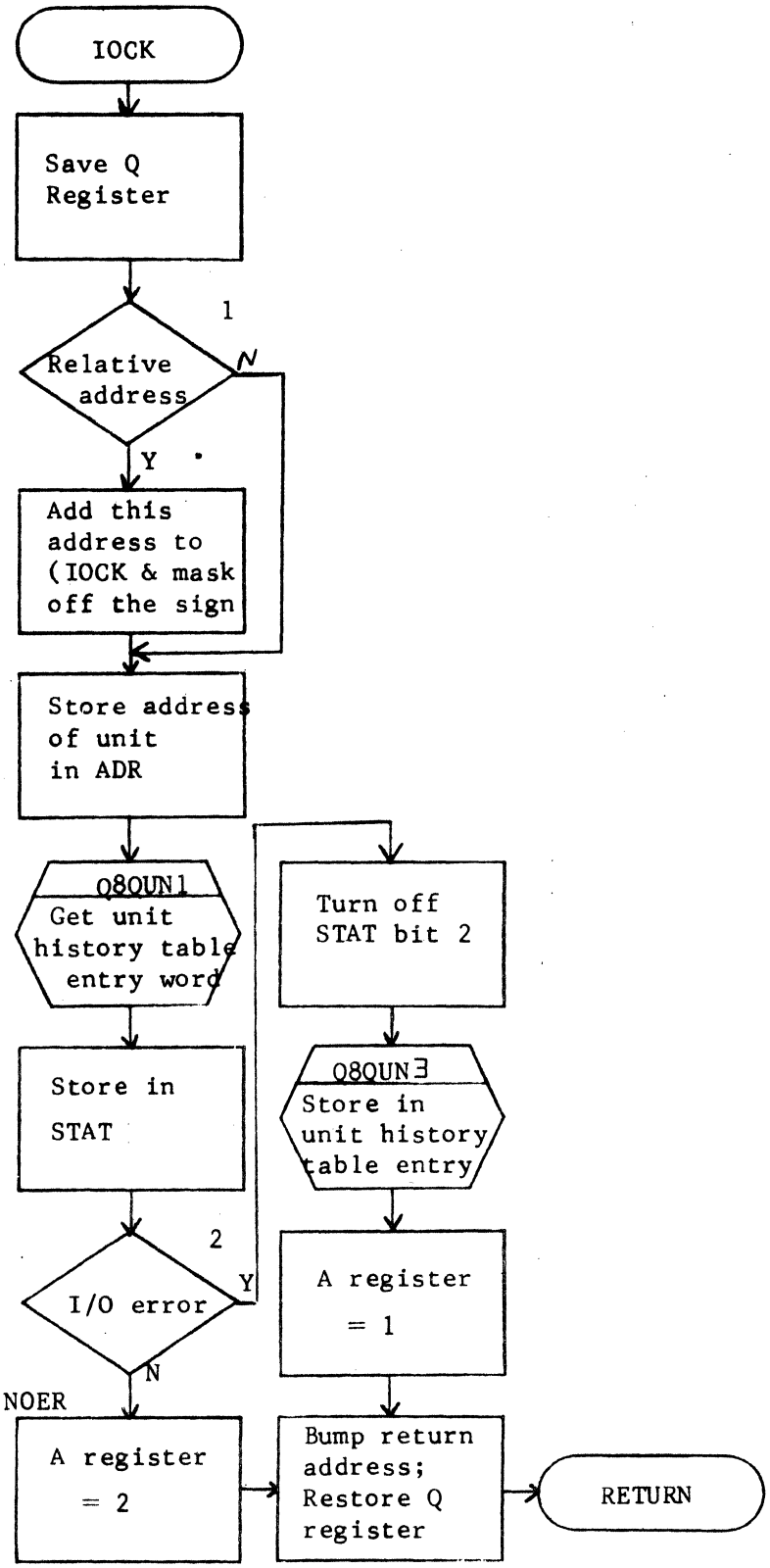


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☑ DECISION TABLE ☐ OTHER ☐

REV	APPROVED	DATE

DOCUMENT CLASS: 711
DOCUMENT TITLE: TAPCON
NUMBER:
DRAWN BY:
MACH TYPE:
PROJECT NO.:
PROJECT MGR.:
PROJECT NAME:
TASK NO.:
TASK NAME:
PAGE 86 OF 89
ISSUE DATE:
DATE:

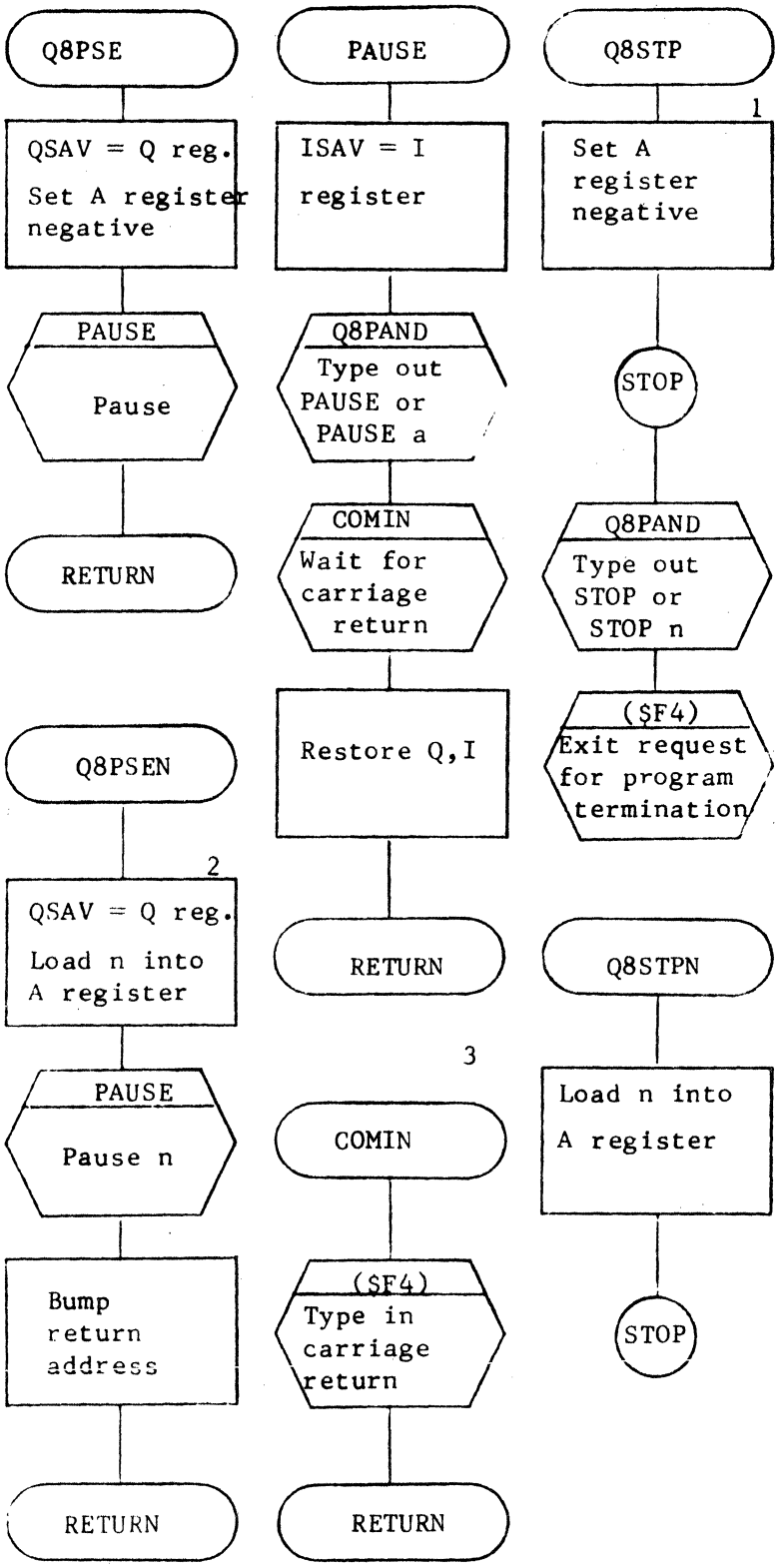
1. A negative address means this address is relative to the given address.
2. Bit 2 in the unit history table entry word set indicates an I/O error occurred.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS: ZMS
MACH TYPE: 1700
PAGE 87 OF 89
DOCUMENT TITLE: IOCK
NUMBER
ISSUE DATE
DRAWN BY
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
REV
APPROVED
DATE

1. A register negative flags no digit conversion for Q8PAND.
2. A register contains n as a hexadecimal number.
3. COMIN is equivalenced to Q8COMI. They are the same entry point.



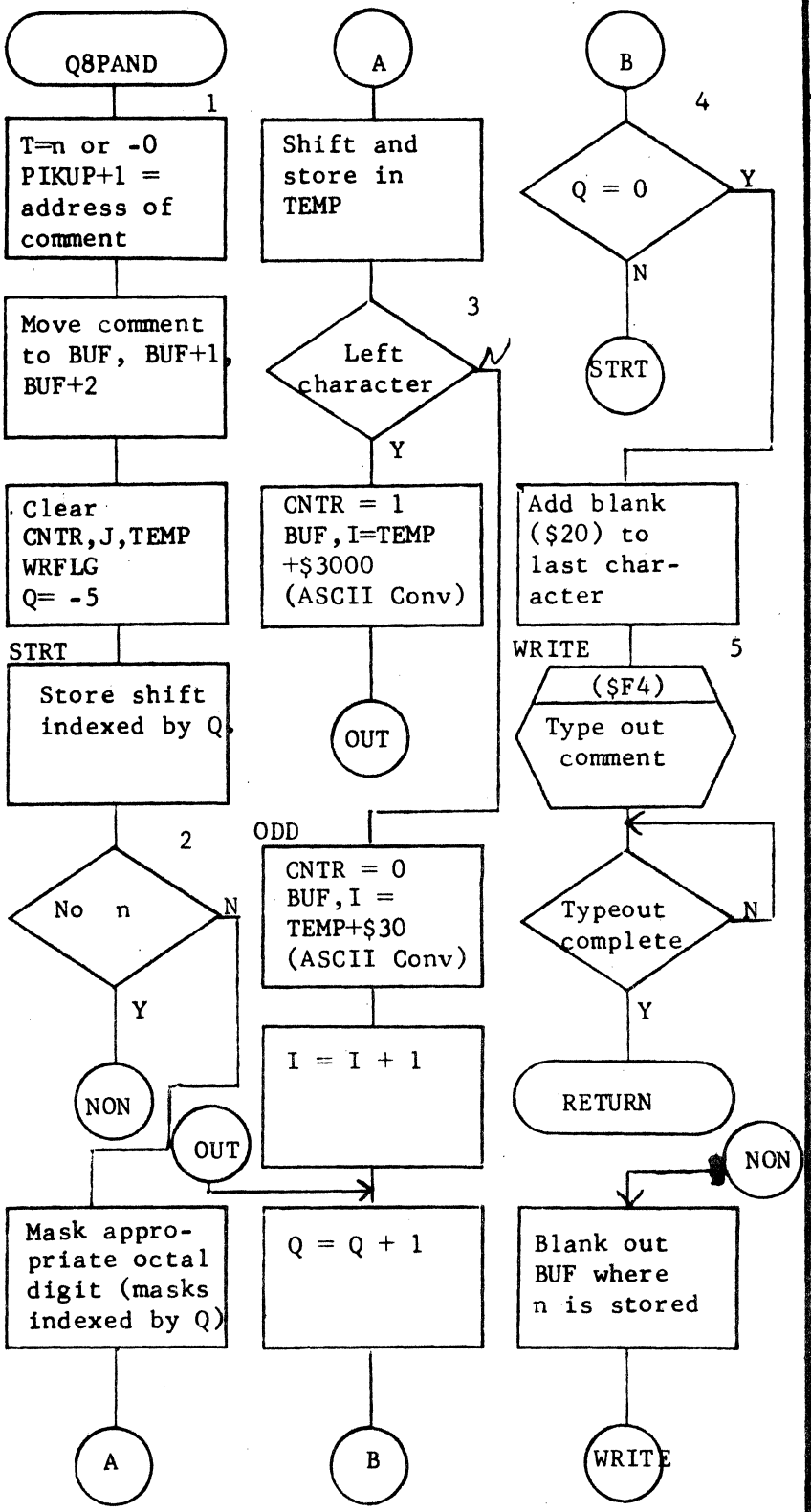
REV	APPROVED	DATE

PROJECT NO. | PROJECT MGR | PROJECT NAME | TASK NO. | TASK NAME

DOCUMENT CLASS | MACH. TYPE | DOCUMENT TITLE: PSSTOP | PAGE 8 OF 89 | ISSUE DATE | NUMBER | DATE | DRAWN BY

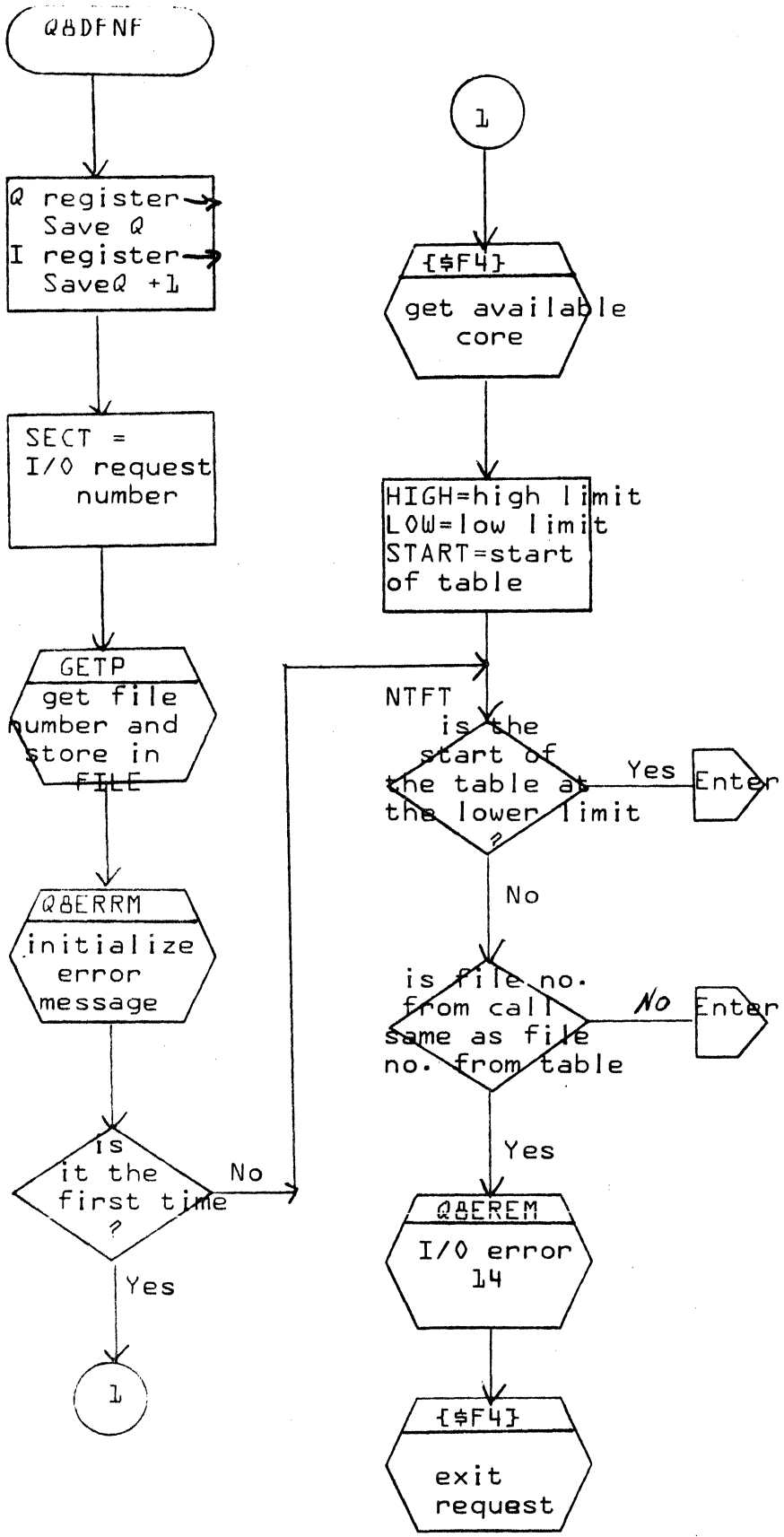
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

1. Either PAUSE or STOP is typed out.
2. T 0 means no n is required.
3. CNTR is a flag indicating left or right half of ASCII equivalent of the octal 5 digit number. CNTR=0 for left half.
4. Conversion to ASCII octal complete?
5. PAUSE, PAUSE n, STOP, STOP n.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS
DOCUMENT TITLE
NUMBER Q8PAND
DRAWN BY
MACH. TYPE
PAGE 89 OF 89
ISSUE DATE
DATE
PROJECT NO.
PROJECT MGR
PROJECT NAME
TASK NO.
TASK NAME
REV
APPROVED
DATE



REV	APPROVED	DATE

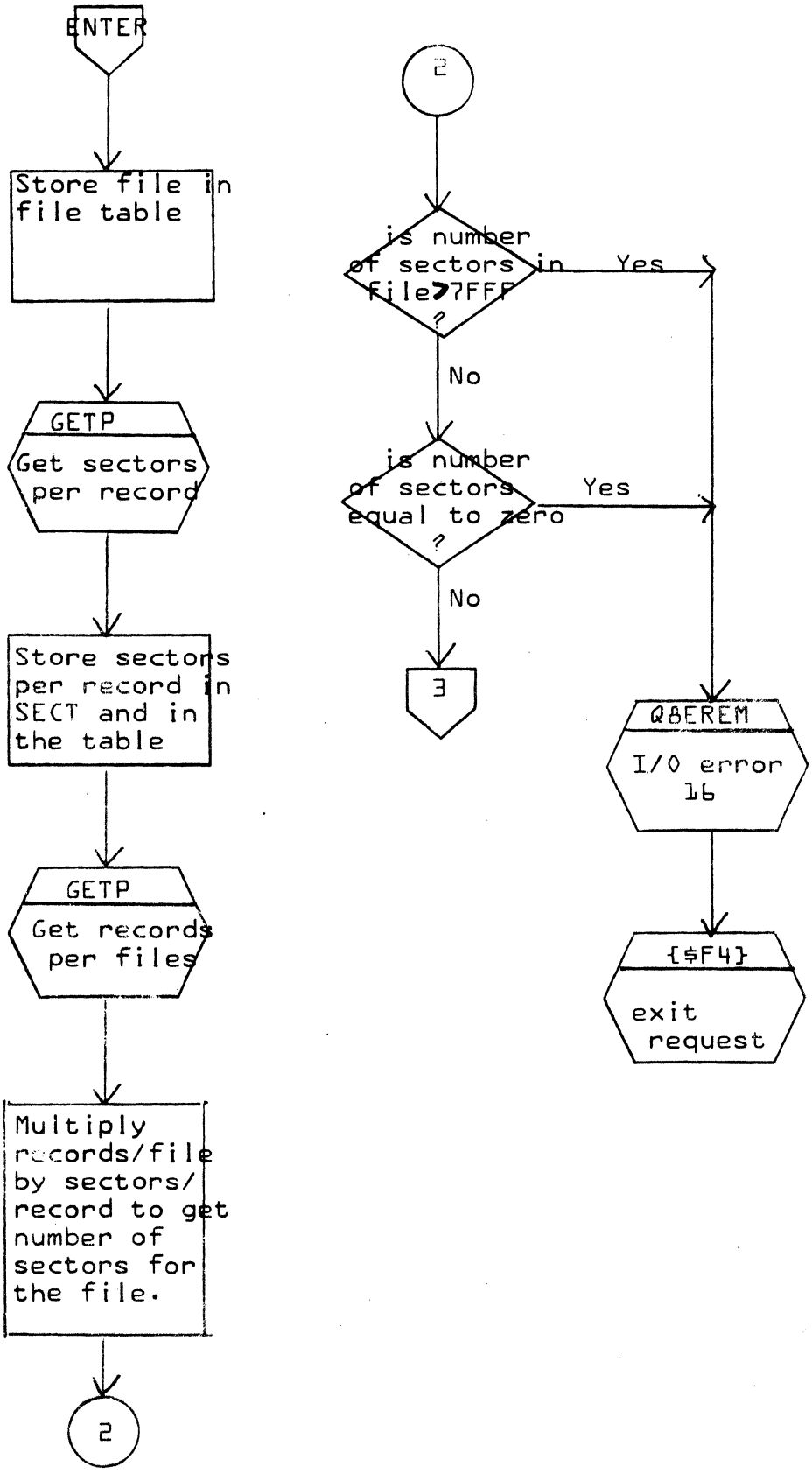
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	QBDFIO
PAGE	1 OF 8
NUMBER	
ISSUE DATE	
DRAWN BY	
DATE	
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

PRINTED IN U.S.A.

CA1385 (FORMERLY CA127-1)

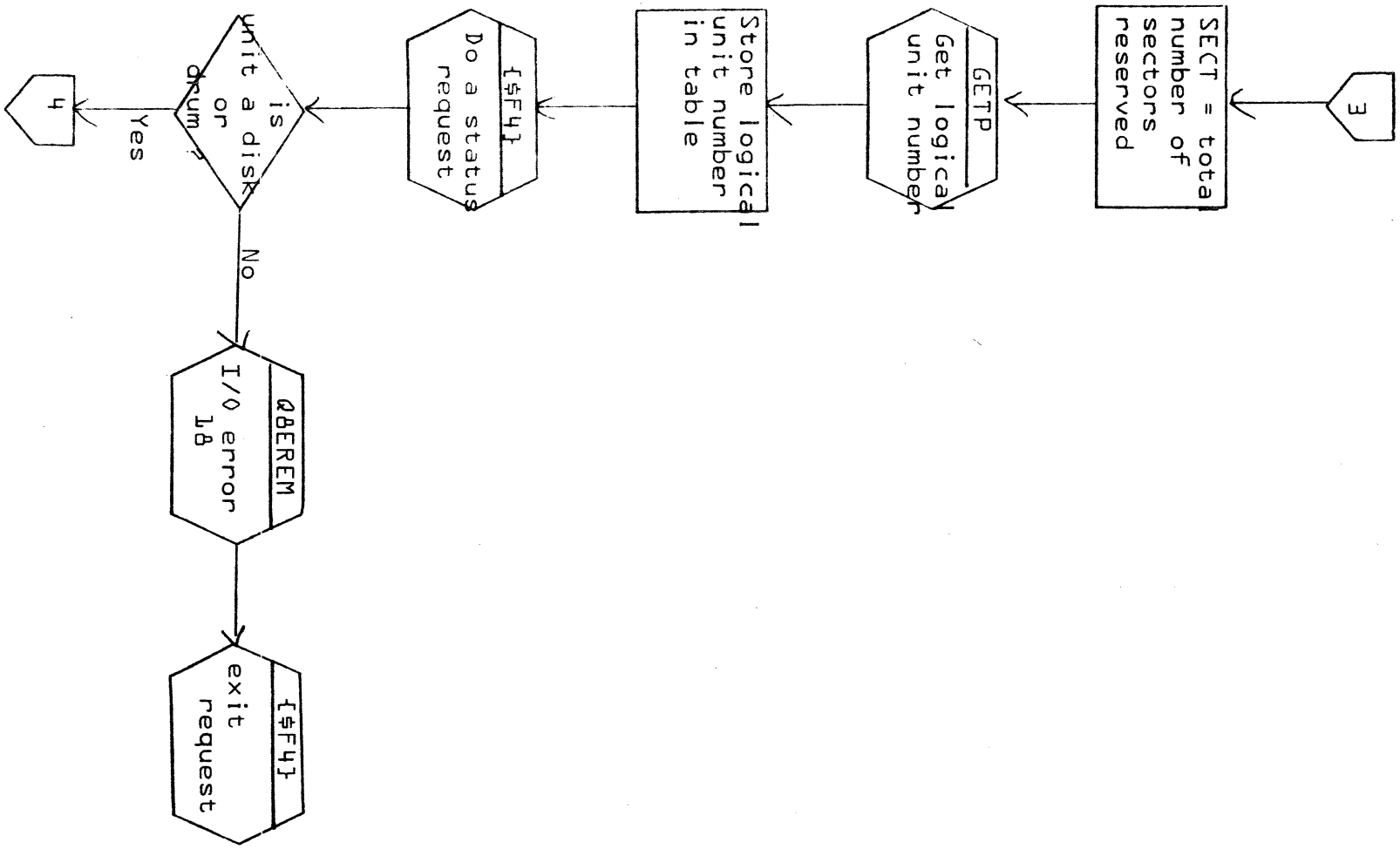
A B C D



REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

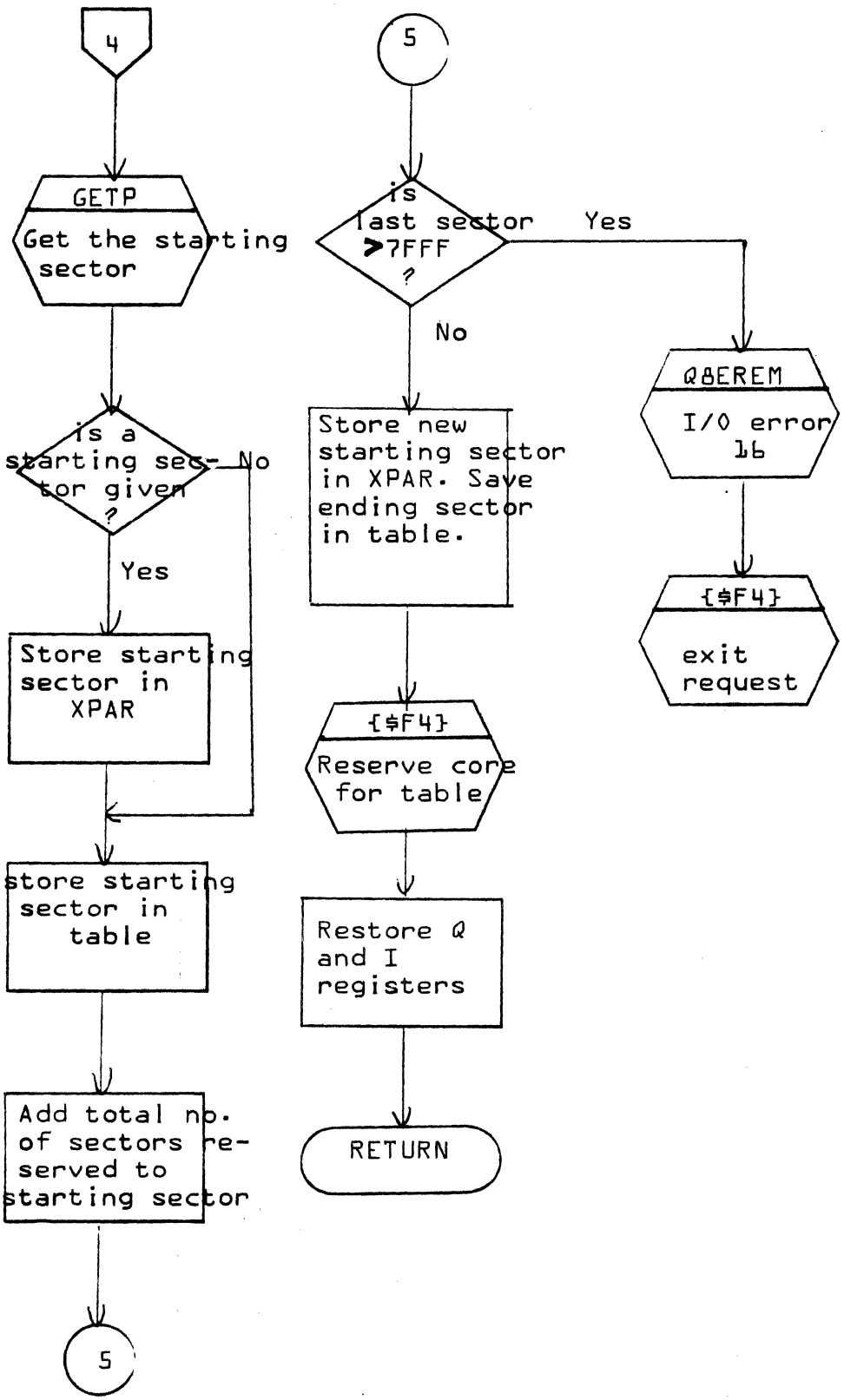
DOCUMENT CLASS: IMS	MACH TYPE: 1700	PROJECT NO.	
DOCUMENT TITLE: Q8DFIO	PAGE 2 OF 8	PROJECT MGR.	
NUMBER	ISSUE DATE	PROJECT NAME	
DRAWN BY	DATE	TASK NO.	
		TASK NAME	



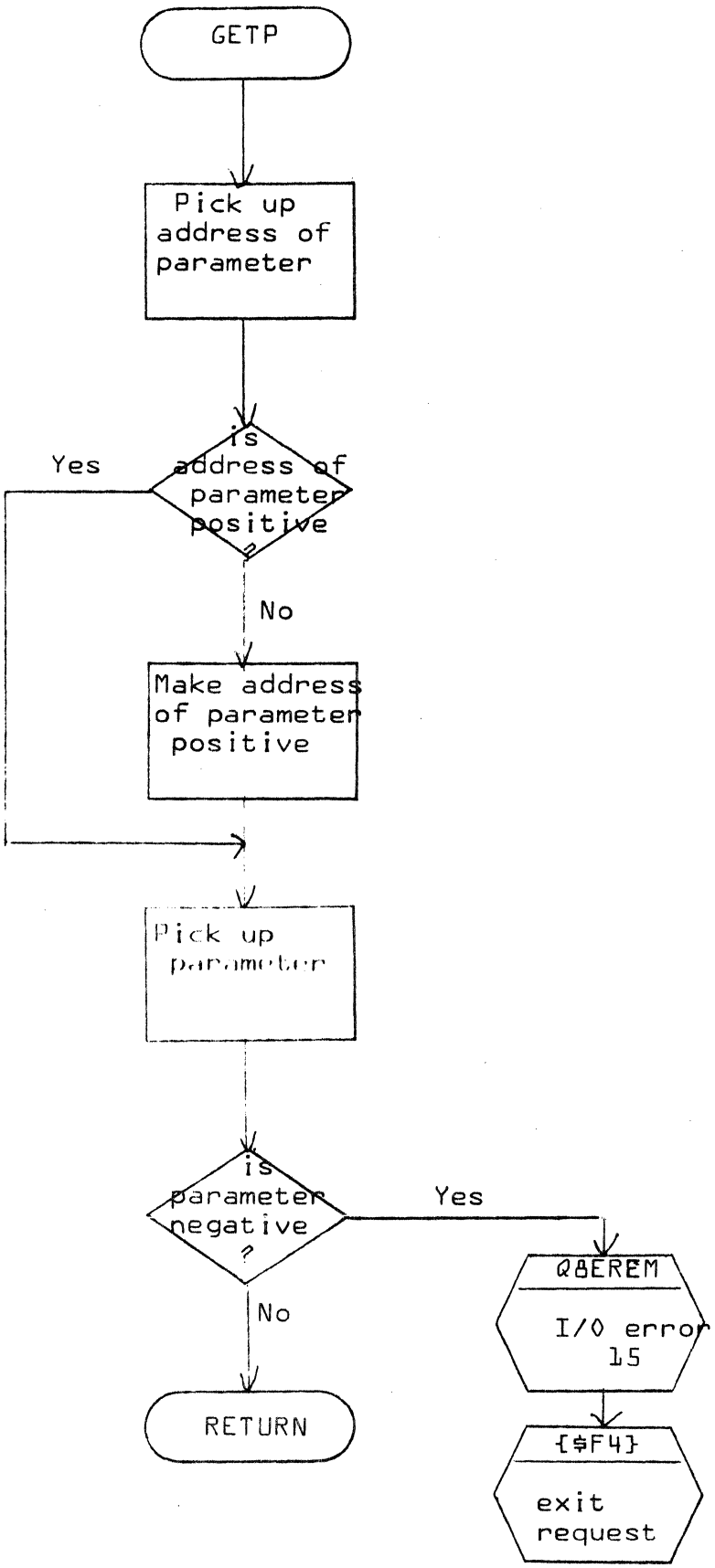
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8DFIO PAGE 3 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

CA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE: Q8DFIO	PAGE 4 OF 8	PROJECT MGR.	APPROVED
	NUMBER	ISSUE DATE	PROJECT NAME	REV
	DRAWN BY	DATE	TASK NO.	
			TASK NAME	



REV	APPROVED	DATE

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	QBDFIO	PAGE 5 OF 8	
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

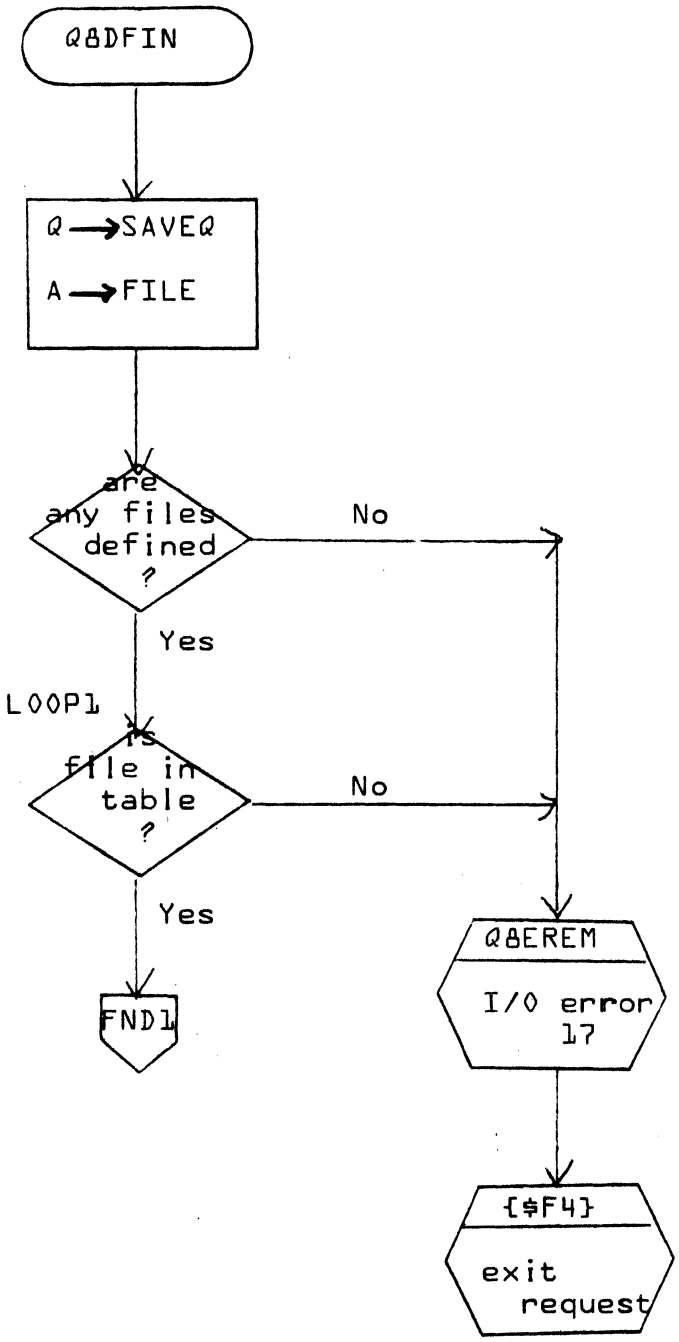
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

PRINTED IN U.S.A.

CA1385 ASSEMBLY CA127-11

A B C D

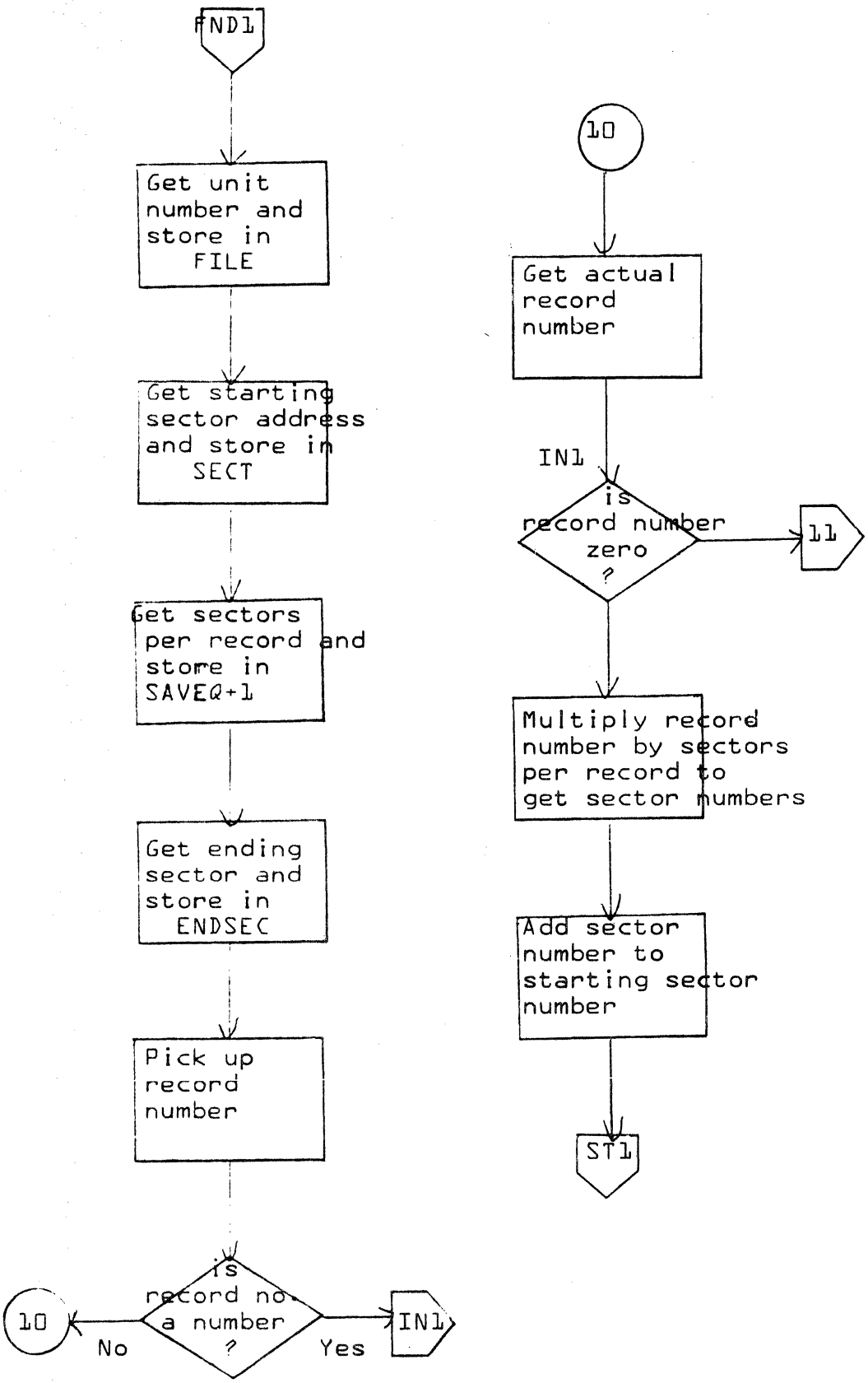


REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	
DOCUMENT TITLE		PROJECT MGR.	
Q8DFIO	PAGE 6 OF 8	PROJECT NAME	
NUMBER	ISSUE DATE	TASK NO.	
DRAWN BY	DATE	TASK NAME	



REV	APPROVED	DATE

PROJECT NO. | PROJECT MGR. | PROJECT NAME | TASK NO. | TASK NAME

DOCUMENT CLASS: IMS | MACH. TYPE: 1700 | PAGE 7 OF 8 | ISSUE DATE | DATE

DOCUMENT TITLE: QADFIO

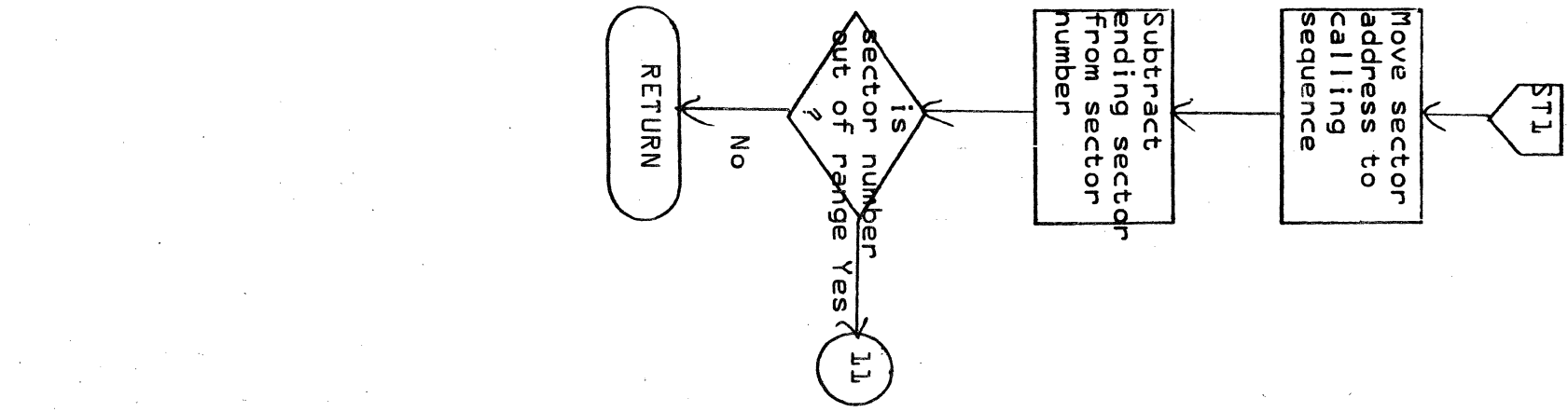
NUMBER | DRAWN BY

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐☐☐☐
FLOWCHART
DECISION TABLE
OTHER

PRINTED IN U.S.A.

A1385 (FORMERLY CA127-1)

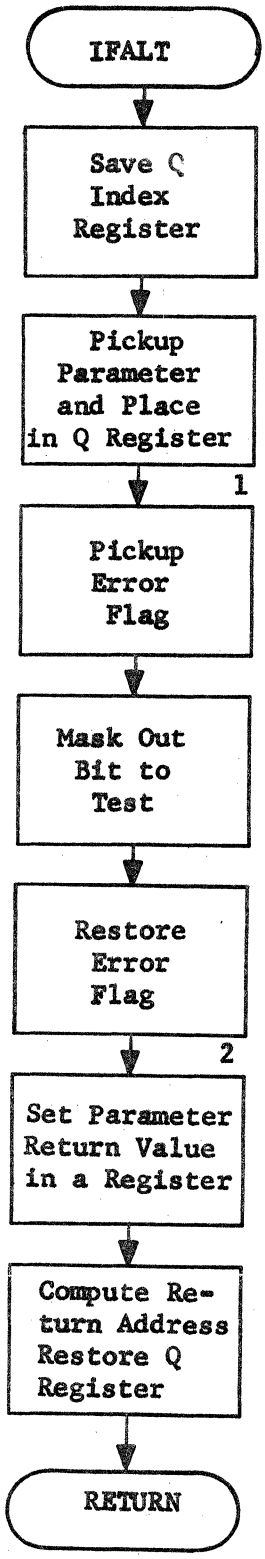
A B C D

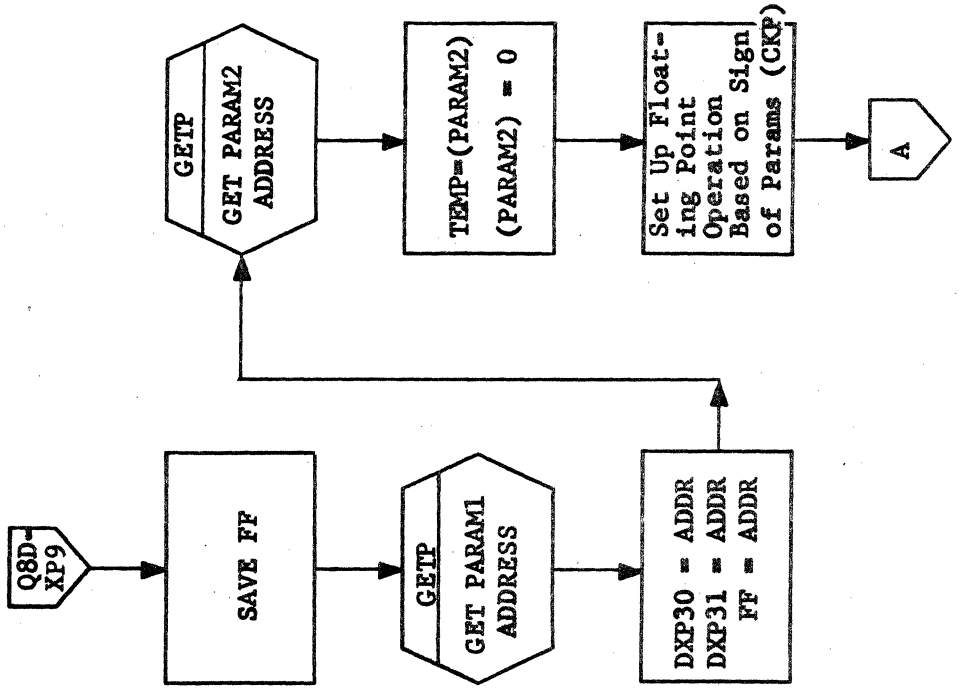


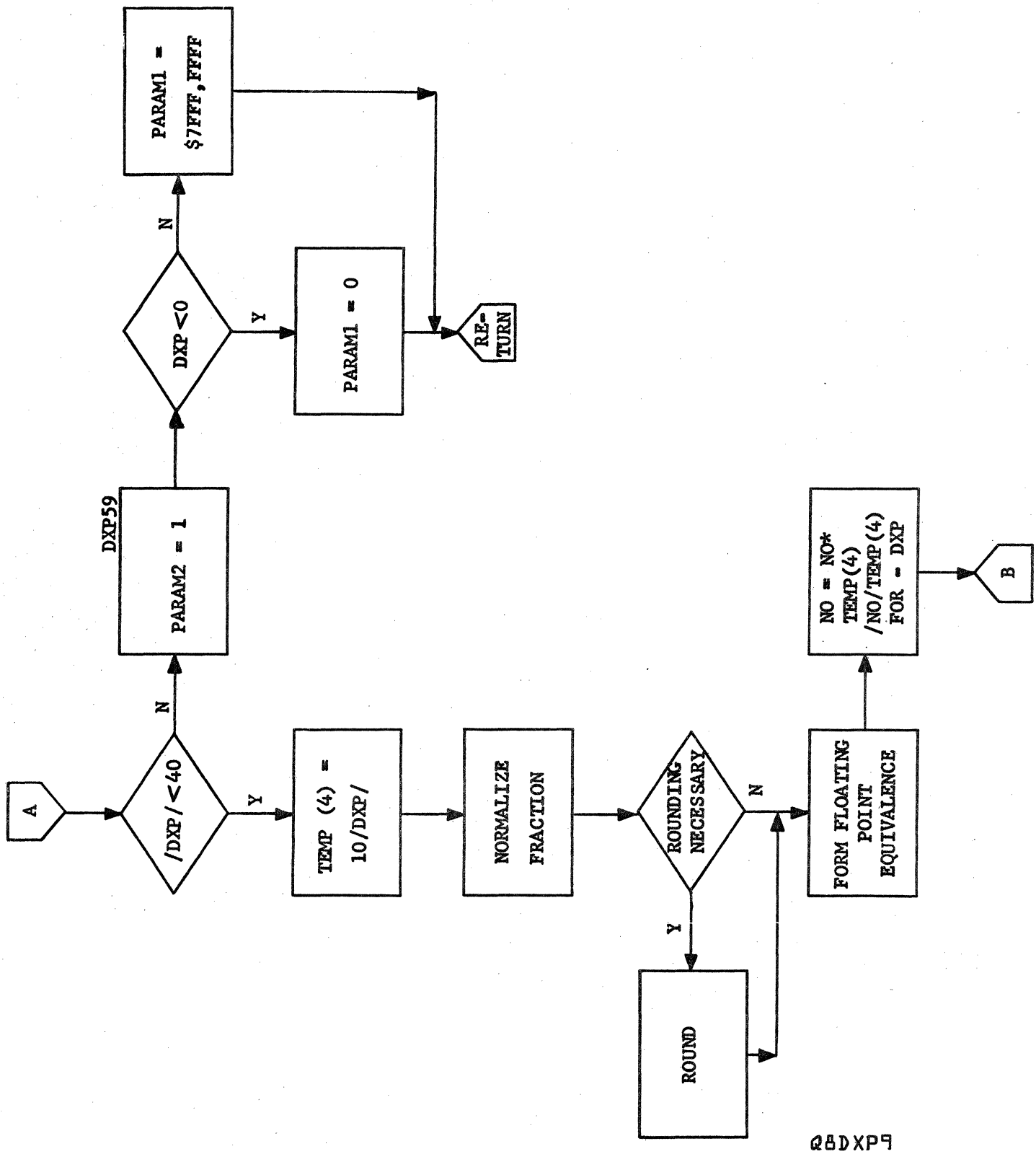
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	@8DFIO PAGE 8 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

1. The routines FLOAT and DFLOT puts the error flag in low core location $C8 from which IFALT can pick it up.

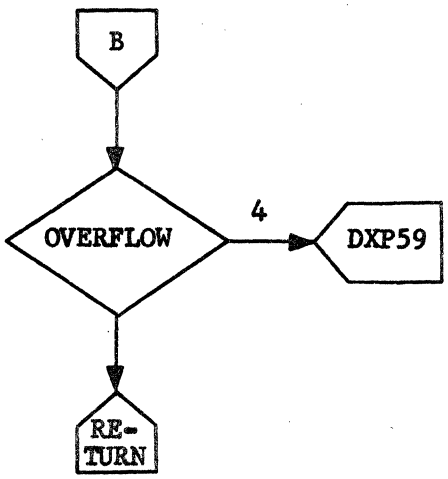
2. IFALT returns the information as to whether the error occurred in the A register.

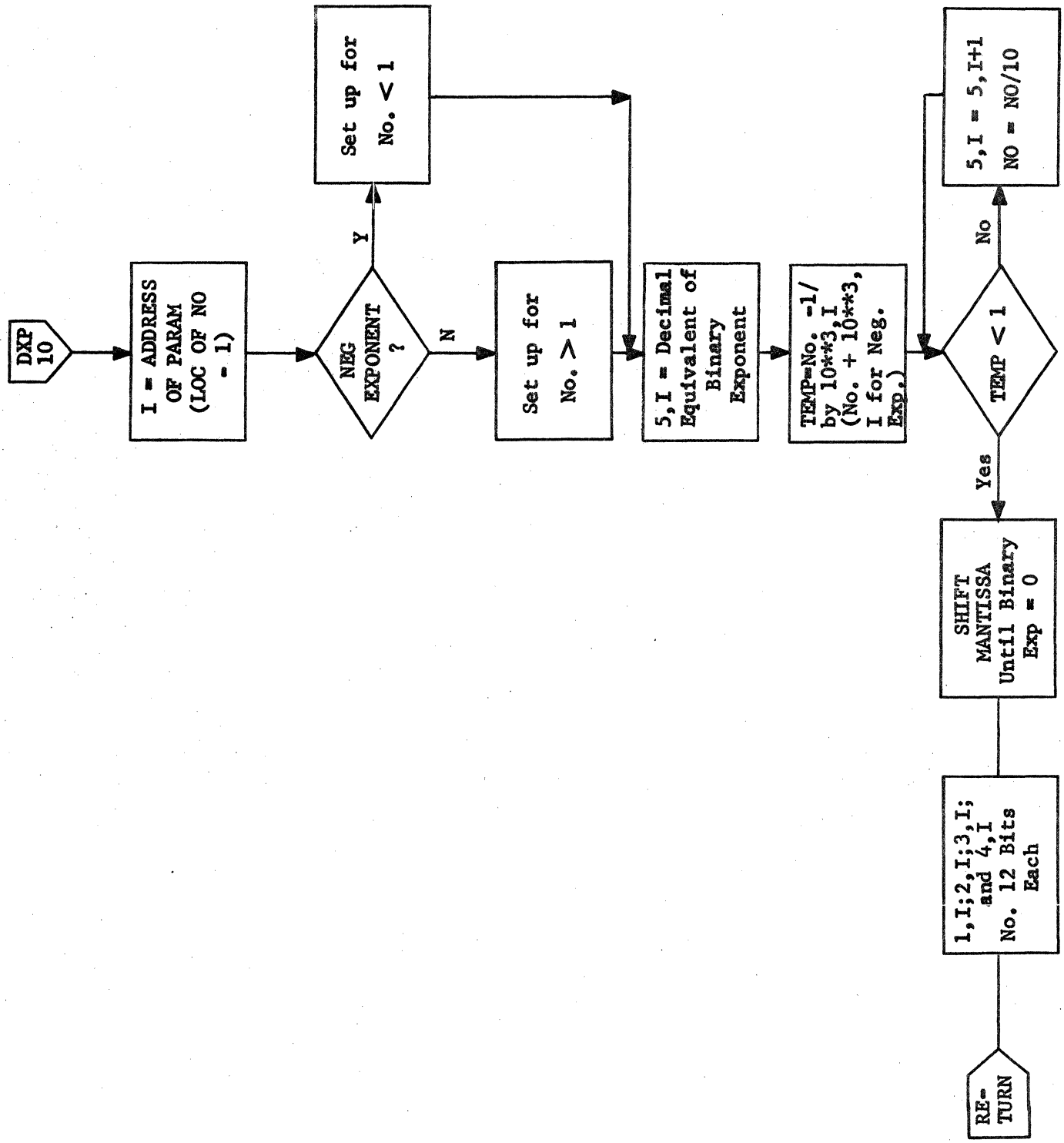



GETP
GET PARAM2 ADDRESS
TEMP=(PARAM2)
(PARAM2) = 0
Set Up Floating Point Operation Based on Sign of Params (CKP)
A
Q8D-XP9
SAVE FF
GETP
GET PARAM1 ADDRESS
DXP30 = ADDR
DXP31 = ADDR
FF = ADDR



Q8DXP9
Page 2 of 3


B
OVERFLOW
4
DXP59
RE-
TURN



Q8DXP1
Page 1 of 1

CHAPTER 7

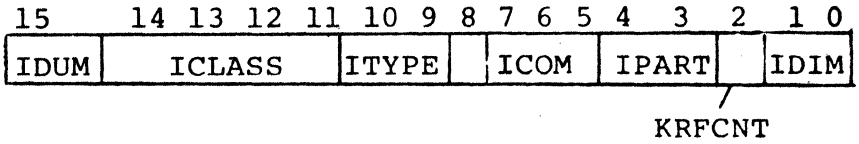
TABLE OF CONTENTS

7.0 TABLES

7.1 SYMBOL TABLE

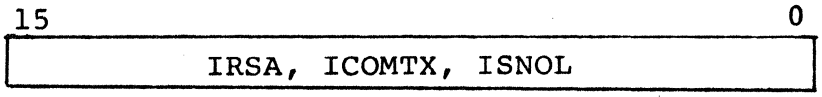
7.1.1 Symbol Table Format

WORD 1

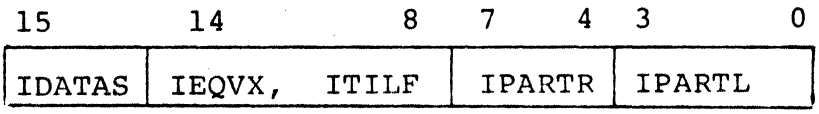


Bit 15 = KDUMY
Bit 8 = ISNGL, KELSIZ
Bit 1 = IREL
Bit 0 = IEXT

WORD 2



WORD 3



Bit 7 = INDUCV
Bit 6 = ISFARG
Bits 5-0 = IARGNO
Bit 15 = IREF

WORD 4 (Optional) ISYM

WORD 5 (Optional)

7.1.2 Meaning and Use of Symbol Table Entries

ITEM NAME	PHASES IN WHICH SET	PHASES IN WHICH USED	DESCRIPTION AND USE
*ISYM	A	A,B,C,E	The symbol name. After Pass 2 an index to the symbol's entry in SYMTAB is substituted for all symbol references. External names and constants are retained in this entry throughout compilation
ICLASS	A	A,B,C,E	Specifies the type of entry. 0 = Unassigned 1 = Variable or FUNCTION definition name 2 = Constant 3 = Intrinsic function 4 = Statement function 5 = External function 6 = External subroutine 7 = Statement label 8 = Subroutine definition name 9 = Program definition name
ITYPE	A	A,B,C,E	If applicable, specifies the arithmetic mode of the symbol. 0 = Unassigned 1 = Integer 2 = Real (floating point) 3 = Double Precision
*IPART	A	A,B	If 0, symbol is not a byte. If 1, symbol is a BYTE. If 2, symbol is a SIGNED BYTE.
IPARTL	A	A,B	The left most bit of a byte.
IPARTR	A	A,B	The right most bit of a byte.
IDUM	A	A,B,C,E	If non-zero, symbol is a dummy argument.
IDIM	A	A,B,C,E	When ICLASS is 1, entry contains the dimensionality of the array (if undimensioned = 0). The actual dimensions are in the ISTAB table. The index to this table for this array is in the ISTABX table.

*See note 1 (page 7-5)

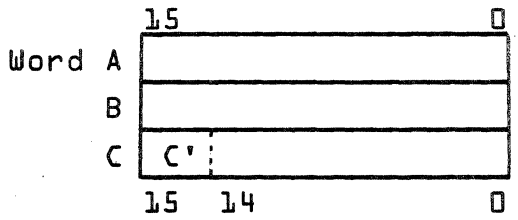
ITEM NAME	PHASES IN WHICH SET	PHASES IN WHICH USED	DESCRIPTION AND USE
KRFCNT	A,B	A,B,C,E	Number of references to this array or variable or constant. Used in placing this entity optimally vis-a-vis one word relative addressing.

ITEM NAME	PHASES IN WHICH SET	PHASES IN WHICH USED	DESCRIPTION AND USE
ISNOL	A	A,B	If ICLASS is 7, the statement number of this label.
IREF	A	B	If ICLASS is 2, non-zero means the constant is to be materialized.
ICOM	A	A,B,C,D,E	ICOM is the index to the ICOMT entry for the COMMON variable. If zero, this entry is not in COMMON.
IEQVX	A	A,B	IEQVX is the IEQV table index for this equivalenced entry. Used in generating definition points and for storage allocation.
KDUMY	A,B	A,B	This entry is a dummy argument-increment entry.
IDATAS	A	E	If non-zero, the variable was initialized in a data statement.
IRSA	C,D,E	C,D,E	The relative storage location of this entry.
ICOMTX	A	A,B	The SYMTAB index to a thread of the variables and arrays in a COMMON block. The first array of the thread is ICOMBX in the table ICOMT. ICOMTX is the SYMTAB index to the array declared prior to this array.
IEXT	A	A,B,C,D,E	If non-zero, this symbol appears in an EXTERNAL statement.
ISNGL	A	A,B	If non-zero, this symbol appears in a SINGLE statement.
IREL	A	A,B,C,D,E	If non-zero, this symbol appears in a RELATIVE statement.
INDUCV	A	A	Non-zero if variable is currently an induction variable.

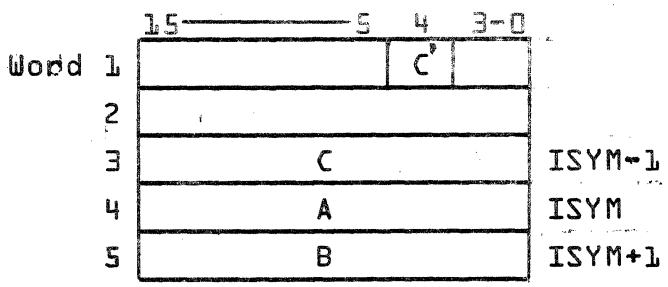
ITEM NAME	PHASES IN WHICH SET	PHASES IN WHICH USED	DESCRIPTION AND USE
IARGNO	A	A,B	Number of arguments in function call.
ISFARG	A	A,B	Set for statement function argument.
KELSIZ	A,B	A,B	If ICLASS is 1, the element size minus one.
ITILF	A	A	If ICLASS is 3, is the index to the in-line functions table.

Note 1: Double precision constants are 3 words long and are stored in the symbol table as follows:

Double Precision Constant



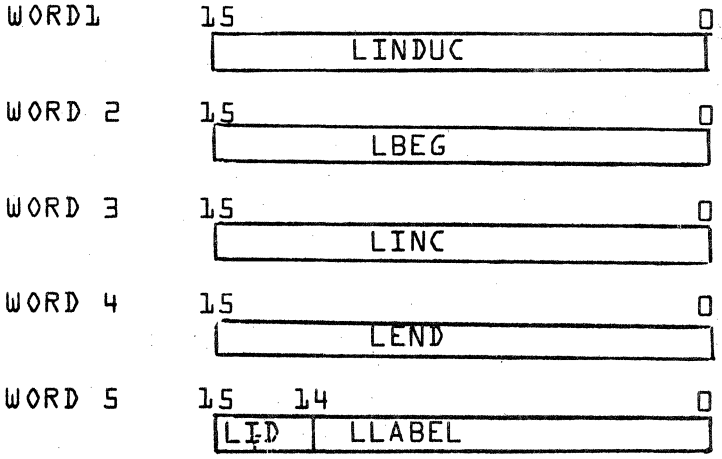
Symbol Table Entry



C' = High order bit of word 3 which must be preserved in the high order bit of IPART.

7.2 Loop Structure Table {Loopt}

7.2.1 Loop Structure Table Format



7.2.2 LOOP Structure Table {LOOPT}

The loop structure table is available in pass A. A BEGIN DO entry is entered in it when Pass A encounters a DO statement. An END DO entry is entered upon encountering a label which terminates the loop. LOOPT allows room for 30 loops. The LOOPT index is LOOPTX.

BEGIN DO ENTRY

ITEM	NO. BITS	DESCRIPTION AND USE
LINDUC	15	The SYMTAB index to the induction variable.
LBEG	15	The SYMTAB index to the initial value.
LINC	15	The SYMTAB index to the increment.
LID	1	≠0 decrementing loop
LLABEL	14	The SYMTAB index to label entry for terminating statemet.

7.3 FORTRAN Internal Code

	CHARACTER	CODE (Decimal)
Digits	0-9	0-9
Letters	A-Z	10-35
Dollar sign	$	36
Period	.	37
Plus sign	+	38
Minus sign	-	39
Equal sign	=	40
Left parenthesis	(	41
Right parenthesis	)	42
Comma	,	43
Slash	/	44
Asterisk	*	45
Blank	⊿	46
End of statement	⊿	47
Single quote	'	48

All other characters should be translated to blank and an appropriate message output.

7.4 Other Tables

7.4.1 Specification Table (ISTAB)

The current index for this table is ISTAB2. The maximum table size (to be compared against ISTAB2) is ISTABS and is 150. The entries are pointers to integer constants which are dimensions. For a given symbol, its SYMTAB entry is used to compute the entry in ISTABX table (ISYMX/(2*ISYMFL)+1) which gives the index to ISTAB of the first dimension. The SYMTAB entry IDIM gives the dimensionality.

7.4.2 Table of Symbol Table Presets {ISET}

Entry	Pointer To
1	constant 1
2	constant 1.0
3	Subroutine Q8QFLT
4	Subroutine Q8QFIX
5	Subroutine FLOT
6	constant 0
7	Subroutine Q8QF2F
8	Subroutine Q8QF2I
9	Subroutine Q8QI2F
10	Subroutine Q8STP
11	Subroutine Q8STPN
12	Subroutine Q8PSE
13	Subroutine Q8PSEN
14	Subroutine Q8PKUP
15	Subroutine Q8PREP
16	Subroutine Q8QFLE
17	Subroutine Q8QWND
18	Subroutine Q8QBCK
19	Subroutine Q8QINI
20	Variable Q8QX1
21	Variable Q8QX2
22	Variable Q8QX3
23	Subroutine Q8QX
24	Subroutine Q8QEND
25	Subroutine Q8DFNF
26	Subroutine Q8QY

7.5 OUTPUT FORMAT, PHASE A

7.5.1 Statement Format of PHASE A Output Files

The first four words are the same for all statements. Word 1 contains the number of words in this entry. This is zero for the first entry of the file. Word 2 is the generated statement number. Word three is the type number of the statement. Word 4 is a switch used to signal the generated statement or statements which comprise the second half of a logical IF. The file is terminated by an END card entry.

Statements with statement labels will have a negative statement number (its absolute value being sequential in the file). Word 5 will then be the SYMTAB index to the label.

The remaining words in each entry are described below. Statement labels, variables, and arrays are represented by their SYMTAB indices.

ASSEM	$parameter_1$, $parameter_2$, ..., parameter n
BLOCK DATA	no additional file entries
SUBROUTINE and FUNCTION	SYMTAB pointer to SUBROUTINE name (in complemented form) SYMTAB to FUNCTION (if applicable) SYMTAB pointer to formal parameter 1 : SYMTAB pointer to formal parameter n
DATA	pointer to entry to be set value pointer value :

FORMAT - contains the part of the FORMAT STATEMENT beginning with the first left parenthesis and ending with the last right parenthesis. This is represented in internal code, two characters per word.

REPLACEMENT	$tree_1$
	$tree_2$

where $tree_1$ is the tree of the variable to be stored into

$tree_2$ is the tree of the arithmetic expression

STATEMENT FUNCTION		SYMTAB pointer to statement function name SYMTAB pointer to formal parameter 1 . . . SYMTAB pointer to formal parameter n tree
ASSIGN I to n	I	SYMTAB pointer to ASSIGN variable
	n	SYMTAB pointer to entry assigned
CALL		tree
RETURN		no additional file entries
Unconditional GO TO		SYMTAB pointer to label
COMPUTED GO TO		SYMTAB pointer to label 1 . . . SYMTAB pointer to label n tree
Assigned GO TO		variable
CONTINUE		no additional file entries
STOP		no additional file entries
STOP n		SYMTAB pointer to n
PAUSE		no additional file entries
PAUSE n		SYMTAB pointer to n
END FILE n		translated to CALL Q8QFLE{N}
REWIND n		CALL Q8QWND{N}
BACKSPACE n		CALL Q8QBCK{N}
END		no additional file entry needed If END is not directly preceded by STOP or RETURN, a STOP or RETURN statement is generated before END.

BEGIN DO	SYMTAB indices for index variable, initial increment and final values, termination label
END DO	Index to LOOPT END DO
ARITHMETIC IF	SYMTAB index to $label_1$
	SYMTAB index to $label_2$
	SYMTAB index to $label_3$
	tree

LOGICAL IF tree

7.5.2 Statement Types

TYPE	STATEMENT NAME
0	DIMENSION
1	COMMON
2	INTEGER
3	REAL
4	INTEGER FUNCTION
5	REAL FUNCTION
6	PROGRAM
7	SINGLE
8	BYTE
9	SIGNED BYTE
10	EXTERNAL
11	RELATIVE
12	EQUIVALENCE
13	BLOCK DATA
14	FUNCTION
15	SUBROUTINE
16	DATA
17	FORMAT
18	Replacement Statement
19	Statement Function
20	ASSIGN
21	CALL
22	RETURN
23	UNUSED
24	GO TO {Unconditional}
25	GO TO {Computed}
26	GO TO {Assigned}
27	CONTINUE
28	STOP
29	STOP n
30	PAUSE
31	PAUSE n
32	END
33	ENDFILE
34	REWIND
35	BACKSPACE
36	READ {Unformatted}
37	READ {Formatted}
38	WRITE {Unformatted}
39	WRITE {Formatted}
40	BEGIN DO
41	END DO
42	Arithmetic IF

TYPE	STATEMENT NAME
43	Logical IF
44	ASSEM
45	OPEN
46	DOUBLE PRECISION
47	DOUBLE PRECISION FUNCTION

7.5.3 ARITHMETIC.TREE

7.5.3.1 General Form of Arithmetic Tree

WORD 1 Tree Indicator {-1}
WORD 2 No. words in expression {from WORD 4 to end}
WORD 3 Mode of expression {0=?, 1=integer, 2=real, 3=double
WORD 4 Operator {see list}
WORD 5 {assuming operator takes only one word} No. Operands
WORD 6 WORD 6+{word 5}-1 operand pointers {base operator this level}
WORD {NEXT} operand normal/operand inverse {0=0≠0}
WORD {NEXT}+1} operand {see list}

NOTE: each operand {operator} except the first is preceded by a normal/inverse indicator

Example I+R+D

WORD 1	-1
WORD 2	14
WORD 3	3
WORD 4	11
WORD 5	3
WORD 6	5
WORD 7	8
WORD 8	11
WORD 9	0
WORD 10	24
WORD 11	D
WORD 12	0
WORD 13	24
WORD 14	R
WORD 15	0
WORD 16	24
WORD 17	I

$$6+\{\text{OPERAND POINTER}\}^{5}_{\{1\}}\text{operand}$$
= 11

$$6+\{\text{OPERAND POINTER}\}_{\{2\}}^{8}\text{operand} = 14$$

$$6+\{\text{OPERAND POINTER}\}_{\{3\}}^{11}\text{operand} = 17$$

Types 22 - 24, 29, 34, 35 are two word entries

WORD 1 operand type
WORD 2 symbol table pointer

Types 18 - 21 take two words to express type

WORD 1 operand type
WORD 2 symbol table pointer
WORD 3 no. operands etc.

Types 25 - 30 are of the following form

WORD 1 25, 30
WORD 2 symbol table pointer
WORD 3 24, 29 for subscript variable
WORD 4 symbol table pointer

Types 26 and 31 are of the following form

WORD 1 26, 31
WORD 2 symbol table pointer
WORD 3 symbol table pointer for constant subscript

Types 27 and 32 are of the following form

WORD 1 27, 32
WORD 2 symbol table pointer
WORD 3 symbol table pointer for constant portion
WORD 4 24, 29 for subscript variable
WORD 5 symbol table pointer for subscript variable

Types 28 and 33 are of the following form

WORD 1	28,33
WORD 2	symbol table pointer
WORD 3	symbol table pointer for constant portion
WORD 4	operator (+ or *)
WORD 5+ff	no. operands, pointers, operands

7.5.3.2 Operand Types (Operators are Also Operands)

1. COMMA - does not appear in File 2
2. AND
3. OR
4. NOT - does not appear in
5. LT
6. GT
7. LE
8. GE
9. EQ
10. NE
11. +
12. - does not appear in File 2 - represented by inverse +
13. *
14. / does not appear in File 2 - represented by inverse *
15. **
16. (does not appear in File 2
17.) does not appear in File 2
18. function
19. unused
20. subroutine
21. unused
22. function with no argument
23. subroutine with no argument
24. non-subscripted variable
25. variable only subscripted variable
26. increment only subscripted variable
27. variable and increment subscripted variable
28. complex subscripted variable
29. non-subscripted partial variable
30. variable only subscripted partial variable
31. increment only subscripted partial variable
32. variable and increment subscripted variable
33. complex subscripted partial variable
34. Hollerith constant
35. numeric constant
36. calling sequence label
37. material constant

7.6.1 Mass Storage Fortran 3.1A Loader Maps

7.6.1.1 Phase A1

FTN	3E63
GOA	4515
CNVT	4576
CONV	45BB
DIAG	45EE
DXP9	4686
DFLOT	47AF
DUMVOL	49EA
GETSYM	4A1A
GPUT	4A53
IOPRBA	4A7C
PACK	4CDB
@8PRMS	4D00
STORE	4D1A
SYMBOL	4D60
SAVEID	4E21
LOCLA1	4EC6
DUMYA1	4F75
STCHAR	4FDC
GETC	500E
ENDDO	5027
GETF	512C
GNST	5472
IGETCF	564E
OPTION	5667
OUTENT	56D5
PHASEA	5709
PLABEL	5C31
@8@BDS	5C87
RDLABL	5C87
TYPE	5D25
ENDLOC	5F52

7.6.1.2 Phase A2

FTN	3E63
GOA	4515
CNVT	4576
CONV	45BB
DIAG	45EE
DXP9	4686
DFLOT	47AF
DUMVOL	49EA
GETSYM	4A1A
GPUT	4A53

IOPRBA	4A7C
PACK	4CDB
Q8PRMS	4D00
STORE	4D1A
SYMBOL	4D60
SAVEID	4E21
LOCLA2	4EC6
DUMYA2	4F78
ARITH	4F85
COMNPR	55FE
GETC	569A
GETF	56B3
DIMPR	59F9
SUBSCR	5BA7
TYPEPR	5E6C
ENDLOC	5E83

7.6.1.3 Phase A3

FTN	3E63
GOA	4515
CNVT	4576
CONV	45BB
DIAG	45EE
DXP9	4686
DFLOT	47AF
DUMVOL	49EA
GETSYM	4A1A
GPUT	4A53
IOPRBA	4A7C
PACK	4CDB
Q8PRMS	4D00
STORE	4D1A
SYMBOL	4D60
SAVEID	4E21
LOCLA3	4EC6
DUMYA3	4F78
BYEQPR	4F85
CHECKF	517C
CONSUB	521F
DATAPR	52A6
OUTENT	5777
FGETC	57AB
FORK	586F
GETC	5A09
GETF	5A22
STCHAR	5D68
FREE	5D9A
ENDLOC	62B9

7.6.1.4 Phase A4

FTN	3E63
GOA	4515
CNVT	4576
CONV	45BB
DIAG	45EE
DXP9	4686
DFLOT	47AF
DUMVOL	49EA
GETSYM	4A1A
GPUT	4A53
IOPRBA	4A7C
PACK	4CDB
Q8PRMS	4D00
STORE	4D1A
SYMBOL	4D60
SAVEID	4E21
LOCLA4	4EC6
DUMYA4	4F78
ARAYSZ	4F7F
ASGNPR	5001
BDOPR	5047
CFIVOC	5184
CKIVC	51E3
CKNAME	51F3
CPLOOP	5203
ENDDO	52B0
GETC	53B5
GETF	53CE
IOSPR	5714
OUTENT	5DE7
RDLABL	5E1B
STCHAR	5EB9
ENDLOC	5EEB

7.6.1.5 Phase A5

FTN	3E63
GOA	4515
CNVT	4576
CONV	45BB
DIAG	45EE
DXP9	4686
DFLOT	47AF
DUMVOL	49EA
GETSYM	4A1A
GPUT	4A53
IOPRBA	4A7C

PACK	4CDB
Q8PRMS	4D00
STORE	4D1A
SYMBOL	4D60
SAVEID	4E21
LOCLA5	4EC6
DUMYA5	4F75
ARITH	4F82
GETC	55FC
GETF	5615
SUBSCR	595B
ENDLOC	5C20

7.6.1.6 Phase A6

FTN	3E63
GOA	4515
CNVT	4576
CONV	45BB
DIAG	45EE
DXP9	4686
DFLOT	47AF
DUMVOL	49EA
GETSYM	4A1A
GPUT	4A53
IOPRBA	4A7C
PACK	4CDB
Q8PRMS	4D00
STORE	4D1A
SYMBOL	4D60
SAVEID	4E21
LOCLA6	4EC6
DUMYA6	4F78
ERBPR	4F89
STCHAR	4FDC
GETC	500E
CFIVOC	5027
CKIVC	5086
GETF	5096
MODMXR	53DC
RDLABL	5841
SUBPPR	58DF
TREE	5C93
ENDLOC	6182

7.6.1.7 Phase A7

FTN	3E63
GOA	4515

CNVT	4576
CONV	45BB
DIAG	45EE
DXP9	4686
DFLOT	47AF
DUMVOL	49EA
GETSYM	4A1A
GPUT	4A53
IOPRBA	4A7C
PACK	4CDB
Q8PRMS	4D00
STORE	4D1A
SYMBOL	4D60
SAVEID	4E21
LOCLA7	4EC6
DUMYA7	4F78
ASEMPR	4F78
EXRLPR	5141
GETC	51A1
GETF	51BA
IGETCF	5500
PE@VS	5519
PRNTNM	593A
PUNT	59C9
RDLABL	5A01
SYMSCN	5A9F
ENDLOC	5ABB

7.6.1.8 Phase B1

FTN	3E63
GOB	457F
CNVT	4597
DUMMY	45DC
FCMSTK	46EF
GETSYM	478D
IOPRBB	47C6
KCPART	4997
KOUTPT	49C8
KPCSTK	49DA
KPC3PR	4F87
KSYMGN	4F9F
LABKPC	4FE7
LABLER	4FFB
PUNT	5019
Q8PRMS	502F
STOREB	5049
SYMBOL	507D
TSALOC	511A

LOCLB1	51B5
DUMYB1	5248
ARAYSZ	5271
ASSEM	52F3
BANANA	535C
BGINDO	5426
END	5536
ENTCOD	5587
HELEN	5636
INXRST	5790
NOPROC	57A4
PHASEB	57D5
READIR	5C76
SUBFUN	5CCE
SYMSCN	5D35
ENDLOC	5D51

7.6.1.9 Phase B2

FTN	3E63
GOB	457F
CNVT	4597
DUMMY	45DC
FCMSTK	46EF
GETSYM	478D
IOPRBB	47C6
KCPART	4997
KOUTPT	49C8
KPCSTK	49DA
KPC3PR	4F87
KSYMGN	4F9F
LABKPC	4FE7
LABLER	4FFB
PUNT	5019
Q8PRMS	502F
STOREB	5049
SYMBOL	507D
TSALOC	511A
LOCLB2	51B5
ACP	524A
AFIDL	5759
ASUPER	57BA
CGOTO	5875
FINK	58D6
INTRAM	598C
PARTSB	5B9B
SUBPR1	5C4F
SUBPR2	5C93
SUBPR3	5D21
ENDLOC	5D68

7.6.1.10 Phase B3

FTN	3E63
GOB	457F
CNVT	4597
DUMMY	45DC
FCMSTK	46EF
GETSYM	478D
IOPRBB	47C6
KCPART	4997
KOUTPT	49C8
KPCSTK	49DA
KPC3PR	4F87
KSYMGN	4F9F
LABKPC	4FE7
LABLER	4FFB
PUNT	5019
Q8PRMS	502F
STOREB	5049
SYMBOL	507D
TSALOC	511A
LOCLB3	51B5
ACP	5248
ARITHR	5757
ASUPER	59C7
FINK	5A82
INTRAM	5B38
PARISB	5D47
SUBPR1	5DFB
SUBPR2	5E3F
SUBPR3	5ECD
ENDLOC	5F14

7.6.1.11 Phase C1

FTN	3E63
GOC	4A55
BKDWN	4A6C
BLDUP	4ACB
BSS	4B0E
CHKWD	4B2C
CHOP	4CB6
CL12	4EE1
CON	4FD7
COUNT	5031
DATAST	5048
GETSYM	5130
INOUT	51D4
IXOPT	5243

PHASEC	5384
LABEL	5734
LABIN	5756
QXLD	57BC
REED	584C
SKIP	58B0
SYMSCN	5906
IOPRBC	5922
Q8PRMS	5BAD
ENDLOC	5BC7

7.6.1.12 Phase D1

FTN	3E63
GOOD	42D5
INDEX	42FA
IOPRBD	4316
NPUNCH	45C7
Q8PRMS	470F
PHASE6	4729
LOCLD1	47D1
DUMYD1	4895
AMT	48A3
AMOUT	48AC
ADMAX	4E86
BKDWN	5089
COUNT	5122
LABOUT	5139
NP2OUT	521C
RBDX	524B
RBPK	528D
TABDEC	52B7
UNPUNC	5341
GETSYM	5357
SYMSCN	5393
ENDLOC	53BA

7.6.1.13 Phase D2

FTN	3E63
GOOD	42D5
INDEX	42FA
IOPRBD	4316
NPUNCH	45C7
Q8PRMS	470F
PHASE6	4729
LOCLD2	47D1
DUMYD2	4896

AMT	489D
GETSYM	48A4
IACON	48D2
IHCON	492C
NWRITE	4959
PACK	4994
SYMSCN	49B9
BEGINO	49D5
FINISH	4BB9
ENDLOC	4D91

7.6.1.14 Phase E1

FTN	3E63
GOE	42D5
INDEX	42FA
IOPRBD	4316
NPUNCH	45C7
Q8PRMS	470F
PHASE6	4729
LOCLD1	47D1
DUMYD1	4895
AMT	48A3
AMOUT	48AC
ADMAX	4EBB
BKDWN	50BE
COUNT	5127
LABOUT	513E
NP2OUT	5260
RBDX	5298
RBPK	52DB
TABDEC	5305
UNPUNC	538F
CONV	53A5
GETSYM	53DE
IACON	542B
IHCON	5485
NWRITE	54B1
PACK	54EC
SETPRT	5511
SYMSCN	5699
ENDLOC	56B5

7.6.1.15 Phase E2

FTN	3E63
GOE	42D5
INDEX	42FA

IOPRBD	4316
NPUNCH	45C7
Q8PRMS	470F
PHASE6	4729
LOCLD2	47D1
DUMYD2	4896
AMT	489D
CONV	48A4
GETSYM	48DD
IACON	492A
IHCON	4984
NWRITE	49B0
PACK	49E8
SETPRT	4A10
SYMSCN	4B98
BEGIN0	4BB4
FINISH	4D3E
ENDLOC	4EEA

7.6.2 Mass Storage Fortran 3.1B Loader Maps

7.6.2.1 Phase A1

FTN	3E63
G0A	4519
CFIVOC	457A
CKNAME	45D9
CNVT	45E9
CONV	462E
DIAG	4661
DXP9	46F9
DFLOT	4822
DUMVOL	4A5D
GETC	4A8D
GETF	4AB8
GETSYM	4E22
GPUT	4E5B
IGETCF	4E84
IOPRBA	4E9D
PACK	51AF
Q8PRMS	51D4
RDLABL	51EE
STORE	528C
SYMBOL	52D2
ENDDO	5393
GNST	5498
OPTION	5674
OUTENT	56E2
PHASEA	5716

PLABEL	5C3E
STCHAR	5C94
TYPE	5CC6
SAVEID	5EF3
LOCLA1	5F98
DUMYA1	6056
Q8QBDS	60BD
ENDLOC	60BD

7.6.2.2 Phase A2

FTN	3E63
GOA	4519
CFIVOC	457A
CKNAME	45D9
CNVT	45E9
CONV	462E
DIAG	4661
DXP9	46F9
DFLOT	4822
DUMVOL	4A5D
GETC	4A8D
GETF	4AB8
GETSYM	4E22
GPUT	4E5B
IGETCF	4E84
IOPRBA	4E9D
PACK	51AF
Q8PRMS	51D4
RDLABL	51EE
STORE	528C
SYMBOL	52D2
ENDDO	5393
GNST	5498
OPTION	5674
OUTENT	56E2
PHASEA	5716
PLABEL	5C3E
STCHAR	5C94
TYPE	5CC6
SAVEID	5EF3
LOCLA2	5F98
DUMYA2	6056
BYEQPR	60BD
CHECKF	62B4
COMNPR	6357
CONSUB	63F3
DATAPR	647A
DIMPR	694B

EXRLPR	6AF9
FGETC	6B59
FORK	6C1D
PEQVS	6DB7
PRNTNM	71E5
SUBPPR	7274
SYMSCN	731C
TYPEPR	7338
ENDLOC	734F

7.6.2.3 Phase A3

FTN	3E63
GOA	4519
CFIVOC	457A
CKNAME	45D9
CNVT	45E9
CONV	462E
DIAG	4661
DXP9	46F9
DFLCT	4822
DUMVOL	4A5D
GETC	4A8D
GETF	4AB8
GETSYM	4E22
GPUT	4E58
IGETCF	4E84
IOPRBA	4E9D
PACK	51AF
Q8PRMS	51D4
RDLABL	51EE
STORE	528C
SYMBOL	52D2
ENDDO	5393
GNST	5498
OPTION	5674
OUTENT	56E2
PHASEA	5716
PLABEL	5C3E
STCHAR	5C94
TYPE	5CC6
SAVEID	5EF3
LOCLA3	5F98
DUMYA3	6056
ARAYSZ	60BD
ASEMPR	613F
ASGNPR	6308
BDOPR	634E
CHECKF	648B

CKIVC	652E
CONSUB	653E
CPLOOP	65C5
FGETC	6672
FORK	6736
ERBPR	68D0
MODMXR	6923
PUNT	7088
ENDLOC	70C0

7.6.2.4 Phase A4

FTN	3E63
GOA	4519
CFIVOC	457A
CKNAME	45D9
CNVT	45E9
CONV	462E
DIAG	4661
DXP9	46F9
DFLOT	4822
DUMVOL	4A5D
GETC	4A8D
GETF	4AB8
GETSYM	4E22
GPUT	4E5B
IGETCF	4E84
IOPRBA	4E9D
PACK	51AF
Q8PRMS	51D4
RDLABL	51EE
STORE	528C
SYMBOL	52D2
ENDDO	5393
GNST	5498
OPTION	5674
OUTENT	56E2
PHASEA	5716
PLABEL	5C3E
STCHAR	5C94
TYPE	5CC6
SAVEID	5EF3
LOCLA4	5F98
DUMYA4	6056
ARITH	60BD
SUBSCR	6769
TREE	6A2E
ENDLOC	6F43

7.6.2.5 Phase A5

FTN	3E63
G0A	4519
CFIV0C	457A
CKNAME	45D9
CNVT	45E9
CONV	462E
DIAG	4661
DXP9	46F9
DFLCT	4822
DUMV0L	4A5D
GETC	4A8D
GETF	4AB8
GETSYM	4E22
GPUT	4E5B
IGETCF	4E84
I0PRBA	4E9D
PACK	51AF
Q8PRMS	51D4
RDLABL	51EE
STORE	528C
SYMB0L	52D2
ENDD0	5393
GNST	5498
OPTION	5674
OUTENT	56E2
PHASEA	5716
PLABEL	5C3E
STCHAR	5C94
TYPE	5CC6
SAVEID	5EF3
L0CLA5	5F98
DUMYA5	6056
BD0PR	608D
CKIVC	61FA
I0SPR	620A
ENDL0C	68F1

7.6.2.6 Phase B1

FTN	3E63
G0B	4583
CNVT	4599

DUMMY	45DE
FCMSTK	56F1
GETSYM	478F
IOPRBB	47C8
KCPART	4D52
KOUTPT	4D83
KPCSTK	4D95
KPC3PR	5342
KSYMGN	535A
LABKPC	53A2
LABLER	53B6
PUNT	53D4
Q8PRMS	53EA
STOREB	5404
SYMBOL	5438
TSALOC	54D5
ARAYSZ	5570
ASSEM	55F2
BANANA	565B
BGINDO	5725
END	5835
ENTCOD	5886
HELEN	5935
INXRST	5A8F
NOPROC	5AA3
PHASEB	5AD4
READIR	5F76
SUBFUN	5FCE
SYMSCN	6035
ACP	6051
AFIDL	6560
ASUPER	66C1
CGOTO	667C
FINK	66DD
INTRAM	6793
PARTSB	69A2
SUBPR1	6A56
SUBPR2	6A9A
SUBPR3	6B28
ARITHR	6B6F
ENDLOC	6DDF

7.6.2.7 Phase C1

FTN	3E63
GOC	4A59
BKDWN	4A77
BLDUP	4AD6
BSS	4B19

CHKWD	4B37
CHOP	4CC1
CLI2	4EEC
CON	4FE2
COUNT	503C
DATAST	5053
GETSYM	513B
INOUT	51DF
IOPRBC	524E
IXOPT	60FC
LABEL	623D
LABIN	625F
PHASEC	62C5
Q8PRMS	6626
QXLD	6640
REED	66D0
SKIP	6734
SYMSCN	678A
ENDLOC	67A6

7.6.2.8 Phase D1

FTN	3E63
GOOD	4A59
AMOUT	4A7C
ADMAX	507A
GEGINO	527D
BKDWN	53C5
COUNT	542E
FINISH	5445
GETSYM	55F1
IACON	5695
IHCON	56EF
INDEX	571C
IOPRBD	5738
LABOUT	5CE6
NP2OUT	5DC9
NPUNCH	5DF8
NWRITE	5F40
PACK	5F7B
PHASE6	5FA0
Q8PRMS	6043
RBDX	605D
RBPK	609F
SYMSCN	60C9
TABDEC	60E5
UNPUNC	616F
ENDLOC	6185

7.6.2.9 Phase E1

FTN	3E63
GOE	4A59
AMOUT	4A7C
ADMAX	508B
BEGIN0	528E
BKDWN	540A
CONV	5473
COUNT	54AC
FINISH	54C3
GETSYM	566F
IACON	5713
IHCON	576D
INDEX	5799
IOPRBD	57B5
LABOUT	5D63
NP20UT	5E85
NPUNCH	5EBD
NWRITE	6005
PACK	6040
PHASE6	6065
Q8PRMS	6108
RBDX	6122
RBPK	6165
SETPRT	618F
SYMSCN	6317
TABDEC	6333
UNPUNC	63BD
ENDLOC	63D3

```
   LOCLAB   5C29  FORTRAN 3.0B SUMMARY-REL   20F
   DUMYAB   5CE7  FORTRAN 3.0B SUMMARY-REL   21F
   ARAYSZ   5D4E  FORTRAN 3.0B SUMMARY-REL   42A
   ASEMPR   5D8C  FORTRAN 3.0B SUMMARY-REL   40A
   ASGNPR   5F6A  FORTRAN 3.0B SUMMARY-REL   32A
   BDUPR    5FB1  FORTRAN 3.0B SUMMARY-REL   33A
   CHECKF   60EA  FORTRAN 3.0B SUMMARY-REL   29A
   CKIVC    618E  FORTRAN 3.0B SUMMARY-REL   35A
   CONSOB   619F  FORTRAN 3.0B SUMMARY-REL   30A
   CPLOOP   6225  FORTRAN 3.0B SUMMARY-REL   43A
   DATAPR   62CA  FORTRAN 3.0B SUMMARY-REL   31A
   FGETC    645B  FORTRAN 3.0B SUMMARY-REL   21A
   FORK     64EB  FORTRAN 3.0B SUMMARY-REL   22A
   ERRPR    6688  FORTRAN 3.0B SUMMARY-REL   38A
   MODMAR   66DA  FORTRAN 3.0B SUMMARY-REL   39A
   PUNT     6B3C  FORTRAN 3.0B SUMMARY-REL   27A
   ENDLOC   6B75  FORTRAN 3.0B SUMMARY-REL   17F
IN

*K,Io
IN

*N,FURIA3,,,B
IN

*K,Io
IN

*P
   FTN      3DC4  COPYRIGHT CONTROL DATA CORP. 1972
   GUA      447A  FORTRAN 3.0B SUMMARY-REL   02F
   CFIVOC   44DB  FORTRAN 3.0B SUMMARY-REL   34A
   CKNAME   4539  FORTRAN 3.0B SUMMARY-REL   36A
   CNVT     4549  FORTRAN 3.0B SUMMARY-REL   01A
   CONV     4587  FORTRAN 3.0B SUMMARY-REL   03F

   DIAG     45BA  FORTRAN 3.0B SUMMARY-REL   04F
   EXP9     464A  FORTRAN 3.0B SUMMARY-REL   05F
   FLOAI    46F3  FORTRAN 3.0B SUMMARY-REL   06F
   GETC     4930  FORTRAN 3.0B SUMMARY-REL   14F
   GETF     48BA  FORTRAN 3.0B SUMMARY-REL   04A
   GETSYM   4B03  FORTRAN 3.0B SUMMARY-REL   07F
   GPOT     4B9C  FORTRAN 3.0B SUMMARY-REL   02A
   IGETCF   4BC5  FORTRAN 3.0B SUMMARY-REL   15F
   IOPRBA   4BDE  FORTRAN 3.0B SUMMARY-REL   08F
   PACK     4EC6  FORTRAN 3.0B SUMMARY-REL   09F
   QBFRMS   4F0A  FORTRAN 3.0B SUMMARY-REL   10F
```

```
   RDLABL   4F25  FORTRAN 3.0B SUMMARY-REL   10A
   STORE    4FC3  FORTRAN 3.0B SUMMARY-REL   11F
   SYMBOL   4FF7  FORTRAN 3.0B SUMMARY-REL   03A
   ENDDO    5099  FORTRAN 3.0B SUMMARY-REL   29A
   GNST     51A0  FORTRAN 3.0B SUMMARY-REL   05A
   OPTION   5301  FORTRAN 3.0B SUMMARY-REL   16F
   OUTENT   53CE  FORTRAN 3.0B SUMMARY-REL   06A
   PHASEA   5403  FORTRAN 3.0B SUMMARY-REL   07A
   PLABEL   54F4  FORTRAN 3.0B SUMMARY-REL   08A
   STCHAR   594E  FORTRAN 3.0B SUMMARY-REL   11A
   TYPE     5993  FORTRAN 3.0B SUMMARY-REL   12A
   SAVELD   5B63  FORTRAN 3.0B SUMMARY-REL   13A
   LOCLA4   5C29  FORTRAN 3.0B SUMMARY-REL   22F
   DUMYA4   5CE7  FORTRAN 3.0B SUMMARY-REL   23F
   ARITH    5D4E  FORTRAN 3.0B SUMMARY-REL   14A
   SUBSCR   6303  FORTRAN 3.0B SUMMARY-REL   17A
   TREE     6690  FORTRAN 3.0B SUMMARY-REL   41A
   ENDLOC   6B95  FORTRAN 3.0B SUMMARY-REL   17F
IN

*K,Io
IN

*N,FURIA4,,,B
IN

*K,Io
IN

*P
   FTN      3DC4  COPYRIGHT CONTROL DATA CORP. 1972
   GUA      447A  FORTRAN 3.0B SUMMARY-REL   02F
   CFIVOC   44DE  FORTRAN 3.0B SUMMARY-REL   34A
   CKNAME   4539  FORTRAN 3.0B SUMMARY-REL   36A
   CNVT     4549  FORTRAN 3.0B SUMMARY-REL   01A
   CONV     4587  FORTRAN 3.0B SUMMARY-REL   03F
   DIAG     45BA  FORTRAN 3.0B SUMMARY-REL   04F
   EXF9     464A  FORTRAN 3.0B SUMMARY-REL   05F
   FLOAI    46F3  FORTRAN 3.0B SUMMARY-REL   06F
   GETC     483D  FORTRAN 3.0B SUMMARY-REL   14F
   GETF     4858  FORTRAN 3.0B SUMMARY-REL   04A
   GETSYM   4893  FORTRAN 3.0B SUMMARY-REL   07F
   GPOT     489C  FORTRAN 3.0B SUMMARY-REL   02A
   IGETCF   48C5  FORTRAN 3.0B SUMMARY-REL   15F
   IOERRA   48DE  FORTRAN 3.0B SUMMARY-REL   08F
   PACK     4EE6  FORTRAN 3.0B SUMMARY-REL   09F
   QBERMS   4F04  FORTRAN 3.0B SUMMARY-REL   10F
   RDLABL   4F25  FORTRAN 3.0B SUMMARY-REL   10A
   STORE    4FC3  FORTRAN 3.0B SUMMARY-REL   11F
```

```
   SYMBOL   4FF7  FORTRAN 3.0B SUMMARY-REL   03A
   ENDDO    5099  FORTRAN 3.0B SUMMARY-REL   29A
   GNST     51A0  FORTRAN 3.0B SUMMARY-REL   05A
   OPTION   5301  FORTRAN 3.0B SUMMARY-REL   16F
   OUTENT   53CF  FORTRAN 3.0B SUMMARY-REL   06A
   PHASEA   5403  FORTRAN 3.0B SUMMARY-REL   07A
   PLABEL   58FB  FORTRAN 3.0B SUMMARY-REL   08A
   STCHAR   594E  FORTRAN 3.0B SUMMARY-REL   11A
   TYPE     5980  FORTRAN 3.0B SUMMARY-REL   12A
   SAVELO   5B03  FORTRAN 3.0B SUMMARY-REL   13A
   LOCLAS   5C29  FORTRAN 3.0B SUMMARY-REL   24F
   DUMYAS   5CE7  FORTRAN 3.0B SUMMARY-REL   25F
   BDOPR    5D4F  FORTRAN 3.0B SUMMARY-REL   33A
   CKIVC    5E88  FORTRAN 3.0B SUMMARY-REL   35A
   IOSPR    5E9A  FORTRAN 3.0B SUMMARY-REL   37A
   ENDLOC   6545  FORTRAN 3.0B SUMMARY-REL   17F
IN

*K,I6
IN

*N,FORIAS,,,B
IN

*K,I6
IN

*P
   FTN      3DC4  COPYRIGHT CONTROL DATA CORP. 1972
   GOB      44D4  FORTRAN 3.0B SUMMARY-REL   26F
   CNVT     44EA  FORTRAN 3.0B SUMMARY-REL   01A
   DUMMY    4528  FORTRAN 3.0B SUMMARY-REL   01B
   FCMSTK   4639  FORTRAN 3.0B SUMMARY-REL   02B
   GETSYM   46C2  FORTRAN 3.0B SUMMARY-REL   07F
   IOPROB   46FA  FORTRAN 3.0B SUMMARY-REL   27F
   KCPART   4C85  FORTRAN 3.0B SUMMARY-REL   03B
   KDUIPT   4CB6  FORTRAN 3.0B SUMMARY-REL   04B
   KPCSTK   4CC8  FORTRAN 3.0B SUMMARY-REL   05B
   KPC3PR   5099  FORTRAN 3.0B SUMMARY-REL   06B
   KSYMGN   50B1  FORTRAN 3.0B SUMMARY-REL   07B
   LABKPC   50F9  FORTRAN 3.0B SUMMARY-REL   08B
   LABLER   5100  FORTRAN 3.0B SUMMARY-REL   09B
   PUNT     512A  FORTRAN 3.0B SUMMARY-REL   10B
   QBPRMS   5141  FORTRAN 3.0B SUMMARY-REL   10F
   STORC    5138  FORTRAN 3.0B SUMMARY-REL   11F
   SYMBOL   518F  FORTRAN 3.0B SUMMARY-REL   11B
   TSALOC   522C  FORTRAN 3.0B SUMMARY-REL   12B
   ARAYSZ   52B7  FORTRAN 3.0B SUMMARY-REL   42A
```

```
ASSEM   5322  FORTRAN 3.0B SUMMARY-REL  13B
BANANA  5389  FORTRAN 3.0B SUMMARY-REL  14B

BGINDO  544D  FORTRAN 3.0B SUMMARY-REL  15B
END     5536  FORTRAN 3.0B SUMMARY-REL  16B

ENTCOD  553F  FORTRAN 3.0B SUMMARY-REL  17B
HELEN   5648  FORTRAN 3.0B SUMMARY-REL  18B

INARST  57A1  FORTRAN 3.0B SUMMARY-REL  19B
NOFROC  57B5  FORTRAN 3.0B SUMMARY-REL  20B
PHASEB  57E6  FORTRAN 3.0B SUMMARY-REL  21B
READIR  5C3C  FORTRAN 3.0B SUMMARY-REL  22B

SUBFUN  5C74  FORTRAN 3.0B SUMMARY-REL  23B
SYMSCN  5CEC  FORTRAN 3.0B SUMMARY-REL  28A
```

```
ACF     5D19  FORTRAN 3.0B SUMMARY-REL  24B
AFIOL   6104  FORTRAN 3.0B SUMMARY-REL  25B
ASUPER  613E  FORTRAN 3.0B SUMMARY-REL  26B

CGOTO   6275  FORTRAN 3.0B SUMMARY-REL  27B
FINK    62D1  FORTRAN 3.0B SUMMARY-REL  28B

INTRAM  6396  FORTRAN 3.0B SUMMARY-REL  29B
PARTSB  655F  FORTRAN 3.0B SUMMARY-REL  30B
SUBPR1  660F  FORTRAN 3.0B SUMMARY-REL  31B
SUBPR2  6640  FORTRAN 3.0B SUMMARY-REL  32B
SUBPR3  66DA  FORTRAN 3.0B SUMMARY-REL  33B
ARITHR  6721  FORTRAN 3.0B SUMMARY-REL  34B
ENDLOC  68E5  FORTRAN 3.0B SUMMARY-REL  17F
IN
```

```
*K,I8
IN

*N,FORIO1,,,B
IN

*K,I8
IN

*P
FTN     30C4  COPYRIGHT CONTROL DATA CORP. 1972
GOC     498A  FORTRAN 3.0B SUMMARY-REL  28F
BKDWN   49DB  FORTRAN 3.0B SUMMARY-REL  01C
BLDUP   4A39  FORTRAN 3.0B SUMMARY-REL  02C
BSS     4A7C  FORTRAN 3.0B SUMMARY-REL  03C
CHKWD   4A9A  FORTRAN 3.0B SUMMARY-REL  04C
CHUP    4C1A  FORTRAN 3.0B SUMMARY-REL  05C
CL12    4E3A  FORTRAN 3.0B SUMMARY-REL  06C
CON     4F2F  FORTRAN 3.0B SUMMARY-REL  07C
COUNT   4F66  FORTRAN 3.0B SUMMARY-REL  08C
```

```
   DATAST  4F7E  FORTRAN 3.0B SUMMARY-REL  09C
   GETSTM  5040  FORTRAN 3.0B SUMMARY-REL  10C
   INOUT   50F1  FORTRAN 3.0B SUMMARY-REL  11C
   IOPRBC  5161  FORTRAN 3.0B SUMMARY-REL  29F
   IXOR1   600F  FORTRAN 3.0B SUMMARY-REL  12C
   LABEL   614B  FORTRAN 3.0B SUMMARY-REL  14C
   LABIN   616D  FORTRAN 3.0B SUMMARY-REL  15C
   PHASEC  61D4  FORTRAN 3.0B SUMMARY-REL  13C
   QBFRMS  6529  FORTRAN 3.0B SUMMARY-REL  10F
   QXCD    6543  FORTRAN 3.0B SUMMARY-REL  16C
   REED    65D3  FORTRAN 3.0B SUMMARY-REL  17C
   SKIP    6630  FORTRAN 3.0B SUMMARY-REL  18C
   SYMSCN  6696  FORTRAN 3.0B SUMMARY-REL  19C
   ENDLOC  66A3  FORTRAN 3.0B SUMMARY-REL  17F
IN

*K,Ia
IN

*N,FORIC1,,,B
IN

*K,Io
IN

*P


   FTN     3DC4  COPYRIGHT CONTROL DATA CORP. 1972
   GOOD    49BA  FORTRAN 3.0B SUMMARY-REL  30F
   AMOUT   49DD  FORTRAN 3.0B SUMMARY-REL  01D
   ADMAX   4FC5  FORTRAN 3.0B SUMMARY-REL  02D
   BEGINO  51C7  FORTRAN 3.0B SUMMARY-REL  03D
   BKDWN   530F  FORTRAN 3.0B SUMMARY-REL  04D
   COUNT   537A  FORTRAN 3.0B SUMMARY-REL  05D
   FINISH  5392  FORTRAN 3.0B SUMMARY-REL  06D
   GETSYM  553B  FORTRAN 3.0B SUMMARY-REL  10C
   IACON   550F  FORTRAN 3.0B SUMMARY-REL  07D
   IHCON   563B  FORTRAN 3.0B SUMMARY-REL  08D
   INDEX   5666  FORTRAN 3.0B SUMMARY-REL  09D
   IOPRBD  5683  FORTRAN 3.0B SUMMARY-REL  31F
   LABOUT  5C31  FORTRAN 3.0B SUMMARY-REL  10D
   NP2OUT  5D11  FORTRAN 3.0B SUMMARY-REL  11D
   NPUNCH  5D40  FORTRAN 3.0B SUMMARY-REL  12D
   NWRITE  5E81  FORTRAN 3.0B SUMMARY-REL  13D
   PACK    5EBC  FORTRAN 3.0B SUMMARY-REL  09F
   PHASE6  5EE1  FORTRAN 3.0B SUMMARY-REL  14D
   QBFRMS  5F7E  FORTRAN 3.0B SUMMARY-REL  10F
```

7.7.4 Phase 4 Labeled Common Block

1. INDTAB — preset table of constants. Those indicator words (see output description) which are used several times are preset in this table for easy reference.

2. INFTB — Intrinsic function table. A preset loopup table for determining whether a function is intrinsic (see HELEN).

3. NBSS, NADC — constants preset to the instruction code values: i.e., NBSS = 1, NINA = 31, NTRA@ = 62, etc.

4. INFTBL — INFTB table entry length.

5. INFTBN — INFTBN table length.

6. INFTBX — index to INFTB table (not preset).

7. KODNAM — part of INFTB entry containing function name as coded.

9. KFTYPE — part of INFTB entry containing function type:

   1 = integer
   2 = real
   3 = double precision

10. NLINE — part of INFTB entry containing in-line/non-in-line flag.

11. NPMTRS — part of INFTB entry containing number of parameters of function.

12. KSPTAB — preset table of special pointers to sybmol table; e.g., 0 constant, 1.0 constant, and pointers to implied subroutines.

```
  ENDLOC  62E4  FORTRAN 3.0B SUMMARY-REL  17F
IN

*K,10
IN

*N,FURIE1,,,B
IN

*K,10
IN

*L,FIN
IN

*K,P3
IN

*V,11
IN
```

15.	LTYP	working cell of BGINDO and BANANA
16.	MBEGIN	working cell of PHASE 4 holds label of first address of statement function
17.	IADDIT	parameter to INTRAM (additive)
19.	INTRAX	INTRAS table index
20.	IPTR	parameter to INTRAM (operand pointer)
21.	ISBSCP	parameter to INTRAM (subscript pointer)
22.	JXX	index to input buffer set by READIR
23.	KBEGIN	working cell of PHASE 4 holds label of first address in program
24.	KBIAS	hold label of cell containing program base (PHASE4)
25.	KBYTX	byte index for constructing floating point calls (FCMSTK)
26.	KENTER	working cell of PHASE 4
27.	KENTRY	working cell of PHASE 4
28.	KEXEC	working cell of PHASE 4
29.	KEXTYP	set by ACP as flag to give expression type of tree just processed
30.	KFCETX	used for KPCSTK
31.	KFCSW	floating calling sequence switch (FCMSTK)
32.	KFFSAV	contains label of cell used to save index FF
33.	KFINDX	index used in building floating calling sequences (FCMSTK)
34.	KFLAM	current floating addressing mode (FCMSTK)
35.	KINTYP	instruction type (parameter to INTRAM)

36.	KLLTBX	KLLTB table index
37.	KNTROL	control parameter to INTRAM
38.	KOBX	output buffer index
39.	KPRNAM	program name. Working cell of PHASE 4
40.	K@SAV	contains label of cell used to save @ register
41.	KRETRN	contains label of first cell of RETURN code sequence
42.	KRTNS	set in PHASEB
45.	KSFNAM	statement function name holder
47.	KSTYP	set in PHASEB
48.	KTCATX	KTCAT index
49.	LEVEL	index to level tables for ACP
50.	LIFTX	index to level tables for LOGLIF
51.	LOGIF	logical if switch (LOGLIF)
52.	IDMM	Holder for local variables equivalenced to common
53.	IDPFLG	Indicates location of 'current' dp value. 0 - dp value is in dp pseudo-accumulator 1 - dp value is in locations C5-C7

7.7.6 Phase C, D, E Labelled Common Block

NPCBF	binary output buffer
IDVTAB	table of pointers to places in program where addresses of variables may be found
NPTBF	print image
IXS1 ISX2	index history of program being compiled

7.7.7 Phase C and D/E Blank Common Block (some array dimensions differ in phase D/E)

INBUFF	input buffer
NOBUFF	intermediate output buffer
NRCD	intermediate record holder
NFTAB	future table
IV	table of temporary holders
NSUMD	maximum length of IDVTAB
NDFL	modify, don't modify addressing switch
NADX	IDVTAB index
NSUMVS	maximum size of future table
IXC	current index
IXFLAG	effect of current command on index
IXF	current contents of $FF
IXQ	current contents of Q
IXSX	index history tables index
IUSE	$FF has been used switch
IFIX	point to which addressing is fixed
NOUTOT	number of words in intermediate output buffer (circular in D/E)
NOWO	number of words in current imtermediate output record (-2)
NL	ignore record switch
NT	record is label switch
NA	record has additive switch
NS	record has subscript switch
ND	don't modify addressing switch
NR	2 word instruction switch
NQ	command uses Q register switch
NF	ADC type indicator
NTYPE	command type - second phase
NOWI	number of words in input record (-2)
NOPC	opcode
NOPT	address pointer
NADD	additive pointer
NXPT	variable subscript pointer
NATYPE	command type - first phase
ICT	number of words command occupies
INCT	input buffer index; in phase D/E circular buffer index
NOCT	output buffer index; in phase D/E, next available circular buffer location

NSTYPE	first executable instruction encountered switch
LRS	long right shift 16 needed indicator
NOBW	no. words in binary output, this record
NBINC	current index into binary output image
NPC1	
NSETX	top of list output buffer
NRM	relocation of current command
NPTBFX	list output buffer
NRC1	relocation - words 1, 2 of current command
NRC2	
NW2FL	processing second half of 2 word command
NW2	second binary word of 2 word command
NBFL	last word flag - binary output
NXCON	constant value - used by SETPRT
NASCI	pointer for address field - used by SETPRT
ISCOUN	address counter - second phase
ICOUNT	address counter - both phases
NOFS	future table index
NODS	NDSTAB index
NDXJ	temporary holder for ICOUNT
NX	INBUFF, NOBUFF index
IHEAD	'top' of circular buffer NOBUFF
INOB	size of NOBUFF
INC	NOBUFF subscript (not 'circular')
NDSTAB	table of references to dummy parameters
IXS3	Index history of program being compiled

7.7.8 Compressed Symbol Table Blank Common Block (phase D 2.0A only)

ICSYFL	Length of compressed symbol table entry
ICSYNS	Number of sectors needed for compressed symbol table page
ICSYMP	Compressed symbol table page indicator
ICSYPC	Current compressed symbol table page
ICSYPS	Compressed Symbol table page size
CSYMTB	Compressed Symbol Table

CHAPTER 8

TABLE OF CONTENTS

8.0 OBJECT TIME ARITHMETIC ROUTINES

8.1 Introduction

This chapter deals with the following object time routines:

External Function	Symbolic Name
Trigonometric sine	SIN
Trigonometric cosine	COS
Hyperbolic tangent	TANH
Exponential	EXP
Natural logorithm	ALOG
Arctangent	ATAN
Square Root	SQRT

The above routines are designed to operate under the 1700 Mass Storage Fortran. They are written in 1700 Assembly Language for optimization of code and time of execution. However, they can be used by any other program working under the 1700 Operating System as long as the proper calling sequence is generated, and cells $C5 and $E5 are reserved for the floating point package and temporary storage.

8.2 General Description

The argument to any of the functions is a real value number written in the 1700 Mass Storage Fortran floating point notation. The result is also a real value number in floating point notation.

8.2.1 Temporary Storage

These routines use the unprotected communications area from $D8 to $E2 as temporary storage area.

8.2.2 Calling Sequence

Each object time routine is entered via an RTJ instruction to its symbolic name, followed by a cell containing the address of core where the value of the argument can be found in floating point notation.

8.2.3 Returning Sequence

Each object time routine will return control to the user at the program location immediately following the parameter address with the calculated value of the function in the floating point pseudo-accumulator.

8.3 Function Description

The mathematical method used to approximate the functions will be found in Appendix A of this document. Basically it is a truncated Taylor-Mclaurin series with Chebyshev expansion for the higher terms of the series.

8.3.1 Trigonometric Sine (SIN)

8.3.1.1 Range Reduction

Let Z be the value of the argument.

If $/Z/ > 2^{21}$ then the answer will be assumed zero.

If:

$Z < 0$ let $U = -Z$ and $\sin(Z) = -\sin(U)$
$Z \geq 0$ let $U = Z$ and $\sin(Z) = \sin(U)$

Let I be integral part of $U/2\pi$ and let

$Y = U - I \cdot 2\pi$, then $Y \leq 2\pi$, and $\sin(U) = \sin(Y)$

If:

$Y > \pi$, let $T = Y - \pi$ then $\sin(Y) = -\sin(T)$
$Y \leq \pi$, let $T = Y$ then $\sin(Y) = \sin(T)$

If:

$T > \pi/2$, let $X = \pi - T$
$T \leq \pi/2$, let $X = T$
and $\sin(T) = \sin(X)$, with $X \leq \pi/2$

8.3.1.2 Series for SIN(X) with $X \leq \pi/2$

$$SIN(X) = C_0X + C_1X^3 + C_2X^5 + C_3X^7 + C_4X^9$$

with $C_0 = 1.0$
$C_1 = -.16666647$
$C_2 = .83328836E-3$
$C_3 = -.19799327E-3$

8.3.2 Trigonometric Cosine (COS)

Since COS(Z) = SIN(Z + $\pi/2$) then object time routine COS will simply be a second entry to the SIN routine.

8.3.2.1 Temporary Storage for SIN and COS

Eight cells of temporary storage are required by either SIN and COS. They are:

PARADD	for parameter address $DE
RETADD	for return address $D9
FLAG	for sign indicator $D8
X and X_2	for temporary floating point numbers, $DA to $DD
QS	for saving Q $E2

8.3.3 Hyperbolic Tangent (TANH)

8.3.3.1 If $0 < |X| < \frac{1}{4}$,

TANH(X) = X + A_1*X**3 + A_2*X**5 + A_3*X**7 + A_4*X**9

where

A_1 = -0.33333227
A_2 = 0.13337246
A_3 = -0.053388615
A_4 = 0.025628253

If $\frac{1}{4} \leq |X| < 10$

TANH(X) = 1.0 - 2.0/(EXP(2.0*X)+1.0).

If $|X| \geq 10$, TANH(X) = ± 1.0

If X = 0, TANH(X) = 0

8.3.3.2 Temporary Storage for TANH

Eight cells of temporary storage are required.

They are:

QS	for saving Q $E2
J	for indexing in series 1000 $DF
X, XS, T1	for floating point numbers, two cells each

$DD, $DA, $D8 3

8.3.4 Arctangent (ATAN)

8.3.4.1 Range Reduction

Let Z be value of input argument.

If:

Z < 0, let U = - Z, and arctan(Z) = - arctan(U)
Z ≥ 0, let U = Z, and arctan(Z) = arctan(U)

If:

U ≤ 1, let Y = U, A = 0, and B = 1
U > 1, let Y = 1/U, A = π/2, and B = -1

If:

Y ≤ tan (π/8) let T = π/16
Y > tan (π/8) let T = 3π/16

and arctan (U) = A + B (T + arctan(X))

where $X = \frac{Y - \tan(T)}{1 + Y \tan(T)}$

8.3.4.2 Series for arctan(X), $X \leq \tan(\pi/16)$

$$\arctan(X) = C_0 X + C_1 X^3 + C_2 X^5$$

with $C_0 = .99999900$
$C_1 = .33313333$
$C_2 = .19000000$

8.3.4.3 Temporary Storage for ATAN

Eleven cells of temporary storage are required.

They are:

PARADD	for parameter address	$E2
ARCRET	for return address	$D9
ARCFLG	for path decision	$D8
X, X_2, AF, BF	for floating point numbers,	$DA to $E1

8.3.5 Exponential (EXP)

8.3.5.1 Range Reduction

Let Z be input value of argument

If:

$Z < 0$ let $Y = -Z$, and $e^z = 1/e^y$

$Z > 0$ let $Y = Z$, and $e^z = e^y$

$Z = 0$ put answer equal to 1 and exit.

Now $e^y = 2^N e^x$ if X and N are defined as follows:

Let $T = Y/\ln 2$
$N = T + \frac{1}{2}$ take integral part
$W = T - N$
$X = W \ln 2$

Therefore the maximum absolute value for Y is equal to 88.02968 which will generate the maximum floating point number. If input value is greater than this maximum value then a message will be typed out and the answer is given as the maximum floating number permissible in 1700 Fortran.

8.3.5.2 Series for e^x for $X \leq \ln 2/2$

Series is normal Taylor-Mclaurin series truncated at n = 7; i.e.,

$$e^X = \sum_{n=0}^{6} \frac{X^n}{n!}$$

8.3.5.3 Temporary Storage for EXP

seven cells of temporary storage are required. They are:

PARADD	for parameter address $E0
RETEXP	for return address $DA
FLAG,N	for indicator $DE, $DC
Y	for floating point number, $D8, $D9
QS	for saving Q $E1

8.3.6 Natural Logarithm (ALOG)

8.3.6.1 Range Reduction

Let Z be input value of argument.

If $Z \leq 0$ then the answer will be the maximum floating point number possible in 1700.

Now $Z = 2^N Y$

and $\ln(Z) = (N - \frac{1}{2}) \ln 2 + \ln (\frac{1 + X}{1 - X})$

where $X = \frac{Y - \sqrt{2}/2}{Y + \sqrt{2}/2}$

Thus $/X/ \leq 3 - 2\sqrt{2}$

8.3.6.2 Series for $\ln (\frac{1 + X}{1 - X})$ for $/X/ \leq 3 - 2\sqrt{2}$

$$\ln (\frac{1 + X}{1 - X}) = C_0 X + C_1 X^3 + C_2 X^5$$

with
$C_0 = 2.0312500$
$C_1 = 0.41666667$
$C_2 = 0.90000000$

8.3.6.3 Temporary Storage for ALOG

Nine cells of temporary storage are required.

They are:

PARADD	for parameter address $E0
LNRET	for return address $D8
N,X,X2	for floating point numbers $D9 to $DE
QS	for saving Q $E1

8.3.7 Square Root (SQRT)

8.3.7.1 Range Reduction

Let Z be input value of argument

If $Z < 0$ let $Y = -Z$ and $\sqrt{Z} = -\sqrt{Y}$

also type out error message.

If $Z \geq 0$ let $Y = Z$

Now $Y = 2^{2N} X$ and $\sqrt{Y} = 2^{N}\sqrt{X}$ where $\frac{1}{4} \leq X < 1$

The Padé approximation for a first estimate with a maximum relative error < 2.3 (E-4) is

$$y_i = \frac{25}{7} - \left[\frac{\frac{5000}{343} \quad (X + \frac{15}{49})}{(X + \frac{15}{49}) \quad (X + \frac{235}{49}) - \frac{400}{2401}} \right]$$

8.3.7.2 Iterative method

The square root is computed via a Newton-Raphson iteration starting with a first approximation, y_i above. Successive approximations are found from:

$$y_{i+1} = \frac{1}{2} (y_i + \frac{X}{y_i})$$

$$\text{until} \left| y_{i+1} = y_i \right| \leq 2^{-23}$$

8.3.7.3 Temporary Storage for SQRT

Ten cells of temporary storage are required:

They are:

PARADD	for parameter address $E0
QS	for saving Q $E2
SQRET	for return address $DA
SQEXP	for value of N $DC
SQFLG	for negative input $DB
X, Y0, Y1	for floating point numbers, $DD to $E0 and $D8,$D9

8.4 Type Out Message

Any illegal input value will cause a predetermined value to be given as an answer and the program will exit normally. It is left to the user to take corrective action.

8.4.1

Routine	Argument	Answer
SIN or COS	$\|Z\| > 2^{21}$	0
EXP	$\|Z\| > 88.02968$	∞
ALOG	$Z \leq 0$	∞
SQRT	$Z < 0$	$-\sqrt{\|Z\|}$

8.5 References:

CONTROL DATA 6600 Computer System, Programming System/Library Functions, A Study of Mathematical Approximations, Pub. No. 60114500.

HANDBOOK OF MATHEMATICAL FUNCTIONS, National Bureau of Standards Applied Mathematics Series-55, Library of Congress catalog number 64-60036.

8.3.8 Exponentiation {**} Subroutine Q8EXPN

This subroutine is called by programs which use the exponentiation operator, i.e. the symbol **. This routine is written in 1700 Assembly Language. It has the following entry points:

Q8QF2I	floating number to integer power
Q8QI2F	integer number to integer power
Q8QF2F	floating number to floating power
RETAD	
QSAVE	

The following externals are declared:

FLOT
ALOG
EXP

The calls required for temporary storage are:

FLOFLG	for floating/fixed flag $DC
FRESLT	for intermediate results {floating} $DD to $DE
SIGN	for sign of exponent $DF
COEFF	for address of coefficient $E1
EXPO	for address or val ue of exponent $E2
XLOGC	holds intermediate results $E1
MLTPR	holds powers of coefficient $E3 {floating point number — 2 words} $E4-L & G saved
IRESLT	for intermediate results {fixed} $E5
FLOVEL	for floating point package overflow $C8
FLACC	$C5

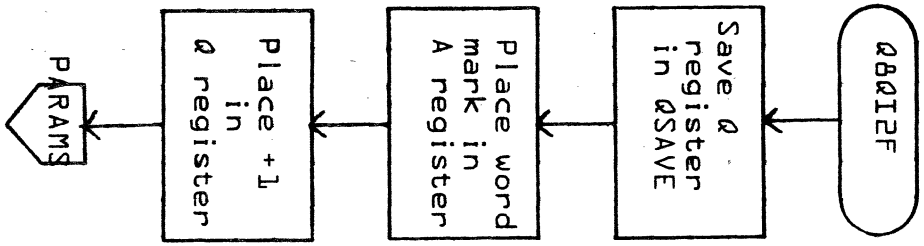
A

B

C

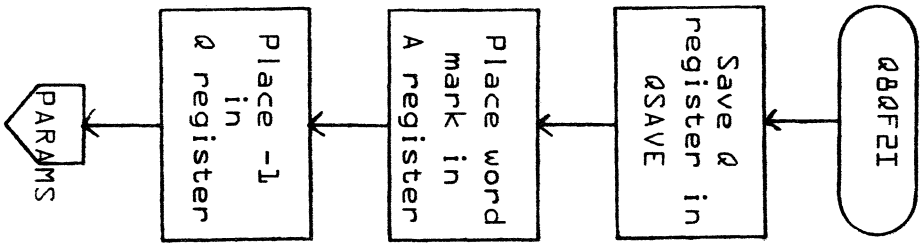
D

Integer to integer



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8EXPN	PAGE 1 OF 18	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

Floating to integer

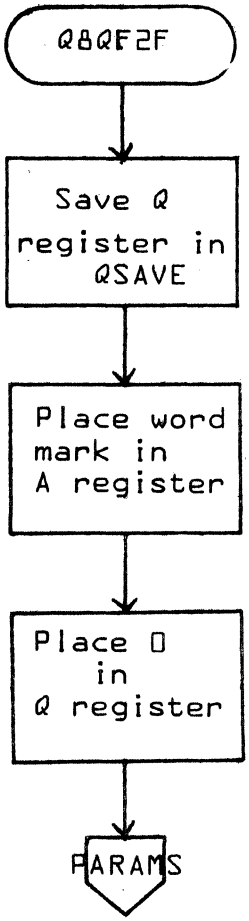


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8EXPN	PAGE 2 OF 18	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

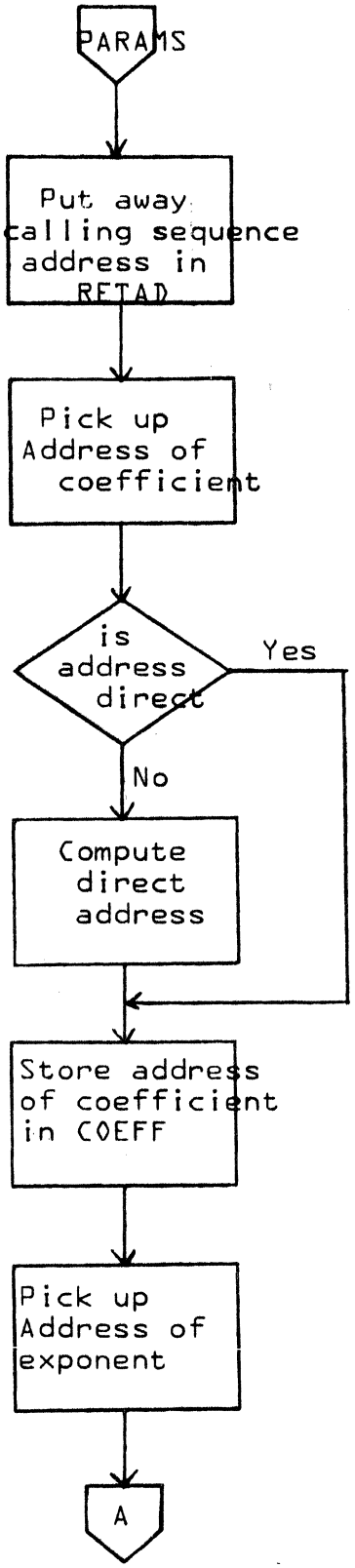
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

Floating to floating



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐☐☐☐	DOCUMENT TITLE: Q8EXPN	PAGE 3 OF 18	PROJECT MGR.			
FLOWCHART			PROJECT NAME			
DECISION TABLE	NUMBER	ISSUE DATE	TASK NO.			
OTHER	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	QBEXPN	PAGE 4 OF 18	
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

PROJECT NO.		REV	APPROVED
PROJECT MGR.			
PROJECT NAME			
TASK NO.			
TASK NAME			

DATE

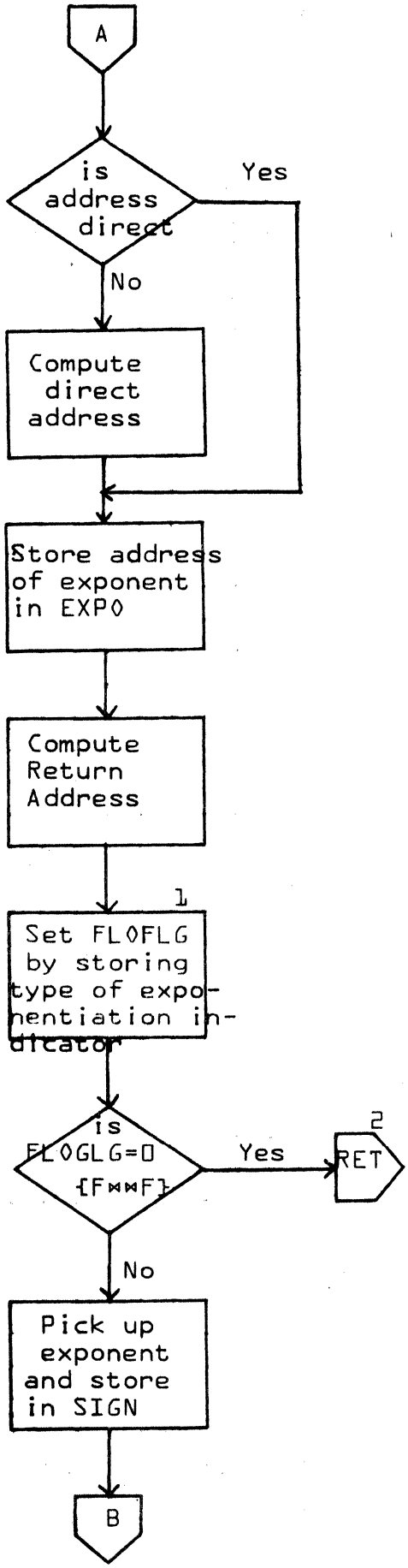
PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

1. FLOFLG equals

   -1 F⋈⋈I
   0 F⋈⋈F
   +1 I⋈⋈I

2. From this place a jump is made to LOGEX.



REV	APPROVED	DATE

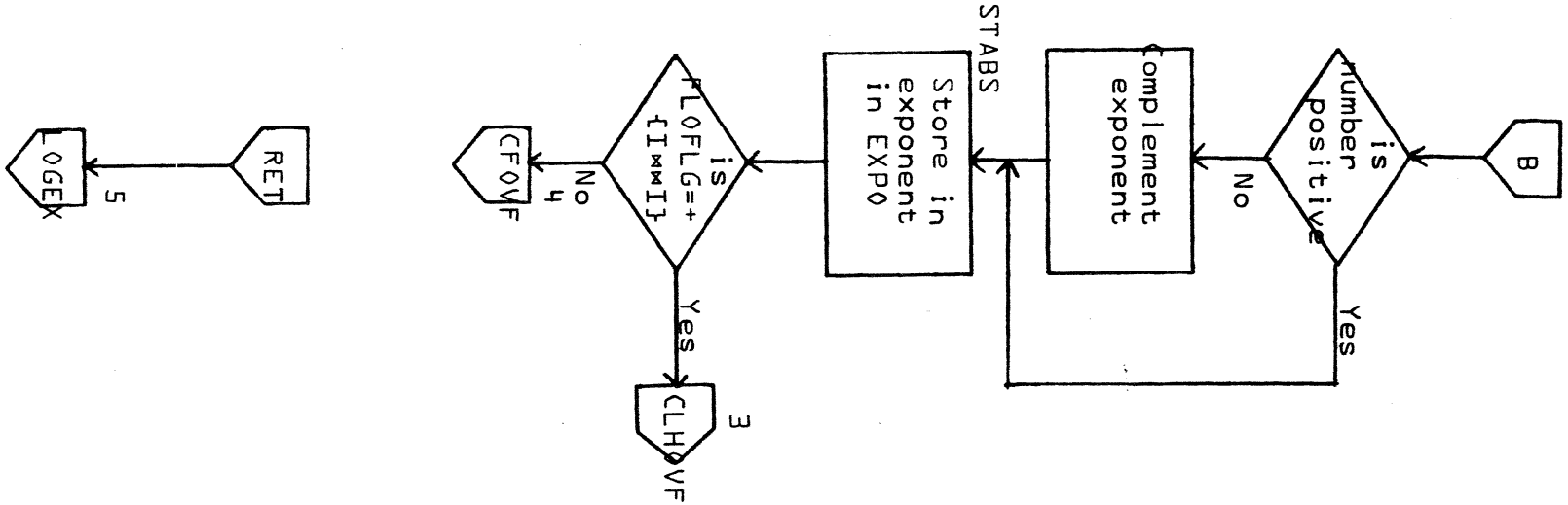
PROJECT NO.		DOCUMENT CLASS	IMS
PROJECT MGR.		DOCUMENT TITLE	QBEXPN
PROJECT NAME		NUMBER	PAGE 5 OF 18
TASK NO.		ISSUE DATE	
TASK NAME		DRAWN BY	DATE

MACH. TYPE 1700

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

3. Process I##I
4. Process F##I
5. Process F##F



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8EXPN	PAGE 6 OF 18	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

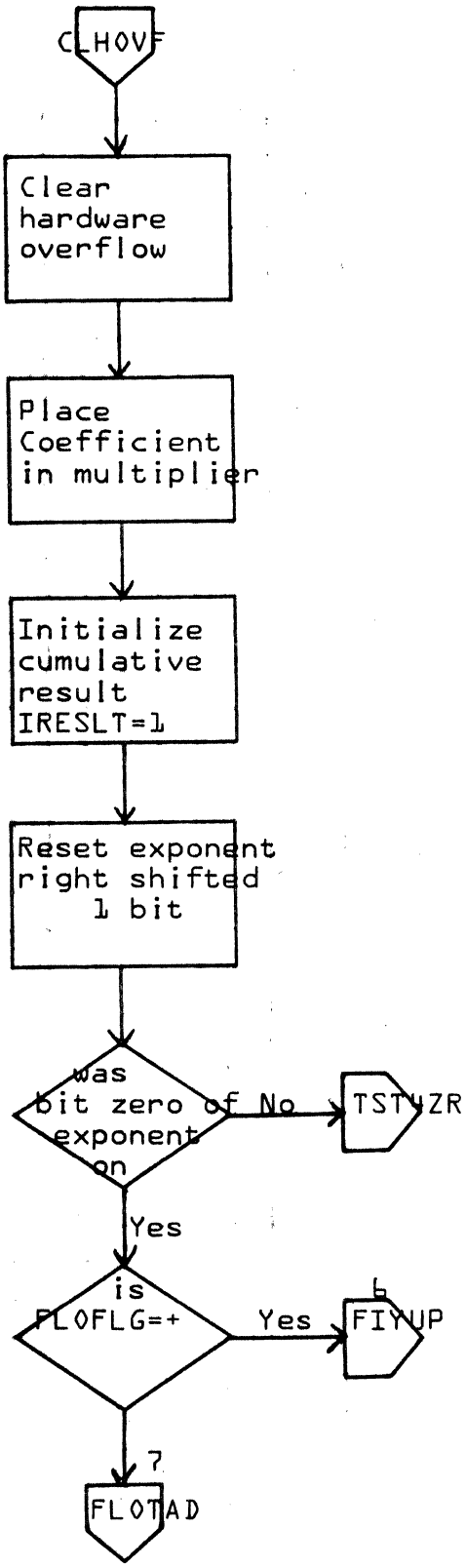
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

Process integer to integer

6. Select I⋈⋈I

7. Select F⋈⋈I

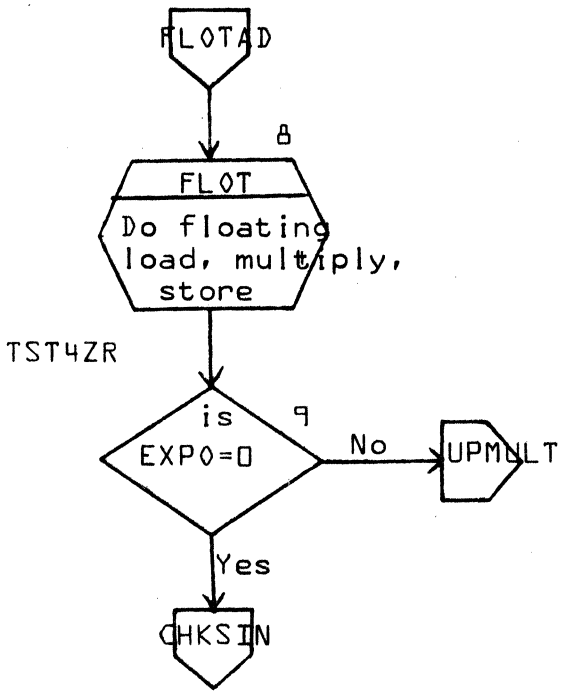


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS	IMS	MACH. TYPE	1700	PROJECT NO.
SAMPLE CODE ☐☐☐☐	DOCUMENT TITLE	Q8EXPN	PAGE	7 OF 18	PROJECT MGR.
FLOWCHART	NUMBER		ISSUE DATE		PROJECT NAME
DECISION TABLE	DRAWN BY		DATE		TASK NO.
OTHER					TASK NAME

REV	APPROVED	DATE

8. Floating
   FRESLT =
   FRESLT*MLTPR

9. Test exponent to see
   if all significant
   bits have been
   shifted out.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	Q8EXPN	PAGE 8 OF	18
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

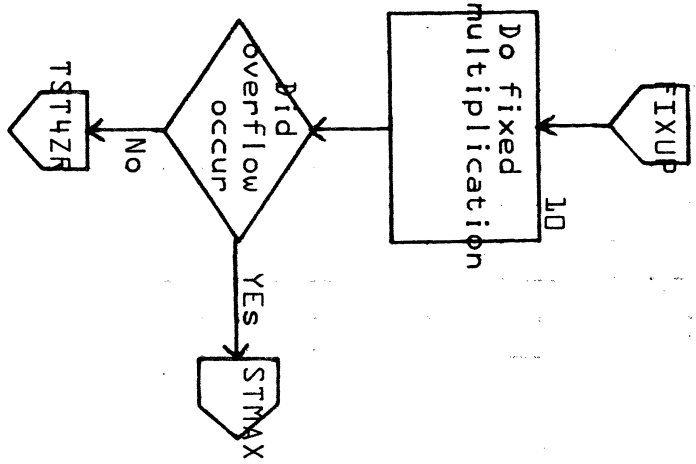
PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

A B C D

A B C D

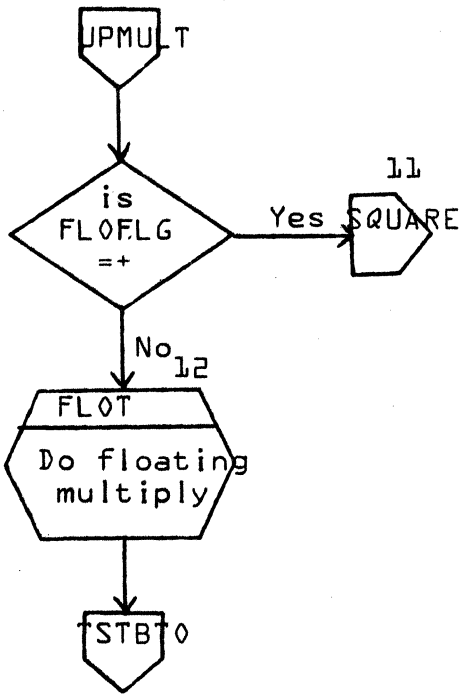
10. Fixed
IRESLT
=IRESLT*MLTPR



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	Q8EXPN	PAGE 9 OF 18	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

11. Processing
I⋈⋈I

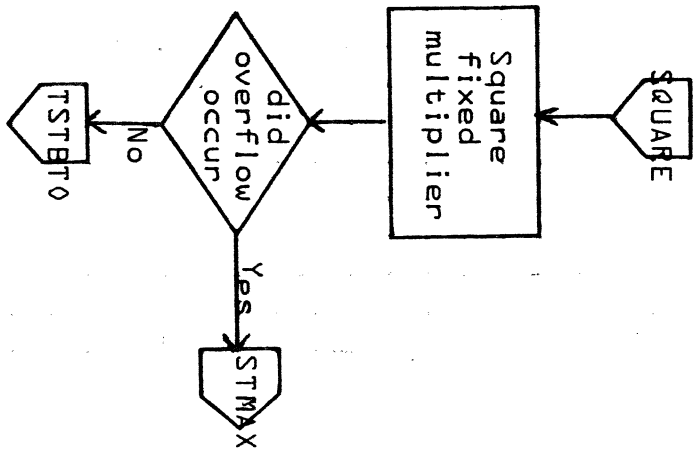
12. Square floating multiplier
MLTPR
=MLTPR⋈MLTPR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE: Q8EXPN	PAGE 20 OF 18	PROJECT MGR.			
	NUMBER	ISSUE DATE	PROJECT NAME			
	DRAWN BY	DATE	TASK NO.			
			TASK NAME			

PRINTED IN U.S.A.

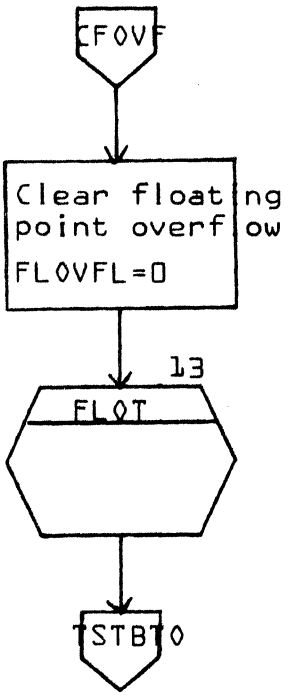
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	Q8EXPN	PAGE 11 OF 18	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

Process floating to integer

13. Place coefficient in multiplier and initialize cumulative result.
MLTPR=COEFF
FRESLT=1.0



REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: QBEXPN
PAGE 13 OF 18
NUMBER
ISSUE DATE
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

PRINTED IN U.S.A.

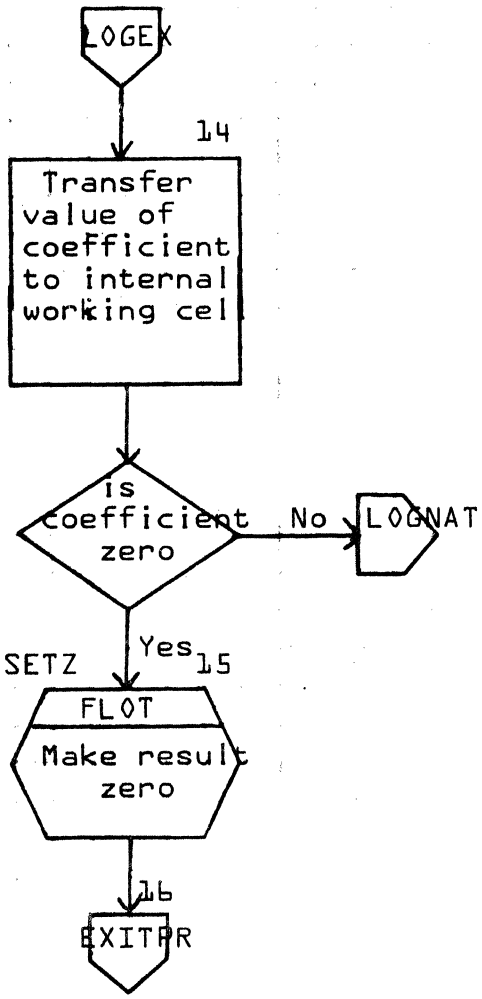
AA1385 (FORMERLY CA127-1)

Process floating to floating

14. First word into FRESLT; Second word into FRESLT+1.

15. Make result zero to avoid a fault in the natural log routine.

16. Exit



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

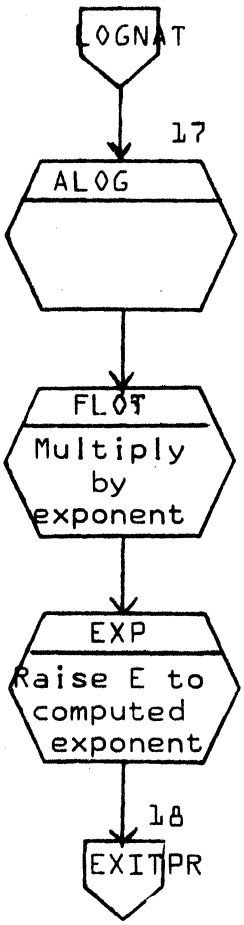
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: QBEXPN
PAGE 13 OF 18
NUMBER
ISSUE DATE
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV | APPROVED | DATE

17. Take natural log of coefficient.

18. Exit



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE: Q8EXPN	PAGE 14 OF 18	PROJECT MGR.			
			PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

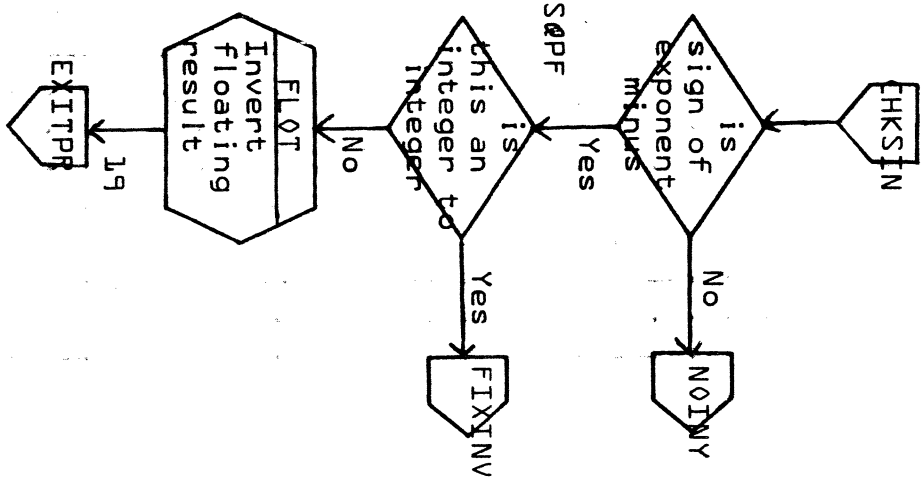
A B C D

A

B

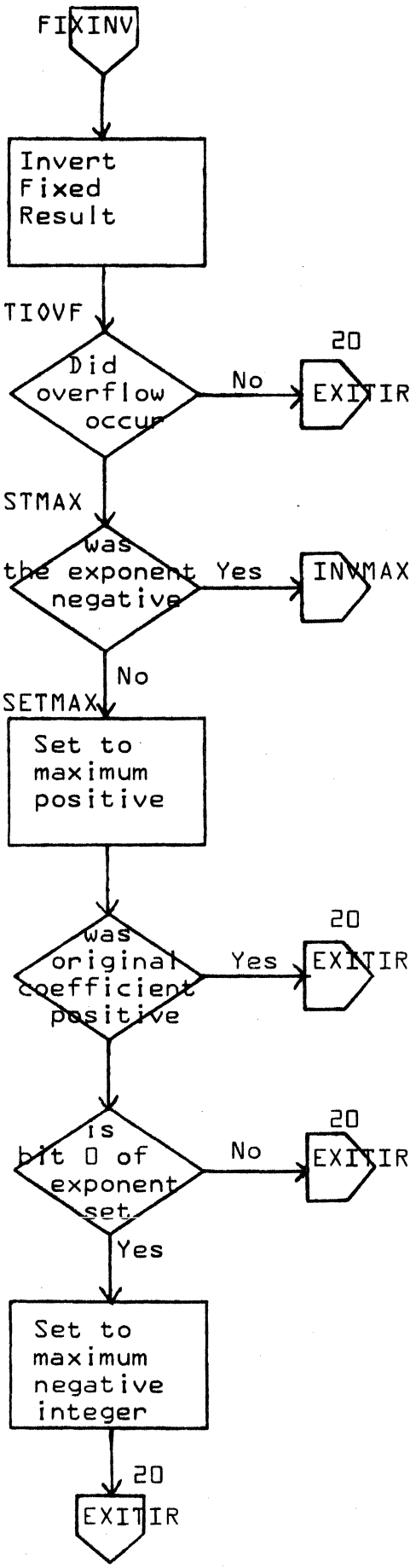
C

D



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐					
DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE		PROJECT MGR.			
Q8EXPN	PAGE 15 OF 18	PROJECT NAME			
NUMBER	ISSUE DATE	TASK NO.			
DRAWN BY	DATE	TASK NAME			

20. Exit



REV	APPROVED	DATE

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	
NUMBER	60EXPN
PAGE 16 OF 18	
ISSUE DATE	
DRAWN BY	
DATE	
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

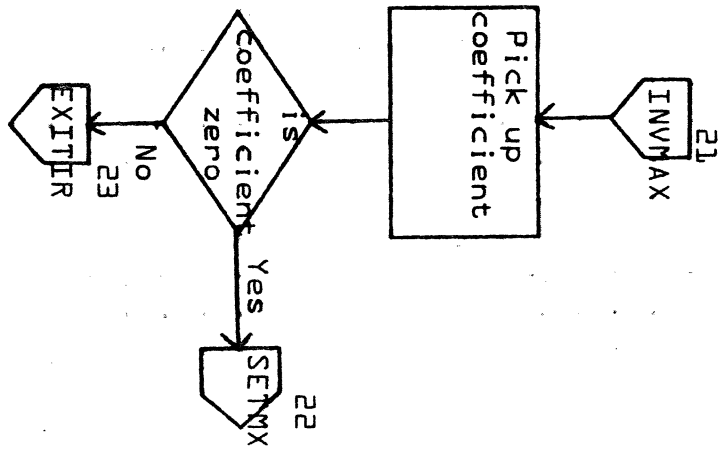
PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

21. Overflow with negative exponent, return zero.

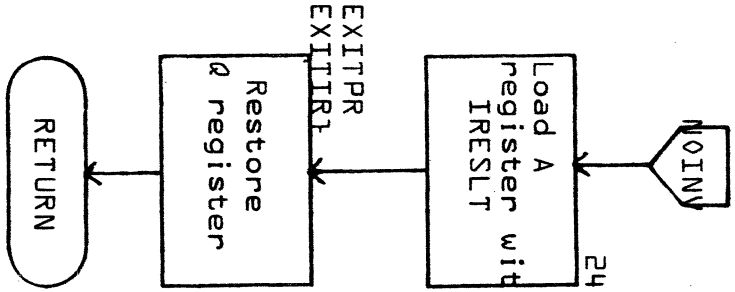
22. Set to maximum value.

23. Exit.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	Q8EXPN	PAGE 176 OF 18	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

24. Returns garbage in A register if F2I.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Q8EXPN PAGE 18 OF 18	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

APPENDIX A

1700 Mathematical Approximations

1.0 Introduction

This paper will derive approximations for object time 1700 Fortran programs which will allow fast evaluations with necessary accuracy and uncomplicated range reduction.

1.1 Accuracy

Since all approximations are to be in floating-point, it shall be required that

$$R < 2^{-23} = 1.192E\text{-}07$$

which is the basic round off due to the size of significant digits in 1700.

2.0 General Description of Method

The method will combine truncated Taylor-Mclaurin Power Series and Chebyshev Polynomials.

2.1 Truncated Taylor-Mclaurin Power Series

A function will be expressed as a truncated Taylor-Mclaurin series such that the error produced is less than R. i.e.,

$$f(N) = \sum_{n=0}^{m} \frac{f^{(n)}(0)}{n!} x^n, \quad /n/ \leqslant a$$

and m is chosen such that

$$R < \left| \frac{f^{(m+1)}}{(m+1)!} x^{m+1} \right| = R_T$$

2.2 Chebyshev Expansions

Denote by $Tn(x) = \cos(nt)$ the Chebyshev polynomial of degree n in $x = \cos t$. Note that $/x/ \leqslant 1$ and that $/Tn(x)/ \leqslant 1$. Writing

$$Tn(x) = \sum_{i=0}^{n} C_n^i \, x^i$$

the coefficients C_n^i are computed from

$$C_n^i = 0 \text{ if } (i + n) \text{ is odd}$$

$$C_n^i = 2^{i-1}\left[2\binom{(n+i)/2}{(n-i)/2} - \binom{(n+i-2)/2}{(n-i)/2}\right](-1)^{\frac{n-i}{2}}$$

if $(i + n)$ is even.

Thus $x^n = Tn(x) - \sum_{i=0}^{n-1}\left(C_n^i \, x^i\right) \cdot \frac{1}{2^{n-1}}$

Table of C_n^i is given at end of paper.

2.3 Use of Chebyshev Expansion

$$f(x) = \sum_{n=0}^{m-1} \frac{f^{(n)} x^n}{n!} + \frac{f^{(m)} x^m}{m!}$$

$$= \sum_{n=0}^{m-1} \frac{f^{(n)} x^n}{n!} + \frac{Tm(x)}{2^{m-1}} - \sum_{i=0}^{m-1} \frac{C_m^i \, x^i}{2^{m-1}} \cdot \frac{f^{(m)}}{m!}$$

If: $R \leq \left|\frac{Tm(x) \quad f^{(m)}}{m! \; 2^{m-1}}\right| + R_T$

then term involving $Tm(x)$ can also be neglected and series now becomes

$$f(x) = \sum_{n=0}^{m-1}\left(\frac{f^{(n)}}{n!} - \frac{f^{(m)}}{m! \; 2^{m-1}} \cdot C_m^n\right)x^n$$

This process can then be repeated as long as the total error does not exceed R.

3.0 SIN (X) for $X < \pi/2$

The Taylor-Mclaurin series truncated at n = 6 is

$$\text{SIN}(X) = \sum_{n=0}^{5} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

with Relative Error at $x = \pi/2 \leq \left| \frac{(\pi/2)^{13}}{13!} \right| = 5.692\text{E-}8$

Let $x = (\pi/2)\, y$, with $y \leq 1$

$$\text{SIN}(X) = \sum_{n=0}^{4} \frac{(-1)^n}{(2n+1)!} y^{2n+1} (\pi/2)^{2n+1} - (\pi/2) \frac{11_y 11}{11!}$$

Now $y^{11} = \frac{T_{11(y)}}{2^{10}} - \frac{C_{11}^{2n+1}}{2^{10}} y^{2n+1}$ $\quad n=0, 1, \ldots, 4$

Error occasioned in omitting T(y) term is

$$Rc \leq (\pi/2)^{11} \left| \frac{T_{11(y)}}{11!\ 2^{10}} \right| = 3.511\text{E-}9$$

Total error so far is 6.043E-8 and series becomes

$$\text{SIN}(X) = \sum_{n=0}^{4} \left[\frac{(-1)^n}{(2n+1)!} \cdot (\pi/2)^{2n+1} + (\pi/2)^{11} \cdot \frac{C11^{2n+1}}{11!\ 2^{10}} \right] \cdot y^{2n+1} \quad (1)$$

$$\text{or} \quad \text{SIN}(X) = \sum_{n=0}^{3} \left[\frac{(-1)^n}{(2n+1)!} \cdot (\pi/2)^{2n+1} + (\pi/2)^{11} \cdot \frac{C11^{2n+1}}{11!2^{10}} \right] \cdot y^{2n+1}$$

$$+ \left(1/9!\ (\pi/2)^9 + (\pi/2)^{11} \cdot \frac{C_{11}^{9}}{11!2^{10}} \right) \cdot y^9$$

$$y^9 = \frac{T_{9(y)}}{2^8} - \frac{C_9^{2n+1}}{2^8} \cdot y^{2n+1} \quad , n = 0,1,2,3$$

Error caused in omitting $T_{9(y)}$ is 6.6E-7 and therefore cannot be neglected.

Thus using (1) and $y = (2/\pi)X$ we get

$$SIN(X) = \sum_{n=0}^{4} \left(\frac{(-1)^n}{(2n+1)!} + \frac{C_{11}^{2n+1}}{11!\ 2^{10}} (\pi/2)^{10-2n} \right) \cdot x^{2n+1}$$

Therefore: $$SIN(X) = \sum_{n=0}^{4} A_n X^{2n+1}$$

Define $B = (\pi/4)^2$ then

$$A_0 = 1 - \frac{B^5}{10!} = 0.99999998$$

$$A_1 = -1/3! + \frac{5B^4}{10!} = -.166666468$$

$$A_2 = 1/5! - \frac{7B^3}{10!} = .83328836E-2$$

$$A_3 = -1/7! + \frac{4B^2}{10!} = -.19799328E-3$$

$$A_4 = 1/9! - \frac{B}{10!} = .25857445E-5$$

4.0 TANH(X) for $x \leqslant \pi/8$

Taylor-Mclaurin series for TANH(X) is

$$TANH(X) = \sum_{n=1}^{\infty} \frac{(2)^{2n}\,(2^{2n}-1)}{(2n)!} \cdot B_{2n}(0)\, X^{2n-1}$$

where $B_{2n}(0)$ is the Bernoulli-Euler number $B_{2n}(X)$ evaluated at $X = 0$.

$$B_{2n}(x) = \frac{(-1)^{n-1}\,2\,(2n)!}{(2\pi)^{2n}} \cdot \sum_{N=1}^{\infty} \frac{\cos 2N\pi X}{N^{2n}}$$

$$B_{2n}(0) = \frac{(-1)^{n-1}2\,(2n)!}{(2\pi)^{2n}} \cdot R_{2n}$$

where $R_{2n} = \sum_{N=1}^{\infty} N^{-2n}$ is the Riemann-Zeta function.

2n	R_{2n}
2	1.64493407
4	1.08232323
6	1.01734306
8	1.00407736
10	1.00099458
12	1.00024609
14	1.00006125

so that series can be written as:

$$TANH(X) = \sum_{n=1}^{\infty} \frac{2(2^{2n}-1)}{\pi^{2n}} \cdot R_{2n}\, X^{2n-1} \cdot (-1)^{n-1}$$

Let $X = (\pi/8) \cdot y$ and truncate series at n = 7

$$TANH(X) = \sum_{n=1}^{6} \frac{2(2^{2n}-1)}{2^{6n-3}} \cdot R_{2n} \, y^{2n-1}$$

with error $\leq \left| \frac{2^{14}-1}{(2)^{38}} \cdot R_{14} \right| = 1.897E\text{-}8$

$$TANH(X) = \sum_{n=1}^{5} \frac{2(2^{2n}-1)}{2^{6n-3}} \cdot R_{2n} \cdot y^{2n-1} + \frac{(2^{12}-1)}{(2)^{32}} \cdot R_{12} \cdot y^{11}$$

The Chebyshev expansion for y^{11} is

$$y^{11} = \frac{T_{11}(y)}{2^{10}} - \frac{C_{11}^{2n-1}}{2^{10}} \cdot y^{2n-1}$$

and error in omitting $T_{11}(y)$ term is

$$\leq \left| \frac{2^{12}-1}{2^{42}} \cdot R_{12} \right| = 2.965E\text{-}10$$

and total error so far 1.927E-8 and series is

$$TAN(X) = \sum_{n=1}^{5} \left[\frac{(2^{2n}-1)}{2^{6n-4}} \cdot R_{2n} - \frac{(2^{12}-1)}{(2)^{42}} \cdot R_{12} \cdot C_{11}^{2n-1} \right] \cdot y^{2n-1} \quad (1)$$

Even though similar expansion for y^9 only gives a total error 3.84E-8 the value of

$$\tan(y) = \frac{(\sqrt{2}+1) + \tan(x)}{1 - (\sqrt{2}+1)\tan(x)}$$

becomes very critical as $y \to \pi/2$ and $\tan(x) \to \sqrt{2} - 1$.

Therefore series will not be reduced any further.

Substituting $y = X\ 8/\pi$ in (1) we get

$$TANH(X) = \sum_{n=1}^{5} \left[2(2^{2n}-1)R_{2n} - \frac{E.\ C_{11}^{\ 2n-1}}{11(2)^{30-6n}} \cdot \frac{X^{2n-1}}{2^n} \right.$$

where $E = \frac{(2^{12} - 1)\cdot 11}{(2)^{15}} \cdot R_{12}$

Define $D = \pi^{-2}$ then

$$A_1 = D \cdot \left[6R_2 + E \cdot 2^{-24}\right] = 1.0000001$$

$$A_2 = D^2 \cdot \left[30R_4 - 5E \cdot 2^{-16}\right] = .33333227$$

$$A_3 = D^3 \cdot \left[126R_6 + 7E.2^{-8}\right] = .13337246$$

$$A_4 = D^4 \cdot \left[510R_8 - 4E\right] = .053388615$$

$$A_5 = D^5 \cdot \left[2046R_{10} + E.2^{8}\right] = .025628253$$

5.0 Arctan(X) for $X \leq \tan(\pi/16)$

Taylor-Mclaurin series is

$$\arctan(X) = \sum_{n=0}^{\infty} \frac{(-1)^n X^{2n+1}}{2n+1}$$

Since $X = \tan(\pi/16) < .2$ then error caused by replacing $X = .2y$ is greater than true error. True value of $\tan(\pi/16) = .19891237$.

Then $$\arctan(X) = \sum_{n=0}^{\infty} \frac{(-1)^n 2^{2n+1}}{(2n+1)(10)^{2n+1}} \cdot y^{2n+1}$$

truncation at n = 4 causes error

$$\leq \left| \frac{2^9}{9\ 10^9} \right| = 5.6888E\text{-}8$$

Using $y^7 = \frac{T_7(y)}{2^6} - \frac{C_7^{2n+1}}{2^6} y^{2n+1}$ in

$$\arctan(X) = \sum_{n=0}^{2} \frac{(-1)^n 2^{2n+1}}{(2n+1)(10)^{2n+1}} \cdot y^{2n+1} - \frac{2^7}{(7)10^7} y^7$$

will give error $\leq \left| \frac{2}{7 \times 10^7} \right| = 1.857E\text{-}8$ if T_7 term is omitted. Then total error so far is $\leq 7.545E\text{-}8$ and series becomes

$$\arctan(X) = \sum_{n=0}^{2} \left[\frac{(-1)^n 2^{2n+1}}{(2n+1)(10)^{2n+1}} + \frac{2 \cdot C_7^{2n+1}}{(7)10^7} \right] \cdot y^{2n+1}$$

Substituting $y = \frac{10X}{2}$ gives

$$\arctan(X) = \sum_{n=0}^{2} \left[\frac{(-1)^n}{(2n+1)} + \frac{C_7}{7} \frac{10^{2n-6}}{2^{2n}} \right] \cdot x^{2n+1}$$

or

$$\text{Arctan}(X) = \sum_{n=0}^{2} A_n X^{2n+1}$$

and

$$A_0 = 1 - 10^{-6} = .99999900$$

$$A_1 = -1/3 + 2 \cdot (10^{-4}) = -.33313333$$

$$A_2 = 1/5 - 10^{-2} = .19000000$$

6.0 Exponential, e^x for $x \leqslant \frac{\ln 2}{2}$

Taylor-Mclaurin series is

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

at $x = \frac{\ln 2}{2}$ and $n = 7$ error is

$$\leqslant \left| \left(\frac{\ln 2}{2}\right)^7 \right| \cdot 1/7! = 1.1916E-7$$

Therefore series is normal series truncated after n = 6, i.e.,

$$e^x = \sum_{n=0}^{6} \frac{x^n}{n!}$$

Now X and N can always be found such that $e^y = 2^N e^X$ where $X \leqslant \frac{\ln 2}{2}$

If X and N are defined as follows

Let $t = y/\ln 2$

N = t + ½ integral part

$w = t - N$

$X = W \ln 2$.

then

$X = (t-N) \cdot \ln 2 = t \cdot \ln 2 - \ln(2^N)$

$e^X = e^y \cdot 2^{-N}$

and $e^y = 2^N e^X$

7.0 Natural Logarithm X for $|X| \leq (3 - 2\sqrt{2})$

The Taylor-Maclaurin series for ln is

$$\ln\left(\frac{1+X}{1-X}\right) = 2\sum_{n=0}^{\infty} \frac{X^{2n+1}}{2n+1}$$

truncating series at n = 4 causes error

$$\left|\frac{2\cdot(.17157287)^9}{9}\right| = 2.86E\text{-}8$$

and $\ln\left(\frac{1+X}{1-X}\right) = \sum_{n=0}^{3} \frac{2X^{2n+1}}{(2n+1)}$. Let X = aY

$$= \sum_{n=0}^{3} \frac{2(aY)^{2n+1}}{2n+1}$$

$$Y^7 = \frac{T_7(Y)}{2^6} - \frac{C_7^{2n+1}}{2^6} \cdot Y^{2n+1}$$

error in omitting $T_7(x)$ term is

$$\leq \left|\frac{(.17157287)^7}{7 \cdot 2^5}\right| = 1.954E\text{-}8$$

Total error so far $\leq$ 4.81E-8

Attempt at further Chebyshev reduction will cause too great an error. Therefore final series is after substitution of Y = X/a

$$\ln\frac{1+X}{1-X} = \sum_{n=0}^{2} \left[\frac{2}{2n+1} - \frac{a^{6-2n} C_7^{2n+1}}{7.2^5}\right] X^{2n+1}$$

$$\ln \frac{1 + X}{1 - X} = \sum_{n=0}^{2} A_n X^{2n+1}$$

$$A_0 = 2 + 2^{-5}_{a}{}^{6} = 2.0000008$$

$$A_1 = 2/3 - 2^{-2}_{a}{}^{4} = 0.66645003$$

$$A_2 = 2/5 + a^2/_2 = 0.41471862$$

Given $Y = 2^N Z$ where $1/2 \leq Z < 1$

Then $\ln(Y) = N \cdot \ln 2 + \ln Z$.

If X is defined as $X = \dfrac{Z - \sqrt{2}/_2}{Z + \sqrt{2}/_2}$

then $|X| \leq 3 - 2\sqrt{2}$

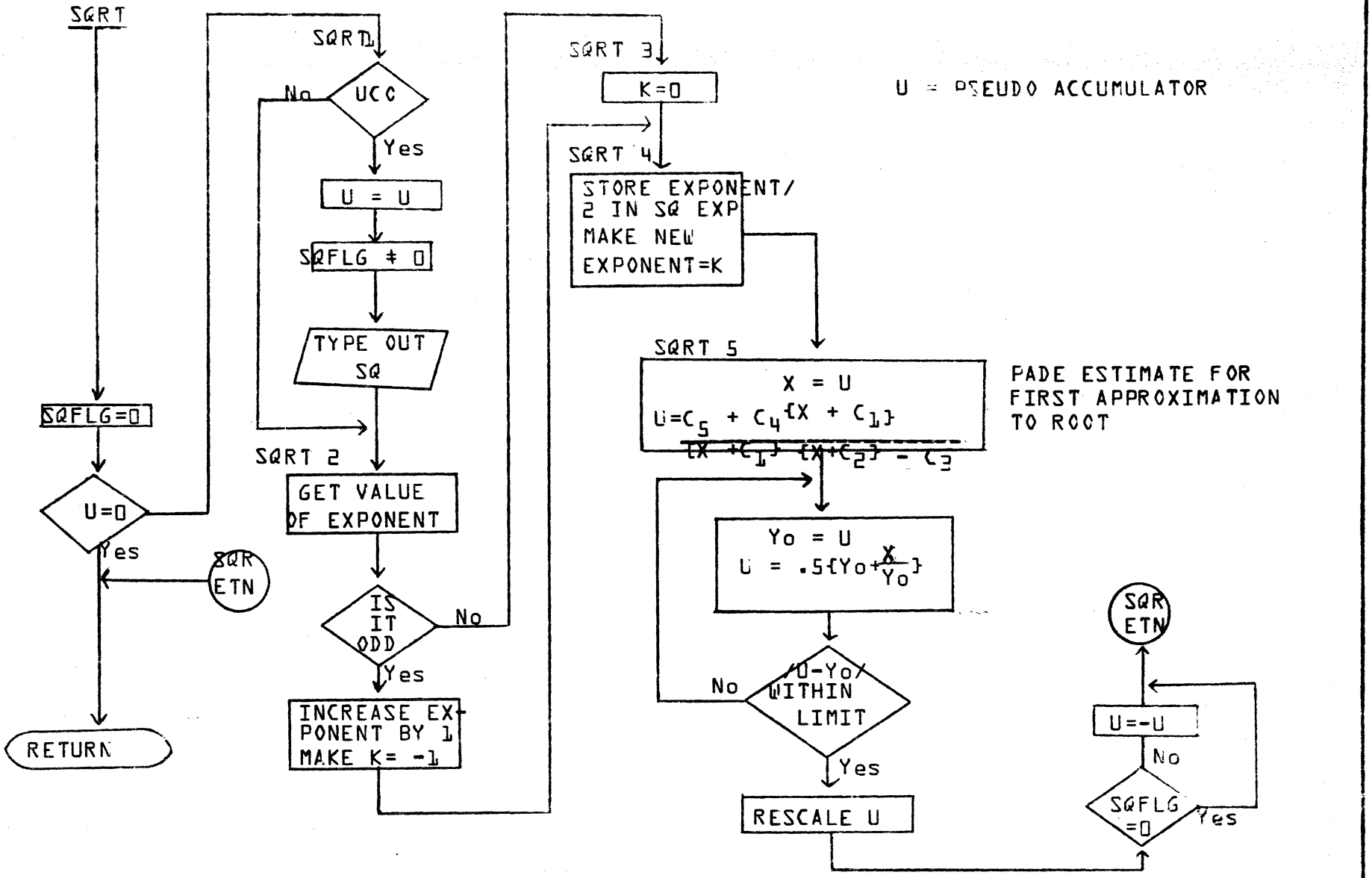
and $\ln Z = -\dfrac{\ln 2}{2} + \ln \dfrac{1 + X}{1 - X}$

Finally $\ln Y = (N - 1/2) \cdot \ln 2 + \ln \dfrac{1 + X}{1 - X}$

TABLE OF C_m^{2n+1}

n \ m		5	7	9	11	13	15	17
1	2^0	5	-7	9	-11	13	-15	17
3	2^2	-5	14	-30	55	-91	140	-204
5	2^4	1	-7	27	-77	182	-378	714
7	2^6		1	-9	44	-156	450	-1122
9	2^8			1	-11	65	-275	935
11	2^{10}				1	-13	90	-642
13	2^{12}					1	-15	119
15	2^{14}						1	-17
17	2^{16}							1

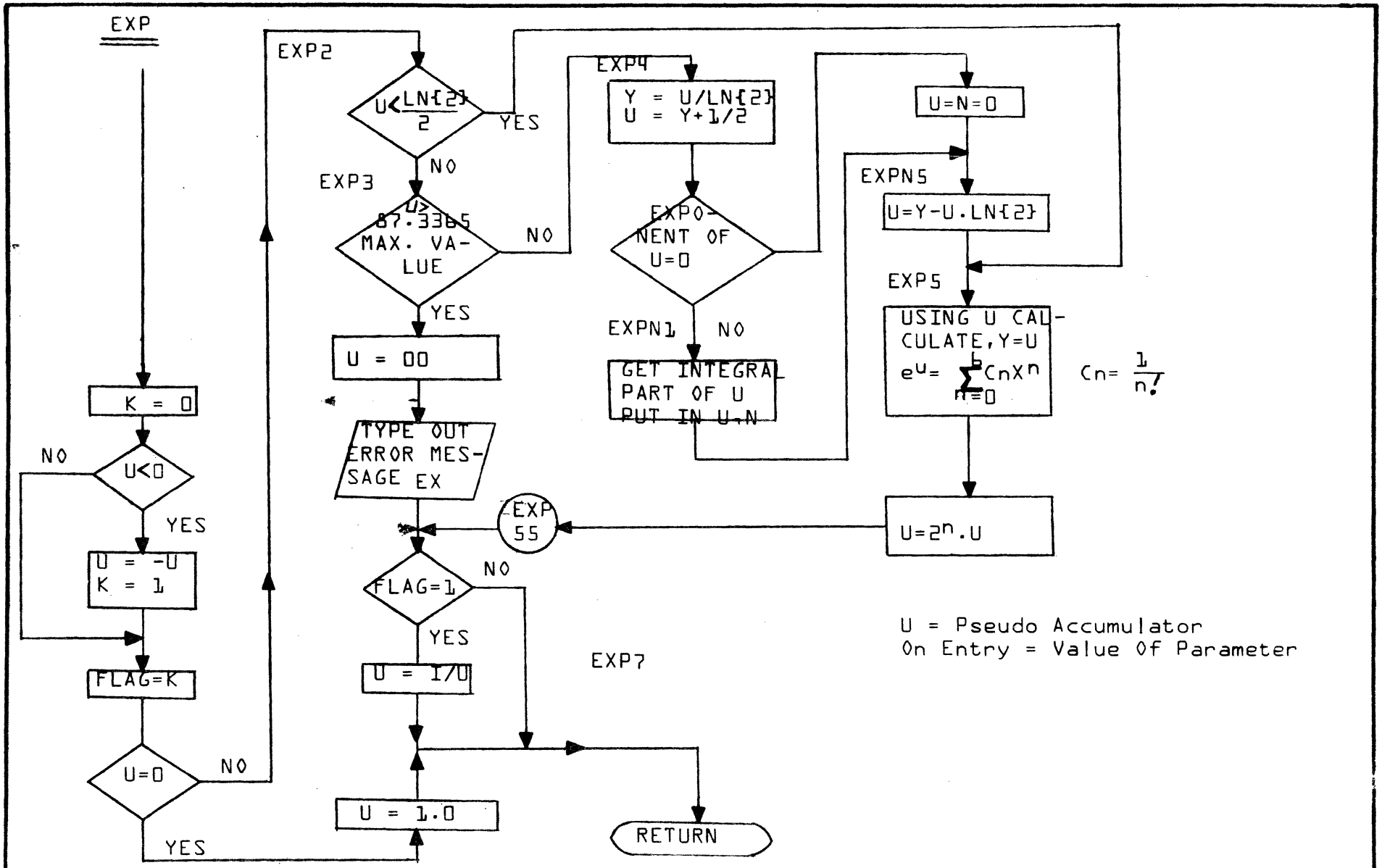
Ex. $C_{13}^{9} = 65 \cdot 2^8$ and $y^{13} = \frac{T_{13}}{2^{12}} - \frac{C_{13}^{2n+1}}{2^{12}} y^{2n+1}$



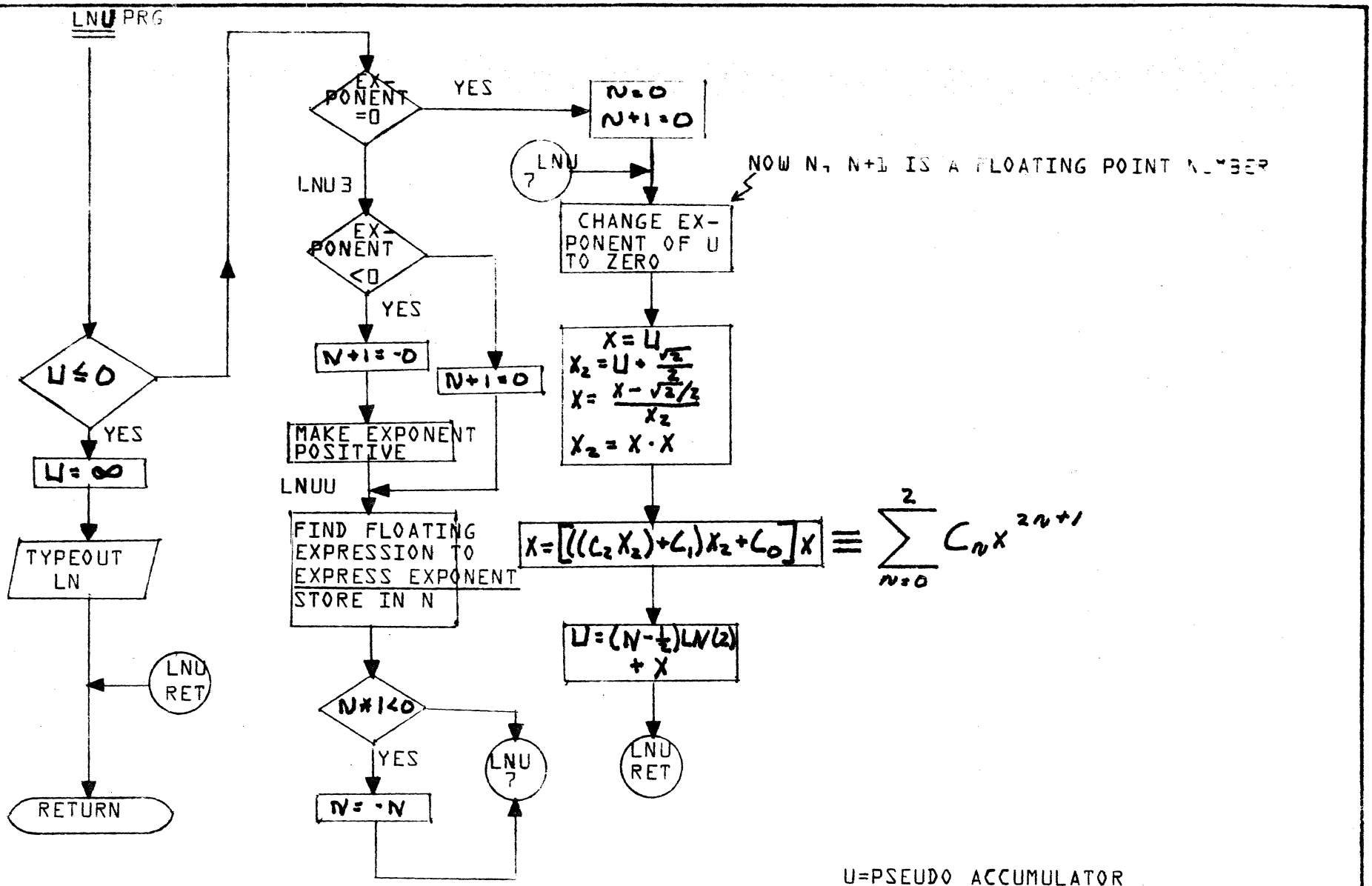
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SQRT		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

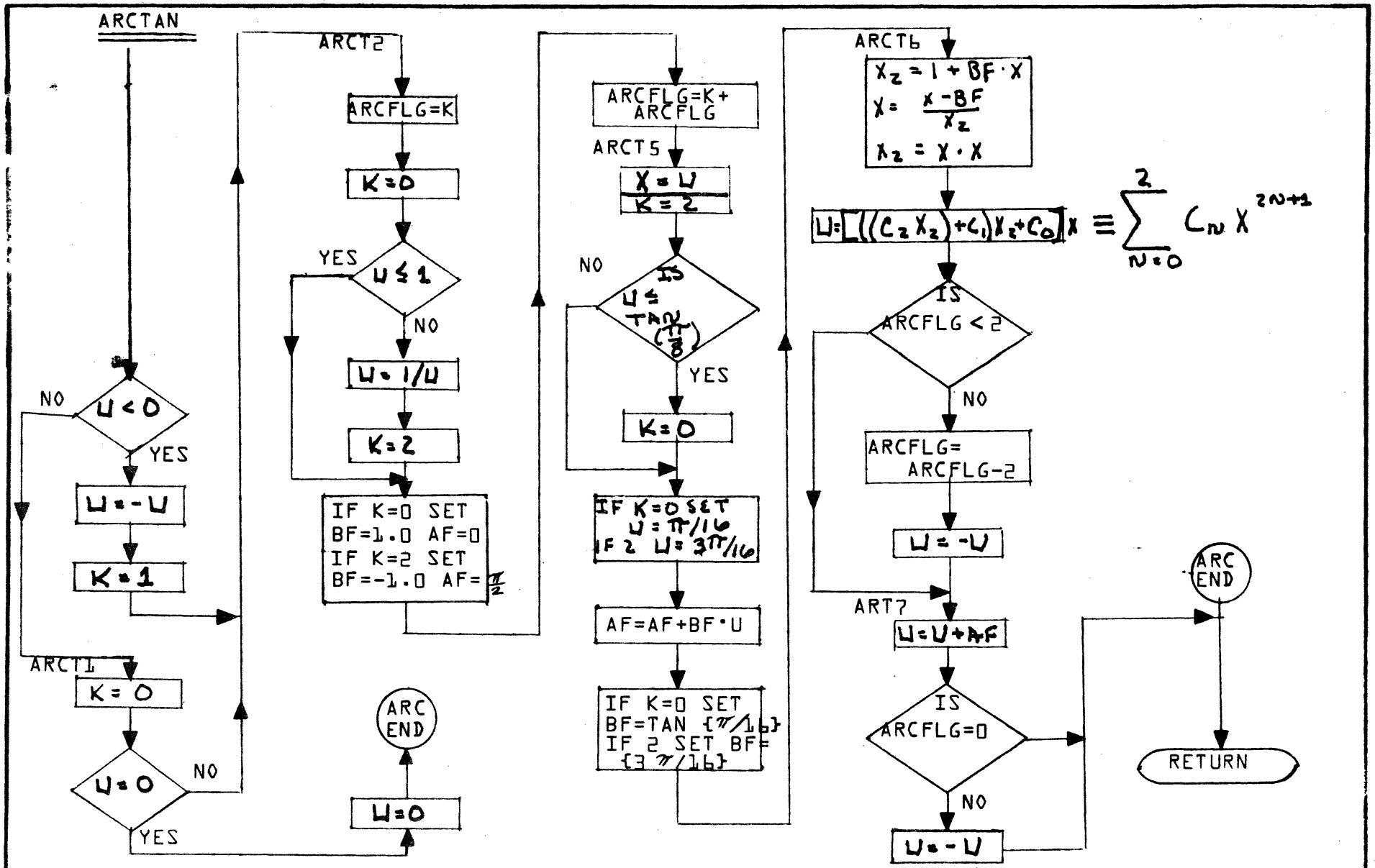
PRINTED IN U.S.A.



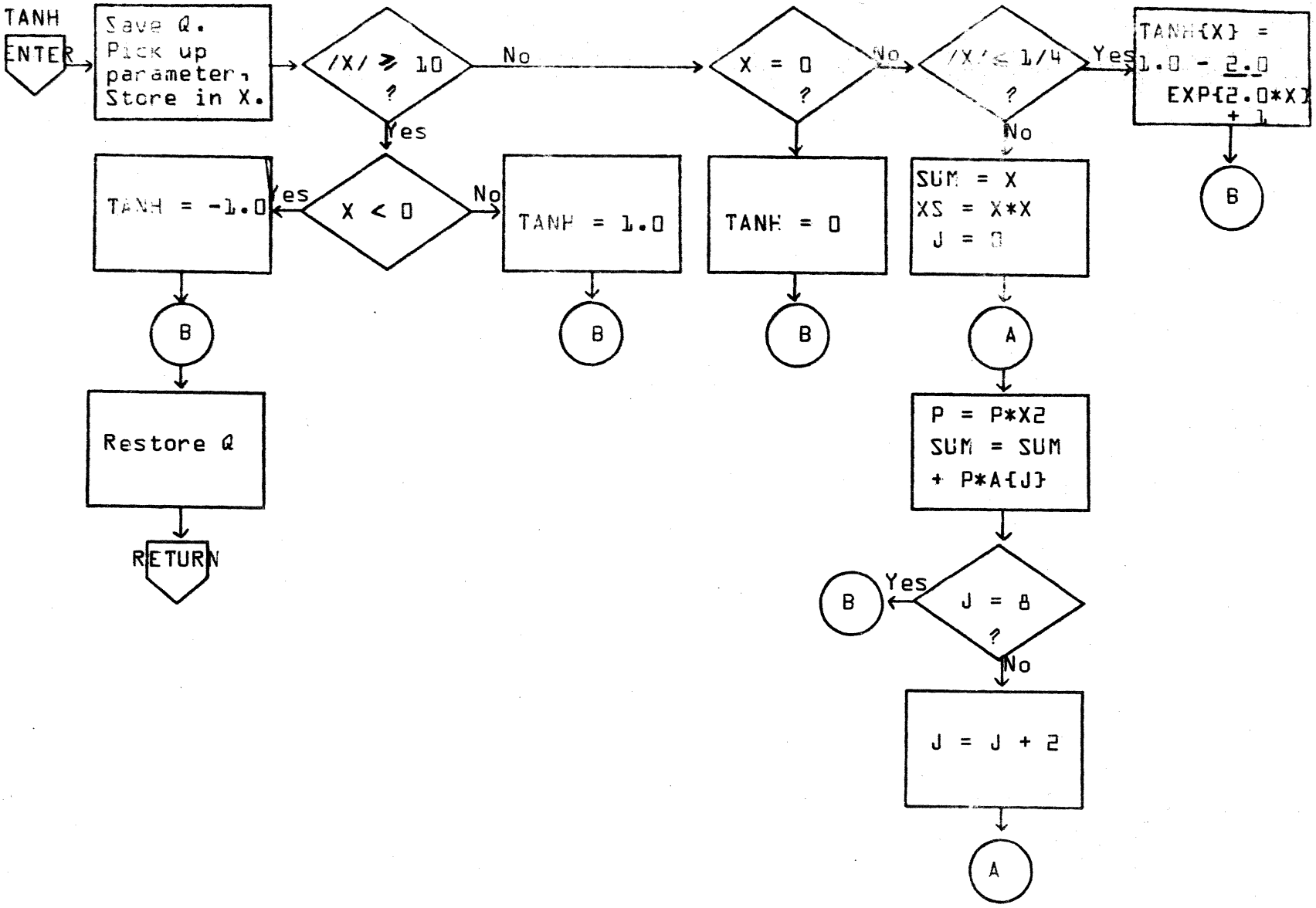
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE EXP		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LNUPRG		PROJECT MGR.		
	PAGE 1 OF 1		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

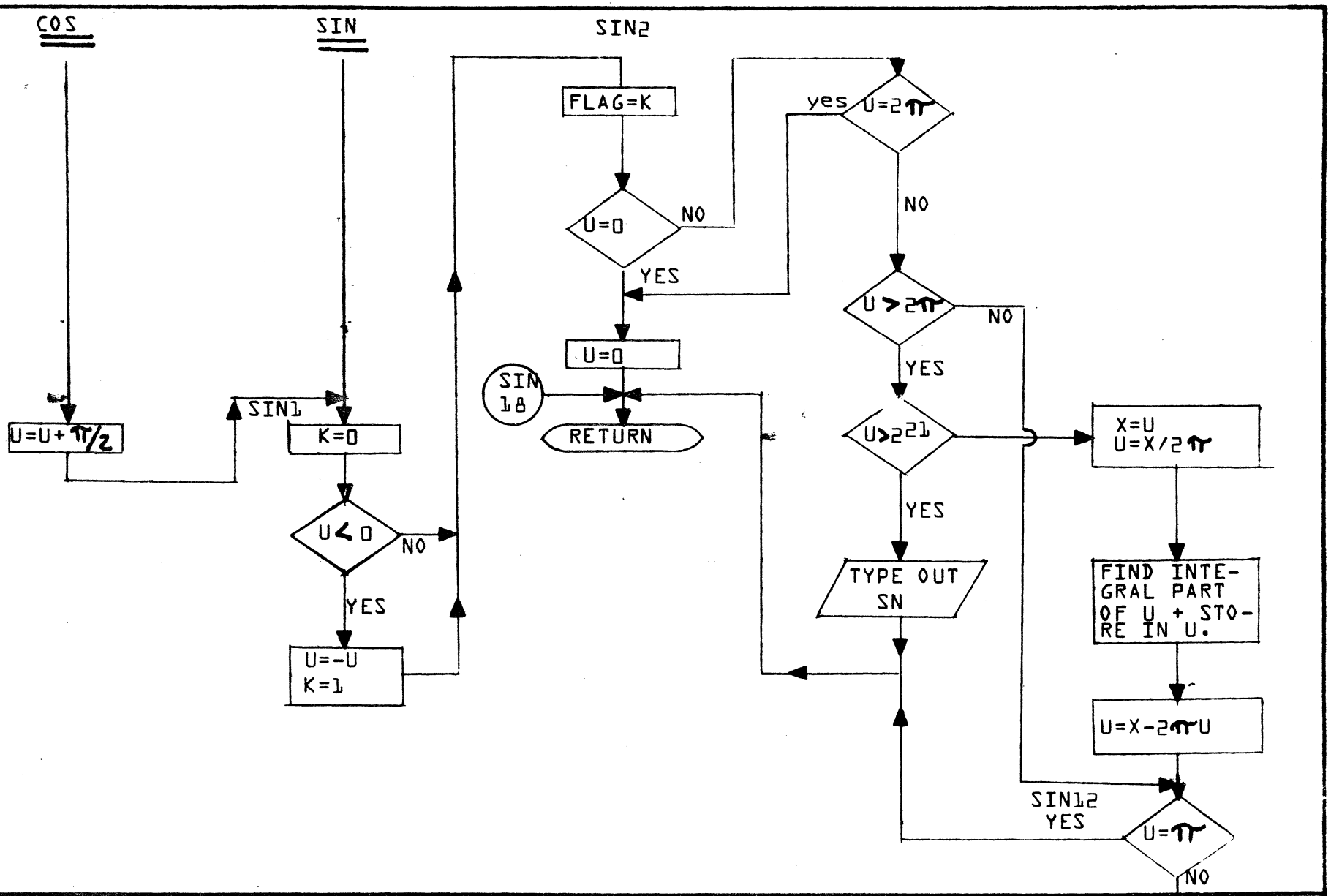


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ARCTAN		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

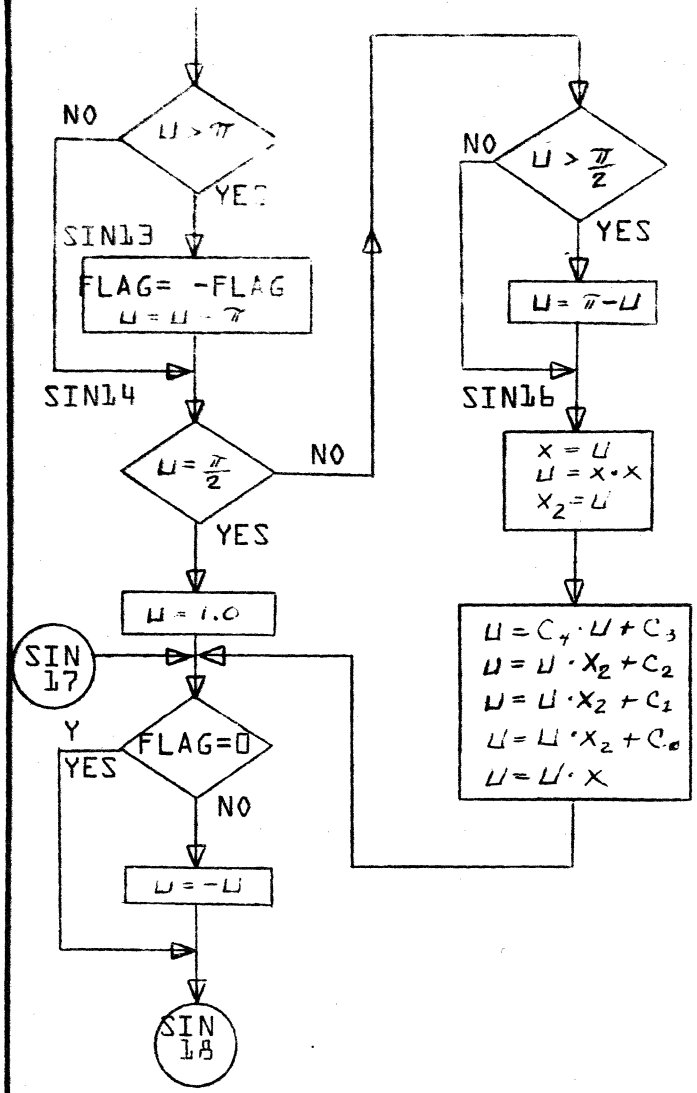


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TANH	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE COS&SIN		PROJECT MGR.		
		PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



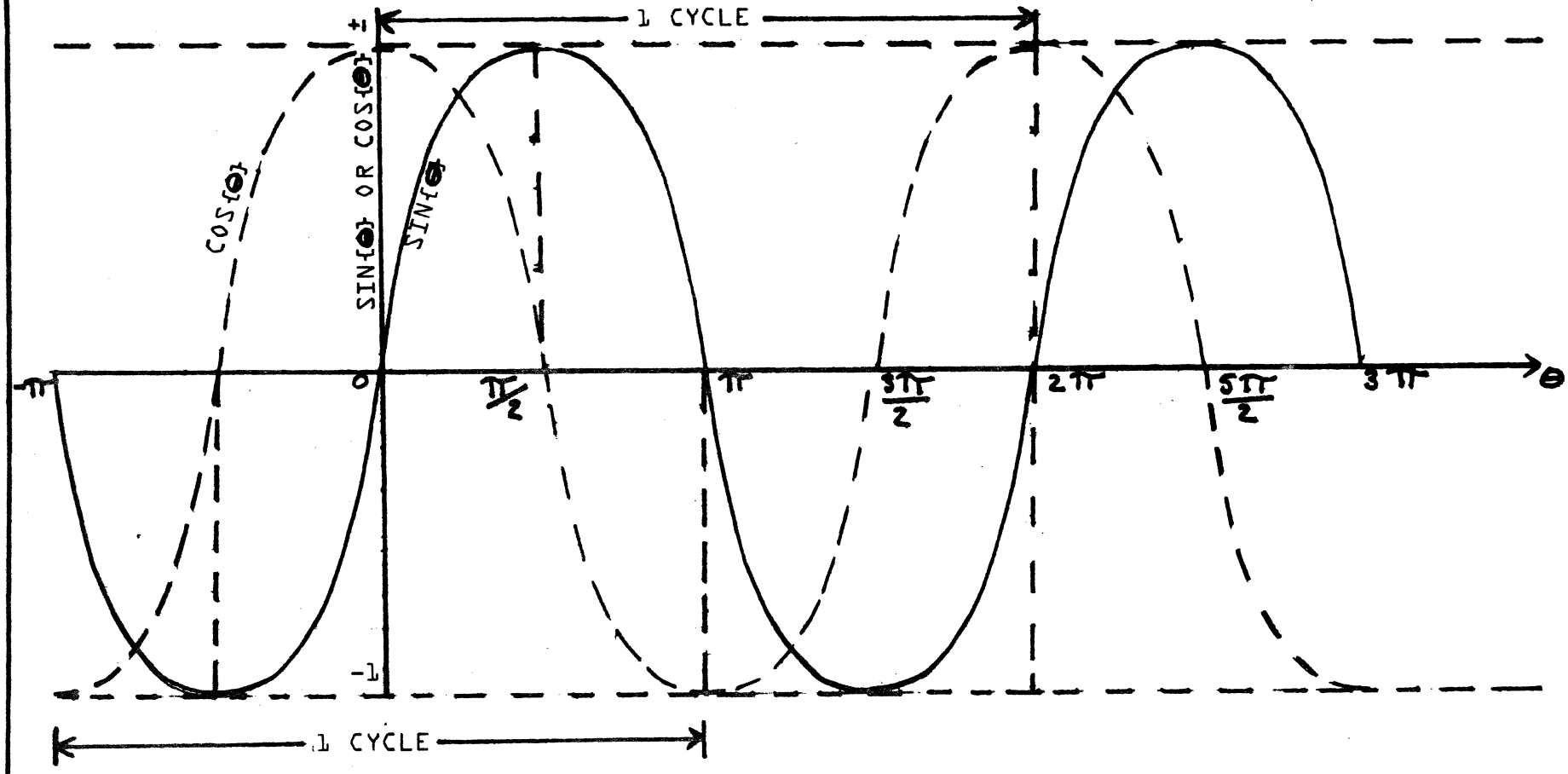
$$\equiv \sum_{n=0}^{4} C_n X^{2n+1}$$

CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE COS&SIN		PROJECT MGR.			
FLOWCHART ☒		PAGE 2 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

1385 (FORM CA127-1)

PRINTED IN U.S.A.

8-48



$COS\{\theta\} = SIN\{\theta + \frac{\pi}{2}\}$

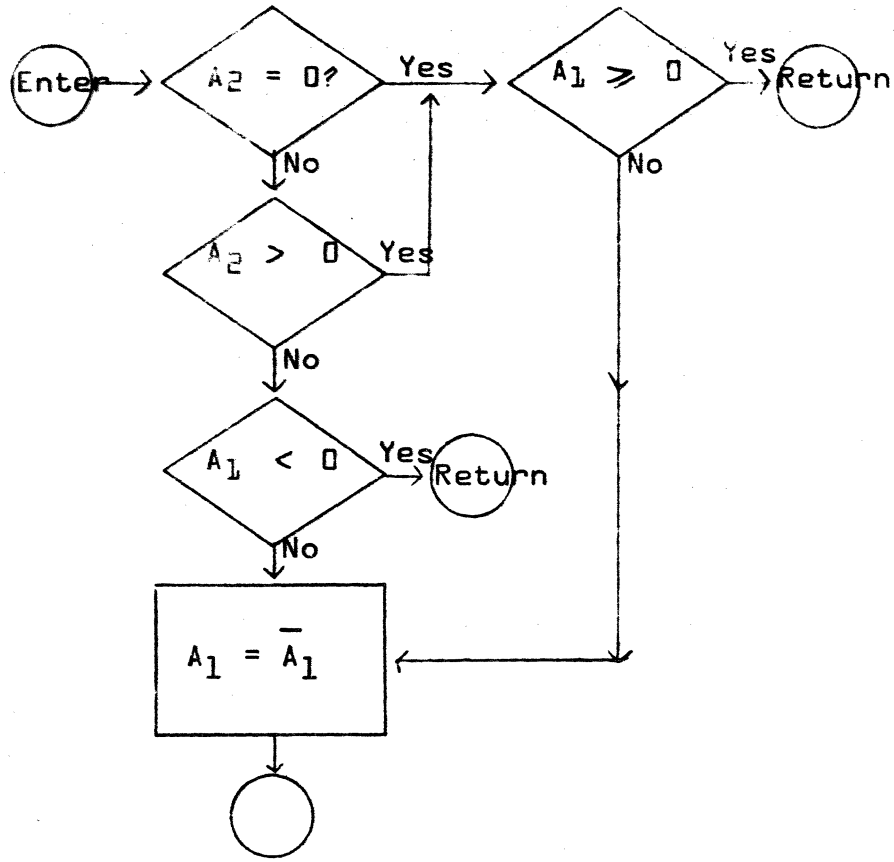
LET θ BE INPUT VALUE. IF $\theta < 0$, $SIN\{\theta\} = -SIN\{|\theta|\}$.

LET I BE INTEGRAL PART OF $\theta / 2\pi$. Then $\beta = \theta - I \cdot 2\pi$ IS $< 2\pi$.

IF $\beta > \pi$ THEN $SIN\{\beta\} = -SIN\{2\pi - \beta\}$, LET $\beta = 2\pi - \beta$

IF $\beta > \pi/2$ THEN $SIN\{\beta\} = SIN\{\pi - \beta\}$.

CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: COS&SIN		PROJECT MGR.			
FLOWCHART ☒		PAGE 3 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SIGN	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

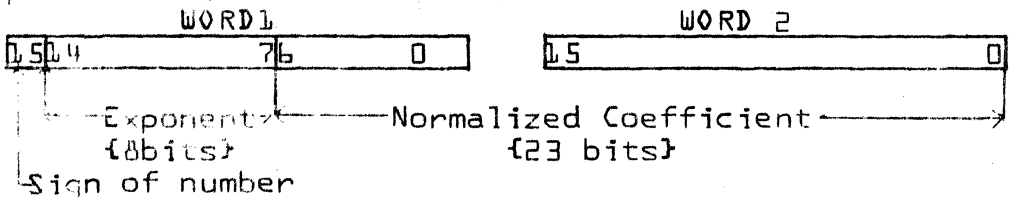
AA1385 (FORMERLY CA127-1)

TABLE OF CONTENTS

9.0 FLOATING POINT PACKAGE

9.1 General Description

Each floating point number requires two consecutive 1700 Computer words. The first word, containing the most significant bits of the number, is the one addressed. Normalized floating point format is as follows:



A floating point number X is in the range given below and is significant to one part in $8X10^{6}$.

$$-2^{127}(1-2^{1/23}) \leq X \leq 2^{127}(1-2^{1/23})$$

If the most significant word is zero (16 bits of zero or one), a floating point zero is assumed.

9.1.1 Exponent

The floating point exponent is an 8-bit quantity with a value ranging from 00 to FF_{16}. Through biasing by 80_{16}, this range expresses both positive and negative exponents.

9.1.2 Coefficient

The coefficient consists of a 23 bit number n, $1-2^{-23} \geq |n| \geq 0$. The high order bit position of the first word is the coefficient sign bit.

A zero denotes a positive coefficient and a one denotes a negative coefficient. When the coefficient is negative, the entire floating point number is stored in complement form.

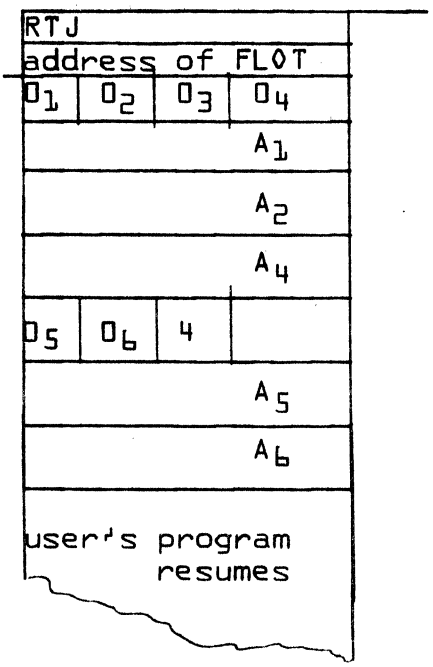
9.1.3 Calling Sequence

FLOT uses an interpretive calling sequence. It does not save Q or FF; it uses the communications locations $C5 and $C6 as a pseudo accumulator. The pseudo accumulator is retained between calls to FLOT.

The interpretation of the calling sequence is based on a string of 4-bit bytes representing the operations, followed by the operand addresses. The leftmost 4-bit byte is the first operation; the operand addresses, if they exist, follow in the same order as the operation bytes, one word

per byte. As many bytes may exist as desired, but the terminating byte must be 4, the operation FEND.

Example:



The O_i's are the operation codes; the A_i's are their operand addresses. An operand address is not needed by all operations, for instance, operation code O_3 does not have a corresponding A_3.

9.1.4 Operations

The following operations are used by the floating point package.

Operation	4-bit Code	Meaning
CHMD	5	Change of mode operation. All operand addresses following this operation code in the calling sequence are made relative if the preceding addresses were absolute, or absolute if the preceding addresses were relative. Addresses are initially absolute. No operand is needed.
INDX	F_{16}	Index. The operand corresponding to INDX is used to increment the operand of the following operations: FLDD, FLST, FADD, FSUB, FMPY, and FDIV.

Each succeeding INDX supersedes the last. No index is initially assumed.

NIDX	6	No index. The succeeding operands do not have indexing increments. NIDX supersedes any preceding INDX, and is superseded by any following INDX. NIDX is assumed initially. No operand is needed.
FLDD	B_{16}	Floating load. The floating point number in the corresponding effective operand address is transferred to the pseudo accumulator.
FLST	D_{16}	Floating store. The floating point number is transferred from the pseudo accumulator to the corresponding effective operand address.
FCOM	7	Floating complement. The pseudo accumulator is complemented. No operand is needed.
FADD	E_{16}	Floating add. The contents of the effective operand address is added to the pseudo accumulator, and the sum is left in the pseudo accumulator.
FSUB	8	Floating subtract. The contents of the effective operand address is subtracted from the pseudo accumulator, and the result is left in the pseudo accumulator.
FMPY	9	Floating multiply. The pseudo accumulator is multiplied by the contents of the effective operand address, and the result is left in the pseudo accumulator.
FDIV	A_{16}	Floating divide. The pseudo accumulator is divided by the contents of the e effective operand address, and the result is left in the pseudo accumulator.
FEND	4 (also 1, 2, 3)	End of calling sequence. This operation terminates the calling sequence. No operand needed.

9.1.5 Operands

All operand addresses, absolute or relative, are placed in bits 14-0. Bit 15 is a flag set to distinguish between direct and indirect addresses.

9.1.5.1 Absolute Addressing

An absolute address may be either direct or indirect. All indirect addressing is executed prior to indexing. Only one level of indirect addressing is allowed.

9.1.5.2 Relative Addressing

A relative address may be indirect to one level of indirectness. The relative address is computed by subtracting the location of the address in the calling sequence from the address contained in that location. For instance, if the relative address, A_1, is contained in location L+3, where L is the first word of the calling sequence, the relative address = A_1 - L + 3.

9.1.5.3 Example

Compute the floating point arithmetic statement:

$$X=-\{A\{I\}+B\{I\}\}*C\{I\}+D\{J\}*E\{J\}$$

Assuming that X, J, D, E, and the temporary location TEMP are absolutely addressed, and the other operands relatively addressed, the generated calling sequence would be:

RTJ			FLOT
F_{16}	B_{16}	9	6
absolute address J			
absolute address D			
absolute address E			
D_{16}	5	F_{16}	B_{16}
absolute address TEMP			
relative address I			

relative address A			
E_{16}	9	7	6
relative address B			
relative address C			
5	E_{16}	D_{16}	4
absolute address TEMP			
absolute address X			

9.1.6 Fault Conditions

Fault conditions are flagged in a communication location {$C8} whenever they occur during execution. The following bits are set to one:

BIT	Fault Condition
15	Exponent overflow; the exponent in an arithmetic operation exceeds the maximum range limit.
14	Divide fault; division by zero is encountered.
13	Exponent underflow; the exponent in an arithmetic operation is less than the minimum range limit.

If exponent underflow occurs, a floating point zero results. If exponent overflow occurs, the largest word of the appropriate sign results. A divide fault is treated as overflow. These error conditions may be tested with the IFALT function.

9.2 FLOATING-POINT ARITHMETIC WITH 23-BIT NUMBERS

9.2.1 Introduction

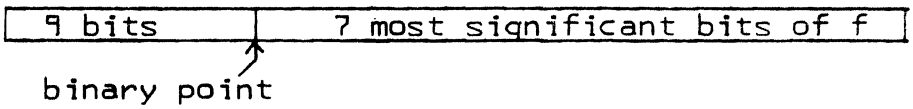
Consider the double precision floating-point number:

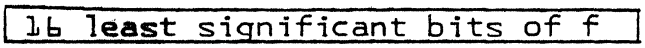
$$F = f \cdot 2^{B} \qquad \{1\}$$

where $|f|$ lies in the range

$$1/2 \leq |f| \leq 1 - 2^{-23}. \qquad \{2\}$$

We have a machine with a word length of 16 bits and that the 32 bits in the double length word are divided in the following way:





The leftmost block of nine bits is divided into three parts:

1. The first {leftmost} bit represents the sign of F.
2. The second bit represents the sign of β .
3. The next 7 bits represent the magnitude of β. This allows 23 bits for the representation of f. We shall assume that the binary point lies at the left and of the 23 bits representing f so that the 7 most significant bits of f are stored in the first word of the pair, and the 16 least significant bits of f are stored in the second word of the pair.

If we write

$$|f| = c + d.2^{-7} \qquad \{3\}$$

where c lies in the range

$$1/2 \leq c \leq 1 - 2^{-7} \qquad \{4\}$$

and where d lies in the range

$$1/2 \leq d \leq 1 - 2^{-16} \qquad \{5\}$$

then c represents the 7 most significant bits of f and d represents the 16 least significant bits of f.

We wish to condiser the four basic arithmetic operations using floating point numbers of the form discussed. Consequently, in order to have notation for two operands, let us consider a second floating point number.

$$G = g.2^{s} \qquad \{6\}$$

where |g| lies in the range

$$1/2 \leq |g| \leq 1 - 2^{-23} \qquad \{7\}$$

If we write

$$|g| = a + b.2^{-7} \quad \{8\}$$

where a lies in the range

$$1/2 \leq a \leq 1 - 2^{-7} \quad \{9\}$$

where b lies in the range

$$1/2 \leq b \leq 1 - 2^{-16} \quad \{10\}$$

then a represents the 7 most significant bits of |g| and b represents the 16 least significant bits of |g|.

The machine represents negative numbers using a "one's-complement" system. In fact the procedure for changing the sign of a floating point number is to perform a bit-by-bit complement of the entire 32 bits {including the 9 bits representing the sign and exponent}.

9.2.2 THE FOUR ARITHMETIC OPERATIONS

A. Multiplication

$$F \cdot G = \{f.2^{\mathcal{B}}\}\ \{g.2^{\mathcal{S}}\}$$
$$= \{\text{sign of } F.G\}\ |f| \ .\ |g| \ .2^{\mathcal{B}+\mathcal{S}} \quad \{11\}$$

The computational procedure is primarily concerned with the formation of $|f| \ . \ |g| \ . \ 2^{\mathcal{B}+\mathcal{S}}$ since {sign F.G} can be recorded in advance and used later to apply the correct sign to the product. In addition to recording {sign F.G} we record the exponents $\mathcal{B}$ and $\mathcal{S}$ until after the product |f| . |g| is formed. In fact, we propose the following algorithm for multiplying F by G.

1. Determine and record {sign F.G}
2. Form |F| and |G|.
3. Record the leftmost nine bits of |F| and |G|. This, in effect, records $\mathcal{B}$ and $\mathcal{S}$.
4. Shift the 23 bits of |f| and |g| left until each has the bit pattern:

+	15 most significant bits	C

0	8 least significant bits	7 zeros	D

If this procedure is followed, |f| is no longer represented by {3} during the computation is step 5, below, but has the form

$$|f| = C + D.2^{-15} \qquad \{12\}$$

where C lies in the range

$$1/2 \leq C \leq 1 - 2^{-15} \qquad \{13\}$$

where D lies in the range

$$0 \leq D \leq 1 - 2^{-8} \qquad \{14\}$$

likewise $|g|$ has the form

$$|g| = A + B.2^{-15} \qquad \{15\}$$

and where A lies in the range

$$1/2 \leq A \leq 1 - 2^{-15} \qquad \{16\}$$

and where B lies in the range

$$0 \leq B \leq 1 - 2^{-8} \qquad \{17\}$$

5. Use fixed point operations in forming the product

$$|f| \cdot |g| = \{C+D.2^{-15}\}\{A+B.2^{-15}\}$$

$$= CA + \{CB + DA\}.2^{-15} + DB.2^{-30}$$

$$= CA + \{CB + DA\}.2^{-15} \qquad \{18\}$$

Notice that the term $DB.2^{-30}$ may be ignored because once the product is placed back in standard form only 23 bits are retained. Notice, also, that {18} is written in such a way that it exhibits the efficiency of the following choice of computational steps:

a. Form CA giving a double length product.
b. Form CB and retain the most significant half of the double length product.
c. Form DA.
d. Add the most significant half of DA to the most significant half of CB.
e. Add the least significant half of CA to the sum obtained in {d}. This result is the second half of the double length product. The first half of the double length product is the most significant half of CA which was formed above in a.

6. Next, round and normalize the product obtained using {18} in Step 5. Any adjustment in the

exponent $\beta+S$ which is necessary because of the normalization of $|f| \cdot |g|$ must be performed.

7. Finally, pack the 23 bits of the normalized product and the 9 bits representing the sign and the adjusted exponent into two 16 bit words. If {sign F.G} is negative, the two words must then be complemented to give the correct sign to the product.

B. Division

$$\frac{G}{F} = \frac{q \cdot 2^{S}}{f \cdot 2^{\beta}} \qquad \{19\}$$

$$= \{\text{sign of } G.F\ \left|\frac{q}{f}\right| \cdot 2^{S-\beta}$$

as a matter of fact, since we want

$$\left|\frac{q}{f}\right| < 1, \qquad \{20\}$$

we scale the numerator and write

$$\frac{G}{F} = \{\text{sign } G.F\} \frac{\left|\frac{q}{2}\right|}{|f|} \cdot 2^{S-\beta+1} \qquad \{21\}$$

Thus we propose the following algorithm for dividing G by F.

1. Determine the record {sign F.G}
2. Form |F| and |G|.
3. Record the leftmost nine bits of |F| and |G|. this, in effect, records β and S.
4. Arrange the 23 bits of |f| to give the bit pattern:

+	15 most significant bits of |f|	C

0	8 least significant bits	7 zeros	D

and the 23 bits of |g| to give the bit pattern:

+	0	14 most significant bits of |g|	A

0	9 least significant bits	6 zeros	B

Thus, |f| is represented by {12}, {13}, and {14} as in the case of multiplication. However, in this case we have:

$$\left|\frac{q}{2}\right| = A + B.2^{-15} \qquad \{22\}$$

where A lies in the range

$$1/4 \leq A \leq 1/2 - 2^{-15} \qquad \{23\}$$

and where B lies in the range

$$0 \leq B \leq 1 - 2^{-9} \qquad \{24\}$$

5. Use fixed point operations in forming the quotient

$$\left|\frac{q}{2}\right|_f = \frac{A + B.2^{-15}}{C + D.2^{-15}}$$

$$= \left[\frac{A + B.2^{-15}}{C}\right]\left[\frac{1}{1 + \frac{D}{C}.2^{-15}}\right]$$

$$= \left[\frac{A}{C} + \frac{B}{C}.2^{-15}\right]\left[1 - \frac{D}{C}.2^{-15} + \frac{D^2}{C^2}.2^{-30} \cdots\right]$$

$$= \frac{A}{C} + \frac{B}{C}.2^{-15} - \frac{AD}{C^2}.2^{-15} - \frac{BD}{C^2}.2^{-30} + \frac{AD^2}{C^3}.2^{-30} + \cdots$$

$$= \frac{A}{C} + \left[\frac{B}{C} - \frac{AD}{C^2}\right].2^{-15} + \left[\frac{AD^2}{C^3} - \frac{BD}{C^2}\right].2^{-30} + \cdots$$

$$= \frac{A + \left[B - \frac{AD}{C}\right].2^{-15}}{C} \qquad \{25\}$$

Notice that the terms beginning with

$$\left[\frac{AD^2}{C^3} - \frac{BD}{C^2}\right].2^{-30}$$

may be ignored because only 23 bits of the quotient are retained. Notice, also, that {25} is written in such a way that it exhibits the efficiency of the following choice of computational steps:

a. Form -AD giving a double length product.
b. Divide -AD {as a double length divident} by C.
c. Form $B - \frac{AD}{C}$ {rounded to single length}.

d. Form the double length dividend

$$A + \left[B - \frac{AD}{C}\right].2^{-15}$$

{The sign of the second term makes this tricky.}

e. Divide this double length dividend by C.
f. To obtain the second half of the double length quotient, the remainder resulting from the division in the previous step must not be divided by C.

The class of machines for which this procedure is efficient must have the feature that fixed point multiplication leaves a double length product which can be used as a double length divident for fixed point division.

6. Next, round and normalize the quotient obtained using {25} and the procedure of step 5. Any adjustment in the exponent which is necessary because of the normalization of $\left|\frac{g}{f}\right|$ must be performed.

7. Finally, pack the 23 bits of the normalized quotient and the 9 bits representing the sign and the adjusted exponent into two 16 bit words. If {sign F.G} is negative, the two words must then be complemented to give the correct sign to the quotient.

C. Addition

$$F + G = f.2^{\beta} + g.2^{\delta} \qquad \{26\}$$

The basic problem in floating point addition is to adjust the exponent of F {or G} so that the binary points are lined up before the addition takes place.

Let us introduce the notation L to represent a pair of cells containing the larger of the two numbers F and G, and the notation M to represent a pair of cells containing the smaller of the two numbers. We shall say that F is larger than G if

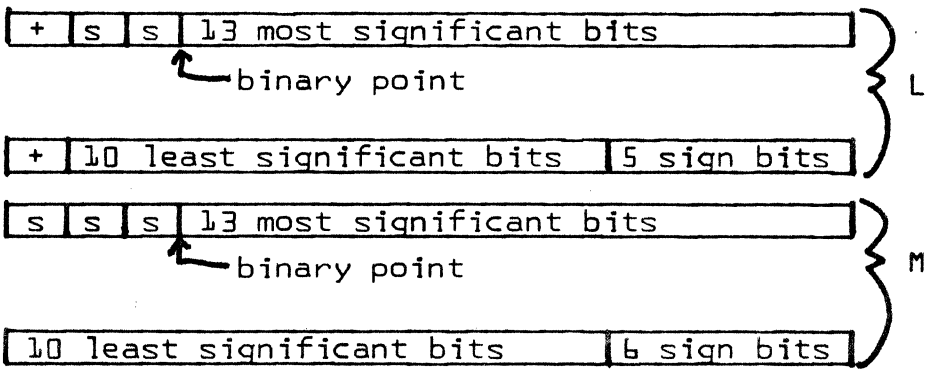
$$\beta \geq \delta \qquad \{27\}$$

and F is smaller than G if

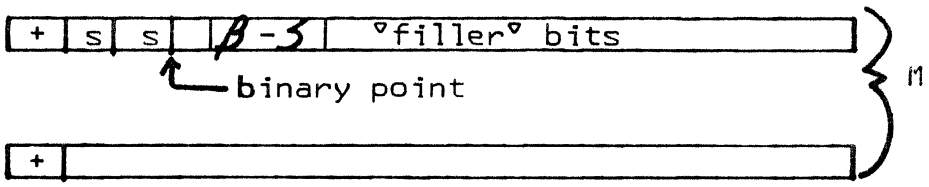
$$\beta < \delta \qquad \{28\}$$

We are not concerned about the relative magnitudes of f and g in case the exponents are equal. Using this convention, we propose the following algorithm for forming F + G.

1. Record the leftmost nine bits of F and G. This, in effect, records β and σ.
2. Determine the sign of $\{\beta-\sigma\}$ and thus determine whether F is smaller or larger than G according to {27}&{28}.
3. Place f and g in L and M. If F is larger than G, then f goes to L; otherwise f goes into M and g goes into L. The following bit patterns should be formed {here s means "sign bit"}:

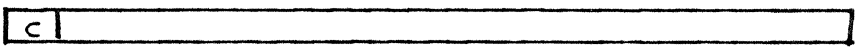


4. Shift M right $|\beta-\sigma|$ places and put a "+" bit at the beginning of each of the two words. If $|\beta-\sigma| \geq 23$ then there is no need to continue since all significant bits in M will be lost.



Notice that the $|\beta-\sigma|$ "filler" bits, between the binary point in M and the most significant bit of the fraction, are sign bits. This is mathematically correct in a one's complement representation of negative numbers.

5. Add the second halves of L and M.



We call the first bit of this sum c. If it is a one we actually have a carry. If it is a zero we do not

have a carry. However, it usually is easier to add c (see step 6) than to test to see whether or not it needs to be added as a carry bit in forming the sum of the first halves of L and M.

6. Add the first halves of L and M and add the carry bit obtained from step 5.

e	s	v	

If e = 1 then an end-around-carry must be performed. This means that a one is added at the right end of the word produced in step 5. Since this might also produce a carry bit, the c in the diagram (see step 5) must be cleared to zero before the end-around-carry. If a carry bit is again produced, then a one must be added at the right end of the word above. It can be shown that this last operation can never produce another e = 1.

If v = s, then v is a sign bit. However, if v ≠ s then there has been overflow during the addition, and and v is the most significant bit of the sum. In the latter case an adjustment of the exponent will be necessary to give the correct answer.

7. Shift of second half of the sum left one place to clear out the carry bit c. Then shift the double length sum left (a) one place if v ≠ s; (b) two places if v = s.

This leaves the sum in the following form:

s	15 most significant bits of sum

at least 8 bits of the sum	sign bits

8. If the double length sum was shifted one place left in step 7 (v ≠ s) then the exponent must be adjusted to take care of the overflow. This means adding one to the exponent β or δ, whichever is larger. (This will be the exponent of the sum.) If the double length sum was shifted two places left in step 7, no adjustment of exponent is necessary.

9. The form of the sum given by step 8 must be checked for normalization since it is possible that several of the leading bits of the sum may be zero. (Cancell-

ation occurs when two numbers of opposite sign but nearly equal magnitude are added.) If the sum is not normalized then it should be normalized at this point and appropriate adjustments in the exponent should be made. If 23 left shifts are not sufficient for normalization then the sum should be made zero.

10. At this point the normalized sum may be rounded although the extra coding involved may not be worth the gain. If rounding is desired then there are two cases to be considered depending on the sign of the sum. These cases require that care be taken in handling any carry bit produced by the rounding operation.

11. Now pack the 23 most significant bits of the sum, along with 9 bits representing the sign and exponent, into two 16 bit words. If the sign of the sum is negative, then the first 9 bits must be complemented before the packing takes place.

D. Subtraction

No special subroutine is necessary since

$$F - G = F + (-G)$$

and one merely complements G before entering the addition subroutine.

REFERENCES:

Gregory, Robert T. and Raney, James L.; Floating Point Arithmetic With 84-bit Numbers, Communications of the ACM, Volume 1/ Number 1/ January 1964.

9.3 Low Core Storage Location

FLOAT uses the low core location $C5 through $D7; they are assigned as follows:

Location Name	Location	Use
SIGN	$C7	Sign register-sign of floating point number.
ERRORS	$C8	Error bits to indicate exponent overflow, divide fault, exponent underflow-which are used by IFALT to check for these floating point errors.
SHIFCT	$D1	Shift counter.
P	$D2	Pseudo program counter which is used by FLOAT to know where it is in the user's program when it wants to pick up more parameters or return.
INDEX	$D3	Effective address of parameter
RELADR	$D4	Absolute/relative indication RELADR { =1 relative address =0 absolute address
OPCNT	$D5	Op counter - keeps track of which op code is being processed.
OPCODE	$D6	Op code holder - holds op codes in the call.
QS	$D7	Saves Q register upon entry into FLOAT.
G G+1	$C5 $C6	Pseudo accumulator - the two words of the floating point number are placed here. The pseudo accumulator is broken up into C, D and DELTA.
C	$CD	Most significant bits of the coefficient of the floating point number.
D	$CE	Least significant bits of coefficient of floating point number.

Location Name	Location	Use
DELTA	$D0	Exponent of floating point number.
F	$C9	1. Address of parameter or 2. First word of a {two-word} floating point number - used in the same manner as G of the pseudo accumulator is used when two operands are needed, e.g. for an add. In this case, F and F+1 are broken up into A, B and BETA.
F+1	$CA	The second word of a {two-word} floating point number.
A	$CB	The most significant bits of the coefficient of the floating point number.
B	$CC	The least significant bits of the coefficient of the floation point number.
BETA	$CF	Exponent of floating point number.

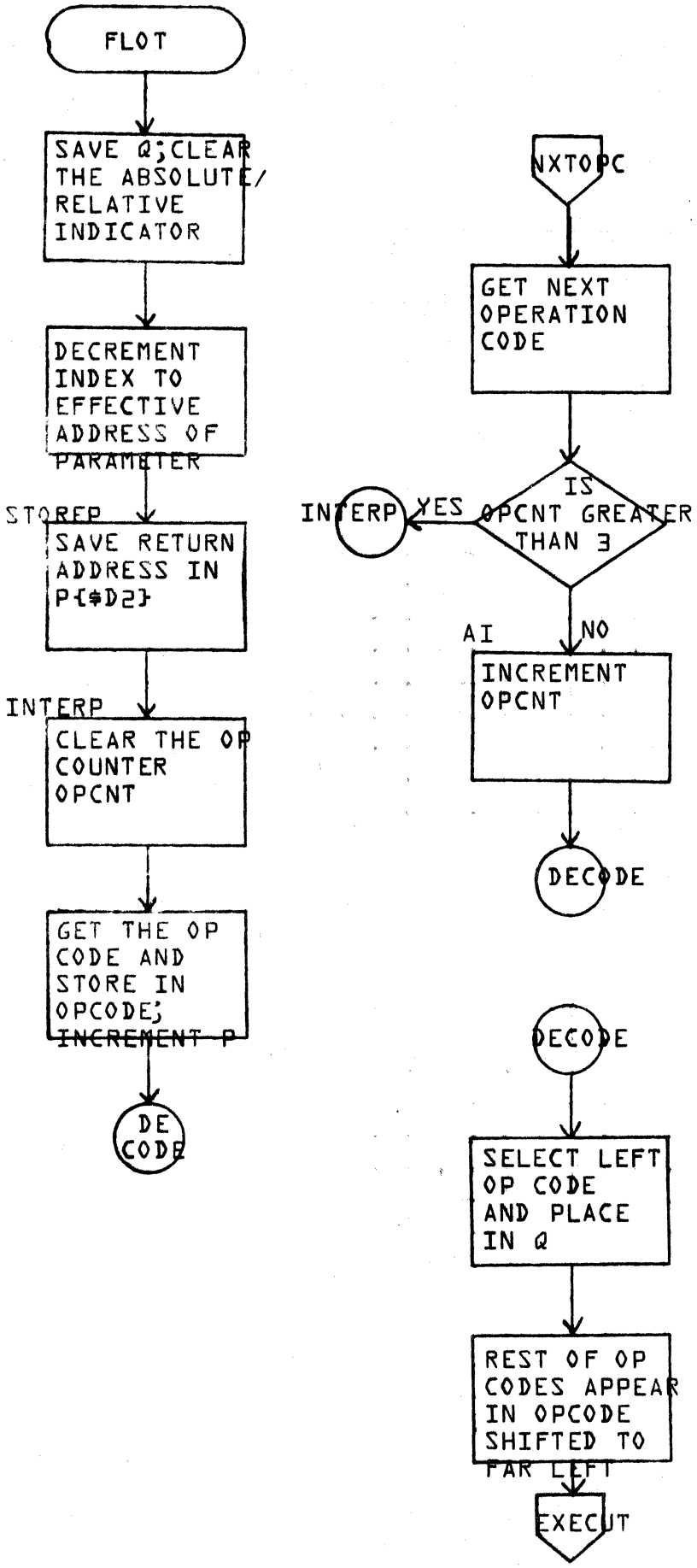
9.4 Entry Points and Externals

The routine FLOAT has one entry point, FLOT, and no externals.

9.5 Flowchart of FLOAT.

The following abbreviations are used:

MSB = most significant bits
LSB = least significant bits



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

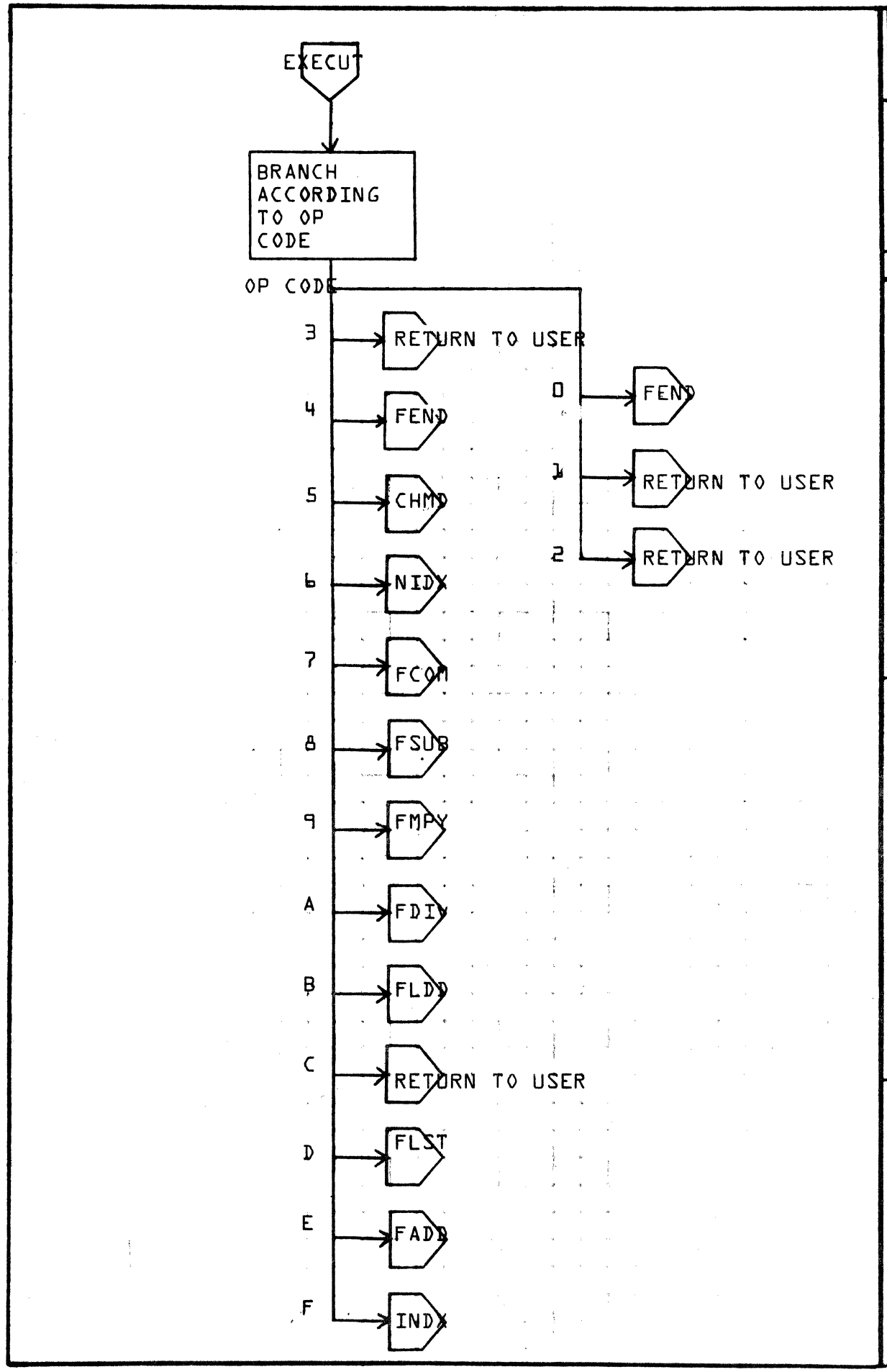
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 1 OF 39
ISSUE DATE
NUMBER
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

A B C D



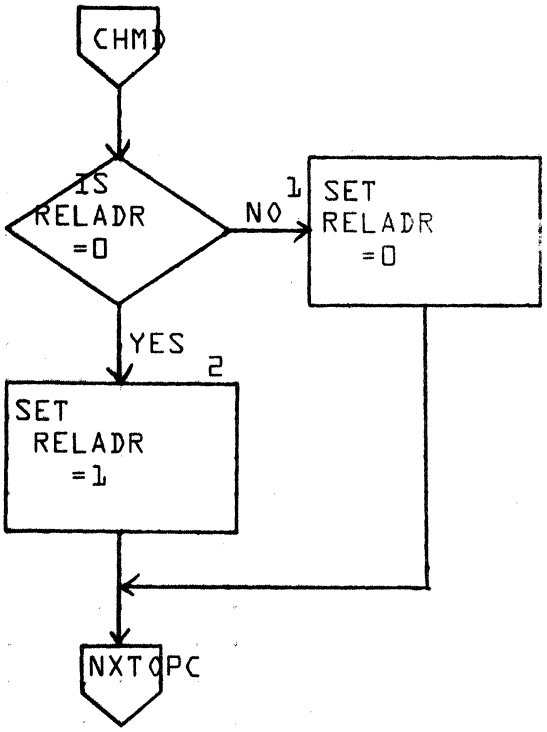
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS
DOCUMENT TITLE	FLOAT
MACH. TYPE	1700
PAGE	2 OF 39
ISSUE DATE	
NUMBER	
DRAWN BY	
DATE	
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

REV	APPROVED	DATE

OPERATION CODE =5
CHANGE MODE OF OPERATION (CHMD)

1. RELATIVE ADDRESS
2. ABSOLUTE ADDRESS



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS IMS
MACH. TYPE 1700
DOCUMENT TITLE FLOAT
PAGE 3 OF 39
ISSUE DATE
NUMBER
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

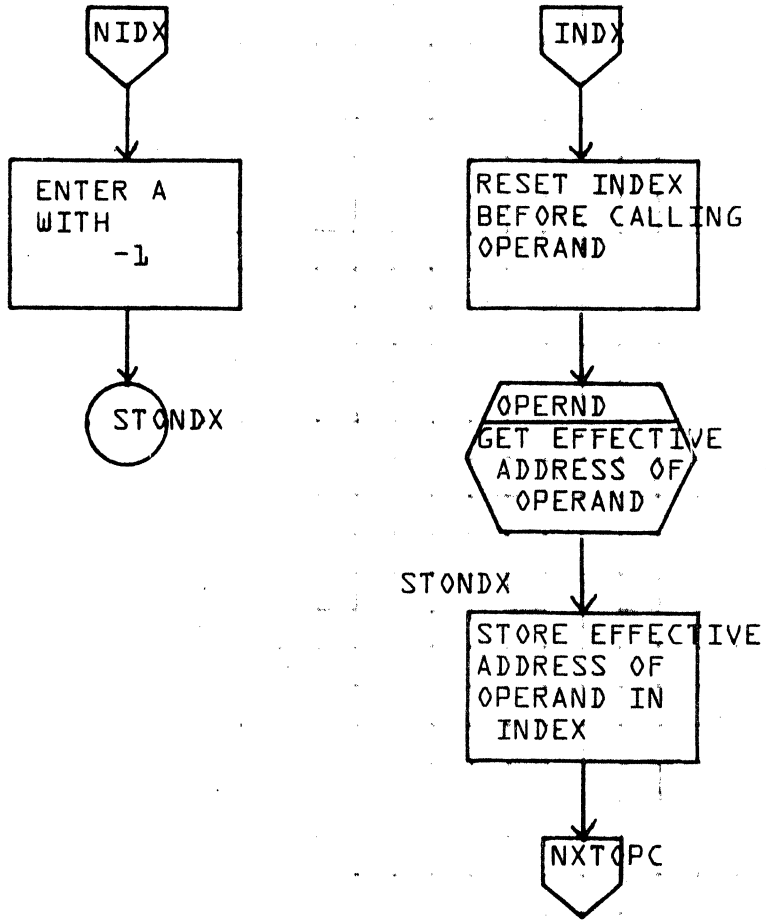
A B C D

OPERATION CODE = 6

NO INDEX {NIDX}

OPERATION CODE = F

INDEX {INDX}



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	FLOAT
PAGE	4 OF 29
ISSUE DATE	
NUMBER	
DRAWN BY	
DATE	

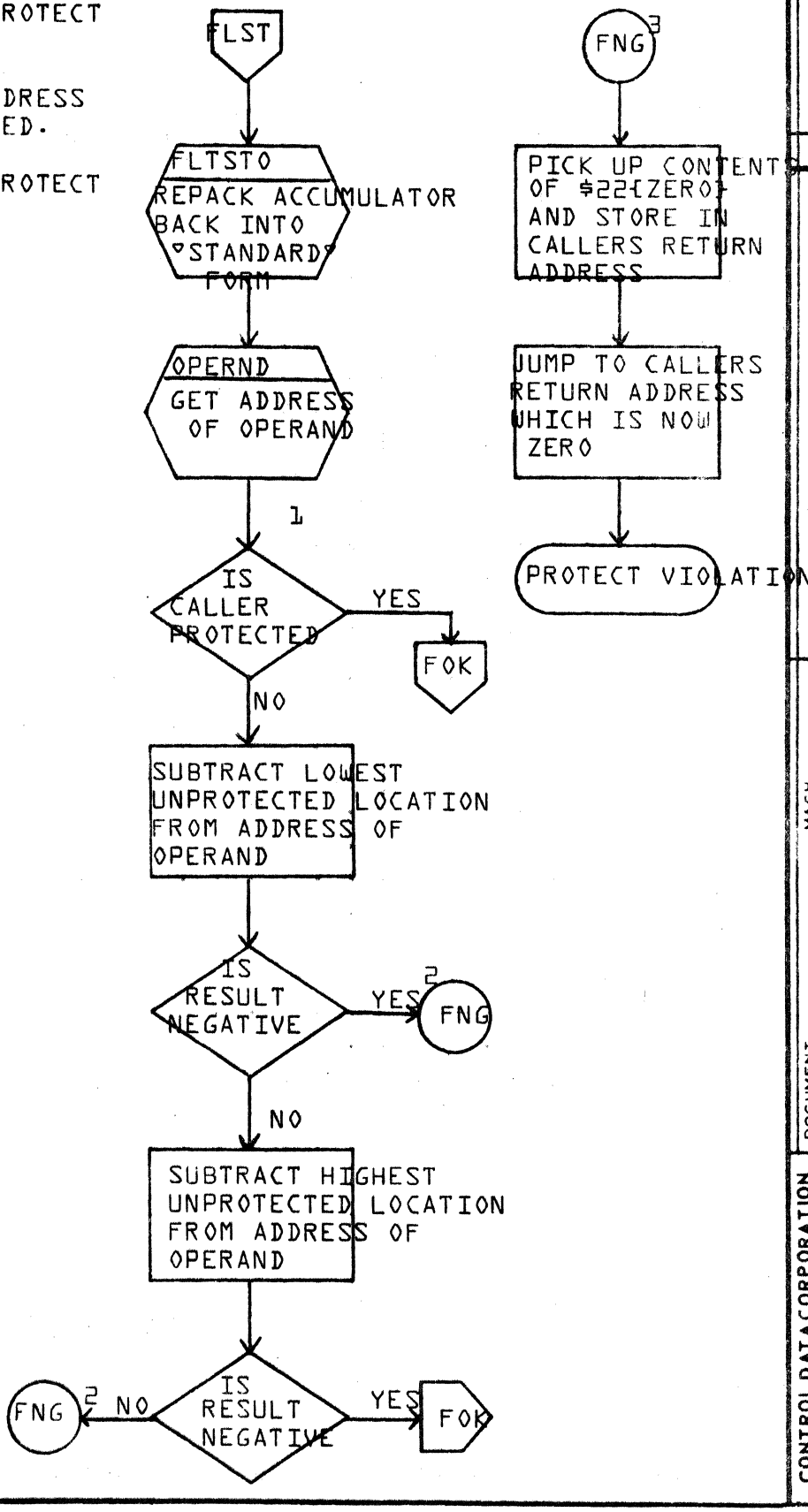
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

OPERATION CODE = D

STORE ACCUMULATOR (FLST)

1. TEST FOR PROTECT VIOLATION.
2. STORAGE ADDRESS IS PROTECTED.
3. SIMULATE PROTECT VIOLATION.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

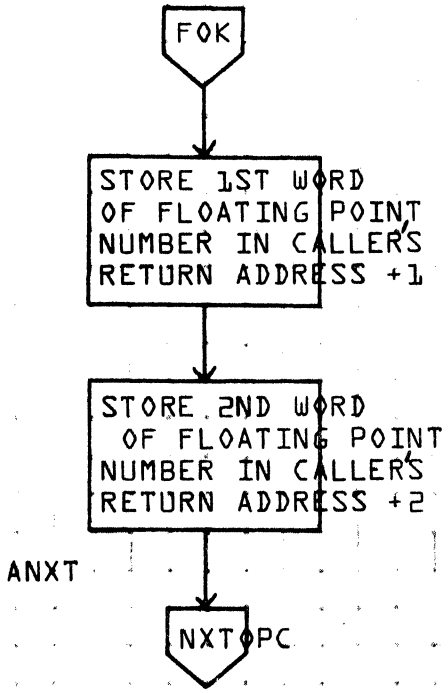
DOCUMENT CLASS: IMS
DOCUMENT TITLE: FLOAT
MACH. TYPE: 1200
PAGE 5 OF 39
ISSUE DATE
NUMBER
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV | APPROVED | DATE

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS	MACH. TYPE	170
DOCUMENT TITLE	FLOAT		
		PAGE 6 OF 39	
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

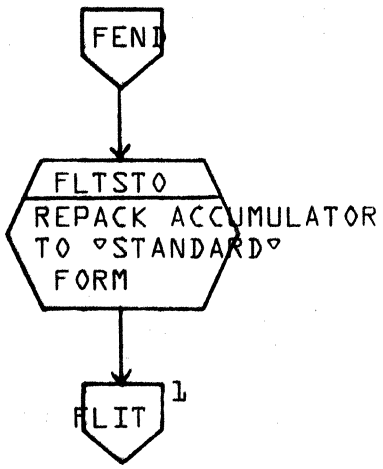
REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

OPERATION CODE 0,4

END OF CALLING SEQUENCE (FEND)

1. RESTORE Q AND
   RETURN TO USER.



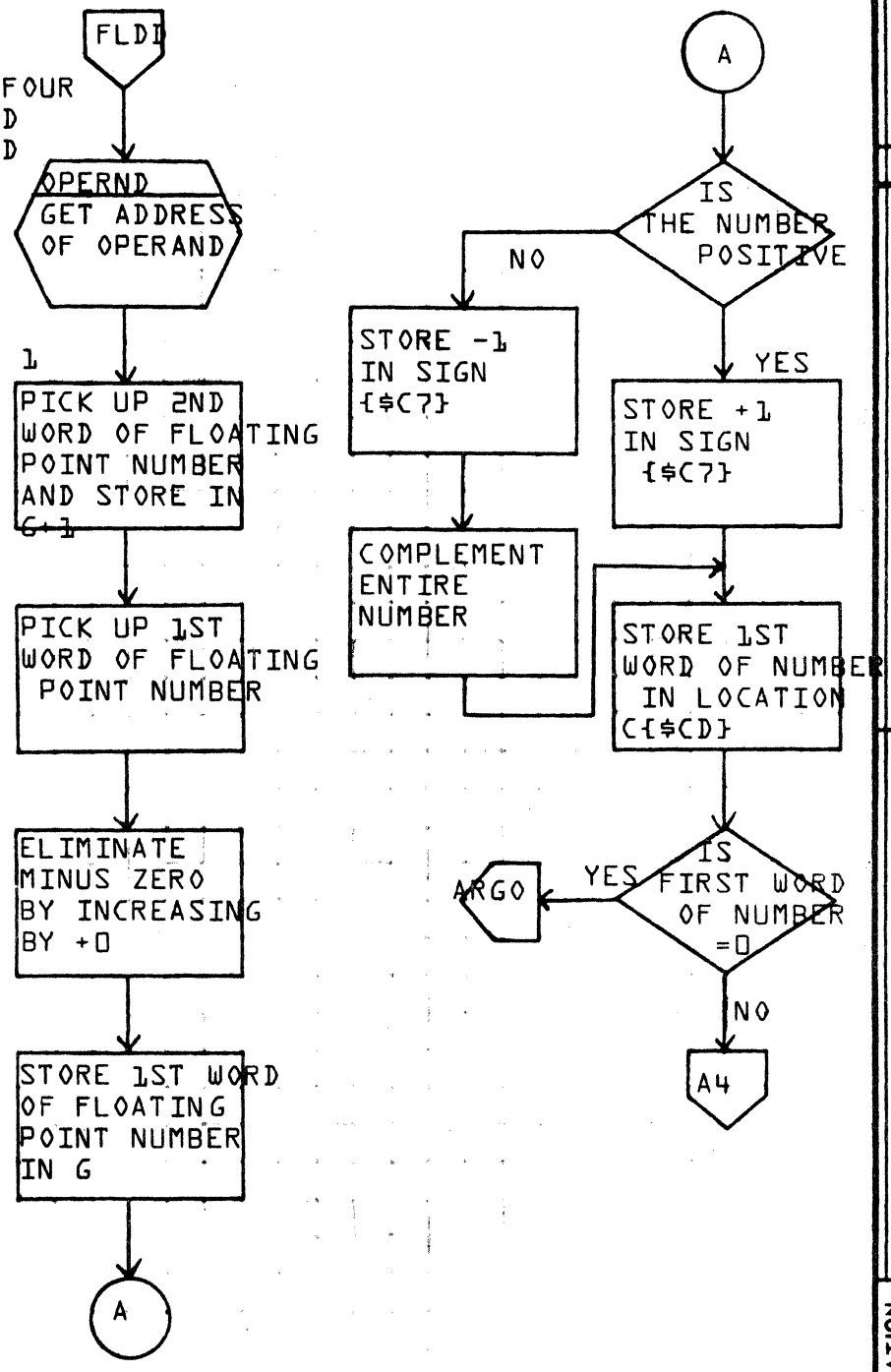
CONTROL DATA CORPORATION SOFTWARE DOCUMENT				REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.			
	DOCUMENT TITLE: FLOAT		PROJECT MGR.			
		PAGE 7 OF 39	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

A B C D

OPERATION CODE = B

FLOATING LOAD {FLDD}

1. FOLLOWING FOUR BOXES LOAD THE OPERAND INTO G AND G+1.



REV	APPROVED	DATE

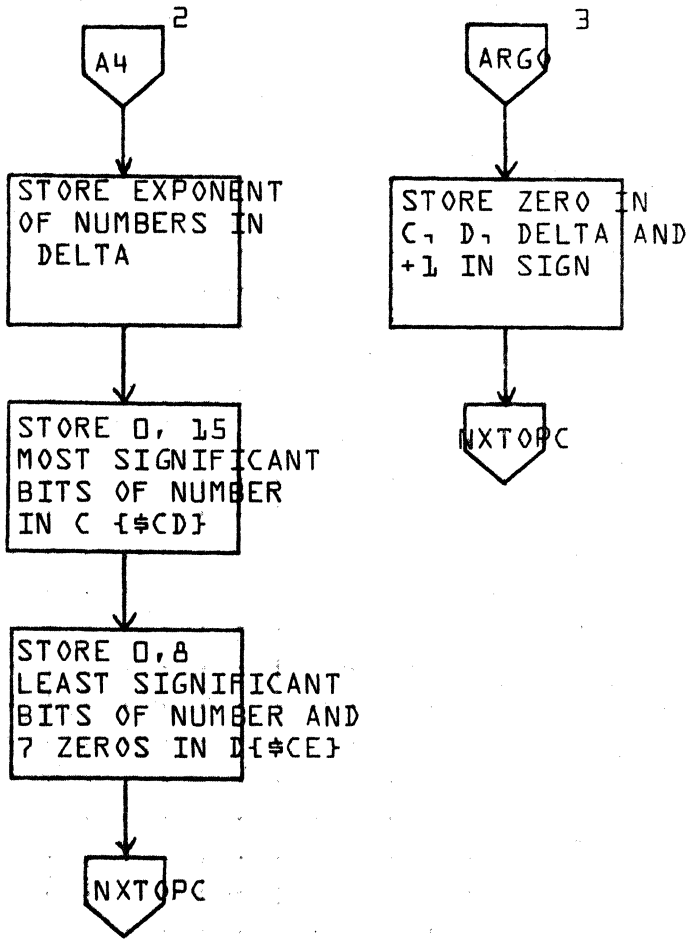
PROJECT NO. | PROJECT MGR. | PROJECT NAME | TASK NO. | TASK NAME

DOCUMENT CLASS: IMS | MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT | PAGE 8 OF 39
NUMBER | ISSUE DATE
DRAWN BY | DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

2. UNPACK OPERAND INTO C, D, AND DELTA.

3. FLOATING POINT NUMBER IS ASSUMED TO BE ZERO SINCE FIRST WORD IS ZERO



REV	APPROVED	DATE

PROJECT NO. | PROJECT MGR. | PROJECT NAME | TASK NO. | TASK NAME

DOCUMENT CLASS: IMS | MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 9 OF 39
ISSUE DATE
NUMBER | DRAWN BY | DATE

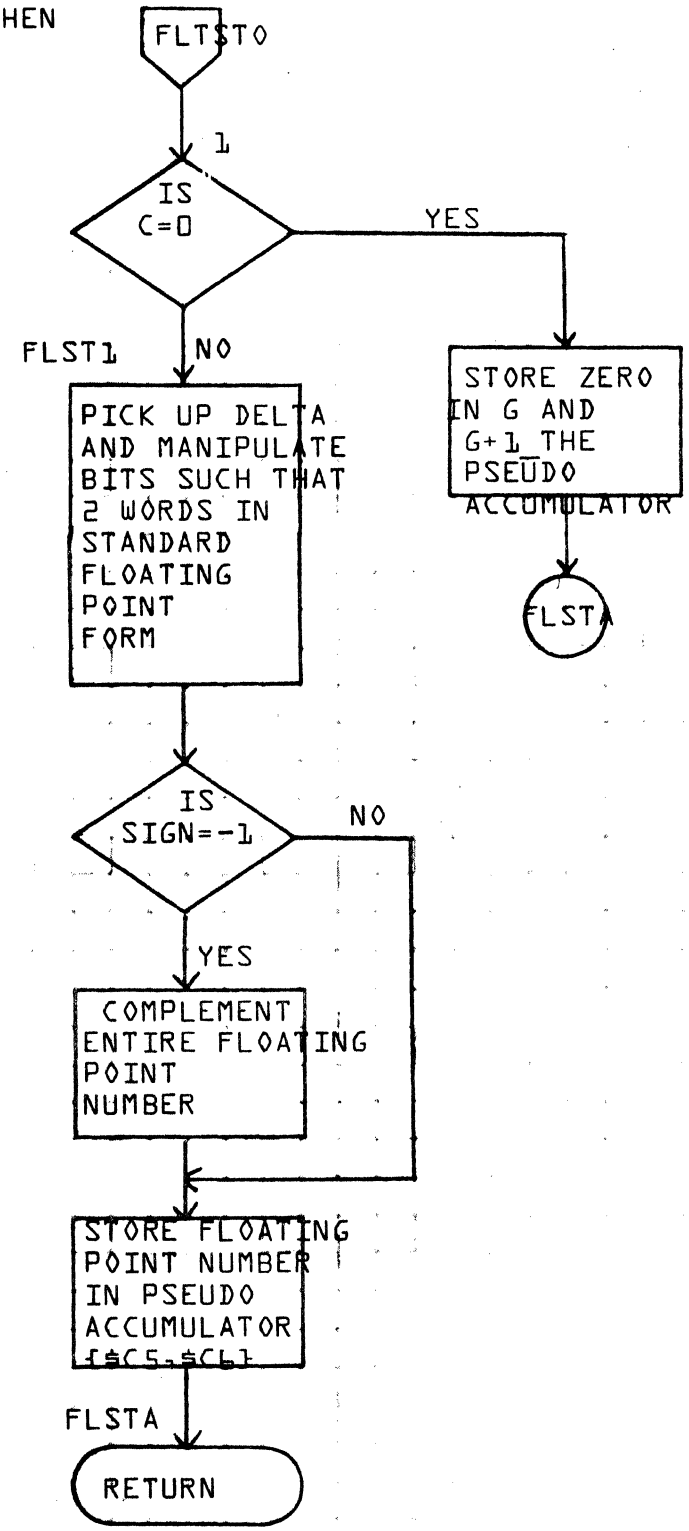
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

REPACKING ACCUMULATOR

1. IF C IS ZERO, THEN THE PSEUDO ACCUMULATOR IS ZERO



REV	APPROVED	DATE			
PROJECT NO.		PROJECT MGR.		PROJECT NAME	
TASK NO.		TASK NAME			
DOCUMENT CLASS	IMS	MACH. TYPE	1700		
DOCUMENT TITLE	FLOAT			PAGE 10 OF 39	
NUMBER		ISSUE DATE			
DRAWN BY		DATE			

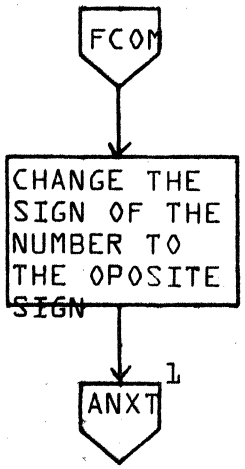
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE [X]
FLOWCHART [ ]
DECISION TABLE [ ]
OTHER [ ]

OPERATION CODE = 7

FLOATING COMPLEMENT (FCOM)

1. A JUMP TO THE LABEL ANXT TAKES YOU TO A JUMP TO THE LABEL NXTOPC-NEXT OP CODE



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 11 OF 39
ISSUE DATE
NUMBER
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER

PRINTED IN U.S.A.

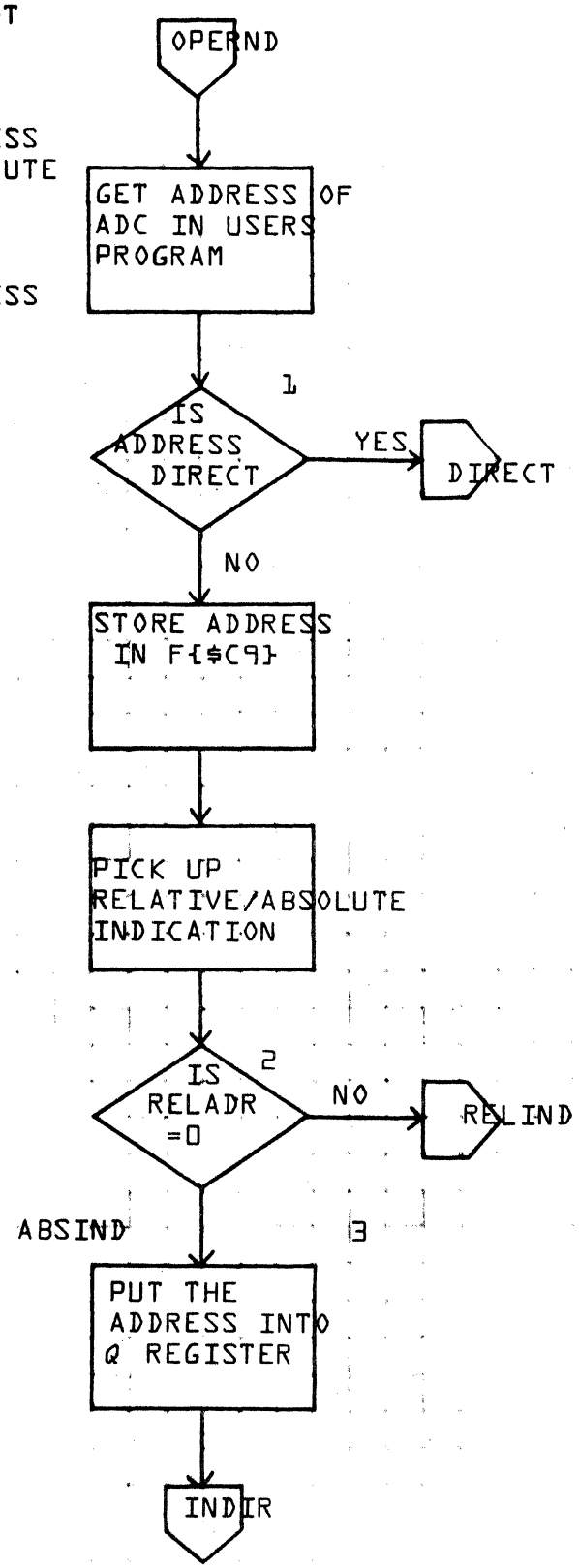
AA1388 (FORMERLY CA127-1)

GET EFFECTIVE ADDRESS OF NEXT OPERAND

1. ADDRESS IS DIRECT IF INDIRECT BIT (BIT 15) IS NOT SET.

2. RELADR=1 IF RELATIVE ADDRESS AND=0 IF ABSOLUTE ADDRESS.

3. ABSOLUTE AND INDIRECT ADDRESS



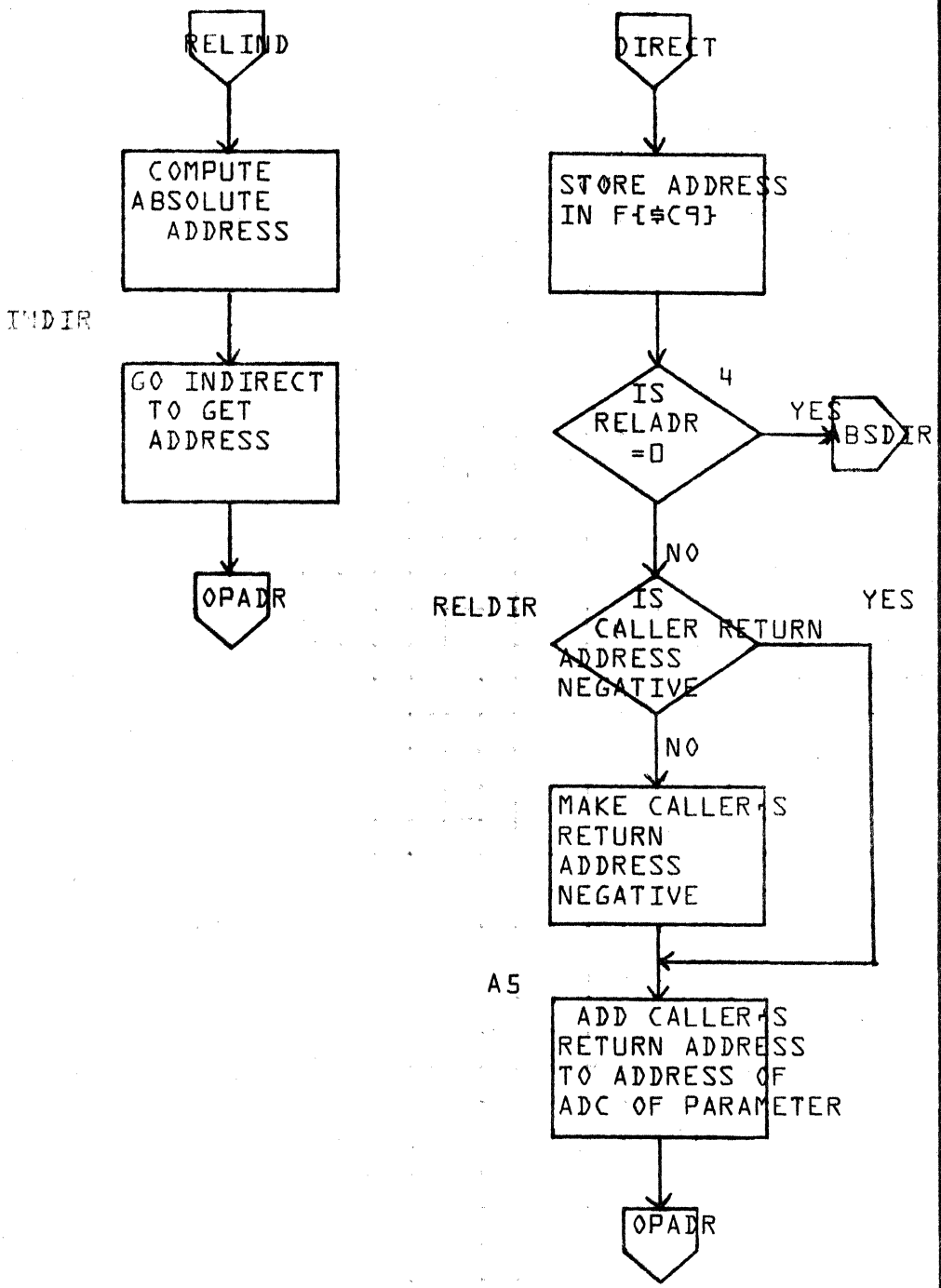
REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS	IMS
DOCUMENT TITLE	FLOAT
NUMBER	
DRAWN BY	
MACH. TYPE	1700
PAGE	12 OF 39
ISSUE DATE	
DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER

4. IS ADDRESS ABSOLUTE?



REV	APPROVED	DATE

PROJECT NO. | PROJECT MGR. | PROJECT NAME | TASK NO. | TASK NAME

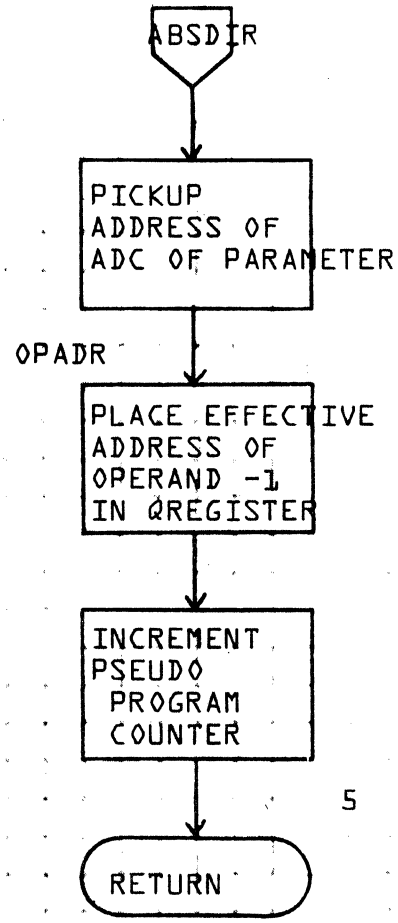
DOCUMENT CLASS: IMS | MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT | PAGE 13 OF 39
NUMBER | ISSUE DATE
DRAWN BY | DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

5. RETURN WITH EFFECTIVE ADDRESS OF OPERAND-1 IN QREGISTER.



REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

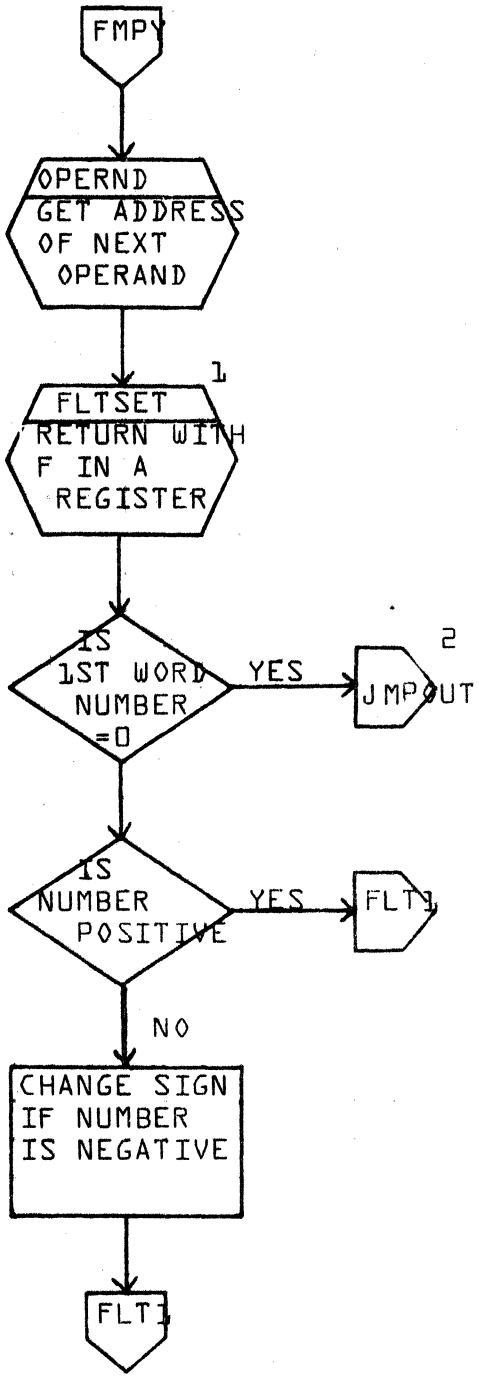
DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	FLOAT
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	
NUMBER	
ISSUE DATE	
DRAWN BY	
DATE	

OPERATION CODE = 9

FLOATING MULTIPLY (FMPY)

1. GETS THE FLOATING POINT NUMBER AND STORES IN F AND F+1; THEN UNPACKS THE NUMBER INTO
   A=0, 15MSB
   B=0, 8LSB, 7 ZEROS
   BETA=EXPONENT

2. A JMP TO JMPOUT CAUSES A JUMP TO ARGO.



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

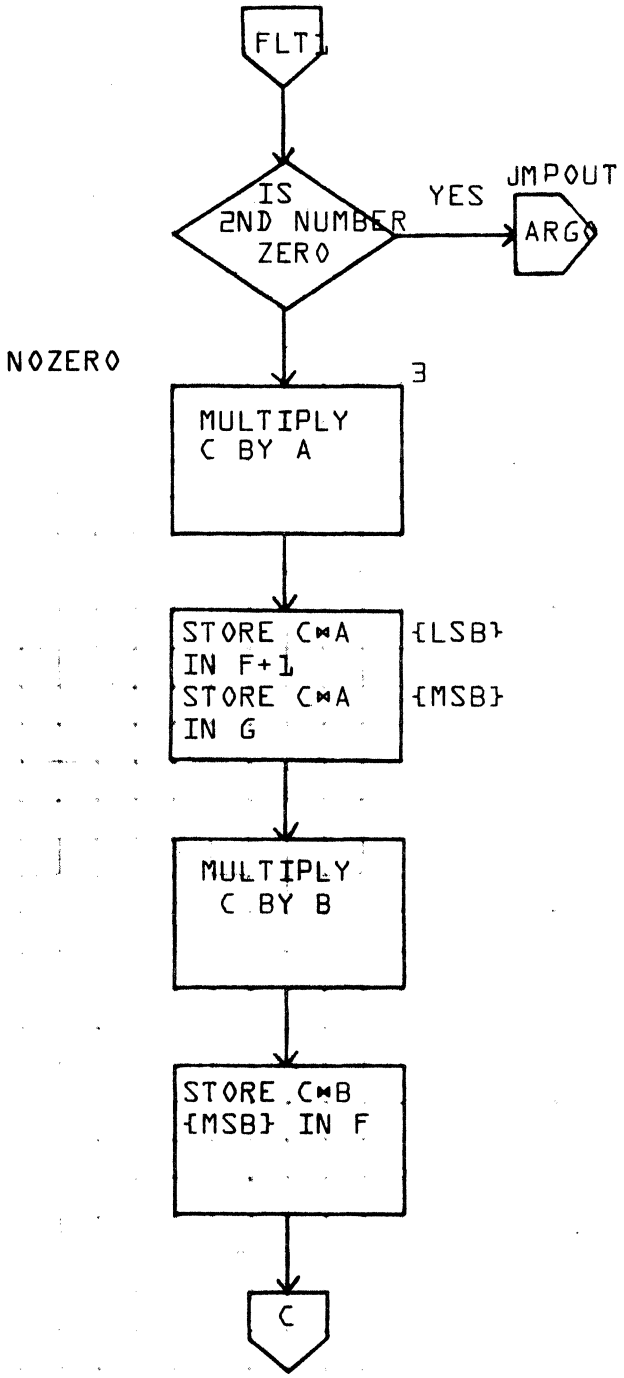
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 35 OF 39
NUMBER
ISSUE DATE
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

3. MULTIPLY THE MSB OF ONE NUMBER BY THE MSB OF THE OTHER NUMBER



REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	FLOAT
PAGE 16 OF 39	
NUMBER	
ISSUE DATE	
DRAWN BY	
DATE	
PROJECT NO.	
PROJECT MGR	
PROJECT NAME	
TASK NO.	
TASK NAME	

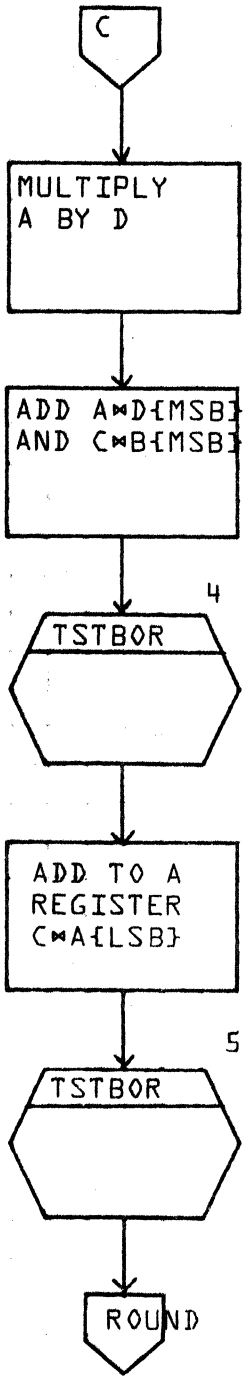
4. UPON ENTRY THE A AND Q REGISTERS ARE:

   A={A⋈D{MSB}}+{C⋈B{MSB}}
   Q=C⋈A{MSB}

   IN THIS SECTION OF CODE, IF A IS NEGATIVE, CARRY INTO G; IF Q IS NEGATIVE, END AROUND.

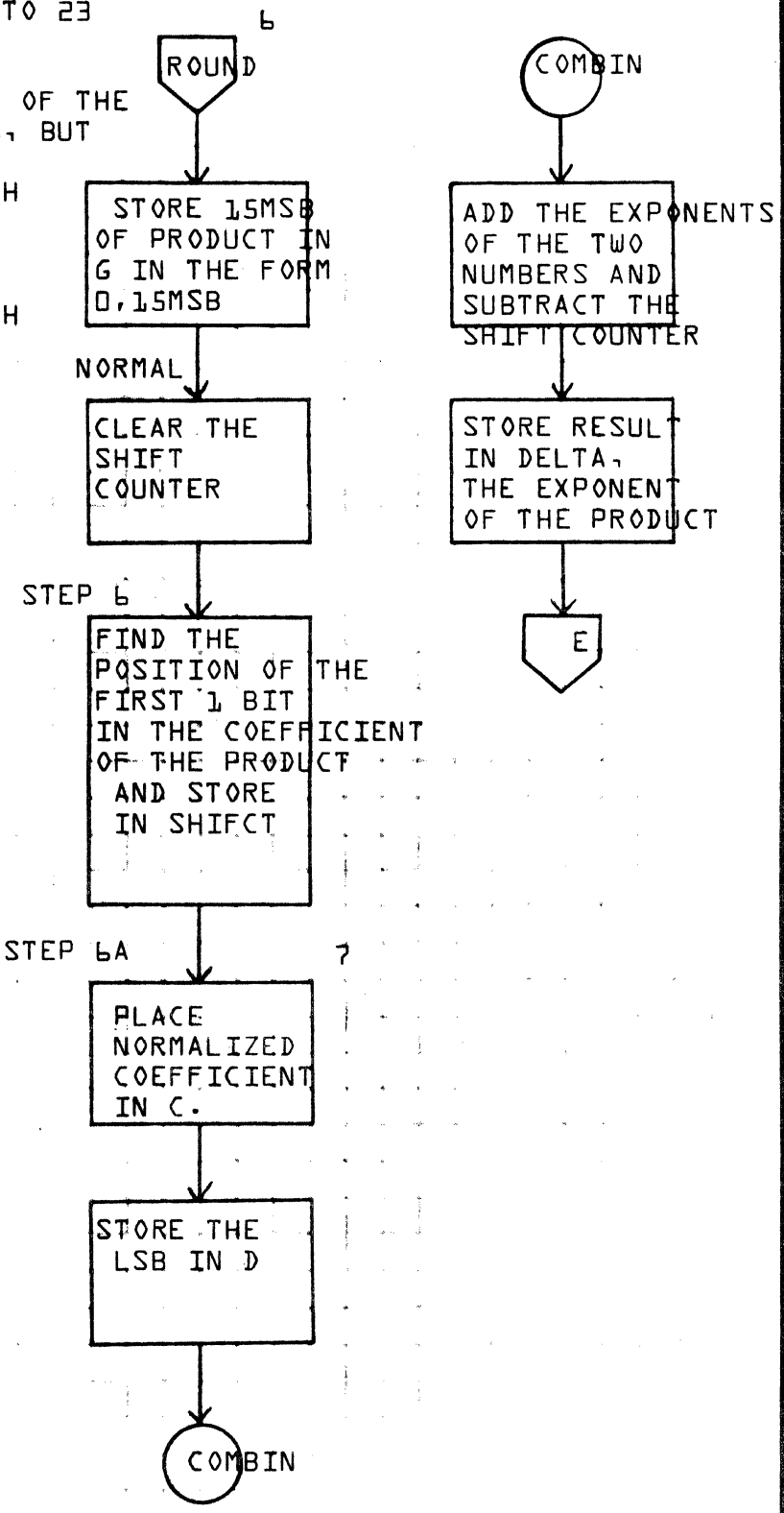
5. SAME AS #4 ABOVE EXCEPT THAT

   A=A⋈D{MSB}
   +C⋈B{MSB}
   +C⋈A{LSB}



6. THIS SECTION ROUNDS AND TRUNCATES TO 23 BITS.

7. COEFFICIENT IS OF THE FORM 0, 15MSB, BUT THE 15MSB ARE NORMALIZED SUCH THAT A ONE APPEARS IN THE FIRST LOCATION OF THE MSB WITH SHIFCT KEEPING TRACK OF HOW MANY PLACES HAVE BEEN SHIFTED.



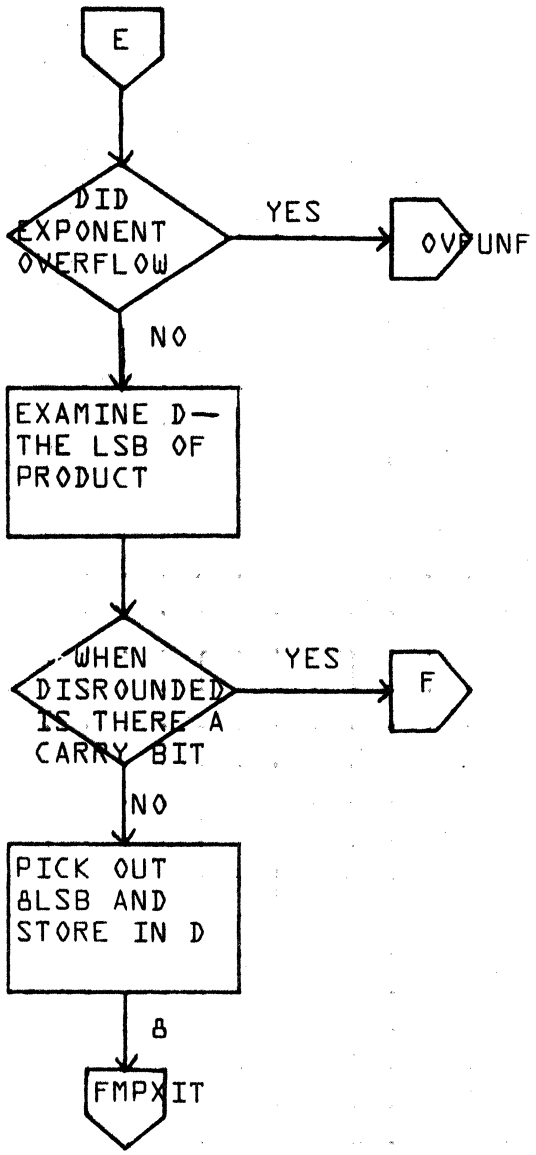
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 18 OF 39
ISSUE DATE
NUMBER
DATE
DRAWN BY

8. A JUMP TO FMPXIT THEN CAUSES A JUMP TO NXTOPC {NEXT OP CODE}.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

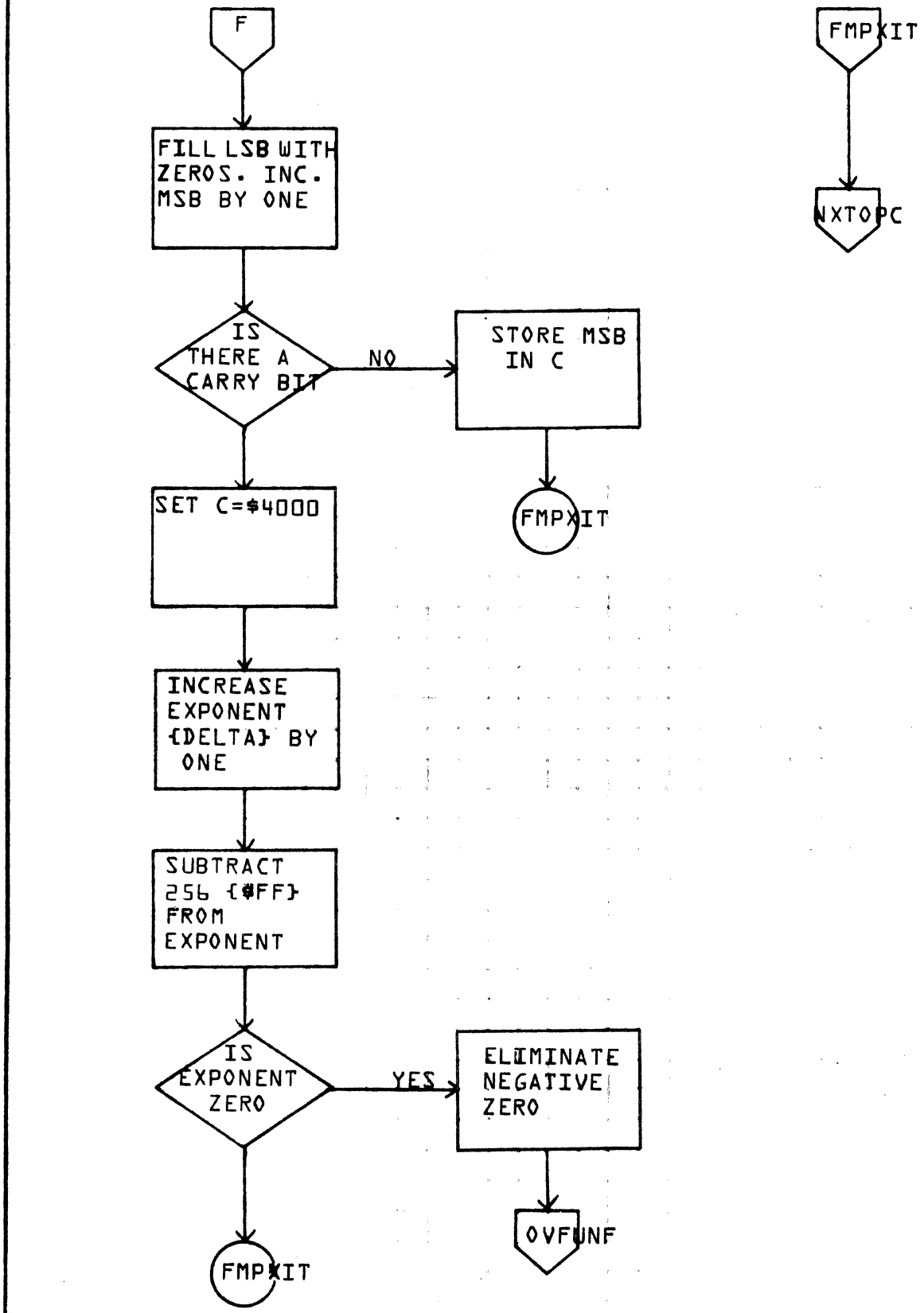
DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	FLOAT	PAGE 9 OF 39	
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)



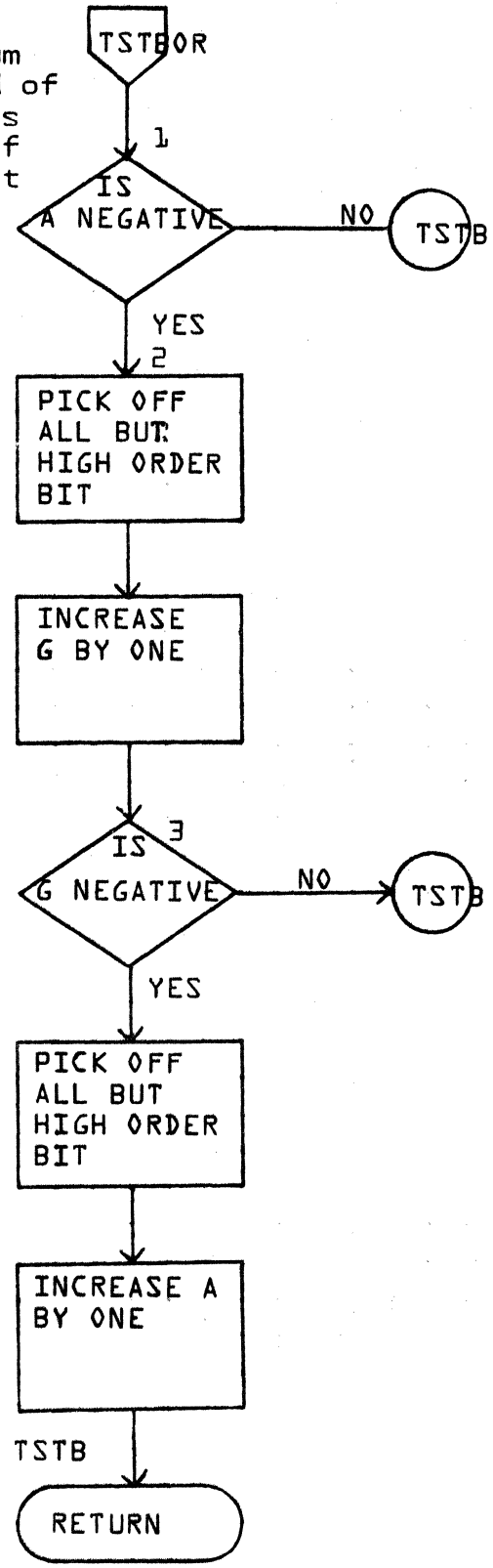
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	FLOAT	PAGE	20 of 29
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

1. A contains the sum of the first word of one argument times the second word of the other argument and vice versa.
2. If A is negative, carry into G.
3. If G is negative, end around.



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 1 OF 39
ISSUE DATE
NUMBER
DRAWN BY
DATE

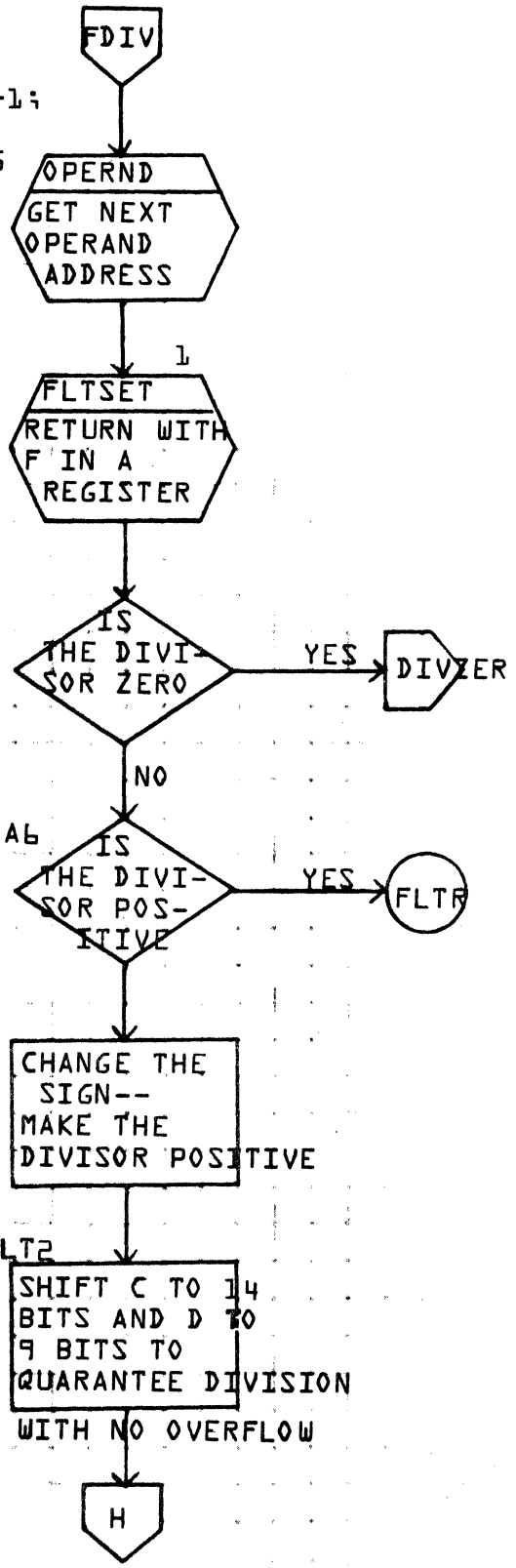
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

Operation Code = A
Floating Divide (FDIV)

1. GETS THE FLOATING POINT NUMBER AND STORES IN F AND F+1; THEN UNPACKS THE NUMBER INTO A=0,15 MSB
B=0,8LSB,7zeroes
BETA=EXPONENT



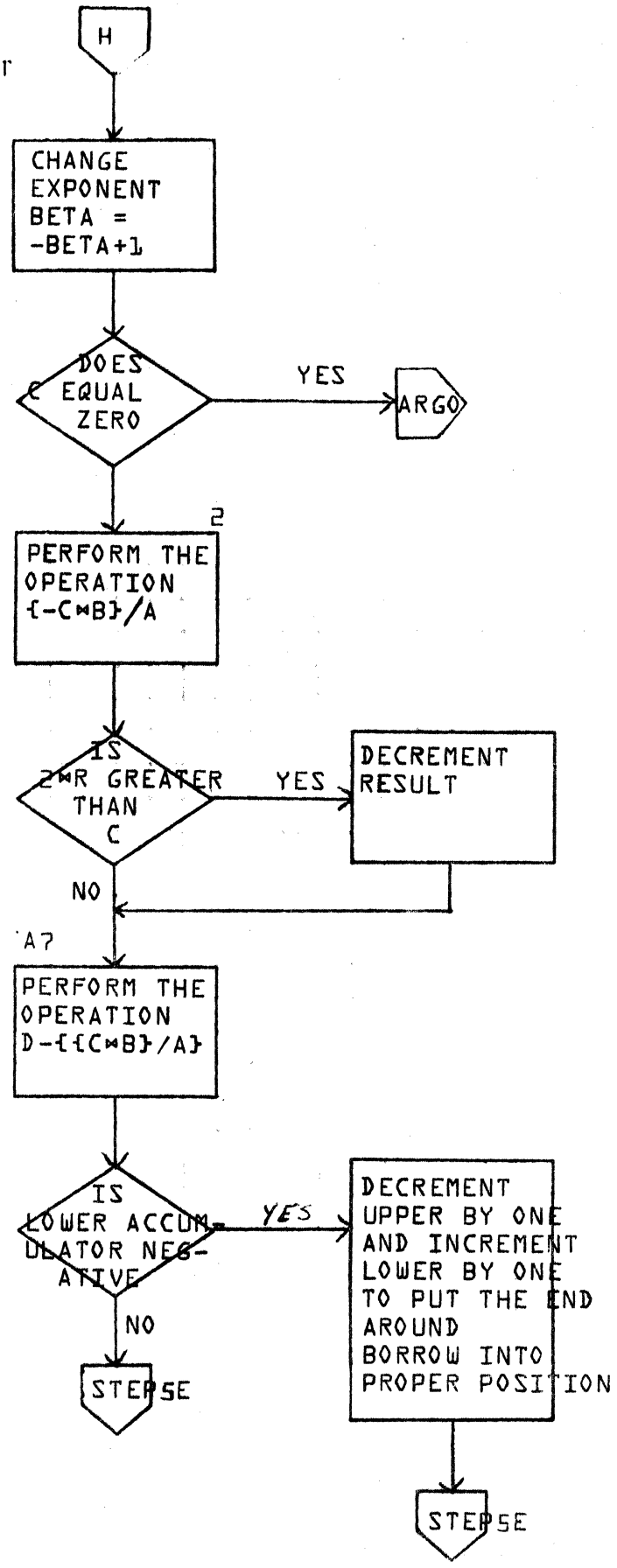
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 22 OF 39
NUMBER
ISSUE DATE
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR
PROJECT NAME
TASK NO.
TASK NAME

REV | APPROVED | DATE

2. THE REMAINDER MUST BE POSITIVE



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	FLOAT
PAGE	23 OF 39
ISSUE DATE	
NUMBER	
DRAWN BY	
DATE	

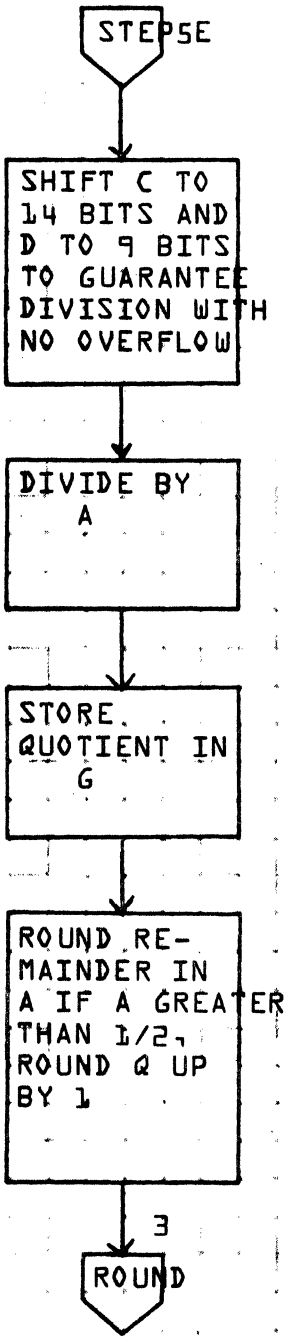
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

- SAMPLE CODE
- FLOWCHART [X]
- DECISION TABLE
- OTHER

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

3. ROUND, TRUNCATE AND NORMALIZE THE RESULTS.



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

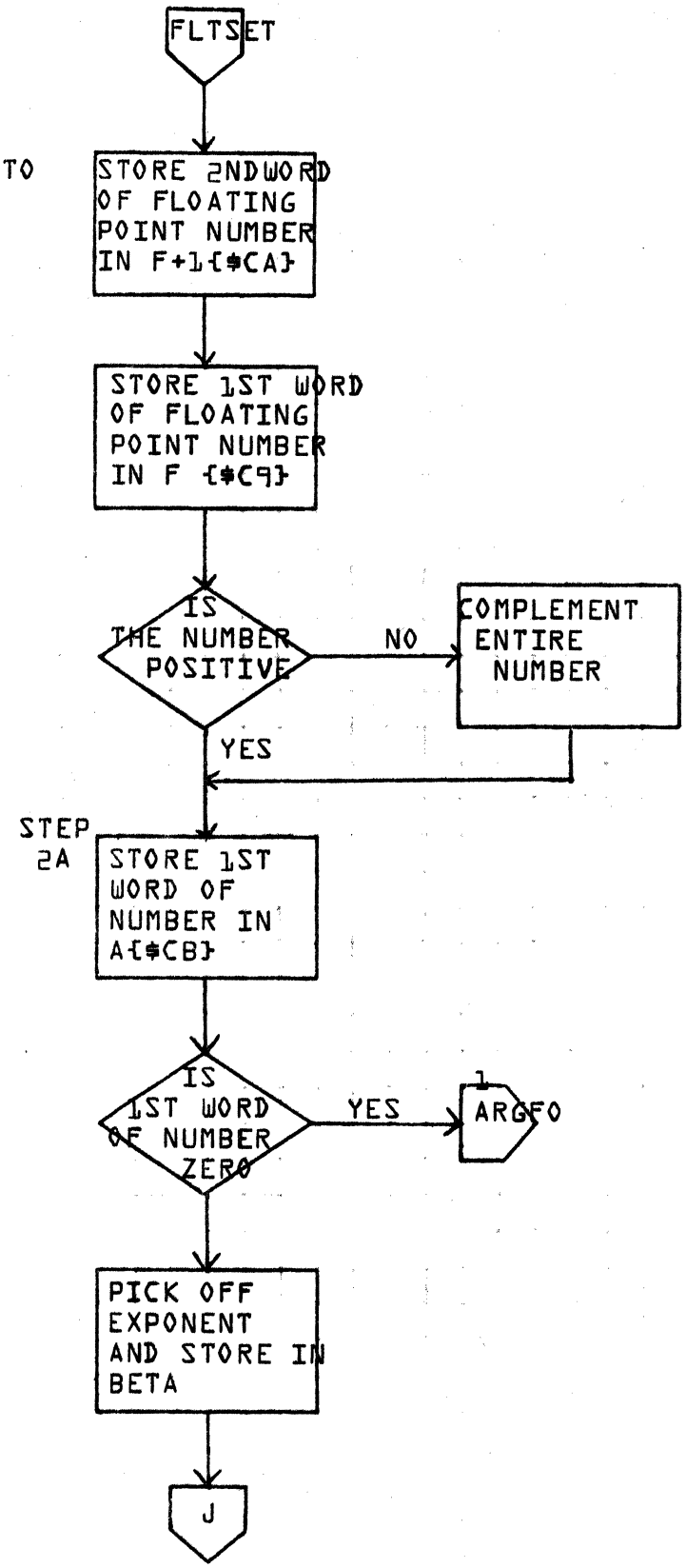
DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	FLOAT	PAGE	24 OF 39
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

PRINTED IN U.S.A.

1. AT ARGFO, THE ENTIRE NUMBER IS SET EQUAL TO +0
   A=+0
   B=+0
   BETA=+1



REV	APPROVED	DATE

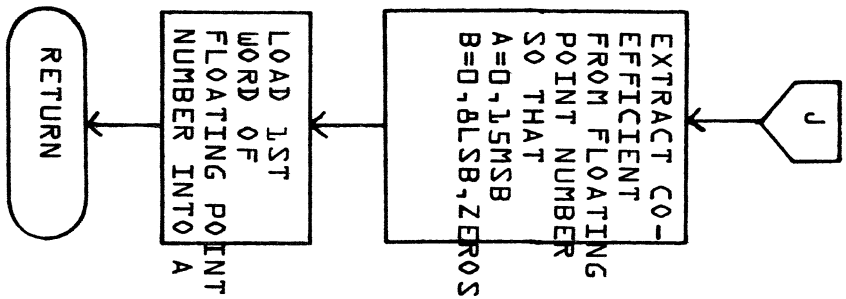
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS	IMS
DOCUMENT TITLE	FLOAT
NUMBER	
DRAWN BY	
MACH. TYPE	1700
PAGE	25 OF 39
ISSUE DATE	
DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

PRINTED IN U.S.A.

AA1388 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	FLOAT PAGE 26 OF 39	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

DIVIDE CHECK WHEN DIVIDING BY ZERO

CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒	FLOAT	PAGE 27 OF 39	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

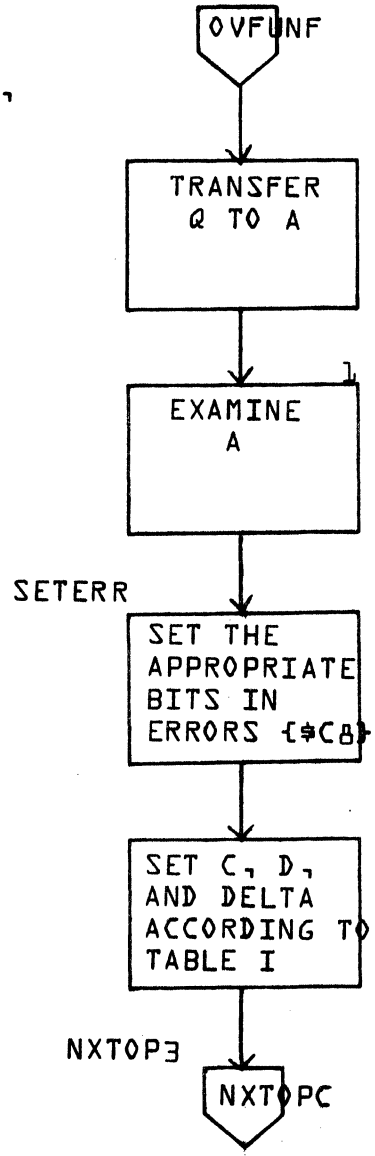
1A1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

OVERFLOW OR UNDERFLOW

1. IF A IS POSITIVE, AN OVERFLOW OCCURRED.

   IF A IS NOT POSITIVE, AN UNDERFLOW OCCURRED.



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	FLOAT
PAGE	28 OF 39
ISSUE DATE	
NUMBER	
DRAWN BY	
DATE	

CONTROL DATA CORPORATION

SOFTWARE DOCUMENT

- SAMPLE CODE
- FLOWCHART [X]
- DECISION TABLE
- OTHER

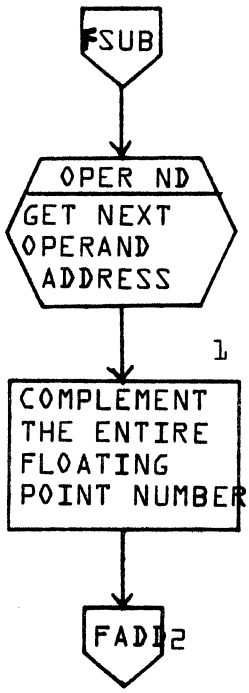
	C	D	DELTA
EXPONENT OVERFLOW	$7FFF	$7F80	$7F
DIVIDE FAULT	$7FFF	$7F80	$7F
EXPONENT UNDERFLOW	$4000	$0000	-$7F

TABLE I

OPERATION CODE = 8

FLOATING SUBTRACT (FSUB)

1. CHANGE THE SIGN BEFORE ENTERING THE ADD ROUTINE.



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

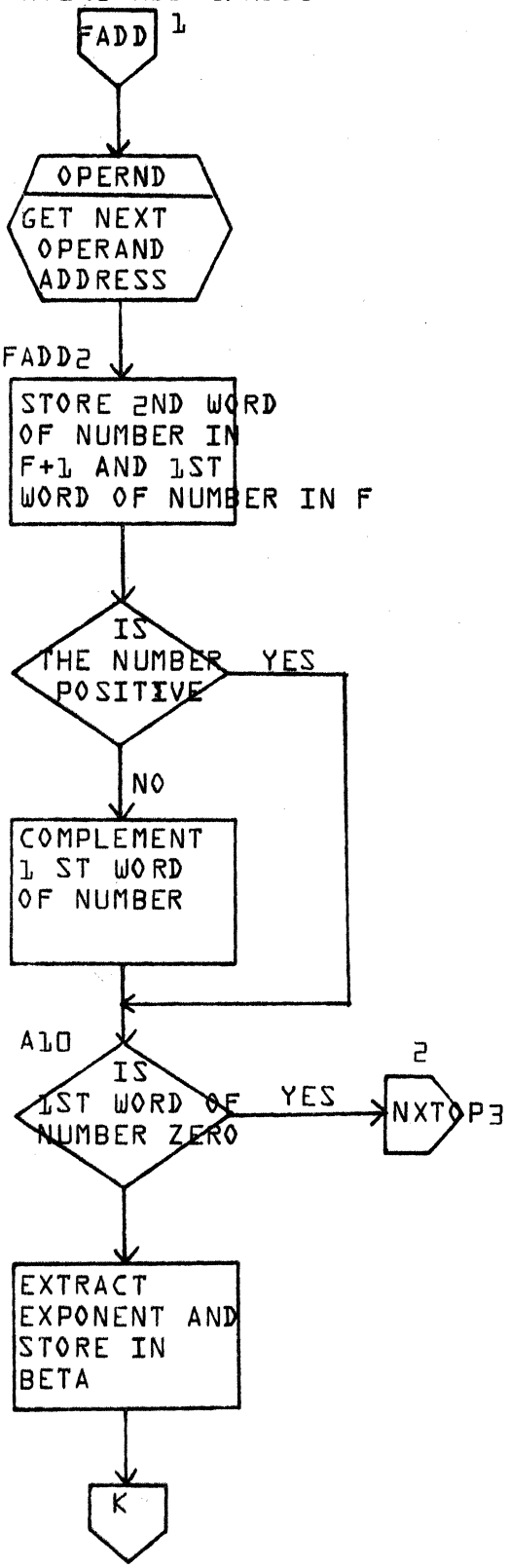
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 30 OF 39
ISSUE DATE
NUMBER
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☒
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

OPERATION CODE = E

FLOATING ADD (FADD)



1. THIS SECTION ADDS THE FLOATING POINT NUMBERS IN G AND F AND PLACES THE RESULT IN G.

2. JUMP OUT - BACK TO NXTOPC, GET NEXT OP CODE.

REV	APPROVED	DATE

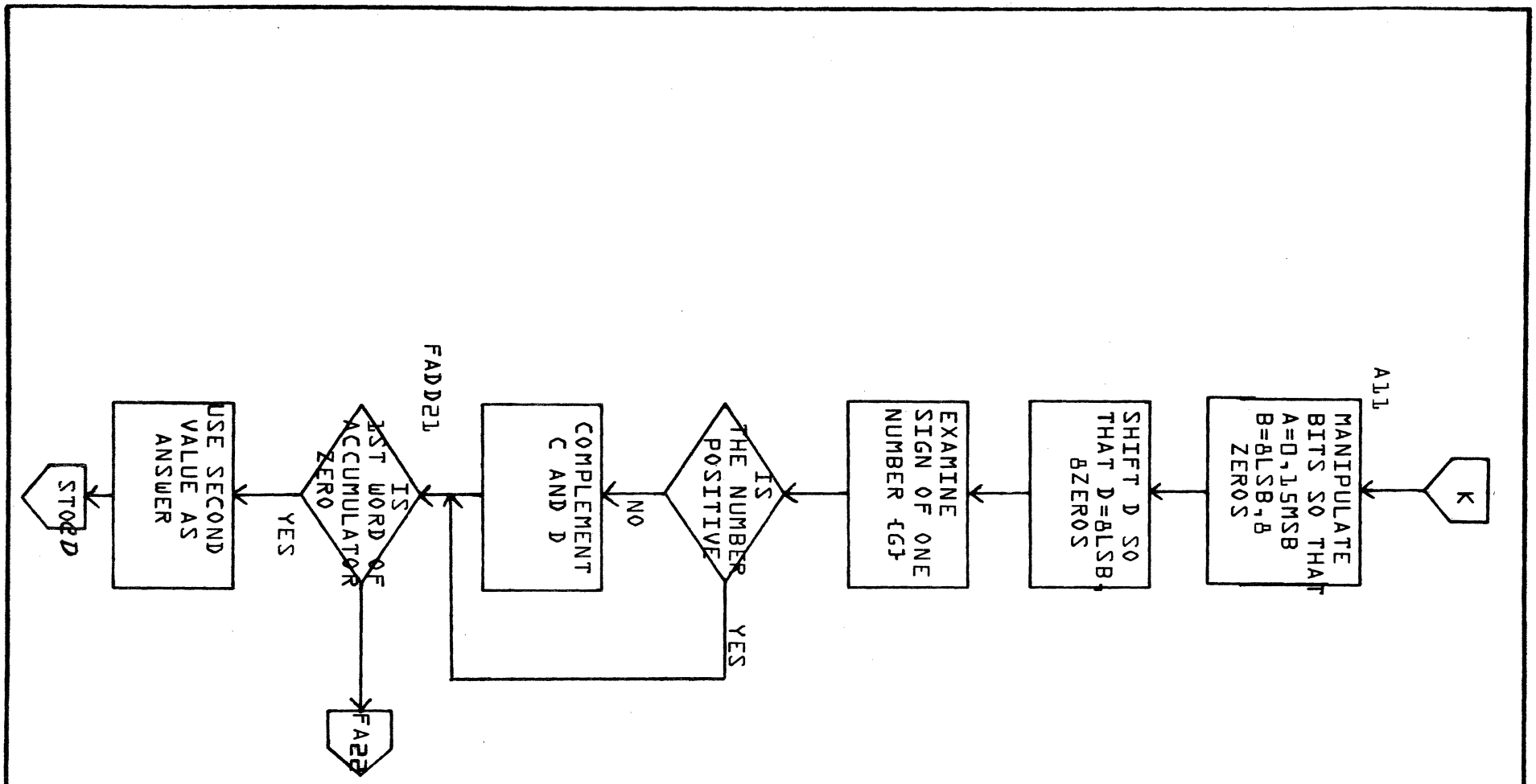
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS IMS
MACH TYPE 1700
DOCUMENT TITLE FLOAT
PAGE 31 OF 39
ISSUE DATE
NUMBER
DRAWN BY
DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER

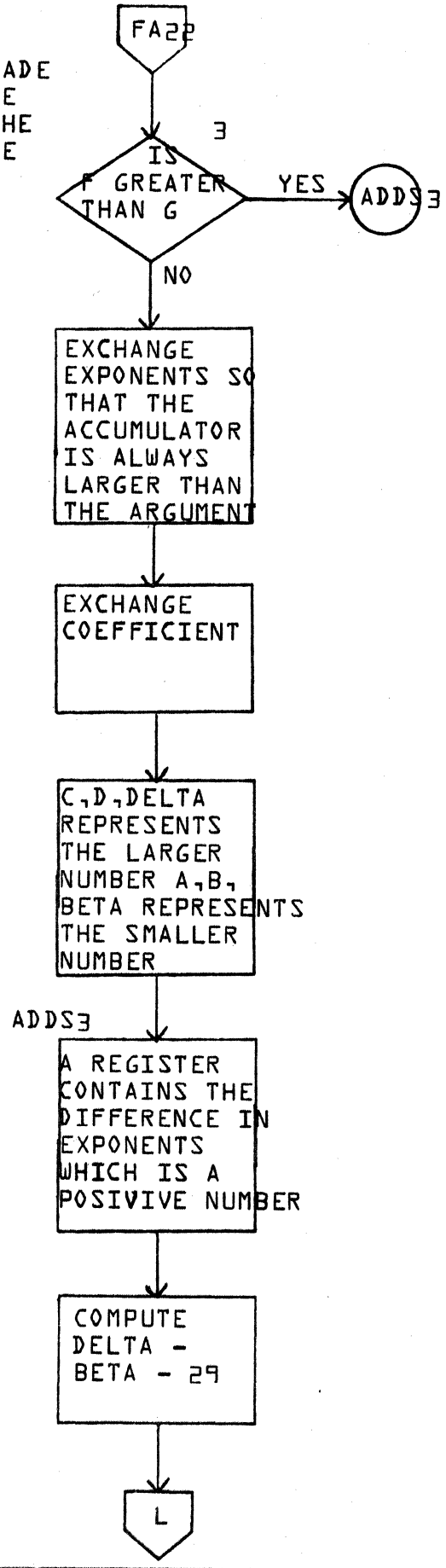
PRINTED IN U.S.A.

CA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☒	FLOAT	PAGE 32 OF 39	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

3. THIS CHECK IS MADE BY EXAMINING THE DIFFERENCE OF THE EXPONENTS OF THE TWO NUMBERS.



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	FLOAT
PAGE	33 OF 39
ISSUE DATE	
NUMBER	
DRAWN BY	
DATE	

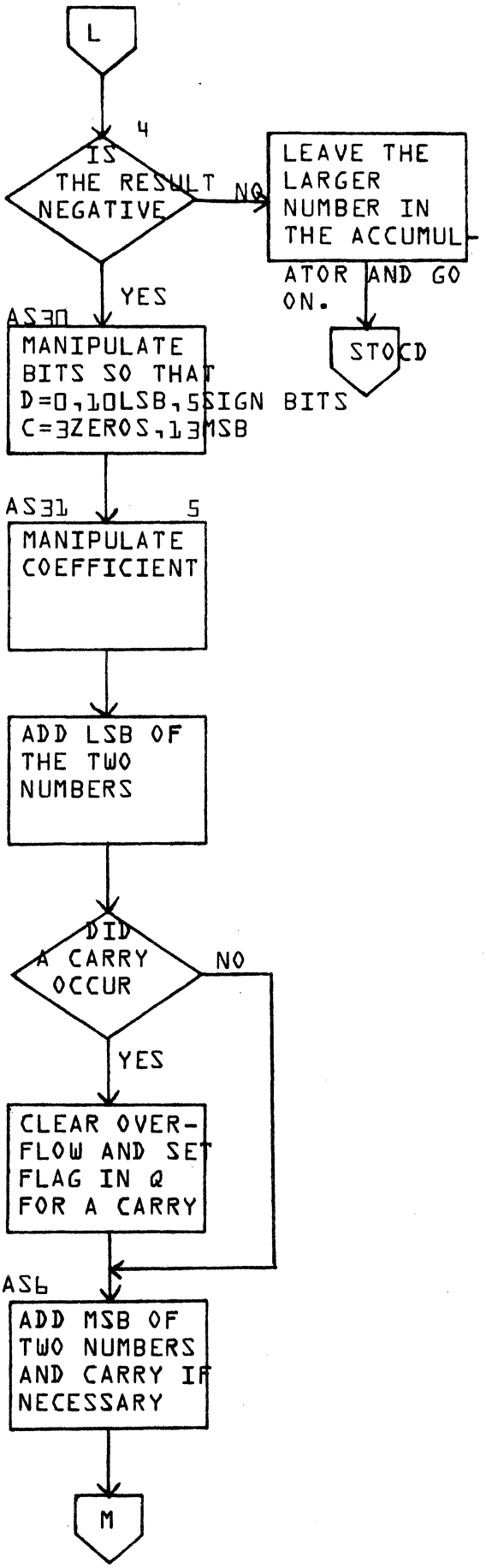
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART [X]
DECISION TABLE
OTHER

PRINTED IN U.S.A.

CA1385 (FORMERLY CA127-1)

4. RESULT IS RESULT OF DELTA-BETA-29; IS THERE ▽TOO▽ GREAT A DIFFERENCE BETWEEN THE NUMBERS.

5. SHIFT SMALLER NUMBER RIGHT {DELTA-BETA} AND SET THE SIGN OF MSB TO POSITIVE. SHIFT LSB RIGHT 1 AND SET THE SIGN OF LSB POSITIVE. CLEAR THE CARRY TO BIT 15 AND ADD 1 TO MSB POSITION THE SMALLER NUMBER.



REV	APPROVED	DATE

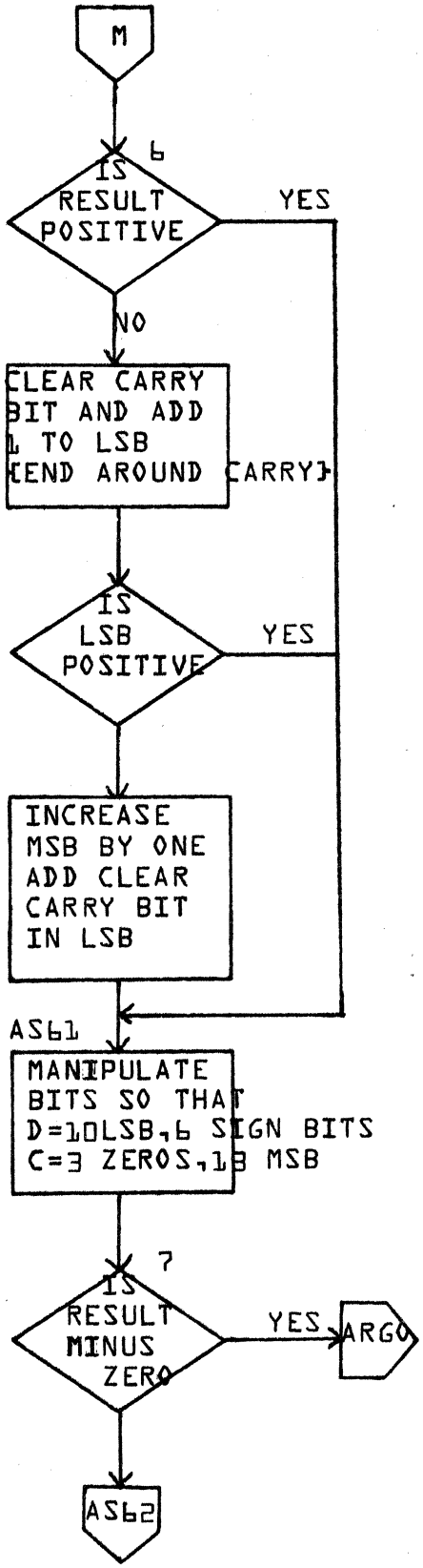
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐

DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	PROJECT NAME
IMS	1700		
DOCUMENT TITLE		PROJECT MGR	TASK NO
FLOAT	PAGE 34 OF 39		TASK NAME
NUMBER	ISSUE DATE		
DRAWN BY	DATE		

PRINTED IN U.S.A.

6. RESULT IS RESULT OF ADDITION OF MSB AND CARRY BIT.

7. MINUS ZERO IS
   Q=7FFF
   A=FFFE



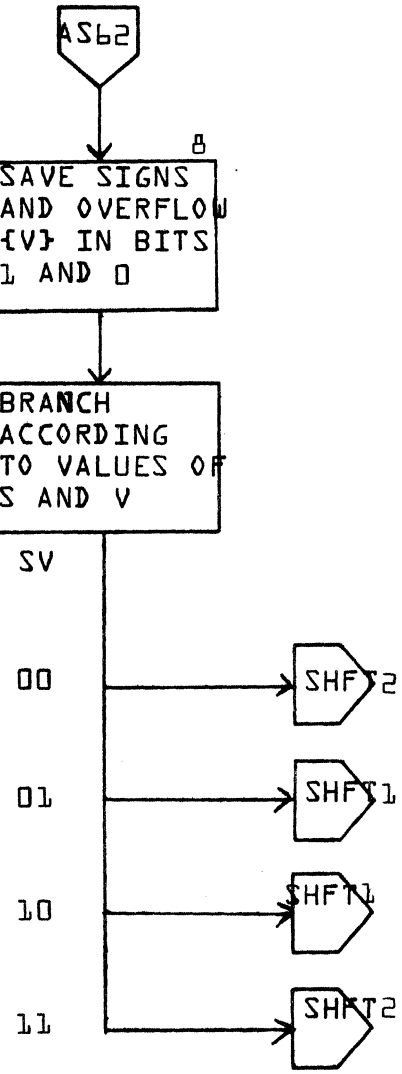
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE [ ]
FLOWCHART [X]
DECISION TABLE [ ]
OTHER [ ]

DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	DATE
DOCUMENT TITLE FLOAT	PAGE 35 OF 39	PROJECT MGR.		APPROVED
NUMBER	ISSUE DATE	PROJECT NAME		
DRAWN BY	DATE	TASK NO.		
		TASK NAME		

PRINTED IN U.S.A.

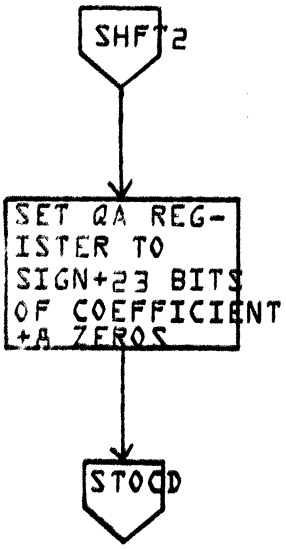
CA1385 (FOR) CA127-11

8. IF S.EQ.V, THEN V IS SIGN EXTENSION.
IF S.NE.V, THEN RESULT OVERFLOWED INTO V.

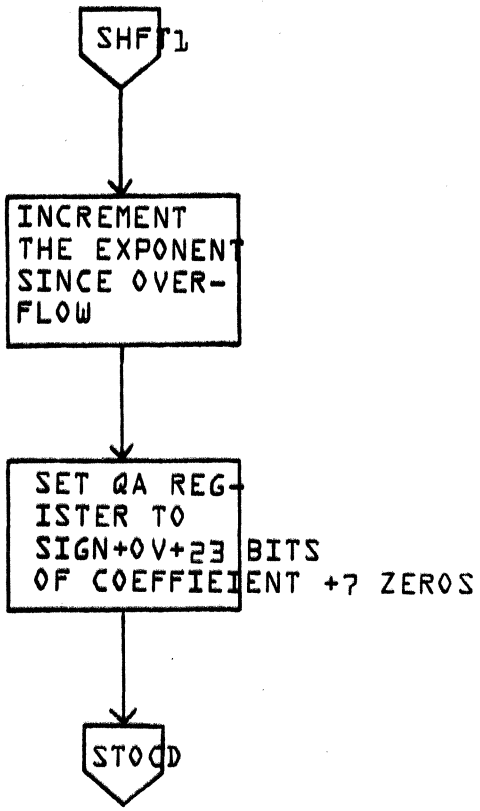


CONTROL DATA CORPORATION	REV	APPROVED	DATE
SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐			
DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	
DOCUMENT TITLE FLOAT	PAGE 36 OF 39	PROJECT MGR.	
		PROJECT NAME	
NUMBER	ISSUE DATE	TASK NO.	
DRAWN BY	DATE	TASK NAME	

NO OVERFLOW



OVERFLOW OCCURRED



REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 37 OF 39
NUMBER
ISSUE DATE
DRAWN BY
DATE

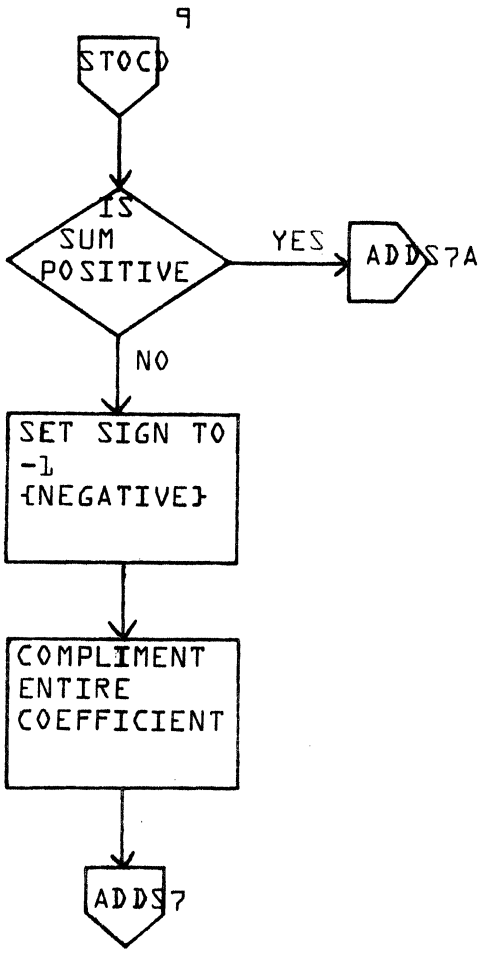
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

PRINTED IN U.S.A.

CA1385 (FORMERLY CA127-1)

9. SAVE THE SIGN AND MAGNITUDE OF THE RESULTS.
ON ENTRY
C=SIGN+15MSB
D=8LSB+8ZEROS



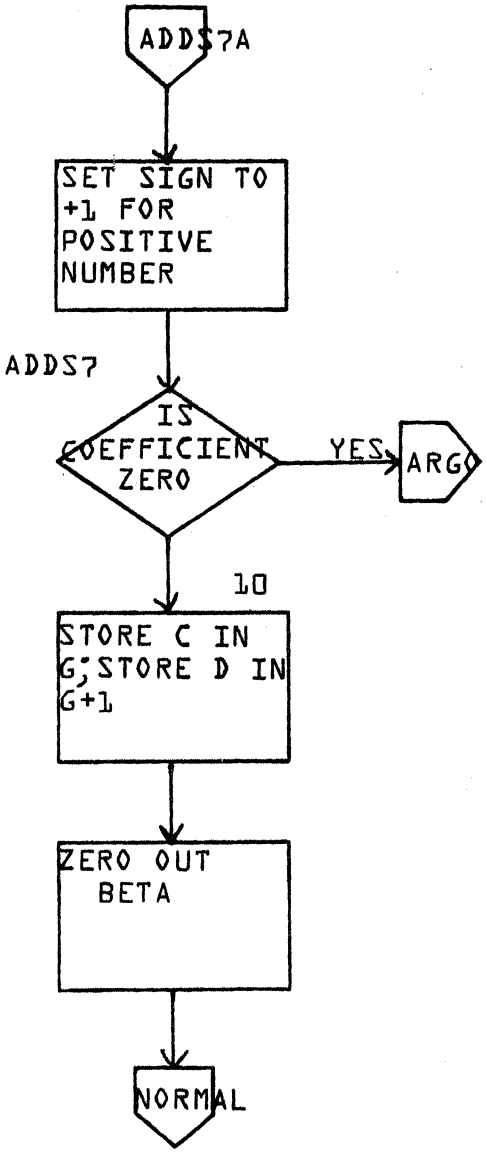
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE [X]
FLOWCHART [ ]
DECISION TABLE [ ]
OTHER [ ]

DOCUMENT CLASS	IMS	MACH. TYPE	1700	
DOCUMENT TITLE	FLOAT	PAGE	38 OF 39	
NUMBER		ISSUE DATE		
DRAWN BY		DATE		

REV	APPROVED	DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

10. C=0+15MSB
    D=8LSB+8ZEROS



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☐
FLOWCHART ☒
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FLOAT
PAGE 29 OF 39
NUMBER
ISSUE DATE
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV	APPROVED	DATE

PRINTED IN U.S.A.

CA1385 (FORMERLY CA127-1)

CHAPTER 10

TABLE OF CONTENTS

10.0 Object Time Intrinsic Functions

10.1 General Description

The following Object Time Intrinsic Functions are included in this chapter:

Intrinsic Functions	I/O Routine
ABS{X}	Q8AB
SIGN{X, Y}	SIGN
IFIX{X}	FXFL
FLOAT{I}	FXFL

10.2 Q8AB Routine

This routine, which is written in 1700 Assembly Language, computer the absolute value of a floating point number and leaves the result in the pseudo accumulation.

The calling sequence is:

```
RTJ  ABS
1.   address of argument
```

The entry points are:

```
Q8AB
ABS  {which is equated to Q8AB}
```

The external declared is:

```
FLOT
```

The low core locations used by this routine are:

ABSPRM	{$E3, $E4} for the parameter of the absolute value function
FLACC	{$C5, $C6} pseudo accumulator

10.3 SIGN Routine

This routine, which is written in 1700 Assembly Language, computer the sign of the second argument times the absolute value of the first argument and leaves the result in the pseudo accumulator.

The calling sequence is:

```
RTJ  SIGN

1.  address of first argument
2.  address of second argument
```

The entry points are:

```
@8SG
SIGN {which is equated to @8SG}
```

The declared external is:

```
FLOT
```

The low core locations used by this routine are:

```
A1   {$D8}
A2   {$D9}
FF   {$E1}  used to save I register
@S   {$E2}  used to save @ register
```

10.4 FXFL Routine

This routine, which is written in 1700 Assembly Language, performs the conversion of a floating point number into an integer {Intrinsic Function IFIX} and of an integer into a floating point number {Intrinsic Function FLOAT}; it is also called by programs which have conversion cross equal signs.

The entry points declared by this routine are:

```
@8@FIX
@8FX
@8@FLT
@8FLOT
FLOAT   {which is equated to @8FLOT}
IFIX    {which is equated to @8FX}
DFIX    {which is equated to IFIX}
```

The declared external is:

```
FLOT
```

The low core locations used by this routine are:

```
CELL2     {$D8}
CELL1     {$DA}
QS        {$E2} used to save Q register
```

The calling sequences for FXFL for the Intrinsic Functions are:

for IFIX{X}:

```
RTJ  IFIX
```

1. address of argument

The resultant integer is in the A register upon return to the user program.

for FLOAT{I}:

```
RTJ  FLOAT
```

1. address of argument

The calling sequence for the conversion across the equal sign of a floating point number into an integer is:

```
RTJ  Q8QFIX
```

In this case the argument is in the pseudo accumulator upon entry and the resultant integer is in the A register upon exit.

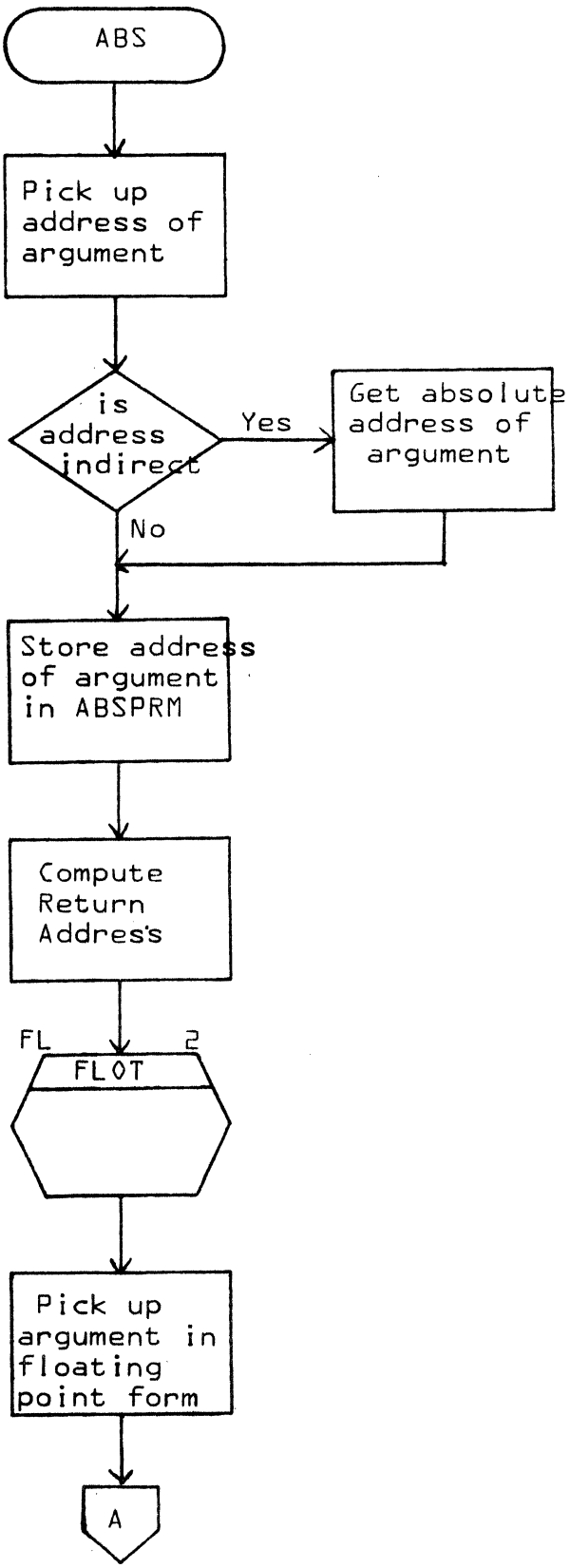
The calling sequence for the conversion across the equal sign of an integer into a floating point number is:

```
RTJ  Q8QFLT
```

In this case the argument is in the A register upon entry.

1. Is address negative?

2. Do a floating point load of argument into pseudo accumulator.

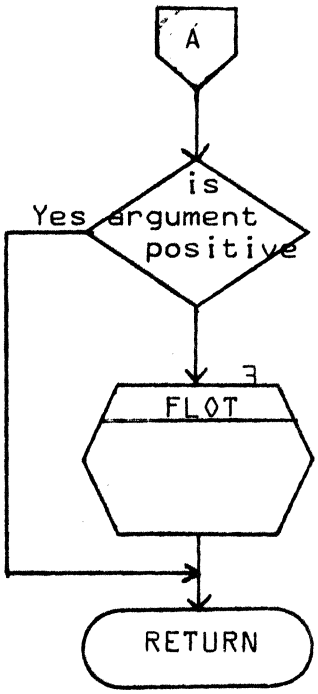


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS: IMS
MACH. TYPE: 1200
DOCUMENT TITLE
NUMBER: QBAB
PAGE 1 OF 2
ISSUE DATE
DRAWN BY
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV | APPROVED | DATE

3. Do a floating point complement of the argument of ABS.

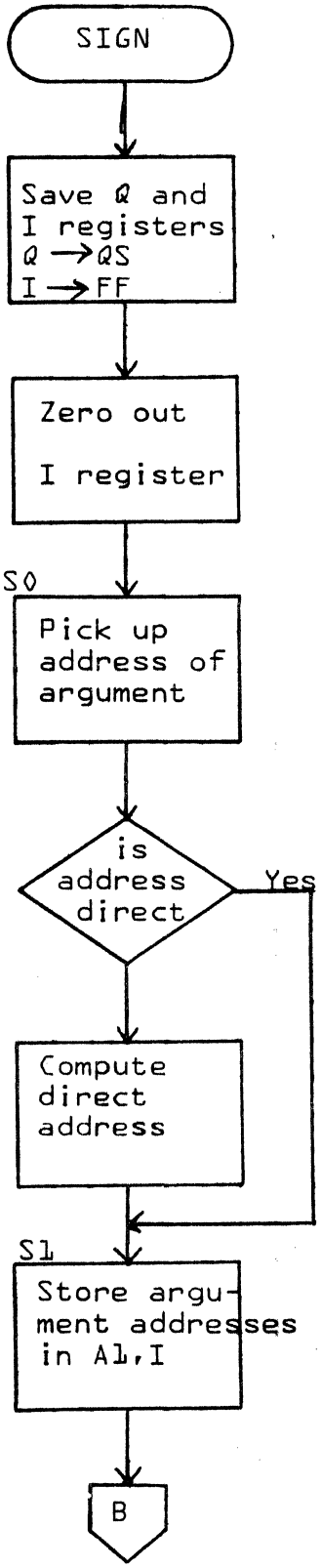


REV	APPROVED	DATE

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	IMS
MACH. TYPE	1700
DOCUMENT TITLE	
	QQAB
PAGE	2 OF 2
NUMBER	
ISSUE DATE	
DRAWN BY	
DATE	
PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	



REV	APPROVED	DATE

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

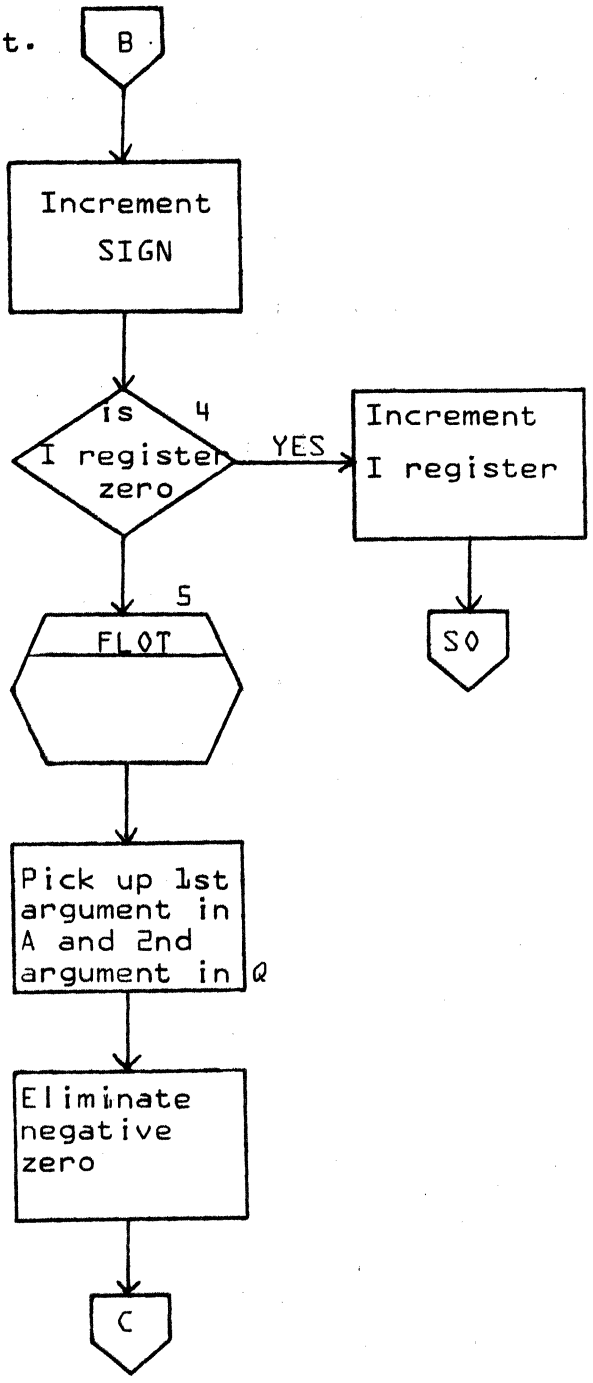
DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE		PAGE 1 OF 3	
NUMBER	SIGN	ISSUE DATE	
DRAWN BY		DATE	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

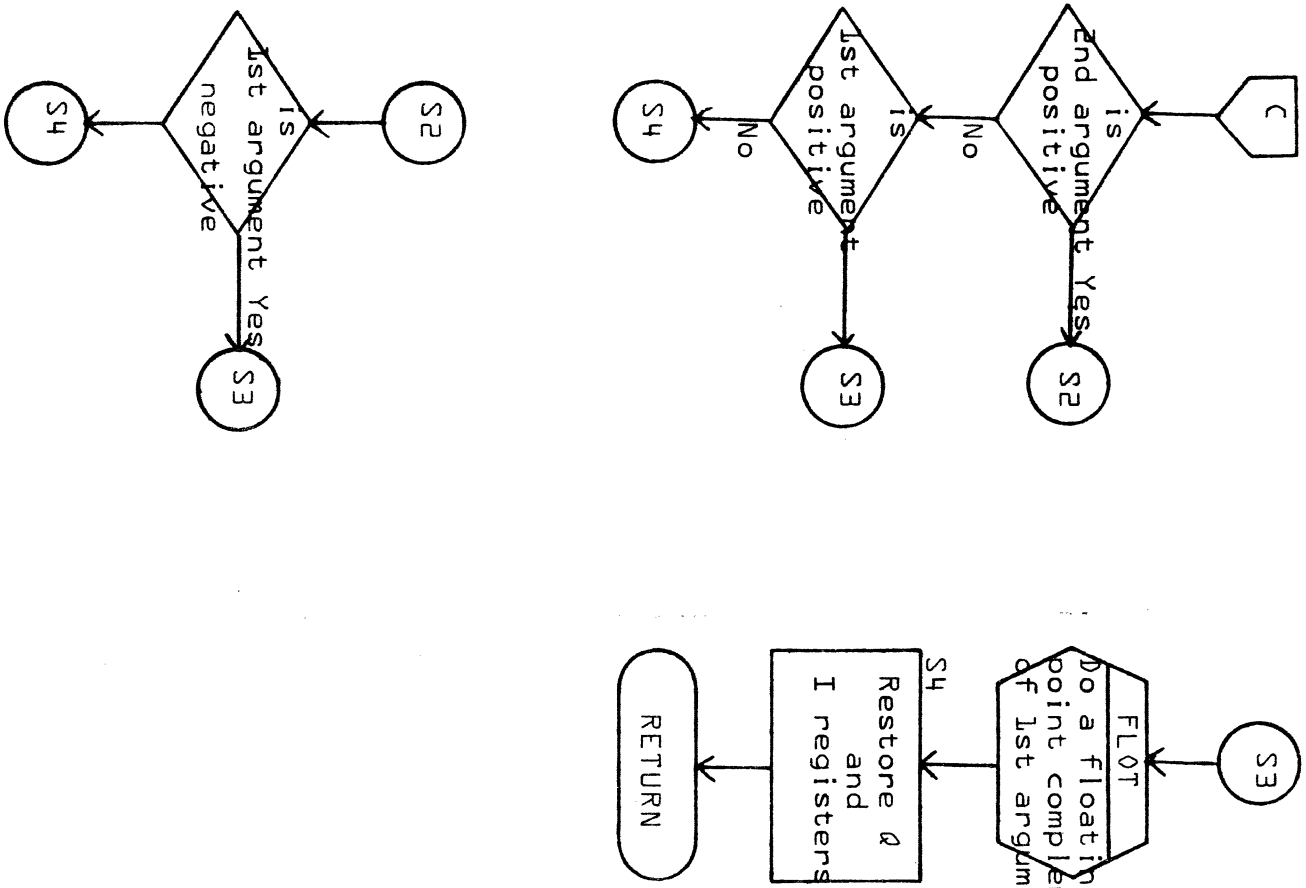
SAMPLE CODE ☐☐☐☐
FLOWCHART
DECISION TABLE
OTHER

4. Have we just processed the first argument's address.

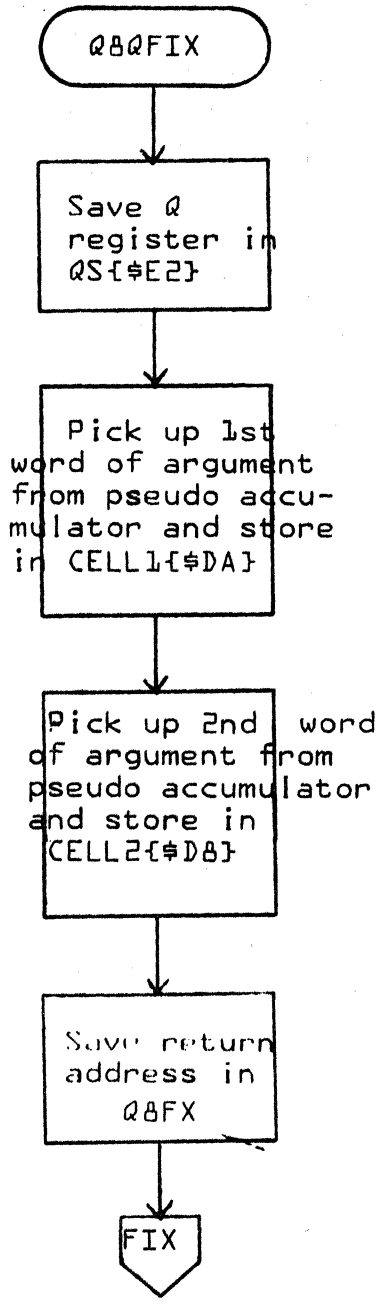
5. Do a floating point load of 1st argument.



CONTROL DATA CORPORATION								
SOFTWARE DOCUMENT	DOCUMENT CLASS	IMS	MACH. TYPE	1700	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE	SIGN	PAGE	2 OF 3			
	NUMBER		ISSUE DATE				
	DRAWN BY		DATE				
	PROJECT NO.		PROJECT MGR.				
	PROJECT NAME		TASK NO.				
	TASK NAME						



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	SIGN	PAGE 3 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



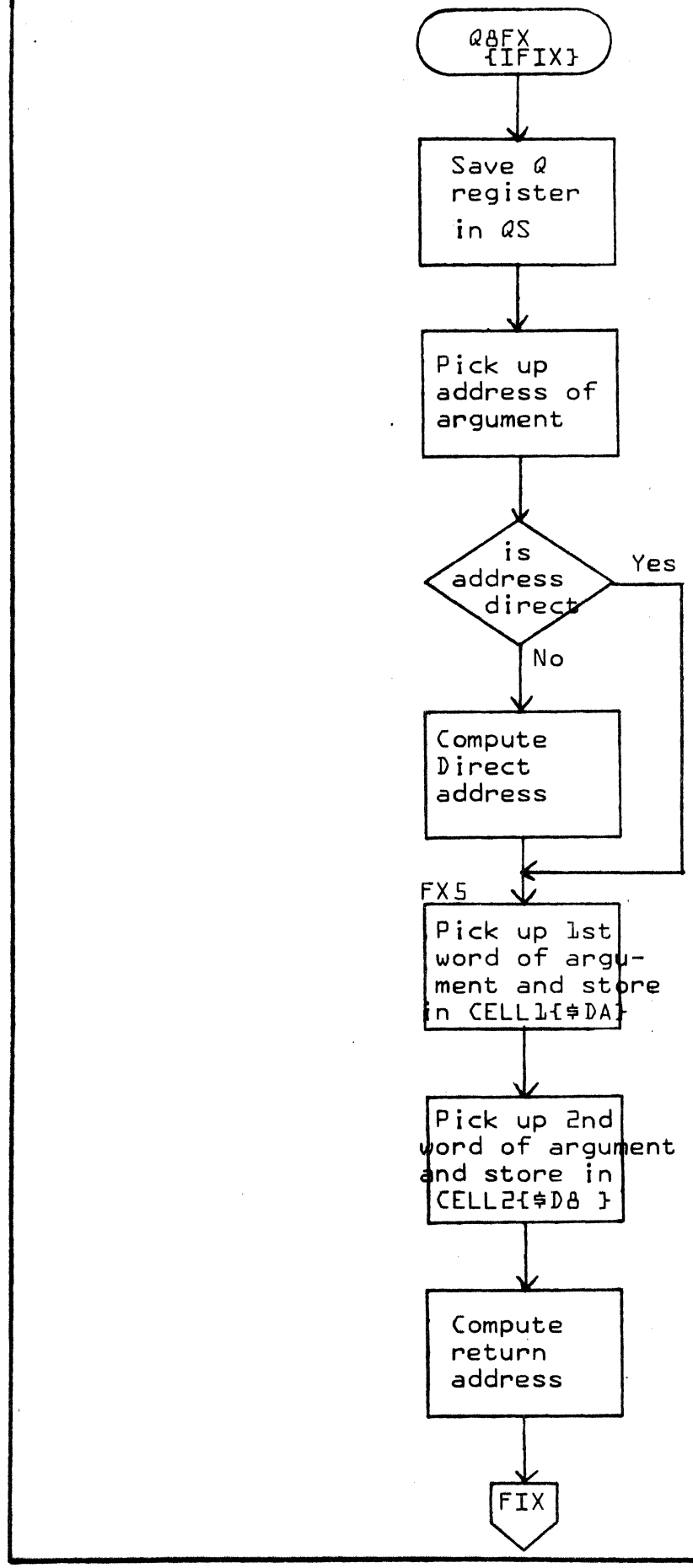
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

REV	APPROVED	DATE

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FXFL
PAGE 1 OF 9
NUMBER
ISSUE DATE
DRAWN BY
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)



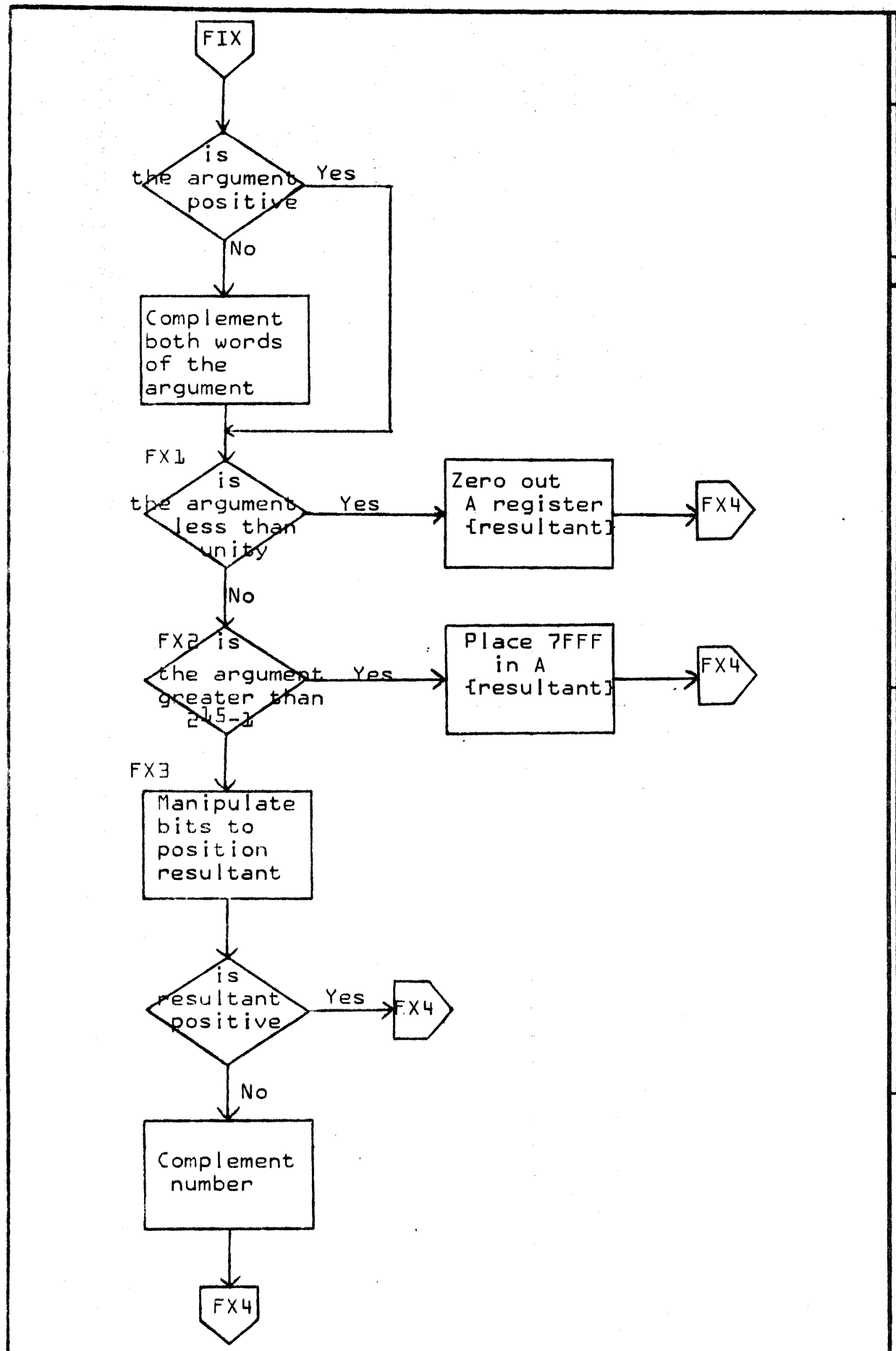
REV	APPROVED	DATE

DOCUMENT CLASS	TMS	MACH. TYPE	1700
DOCUMENT TITLE	FXFL	PAGE 2 OF 9	
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

PROJECT NO.	
PROJECT MGR.	
PROJECT NAME	
TASK NO.	
TASK NAME	

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

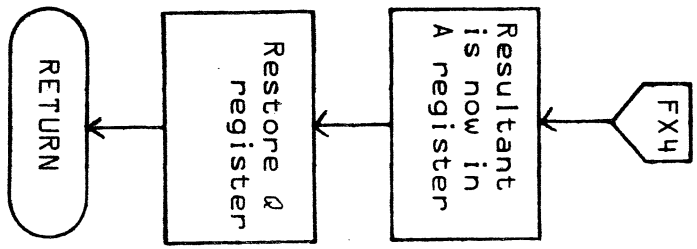
SAMPLE CODE ☐☐☐☐
FLOWCHART
DECISION TABLE
OTHER



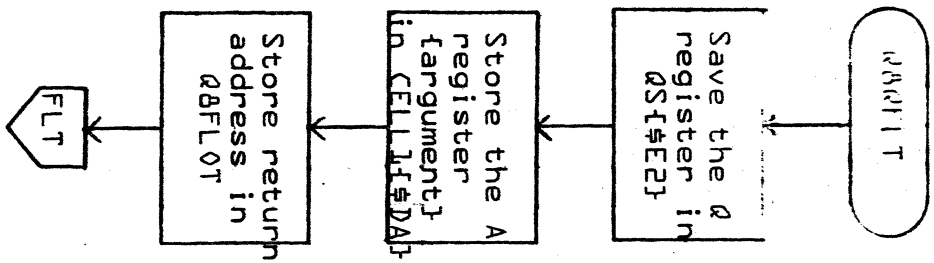
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.		REV	APPROVED	DATE
	DOCUMENT TITLE: FXFL	PAGE 3 OF 9	PROJECT MGR.				
			PROJECT NAME				
	NUMBER	ISSUE DATE	TASK NO.				
	DRAWN BY	DATE	TASK NAME				

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)



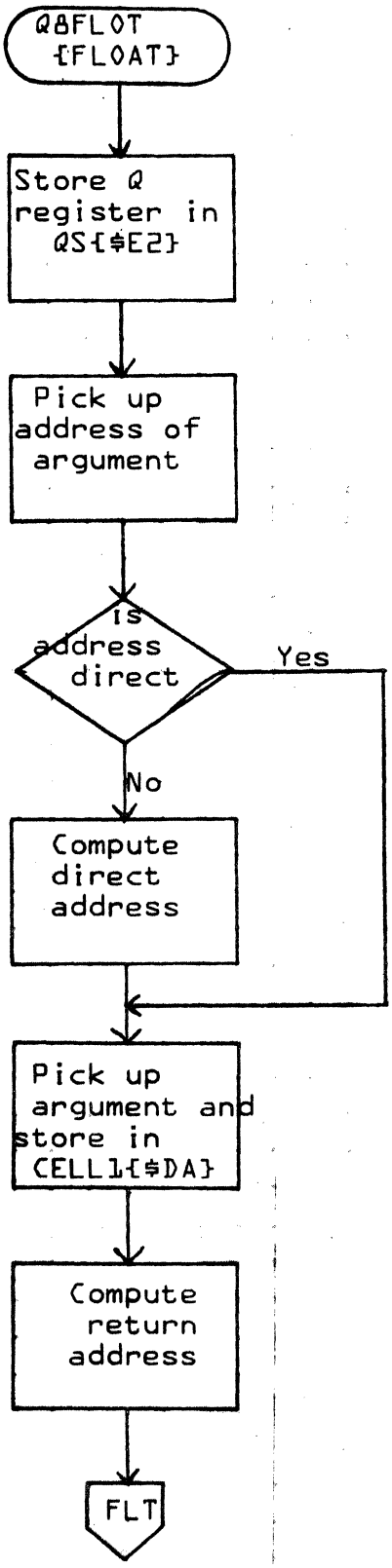
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	FXFL	PAGE 4 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	ISSUED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	FXFL	PAGE 5 OF 9	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

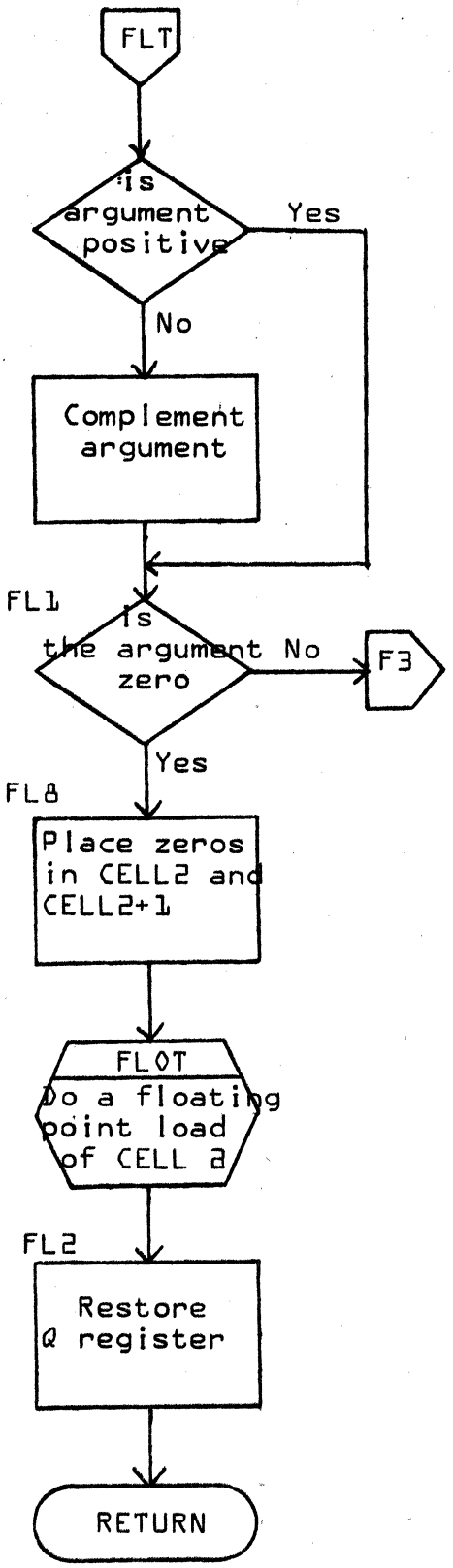


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

REV	APPROVED	DATE

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FXFL
PAGE 6 OF 9
NUMBER
ISSUE DATE
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

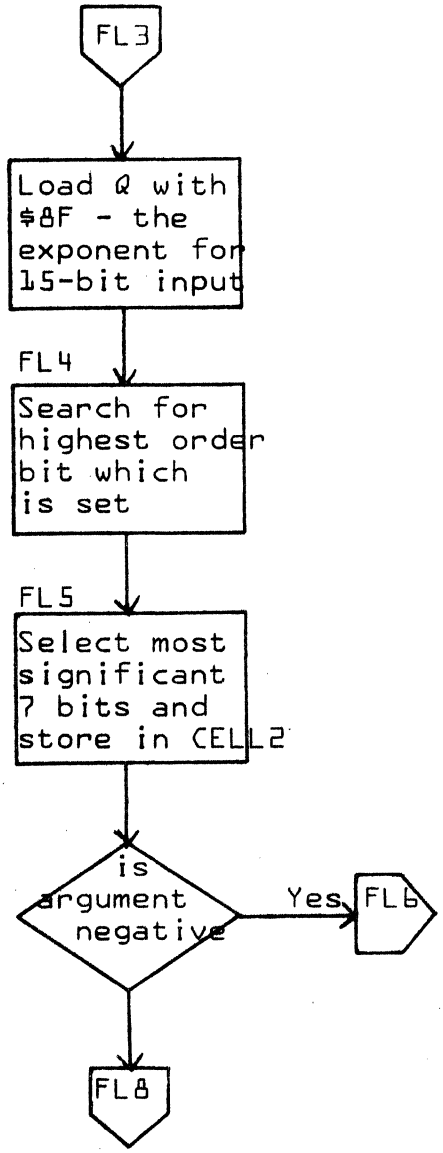
REV	APPROVED	DATE

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: FXFL
PAGE 7 OF 9
ISSUE DATE
NUMBER
DATE
DRAWN BY

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

PRINTED IN U.S.A.

AA1385 (FORMERLY CA127-1)

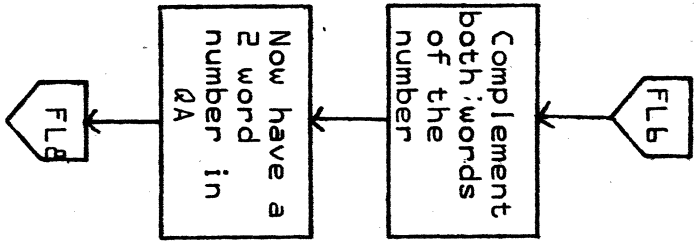


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS	IMS	MACH. TYPE	1700
DOCUMENT TITLE	FXFL	PAGE 8 OF 9	
NUMBER		ISSUE DATE	
DRAWN BY		DATE	

PROJECT NO.		REV	APPROVED
PROJECT MGR.			
PROJECT NAME			
TASK NO.			
TASK NAME			

DATE



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE FXFL PAGE 9 OF 9 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

CHAPTER 11

T A B L E O F C O N T E N T S

11.0 DOUBLE PRECISION OBJECT TIME ARITHMETIC ROUTINES

11.1 Introduction

This chapter deals with the following object time routines:

External Function	Symbolic Name
Trigonometric sine	DSIN
Trigonometric cosine	DCOS
Exponential	DEXP
Natural logarithm	DLOG
Arctangent	DATAN
Square Root	DSQRT

The above routines are designed to operate under the 1700 Mass Storage FORTRAN. They are written in 1700 Assembly Language for optimization of code and time of execution. However, they can be used by any other program working under the 1700 Operating System as long as the proper calling sequence is generated, and cells $C5 and $E5 are reserved for the floating point package and temporary storage.

11.2 General Description

The argument to any of the functions is a real value number written in the 1700 Mass Storage FORTRAN floating point notation. The result is also a real value number in floating point notation.

11.2.1 Temporary Storage

These routines use the unprotected communications area from $C9 to $E3 as temporary storage area.

11.2.2 Calling Sequence

Each object time routine is entered via an RTJ instruction to its symbolic name, followed by a cell containing the address of core where the value of the argument can be found in floating point notation.

11.2.3 Returning Sequence

Each object time routine will return control to the user at the program location immediately following the parameter address with the calculated value of the function in the floating point pseudo-accumulator.

11.3 Function Description

The mathematical method used to approximate the functions will be found in Appendix A of this document. Basically it is a truncated Taylor-Mclaurin series.

11.3.1 Trigonometric Sine {DSIN}

11.3.1.1 Range Reduction

Let Z be the value of the argument.

If $/Z/ > 2^{21}$ then the answer will be assumed zero.

If: $Z < 0$ let $U = -Z$ and $\sin\{Z\} = -\sin\{U\}$
$Z \geq 0$ let $U = Z$ and $\sin\{Z\} = \sin\{U\}$

Let I be integral part of U/2 and let

$Y = U - I \cdot 2\pi$, then $Y \leq 2\pi$, and $\sin\{U\} = \sin\{Y\}$

If: $Y > \pi$, let $T = Y - \pi$ then $\sin\{Y\} = -\sin\{T\}$
$Y \leq \pi$, let $T = Y$ then $\sin\{Y\} = \sin\{T\}$

If: $T > \pi/2$, let $X = \pi - T$
$T \leq \pi/2$, let $X = T$
and $\sin\{T\} = \sin\{X\}$, with $X \leq \pi/2$

11.3.1.2 Series for SIN{X} with $X \leq \pi/2$

Series is a normal truncated Taylor-Mclaurin series:

$$SIN\{X\} = \sum_{n=0}^{8} \frac{\{-1\}^n X^{2n+1}}{\{2n+1\}!}$$

11.3.2 Trigonometric Cosine (DCOS)

Sine COS(Z) = SIN(Z + $\pi/2$) then object time routine DCOS will simply be a second entry to the DSIN routine.

11.3.2.1 Temporary Storage for DSIN and DCOS

Nine cells of temporary storage are required by either DSIN or DCOS. They are:

RETADD	for return address $D7
FLAG	for sign indicator $D8
X and X_2	for temporary floating point numbers, $DA to $DF
QS	for saving Q $E2

11.3.3 Arctangent (DATAN)

11.3.3.1 Range Reduction

Let Z be value of input argument.

If: $Z<0$, let U = - Z, and arctan(Z) = - arctan(U)
$Z\geq 0$, let U = Z, and arctan(Z) = arctan(U)

If: $U<1$, let Y = U, A = 0, and B = 1
$U>1$, let Y = 1/U, A = /2, and B = -1

If: $Y\leq \tan(\pi/8)$ let T = $\pi/16$
$Y> \tan(\pi/8)$ let T = $3\pi/16$

and arctan(U) = A + B (T + arctan(X))

where $X = \dfrac{Y - \tan(T)}{1 + Y \tan(T)}$

11.3.3.2 Series for arctan(X), $X\leq \tan(\pi/16)$

Series is a normal truncated Taylor-Mclaurin series:

$$\text{Arctan}(X) = \sum_{n=0}^{8} \frac{(-1)^n X^{2n+1}}{2n+1}$$

11.3.3.3 Temporary Storage for DATAN

Twelve cells of temporary storage are required.

They are:

ARCRET	for return address $D9
ARCFLG	for path decision $D8
X,X_2,AF,BF	for floating point numbers, $DA to $E2
@S	for saving @ $E3

11.3.4 Exponential (DEXP)

11.3.4.1 Range Reduction

Let Z be input value of argument

If: $Z<0$ let $Y = -Z$, and $e^Z = 1/e^Y$

$Z>0$ let $Y = Z$, and $e^Z = e^Y$

$Z=0$ put answer equal to 1 and exit.

Now $e^Y = 2^N e^X$ if X and N are defined as follows:

Let $T = Y/\ln 2$
$N = T + 1/2$ take integral part
$W = T - N$
$X = W \ln 2$

Therefore the maximum absolute value for Y is equal to 88.02968 which will generate the maximum floating point number. If input value is greater than this maximum value, then a message will be typed out and the answer is given as the maximum floating number permissible in 1700 FORTRAN.

11.3.4.2 Series for e^X for $X \leq \ln 2/2$

Series is normal truncated Taylor-Mclaurin series:

$$e^X = \sum_{n=0}^{9} \frac{X^n}{n!}$$

11.3.4.3 Temporary Storage for DEXP

Seven cells of temporary storage are required. They are:

RETEXP	for return address $DB
FLAG,N	for indicator $DC, DD
Y	for floating point number $D8, $D9, $DA
@S	for saving @ $E1

11.3.5 Natural Logarithm (DLOG)

11.3.5.1 Range Reduction

Let Z be input value of argument.

If $Z \leq 0$ then the answer will be the maximum floating point number possible in 1700.

Now $Z = 2^N Y$

and $\ln(Z) = (N - 1/2) \ln 2 + \ln \left(\frac{1+X}{1-X}\right)$

where $X = \frac{Y - \sqrt{2}/2}{Y + \sqrt{2}/2}$

Thus $|X| \leq 3 - 2\sqrt{2}$

11.3.5.2 Series for $\ln \left(\frac{1+X}{1-X}\right)$ for $|X| \leq 3 - 2\sqrt{2}$

Series is a normal truncated Taylor-Mclaurin series:

$$\ln \left(\frac{1+X}{1-X}\right) = 2 \sum_{n=0}^{8} \frac{X^{2n+1}}{2n+1}$$

11.3.5.3 Temporary Storage for DLOG

Eleven cells of temporary storage are required. They are:

LNRET	for return address $D8
N,X,X2	for floating point numbers $D9 to $E1
@S	for saving @ $E3

11.3.6 Square Root (DSQRT)

11.3.6.1 Range Reduction

Let Z be input value of argument.

If $Z<0$ let $Y = -Z$ and $\sqrt{Z} = -\sqrt{Y}$

also type out error message.

If $Z \geq 0$ let $Y = Z$

Now $Y = 2^{2N} X$ and $\sqrt{Y} = 2^{N}\sqrt{X}$ where $1/4 \leq X < 1$

The Padé approximation for a first estimate is:

$$y_i = \frac{25}{7} - \left[\frac{\frac{5000}{343}}{\left\{X + \frac{15}{49}\right\} \left\{X + \frac{235}{49}\right\} - \frac{400}{2401}} \right]$$

11.3.6.2 Iterative Method

The square root is computed via a Newton-Raphson iteration starting with a first approximation, y_i above. Successive approximations are found from:

$$y_{i+1} = 1/2 \left\{ y_i + \frac{X}{y_i} \right.$$

$$\text{until} \left| y_{i+1} = y_i \right| \leq 2^{-38}$$

11.3.6.3 Temporary Storage for DSQRT

Twelve cells of temporary storage are required. They are:

```
QSAVE      for saving Q $E2
SQRRET     for return address $C9
SQREXP     for value of N $CA
ARG, Y1,   for floating point numbers,
TEMP,      $D8 to $E1
SQRFLG
```

11.4 Type Out Message

Any illegal input value will cause a predetermined value to be given as an answer and the program will exit normally. It is left to the user to take corrective action.

11.4.1

Routine	Argument	Answer
DSIN or DCOS	$/Z/ > 2^{21}$	0
DEXP	$/Z/ > 88.02968$	∞
DLOG	$Z \leq 0$	∞
DSQRT	$Z < 0$	$-\sqrt{/Z/}$

11.5 References

CONTROL DATA 6600 Computer Systems, Programming System/Library Functions, A Study of Mathematical Approximations, Pub. No. 60114500.

HANDBOOK OF MATHEMATICAL FUNCTIONS, National Bureau of Standards Applied Mathematics Series-55, Library of Congress catalog number 64-60036.

11.6 DEXPN Subroutine - Double Precision Exponentiation (**)

This subroutine is called by the FORTRAN compiler for programs that use the exponentiation operator, i.e., the symbol **, in performing double precision exponentiation. This subroutine is written in 1700 Assembly Language. It contains the following entry points:

@@QD2I	Double precision floating number to integer power
@@QD2F	Double precision floating number to single precision floating power
@@QD2D	Double precision floating number to double precision floating power

The following externals are declared:

DFLOT
DLOG
DEXP

The cells required for temporary storage are:

DFLOFG	$E5	Floating/Fixed Flag
DFRSLT	$DB	Intermediate DP Results $DB,$DC,$DD
SIGN	$E1	Sign of Exponent
COEFF	$E2	Address of Coefficient
EXPO	$D7	Address of Exponent
MLTPR	$DE	Powers of Coefficient {double precision floating point number - 3 words}
DFLOVF	$C8	Floating Point Package Overflow Flag
DFLACC	$C5	Double Precision Floating Point Pseudo Accumulation {$C5,$C6,$C7}
CZERO	$22	Low Core Cell containing Zero

The calling sequence is:

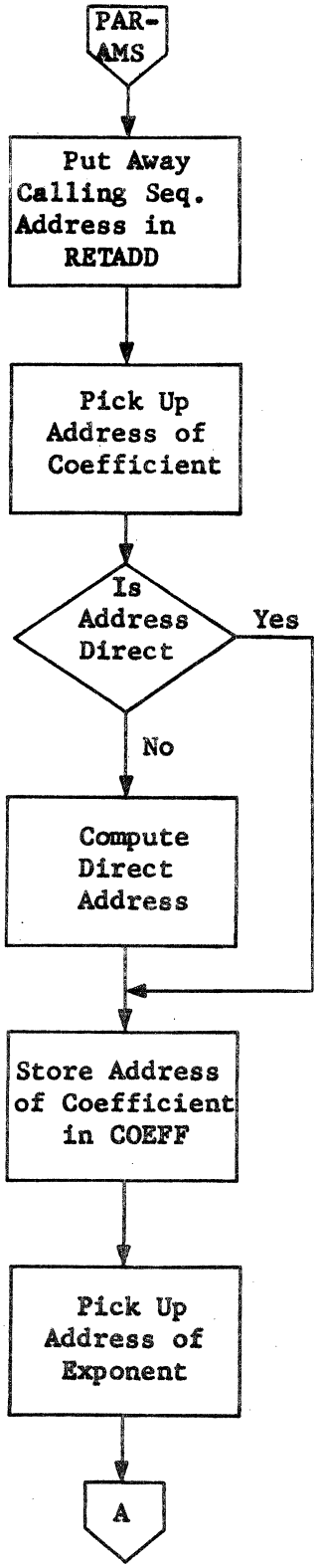
RTJ @8@XXX Where XXX is D2I, D2F, or D2D

1. address of coefficient
2. address of exponent

Double Floating to Single Floating

Double Floating to Integer

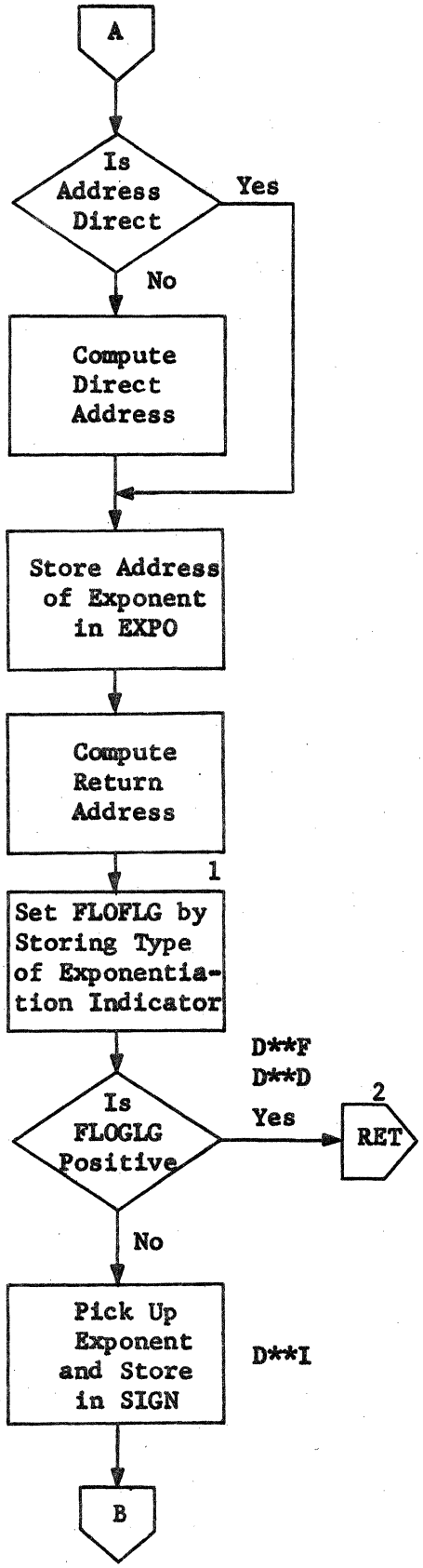
Double Floating to Double Floating


PAR-
AMS
Put Away Calling Seq. Address in RETADD
Pick Up Address of Coefficient
Is Address Direct
Yes
No
Compute Direct Address
Store Address of Coefficient in COEFF
Pick Up Address of Exponent
A

1. DFLOFG equals

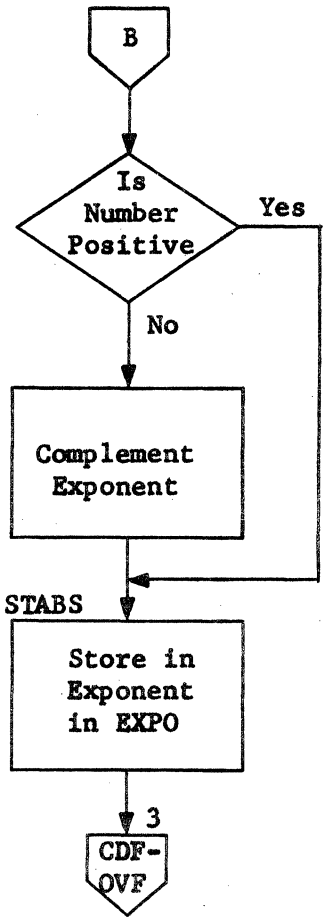
   -1 D**I
    0 D**F
   +1 D**D

2. From this place a jump is made to DL0GEX

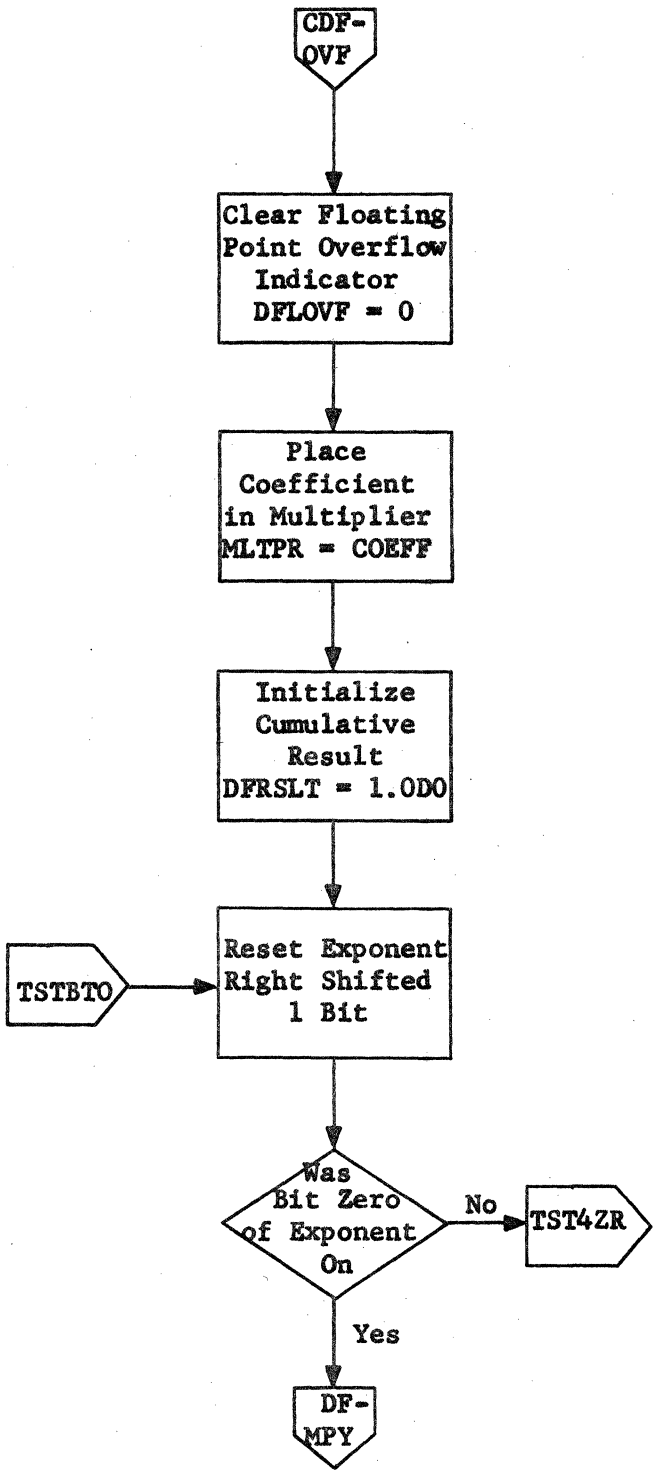


3. Process D**I

4. Process D**F
   or D**D

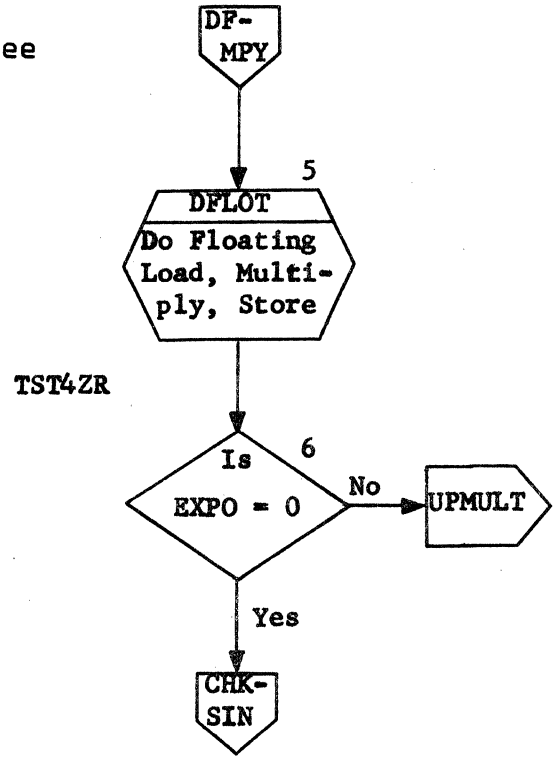


Process Double Floating to Integer

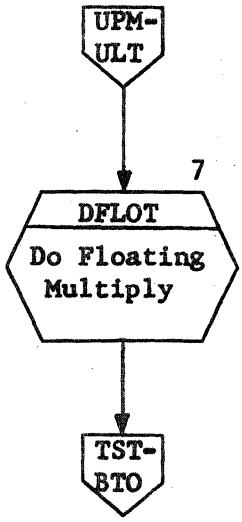


5. Floating
   DFRSLT =
   DFRSLT*MLTPR

6. Test exponent to see if all significant bits have been shifted out.



7. Square floating multiplier
MLTPR = MLTPR*MLTPR

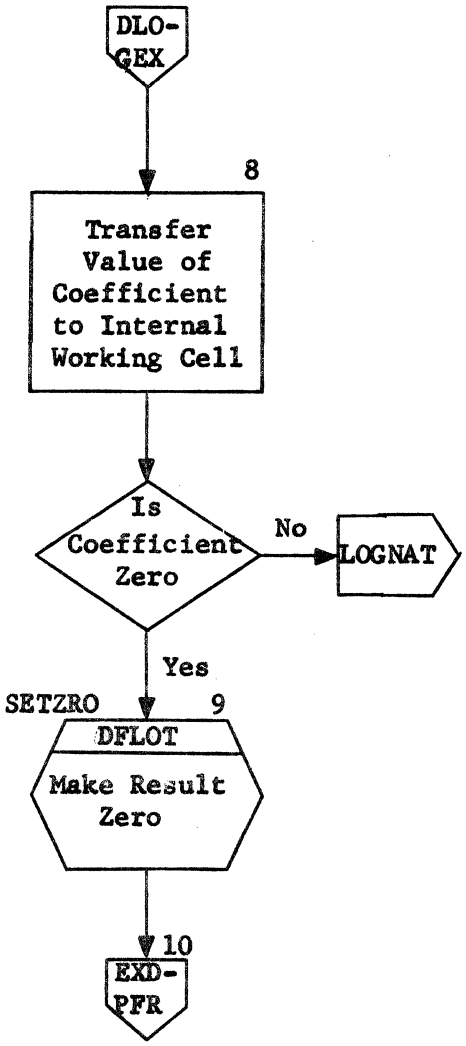


Process Double Floating to Floating or Single Floating to Floating

8. First word into DFRSLT; Second word into DFRSLT+1

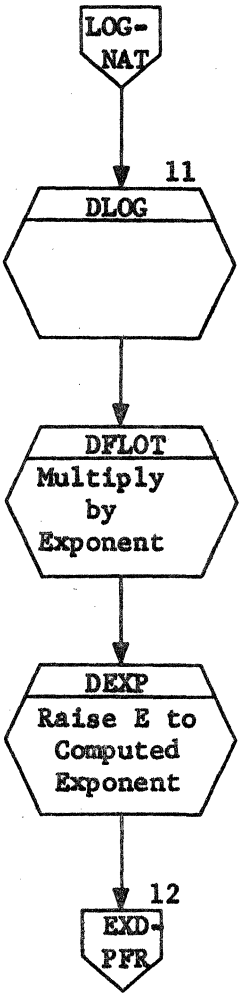
9. Make result zero to avoid a fault in the natural log routine.

10. Exit

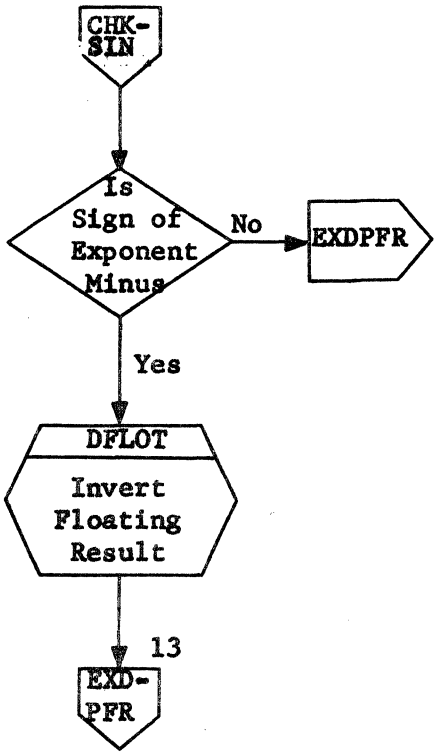


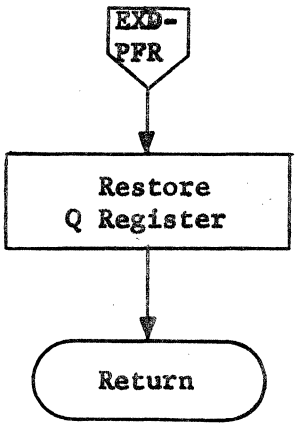
11. Take natural log of coefficient.

12. Exit



13. Exit


CHK-SIN
Is Sign of Exponent Minus
No
EXDPFR
Yes
DFLOT
Invert Floating Result
13
EXD-PFR


EXD-PFR
Restore Q Register
Return

APPENDIX A

1700 Double Precision Mathematical Approximations

1.0 INTRODUCTION

This paper will derive approximations for object time 1700 FORTRAN programs which will allow fast evaluations with necessary accuracy and uncomplicated range reduction.

1.1 Accuracy

Since all approximations are to be in floating-point, it shall be required that:

$$R < 2^{-39} = 1.819E\text{-}12$$

which is the basic round off due to the size of significant digits in 1700.

2.0 GENERAL DESCRIPTION OF METHOD

The method will use normal truncated Taylor-Mclaurin Power Series.

2.1 Truncated Taylor-Mclaurin Power Series

A function will be expressed as a truncated Taylor-Mclaurin series such that the error produced is less than R. That is,

$$f\{N\} = \sum_{n=0}^{m} \frac{f^{\{n\}}\{0\}}{n!} x^n, \quad /n/ \leqslant a$$

and m is chosen such that

$$R < \left| \frac{f^{\{m+1\}}}{\{m+1\}!} x^{m+1} \right| = R_T$$

3.0 SIN{X} for $X < \pi/2$

The Taylor-Mclaurin series is:

$$SIN\{X\} = \sum_{n=0}^{8} \frac{\{-1\}^n \ X^{2n+1}}{\{2n+1\}!}$$

Which expands to:

$$SIN\{X\} = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \frac{X^9}{9!} - \frac{X^{11}}{11!} + \frac{X^{13}}{13!} - \frac{X^{15}}{15!} + \frac{X^{17}}{17!}$$

4.0 Arctan{X} for $X \leq \tan\{\pi/16\}$

The Taylor-Mclaurin series is:

$$Arctan\{X\} = \sum_{n=0}^{8} \frac{\{-1\}^n \ X^{2n+1}}{2n+1}$$

Which expands to:

$$Arctan\{X\} = X - \frac{X^3}{3} + \frac{X^5}{5} - \frac{X^7}{7} + \frac{X^9}{9} - \frac{X^{11}}{11} + \frac{X^{13}}{13} - \frac{X^{15}}{15} + \frac{X^{17}}{17}$$

5.0 Exponential, e^x for $x \leq \frac{\ln 2}{2}$

Taylor-Mclaurin series is a normal series truncated after n = 9, that is,

$$e^x = \sum_{n=0}^{9} \frac{x^n}{n!}$$

Which expanded, gives:

$$e^x = X + \frac{X^2}{2!} + \frac{X^3}{3!} + \frac{X^4}{4!} + \frac{X^5}{5!} + \frac{X^6}{6!} + \frac{X^7}{7!} + \frac{X^8}{8!} + \frac{X^9}{9!}$$

Now X and N can always be found such that $e^y = 2^N e^X$
where $X \leq \frac{\ln 2}{2}$

If X and N are defined as follows

Let t = y/ln 2
N = t + 1/2: integral part
w = t - N
X = W ln 2 .

then

$X = \{t-N\} \cdot \ln 2 = t \cdot \ln 2 - \ln\{2^N\}$

$e^X = e^y \cdot 2^{-N}$

and $e^y = 2^N e^X$

6.0 Natural Logarithm X for $|X| \leq \{3-2\sqrt{2}\}$

The Taylor-Mclaurin series is:

$$\ln\{\frac{1+X}{1-X}\} = 2\sum_{n=0} \frac{X^{2n+1}}{2n+1}$$

Which expands to:

$$\ln\{\frac{1+X}{1-X}\} = 2X + \frac{2X^3}{3} + \frac{2X^5}{5} + \frac{2X^7}{7} + \frac{2X^9}{9} + \frac{2X^{11}}{11} + \frac{2X^{13}}{13} + \frac{2X^{15}}{15} + \frac{2X^{17}}{17}$$

Given $Y = 2^N A$ where $1/2 \leq Z < 1$

Then $\ln\{Y\} = N \cdot \ln 2 + \ln Z$.

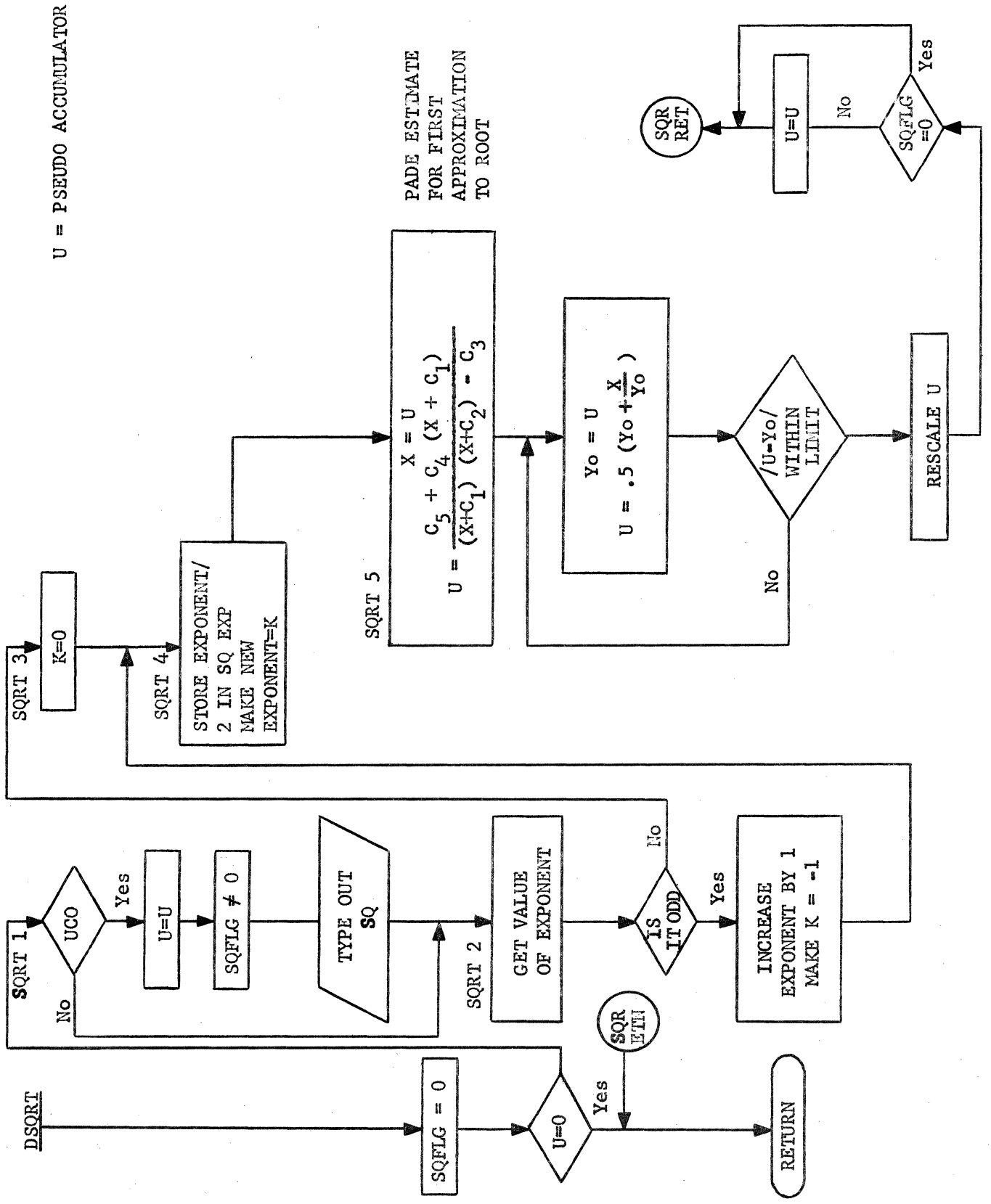
If X is defined as $X = \frac{Z - \sqrt{2/2}}{Z + \sqrt{2/2}}$

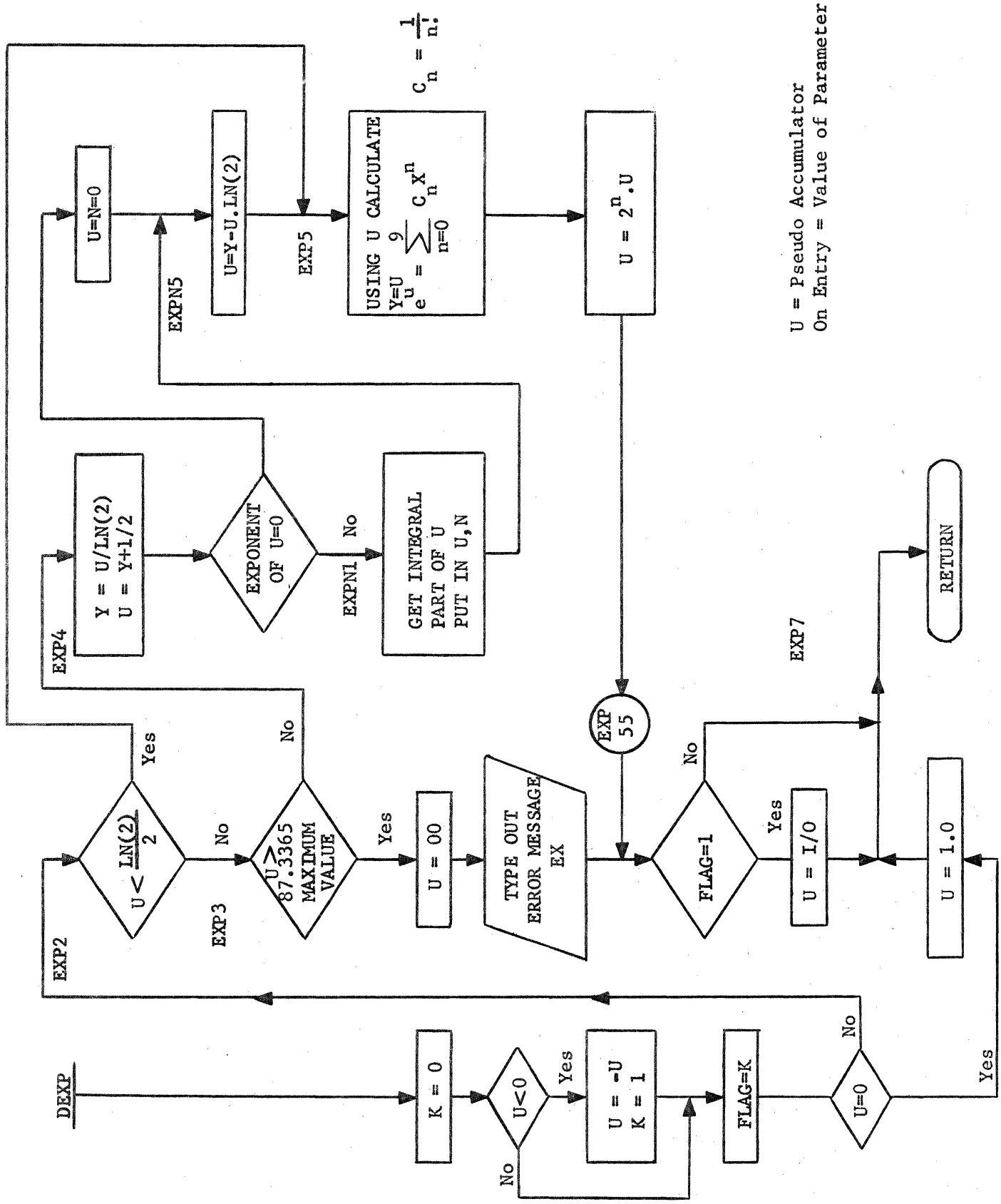
then $/X/ \leq 3 - 2\sqrt{2}$

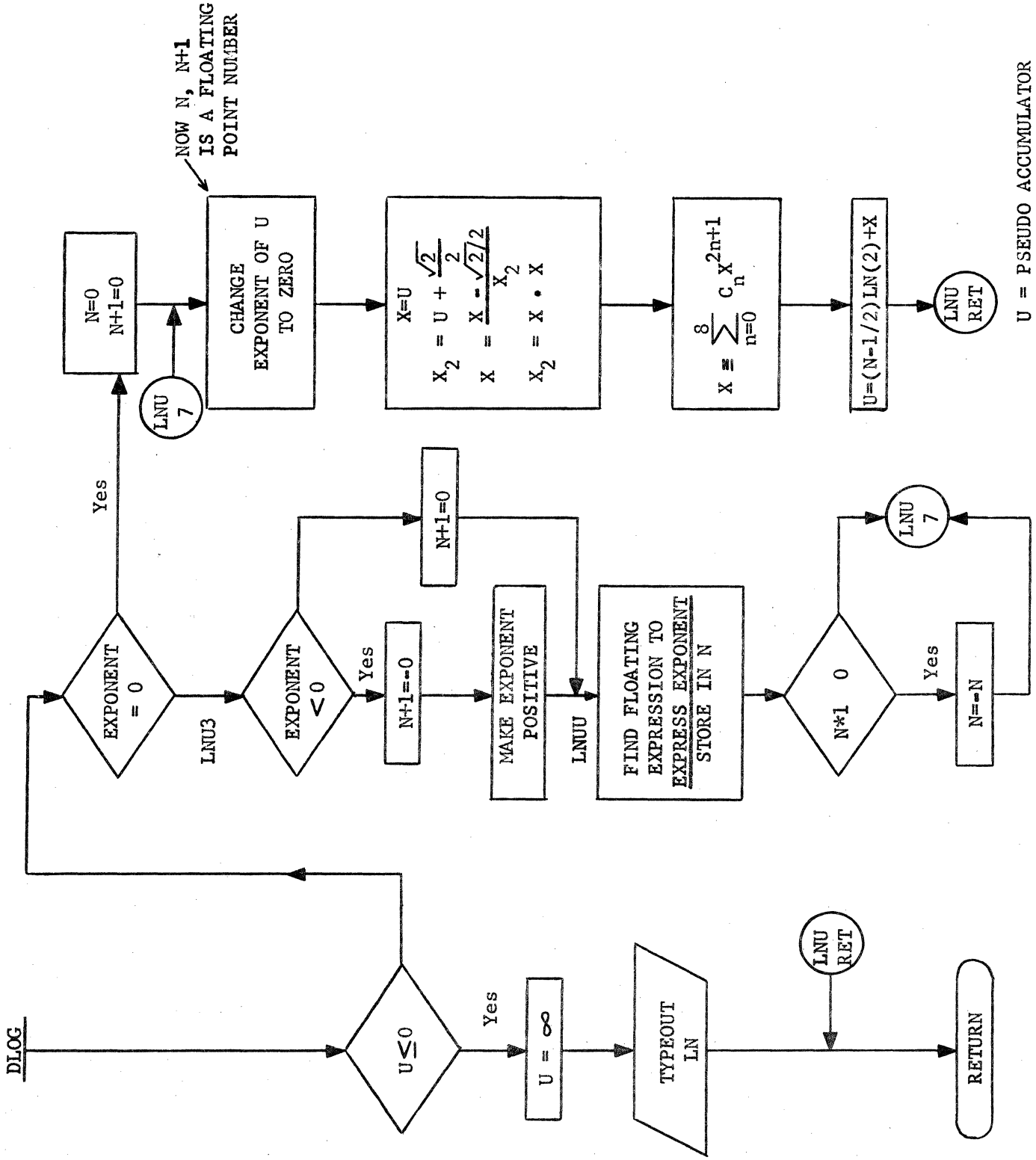
and $\ln Z = -\frac{\ln 2}{2} + \ln \frac{1 + X}{1 - X}$

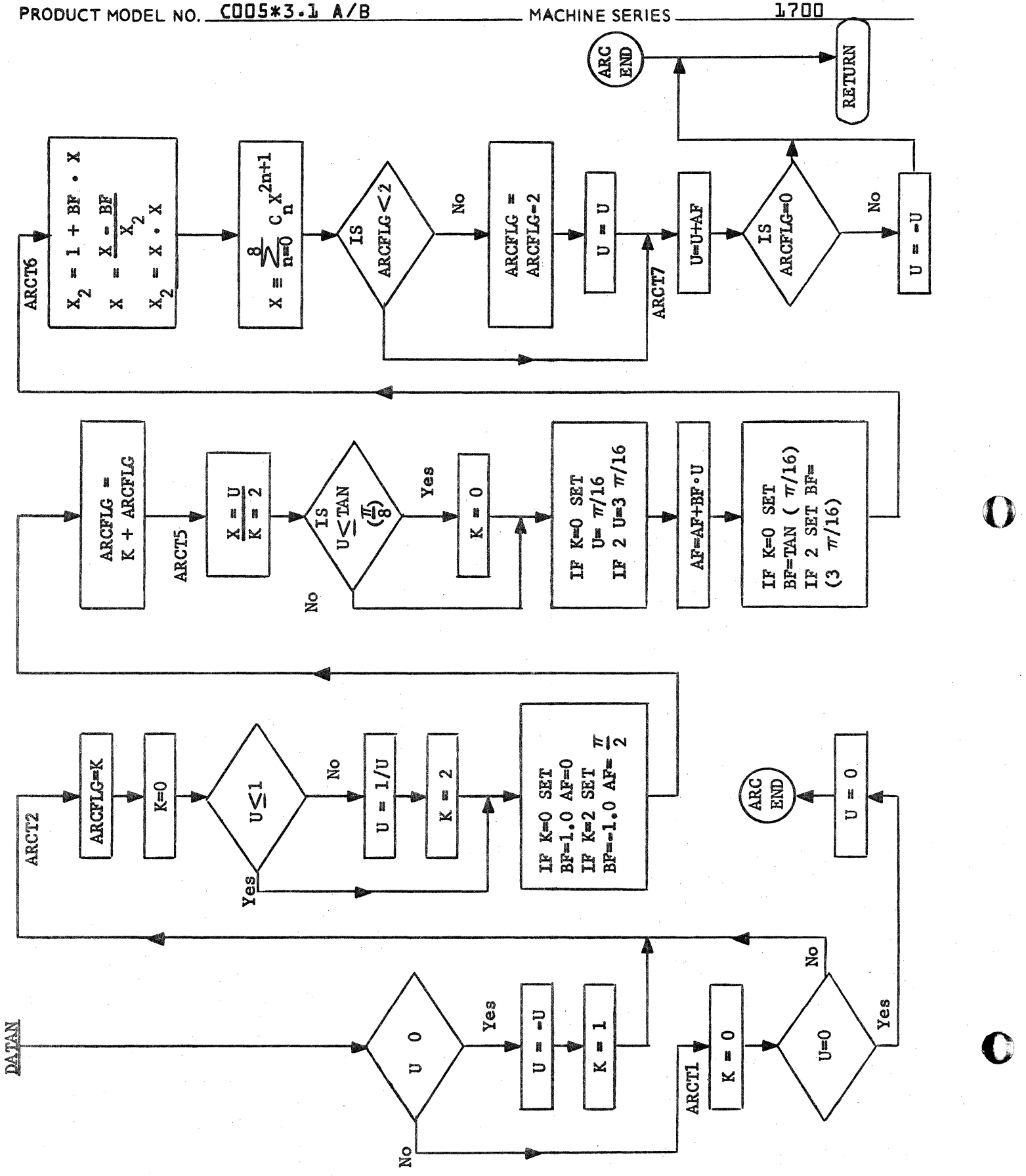
Finally, $\ln Y = \{N - 1/2\} \cdot \ln 2 + \ln \frac{1 + X}{1 - X}$

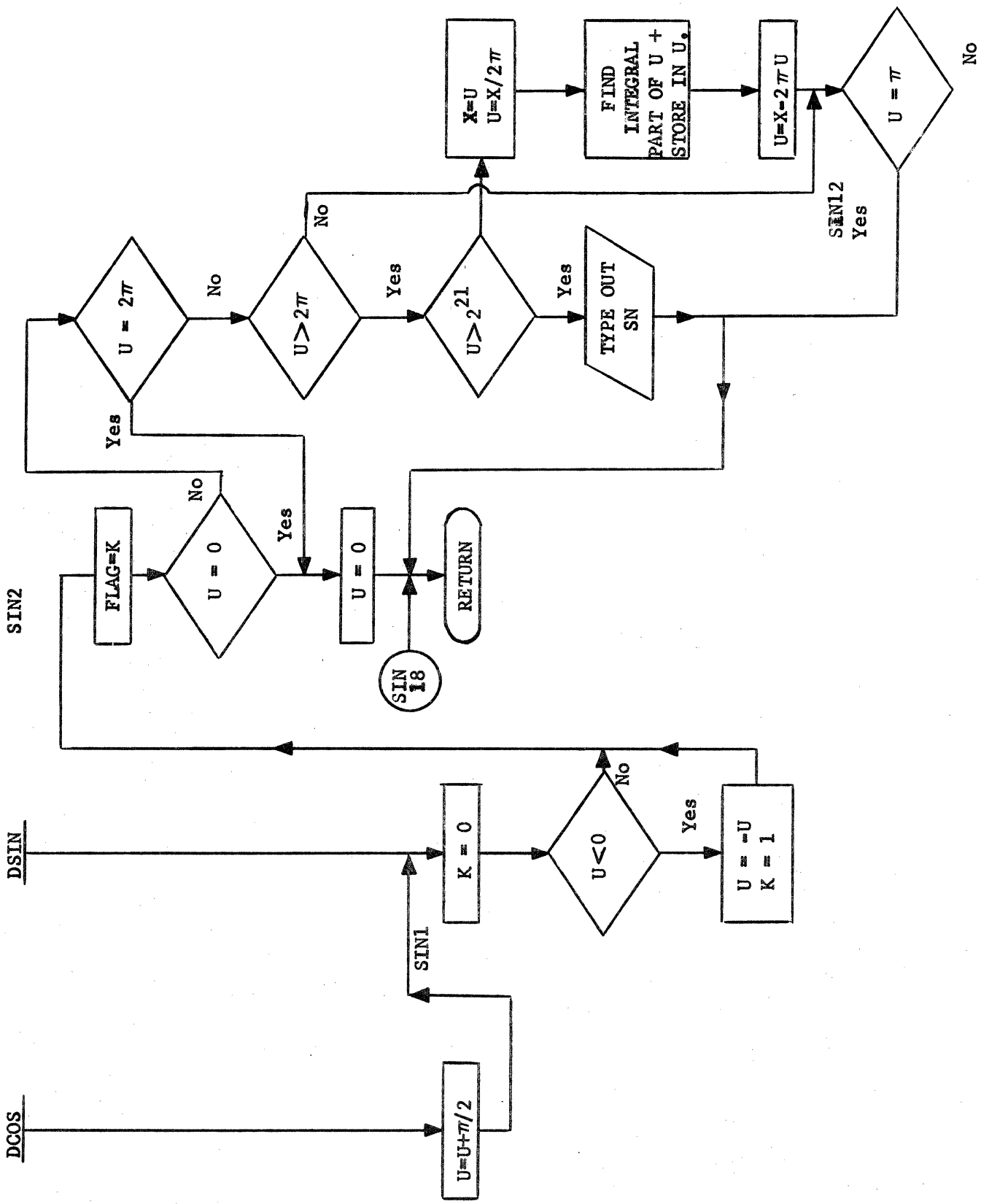
DOCUMENT CLASS IMS PAGE NO. 11-25
PRODUCT NAME 1700 MASS STORAGE FORTRAN
PRODUCT MODEL NO. C005*3.1 A/B MACHINE SERIES 1700

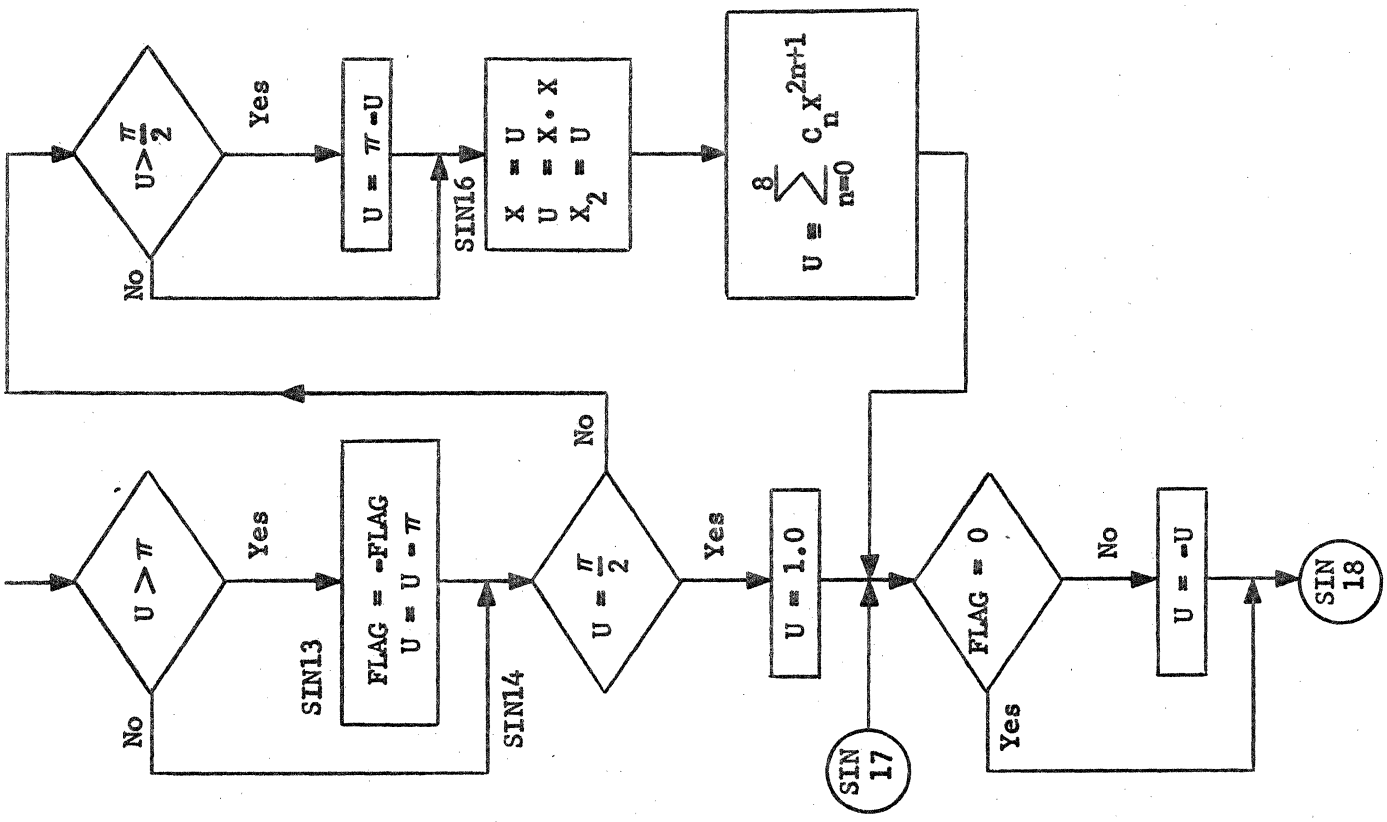


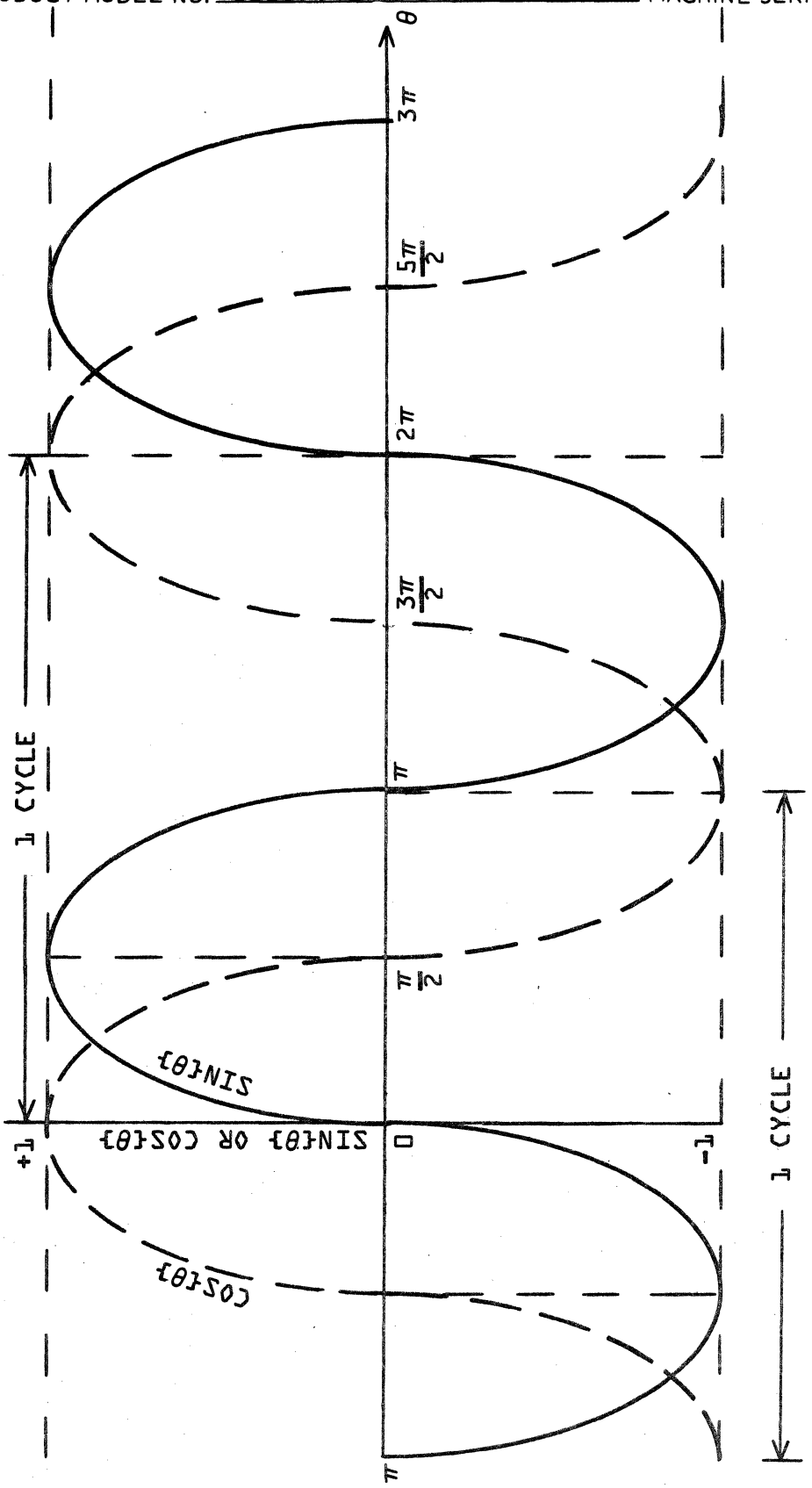

U = Pseudo Accumulator
On Entry = Value of Parameter
$c_n = \frac{1}{n!}$
U=N=0
U=Y-U.LN(2)
EXP5
EXPN5
USING U CALCULATE
$e^{Y=U} = \sum_{n=0}^{9} c_n x^n$
$U = 2^n.U$
EXP4
Y = U/LN(2)
U = Y+1/2
EXPONENT OF U=0
No
EXPN1
GET INTEGRAL PART OF U
PUT IN U,N
EXP7
RETURN
EXP 55
Yes
No
$U < \frac{LN(2)}{2}$
EXP2
No
EXP3
$U \geq 87.3365$ MAXIMUM VALUE
Yes
U = 00
TYPE OUT ERROR MESSAGE EX
No
FLAG=1
Yes
U = I/O
U = 1.0
DEXP
K = 0
U<0
Yes
No
U = -U
K = 1
FLAG=K
No
U=0
Yes


DLOG
U≤0
Yes
U = ∞
TYPEOUT LN
LNU RET
RETURN
EXPONENT = 0
LNU3
EXPONENT < 0
Yes
N+1=-0
MAKE EXPONENT POSITIVE
LNUU
FIND FLOATING EXPRESSION TO EXPRESS EXPONENT STORE IN N
N*1 0
Yes
N=-N
LNU 7
N+1=0
Yes
N=0
N+1=0
LNU 7
NOW N, N+1 IS A FLOATING POINT NUMBER
CHANGE EXPONENT OF U TO ZERO
X=U
$X_2 = U + \frac{\sqrt{2}}{2}$
$X = \frac{X - \sqrt{2}/2}{X_2}$
$X_2 = X \cdot X$
$X \equiv \sum_{n=0}^{8} C_n x^{2n+1}$
U=(N-1/2)LN(2)+X
LNU RET
U = PSEUDO ACCUMULATOR


DATAN
U 0
Yes
No
U = -U
K = 1
ARCT1
K = 0
U=0
Yes
No
U = 0
ARC END
ARCT2
ARCFLG=K
K=0
U≤1
Yes
No
U = 1/U
K = 2
IF K=0 SET
BF=1.0 AF=0
IF K=2 SET
BF=-1.0 AF= π/2
ARCT5
ARCFLG = K + ARCFLG
X = U / K = 2
IS U≤TAN (π/8)
Yes
No
K = 0
IF K=0 SET
U= π/16
IF 2 U=3 π/16
AF=AF+BF•U
IF K=0 SET
BF=TAN (π/16)
IF 2 SET BF=
(3 π/16)
ARCT6
X2 = 1 + BF • X
X = X - BF / X2
X2 = X • X
X = Σ n=0 to 8 Cn X^(2n+1)
IS ARCFLG<2
No
ARCFLG = ARCFLG-2
U = U
ARCT7
U=U+AF
IS ARCFLG=0
No
U = -U
ARC END
RETURN


X=U
U=X/2π
FIND INTEGRAL PART OF U + STORE IN U.
U=X-2πU
U = π
No
SIN12
Yes
U = 2π
No
U>2π
No
Yes
U>2^21
Yes
TYPE OUT SN
Yes
SIN2
FLAG=K
No
U = 0
Yes
U = 0
RETURN
SIN 18
DSIN
K = 0
U<0
No
Yes
U = -U
K = 1
SIN1
DCOS
U=U+π/2


U > π/2
Yes
No
U = π - U
SIN16
X = U
U = X • X
X2 = U
U = Σ (n=0 to 8) Cn X^(2n+1)
U > π
Yes
No
SIN13
FLAG = -FLAG
U = U - π
SIN14
U = π/2
Yes
No
U = 1.0
SIN 17
FLAG = 0
Yes
No
U = -U
SIN 18



COS{θ} = SIN {θ+π/2}

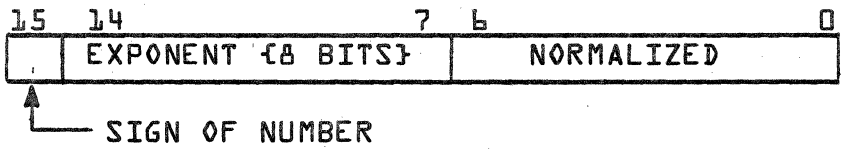
LET θ BE INPUT VALUE. IF θ<0, SIN{θ} = -SIN {|θ|}.
LET I BE INTEGRAL PART OF θ/2π. THEN β = θ -I·2π IS < 2π.
IF β > π THEN SIN{β} = -SIN{2π-β}, LET β = 2π-β
IF β > π/2 THEN SIN{β} = SIN{π-β}.

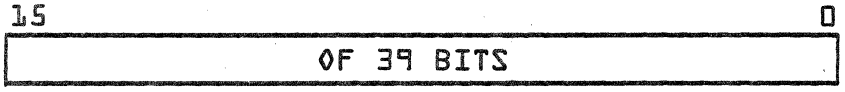
TABLE OF CONTENTS

12.0 DOUBLE PRECISION FLOATING POINT PACKAGE

12.1 General Description

Each double precision floating point number requires three consecutive words of 1700 Computer storage. The first word, containing the most significant bits of the number, is the one that is addressed. Normalized floating point format is as follows:





Thus the numbers, X, expressable are of the range $-2^{127}\{1-2^{-39}\} \leq X \leq 2^{127}\{1-2^{-39}\}$ and are significant to one part in 549 billion. If the most significant word is zero (16 bits of zero or 1), a floating point zero is assumed.

12.1.1 Exponent

The floating point exponent is an 8-bit quantity with a value ranging from 00 to FF_{16}. Through biasing by 80_{16}, this range expresses both positive and negative exponents.

12.1.2 Coefficient

The coefficient consists of a 39 bit number n, $1-2^{-39} \geq |n| \geq 0$. The high order bit position of the first word is the coefficient sign bit.

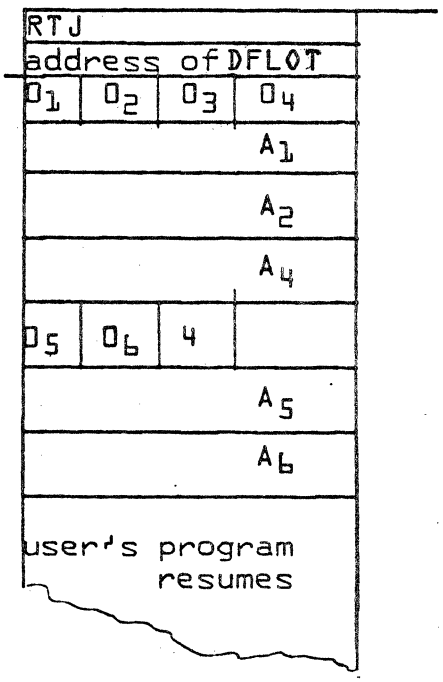
A zero denotes a positive coefficient and a one denotes a negative coefficient. When the coefficient is negative, the entire floating point number is stored in complement form.

12.1.3 Calling Sequence

DFLOT uses an interpretive calling sequence. It saves the Q, A, and I registers in Temporary Storage; it uses the communications locations $C5, $C6, and $C7 as a pseudo accumulator. The pseudo accumulator is retained in volatile or temporary storage between calls to DFLOT.

The interpretation of the calling sequence is based on a string of 4-bit bytes representing the operations, followed by the operand addresses. The leftmost 4-bit byte is the first operation; the operand addresses, if they exist, follow in the same order as the operation bytes, one word per byte. As many bytes may exist as desired, but the terminating byte must be 4, the operation FEND.

Example:



The O_i's are the operation codes; the A_i's are their operand addresses. An operand address is not needed by all operations, for instance, operation code O_3 does not have a corresponding A_3.

12.1.4 Operations

The following operations are used by the floating point package.

Operation	4-bit Code	Meaning
CHMD	5	Change of mode operation. All operand addresses following this operation code in the calling sequence are made relative if the preceding addresses were absolute, or absolute if the preceding addresses were relative. Addresses are assumed initially absolute. No operand is needed.
INDX	F_{16}	Index. The operand corresponding to INDX is used to increment the operand of the following operations: DFLDD, DFLST, DFADD, DFSUB, DFMPY, and DFDIV. Each succeeding INDX supercedes the last. No index is initially assumed.
NIDX	6	No index. The succeeding operands do not have indexing increments. NIDX supercedes any preceding INDX, and is superceded by any following INDX. NIDX is assumed initially. No operand is needed.
DFLDD	B_{16}	Double Floating load. The floating point number in the corresponding effective operand address is transferred to the pseudo accumulator located in temporary storage.
DFLST	D_{16}	Double Floating store. The floating point number is transferred from the pseudo accumulator to the corresponding effective operand address.
DFCOM	7	Double Floating complement. The pseudo accumulator is complemented. No operand is needed.
DFADD	E_{16}	Double Floating add. The contents of the effective operand address is added to the pseudo accumulator, and the sum is left in the pseudo accumulator.

DFSUB	8	Double Floating subtract. The contents of the effective operand address is subtracted from the pseudo accumulator, and the result is left in the pseudo accumulator.
DFMPY	9	Double Floating multiply. The pseudo accumulator is multiplied by the contents of the effective operand address, and the result is left in the pseudo accumulator.
DFDIV	A_{16}	Double Floating divide. The pseudo accumulator is divided by the contents of the effective operand address, and the result is left in the pseudo accumulator.
FEND	4 (also 1, 2, 3)	End of calling sequence. This operation terminates the calling sequence. No operand needed.

12.1.5 Operands

All operand addresses, absolute or relative, are placed in bits 14-0. Bit 15 is a flag set to distinguish between direct and indirect addresses.

12.1.5.1 Absolute Addressing

An absolute address may be either direct or indirect. All indirect addressing is executed prior to indexing Only one level of indirect addressing is allowed.

12.1.5.2 Relative Addressing

A relative address may be indirect to one level of indirectness. The relative address is computed by subtracting the location of the address in the calling sequence from the address contained in that location. For instance, if the relative address, A_1, is contained in location L+3, where L is the first word of the calling sequence, the relative address = $A_1 - L + 3$.

12.1.5.3 Example

Compute the floating point arithmetic statement:

$$X=-\{A\{I\}+B\{I\}\}*C\{I\}+D\{J\}*E\{J\}$$

Assuming that X, J, D, E, and the temporary location TEMP are absolutely addressed, and the other operands relatively addressed, the generated calling sequence would be:

RTJ			DFLOT
F_{16}	B_{16}	9	(6)
absolute address J			
absolute address D			
absolute address E			
D_{16}	(5)	F_{16}	B_{16}
absolute address TEMP			
relative address I			
relative address A			
E_{16}	9	(7)	(6)
relative address B			
relative address C			
(5)	E_{16}	D_{16}	(4)
absolute address TEMP			
absolute address X			

A circle around the operator indicates that the operator has no operand.

12.1.6 Fault Conditions

Fault conditions are flagged in a communication location {$C8} whenever they occur during execution. The following bits are set to one:

BIT	Fault Condition
15	Exponent overflow; the exponent in an arithmetic operation exceeds the maximum range limit.

BIT	Fault Condition
14	Divide fault; division by zero is encountered.
13	Exponent underflow; the exponent in an arithmetic operation is less than the minimum range limit.

If exponent underflow occurs, a floating point zero results. If exponent overflow occurs, the largest word of the appropriate sign results. A divide fault is treated as overflow. These error conditions may be tested with the IFALT function.

12.2 FLOATING-POINT ARITHMETIC WITH 39-BIT NUMBERS

9.2.1 Introduction

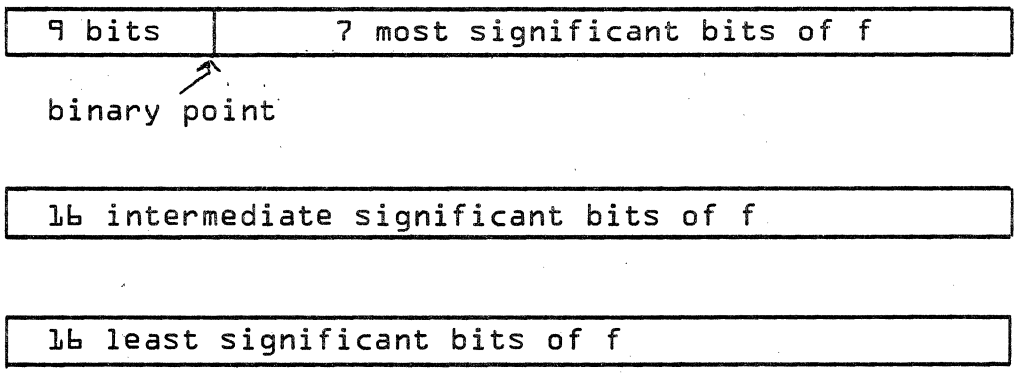
Consider the double precision floating-point number:

$$F = f \cdot 2^{\beta}, \qquad (1)$$

where $|f|$ lies in the range

$$1/2 \leq |f| \leq 1 - 2^{-39}. \qquad (2)$$

We have a machine with a word length of 16 bits and that the 48 bits in the double length word are divided in the following way:



The leftmost block of nine bits is divided into three parts:

1. The first (leftmost) bit represents the sign of F.
2. The second bit represents the sign of β.

3. The next 7 bits represent the magnitude of f. This allows 39 bits for the representation of f. We shall assume that the binary point lies at the left and of the 39 bits representing f so that the 7 most significant bits of f are stored in the first word of the three, and the 16 least significant bits of f are stored in the third word of the three.

If we write

$$|f| = c + ci \times 2^{-7} + d \times 2^{-23} \qquad \{3\}$$

where c lies in the range

$$1/2 \leq c \leq 1 - 2^{-7} \qquad \{4\}$$

ci lies in the range

$$0 \leq ci \leq 1 - 2^{-16} \qquad \{5\}$$

and where d lies in the range

$$0 \leq d \leq 1 - 2^{-16} \qquad \{6\}$$

then c represents the 7 most significant bits of f, ci represents the 16 intermediate significant bits of f, and d represents the 16 least significant bits of f.

We wish to consider the four basic arithmetic operations using double precision floating point numbers of the form discussed. Consequently, in order to have notation for two operands, let us consider a second double precision floating point number.

$$G = g.2^{S}, \qquad \{7\}$$

where $|g|$ lies in the range

$$1/2 \leq |g| \leq 1 - 2^{-39} \qquad \{8\}$$

If we write

$$|g| = a + ai \times 2^{-7} + b \times 2^{-23} \qquad \{9\}$$

where a lies in the range

$$1/2 \leq a \leq 1 - 2^{-7} \qquad \{10\}$$

ai lies in the range

$$0 \leq ai \leq 1 - 2^{-16} \qquad (11)$$

and b lies in the range

$$0 \leq b \leq 1 - 2^{-16} \qquad (12)$$

then a represents the 7 most significant bits of |g|, ai represents the intermediate significant bits of |g|, and b represents the 16 least significant bits of |g|.

The machine represents negative numbers using a "one's-complement" system. In fact the procedure for changing the sign of a floating point number is to perform a bit-by-bit complement of the entire 48 bits (including the 9 bits representing the sign and exponent).

12.2.2 THE FOUR ARITHMETIC OPERATIONS

A. Addition

$$F + G = f \times 2^{\beta} + g \times 2^{\delta} \qquad (13)$$

The basic problem in floating point addition is to adjust the exponent of F (or G) so that the binary points are lined up before the addition takes place.

Let us introduce the notation L to represent three cells containing the larger of the two numbers F and G, and the notation S to represent three cells containing the smaller of the two numbers. We shall say that F is larger than G if

$$\beta \geq \delta \qquad (14)$$

and F is smaller than G if

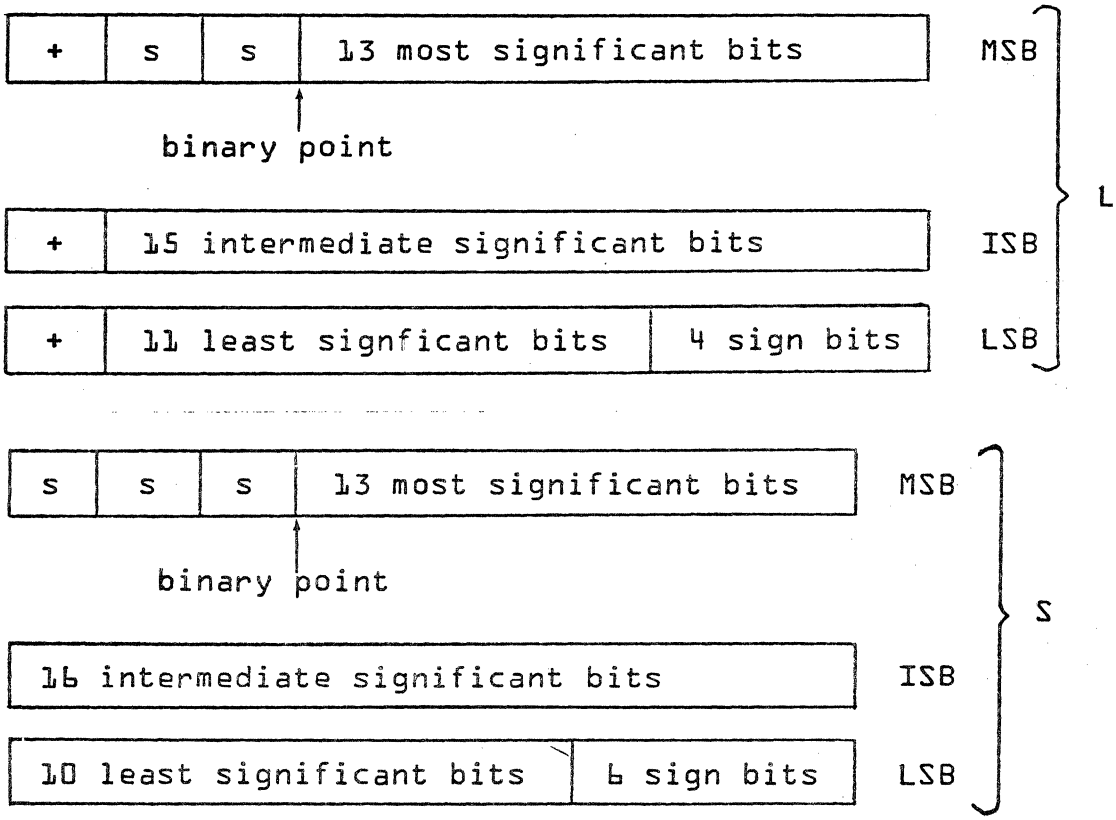
$$\beta < \delta \qquad (15)$$

We are not concerned about the relative magnitudes of f and g in case the exponents are equal. Using this convention, we process the following algorithm for forming F + G:

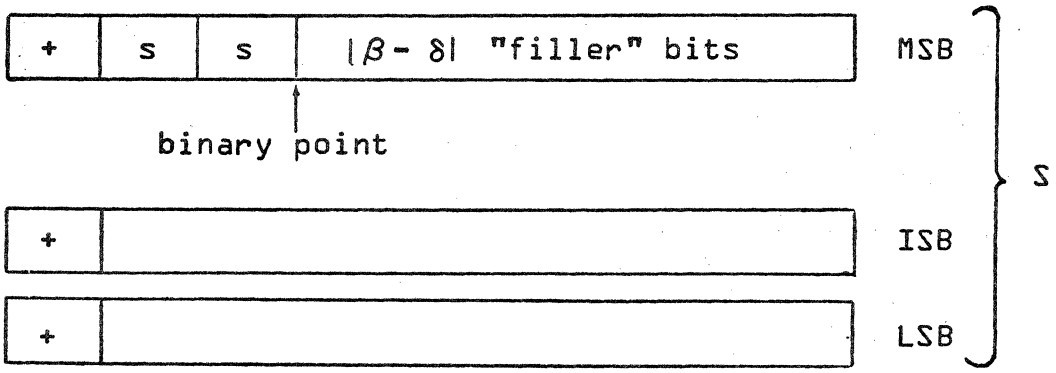
1. Record the leftmost nine bits of F and G. This, in effect, records β and δ.

2. Determine the sign of $\{\beta - \delta\}$ and thus determine whether F is smaller or larger than G according to {L4} and {L5}.

3. Place f and g in L and S. If F is larger than G, then f goes into L; otherwise, f goes into S and g goes into L. The following bit patterns should be formed {here s means "sign bit"}:



4. Shift S right $|\beta - \delta|$ places and put a "+" bit at the beginning of each of the three words. If $|\beta - \delta| = 39$, then there is no need to continue since all significant bits in S will be lost.

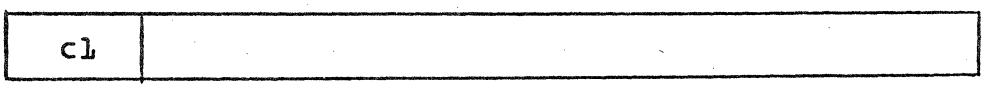


Notice that the $|\beta - \delta|$ "filler" bits, between the binary point in S and the most significant bit of the function, are sign bits. This is mathematically correct in a one's complement representation of negative numbers.

5. Add the LSB portions of L and S.

We call the first bit of this sum c. If it is a one, we actually have a carry. However, it usually is easier to add c (see step 6) than to test to see whether or not it needs to be added as a carry bit in forming the sum of the ISB portions of L and S.

6. Add the ISB portions of L and S and the carry bit from step 5.



If c1 is set to one, we have a carry and we will add c1 to step 7 in forming the sum of the most significant bits of L and S.

7. Add the MSB portions of L and S and add the carry bit obtained from step 6.

If e = 1 then an end-around-carry must be performed. This means that a one is added at the right end of the word produced in step 5. Since this might also produce a carry bit, the c in the diagram (see step 5) must be cleared to zero before the end-around-carry. If a carry bit is again produced, then a one must be added at the right end of the word produced in step 6. Since this might also produce a carry bit, the c1 in the diagram (see step 6) must be cleared to zero before the end-around-carry. If a carry bit is again produced, then a one must be added at the right end of the word above. It can be shown that this last operation can never produce another e = 1.

If v = s then v is a sign bit. However, if v ≠ s then there has been overflow during the addition, and the v is the most significant bit of the sum. In the latter case an adjustment of the exponent will be necessary to give the correct answer.

8. Shift the LSB portion of the sum left one place to clear out the carry bit c. Then shift the ISB portion of the sum left one place to clear out the carry bit c1. Then shift the LSB portion of the sum one place and put the bit shifted off into the right most bit of the ISB portion of the sum. Then shift the triple length sum left (a) one place if v ≠ s; (b) two places if v = s.

This leaves the sum in the following form:

s	15 most significant bits of the sum

16 intermediate significant bits of the sum

at least 8 bits of the sum	sign bits

9. If the triple length sum was shifted one place left in step 7 (v ≠ s) then the exponent must be adjusted to take care of the overflow. This means adding one to the exponent β or δ, whichever is larger. (This will be the exponent of the sum.) If the double length sum was shifted two places left in step 7, no adjustment of exponent is necessary.

10. The form of the sum given by step 8 must be checked for normalization since it is possible that several of the leading bits of the sum may be zero. {Cancellation occurs when two numbers of opposite sign but nearly equal magnitude are added.} If the sum is not normalized at this point, appropriate adjustments in the exponents should be made.

    If 39 left shifts are not sufficient for normalization, then the sum should be made zero.

11. At this point the normalized sum may be rounded although the extra coding involved may not be worth the gain. If rounding is desired, then there are two cases to be considered depending on the sign of the sum. These cases require that care be taken in handling any carry bit produced by the rounding operations.

12. Now pack the 39 most significant bits of the sum, along with 9 bits representing the sign and exponent, into three 16-bit words {in the standard way}. If the sign of the sum is negative, then the first 9 bits must be complemented before the packing takes place.

B. Subtraction

No special subroutine is necessary since

$$F - G = F + \{-G\}$$

and one merely complements G before entering the addition subroutine.

C. Multiplication

$$F \times G = \{f \times 2^{\beta}\}\{g \times 2^{\delta}\} \qquad \{16\}$$

$$= \{\text{sign } F \times G\}\ |f| \times |g| \times 2^{\beta+\delta}$$

The computational procedure is primarily concerned with the formation of $|f| \times |g| \times 2^{\beta+\delta}$ since {sign F x G} can be recorded in advance and used later to apply the correct sign to the product. In addition to recording {sign F x G}, we record the exponents β and δ after the product $|f| \times |g|$ is formed. In fact, we propose the following algorithm for multiplying F by G:

1. Determine and record {sign F x G}

2. Form |F| and |G|

3. Record the leftmost nine bits of |F| and |G|. This, in effect, records β and δ.

4. Shift the 39 bits of |f| and |g| left until each has the bit pattern.

+	15 most significant bits	C and A
0	15 intermediate significant bits	Ci and Ai
0	9 least significant bits	6 zeros	D and B

If this procedure is followed, f is no longer represented by {3} during the computation in step 5, below but has the form:

$$f = C + Ci \times 2^{-15} + D \times 2^{-30} \quad \{17\}$$

where C, Ci, and D lie in the following range

$$2^{-1} \leq C \leq 1 - 2^{-15} \quad \{18\}$$

$$0 \leq Ci \leq 1 - 2^{-15} \quad \{19\}$$

$$0 \leq D \leq 1 - 2^{-9} \quad \{20\}$$

likewise |g| has the form

$$|g| = A + Ai \times 2^{-15} + B \times 2^{-30} \quad \{21\}$$

where A, Ai, and B lie in the following range

$$2^{-1} \leq A \leq 1 - 2^{-15} \quad \{22\}$$

$$0 \leq Ai \leq 1 - 2^{-15} \quad \{23\}$$

$$0 \leq B \leq 1 - 2^{-9} \quad \{24\}$$

5. Use fixed point operations in forming the product.

$$f \times g = \{C + Ci \times 2^{-15} + D \times 2^{-30}\} \{A + Ai \times 2^{-15} + B \times 2^{-30}\}$$

$$= CA + \{CAi + CiA\} \times 2^{-15} + \{DA + CiAi + CB\} \times 2^{-30} + \{DAi + CiB\} \times 2^{-45} + DB \times 2^{-60}$$

$$= CA + \{CAi + CiA\} \times 2^{-15} + \{DA + CiAi + CB\} \times 2^{-30} \quad \{25\}$$

Notice that the terms $\{DAi + CiB\} \times 2^{-45}$ and $DB \times 2^{-60}$ may be ignored because once the product is placed back in standard form, only 39 bits are retained. The following computational steps are performed.

a. Form CA giving a double length product.

b. Form DA and retain the most significant half of the double length product.

c. Form CiAi and retain the most significant half of the double length product.

d. Form CiA giving a double length product.

e. Add the most significant half of DA to the most significant half of CiAi.

f. Form CB and retain the most significant half of the double length product.

g. Add the most significant half of DB to the sum obtained in {e}.

h. Add the least significant half of CiA to the sum obtained in {g}.

i. Form CAi giving a double length product.

j. Add the least significant half of CAi to the sum obtained in {h}. This result is the least significant portion of the triple length product.

k. Add the most significant half of CiA to CAi.

l. Add the least significant half of CA to the sum obtained in {k}. This result is the intermediate significant portion of the triple length product. The first half of the double length product is the most significant half of CA which was formed above in {a}.

6. Next, round and normalize the product obtained using {25} in step 5. Any adjustment in the exponent $\beta + \delta$ which is necessary because of the normalization of {f} x {g} must be performed.

7. Finally, pack the 39 bits of the normalized product and the 9 bits representing the sign and the adjusted exponent into three 16 bit words {in the standard way}. If the {sign F.G} is negative, the two words must then be complemented to give the correct sign to the product.

D. Division

$$\frac{G}{F} = G \times \frac{1}{F} = \{g \times 2^{\delta}\} \times \left\{\frac{1}{f \times 2^{\beta}}\right\}$$

$$= \{\text{Sgn } G \times F\} \times |g| \times \left|\frac{1}{f}\right| \times 2^{\delta-\beta} \qquad \{26\}$$

As a matter of fact, since we want

$$\left|\frac{g}{f}\right| < 1, \qquad \{27\}$$

We scale the numerator and write

$$\frac{G}{F} = \{\text{Sgn } G \times F\} \times |g| \times \left|\frac{\frac{1}{2}}{f}\right| \times 2^{\delta-\beta+1} \qquad \{28\}$$

Thus we propose the following algorithm for dividing G by F:

1. Determine and record {Sgn F x G}.

2. Form {F} and {G}.

3. Record the leftmost eight bits of {F} and {G}. This, in effect, records β and δ.

4. Arrange the 39 bits of {f} to give the bit pattern:

+	15 most significant bits of {f}	A

0	15 intermediate significant bits of {f}	Ai

0	9 least significant bits	6 zeros	B

and the 39 bits which represent the number 1.0 to give the bit pattern:

+	0	14 most significant bits of 1/2	$2000	α

0	15 intermediate significant bits of 1/2	$0000	γ

0	10 least significant bits of 1/2	5 zeros	$0000	ϵ

5. Use fixed point operations in forming the quotient.

$$\frac{1}{A + Ai \times 2^{-15} + B \times 2^{-30}} = \frac{\alpha + \gamma \times 2^{-15} + \epsilon \times 2^{-30}}{A + Ai \times 2^{-15} + B \times 2^{-30}}$$

where: $\alpha = \$2000$
$\gamma = \$0000$
$\epsilon = \$0000$

$$= \frac{\alpha}{A} + \frac{1}{A}\left\{\gamma - \alpha\frac{Ai}{A}\right\} \times 2^{-15} + \frac{1}{A}\left\{\epsilon - \frac{\alpha B}{A}\right\} \times 2^{-30} - \frac{Ai}{A^2}\left\{\gamma - \frac{\alpha Ai}{A}\right\} \times 2^{-30}$$

$$= \frac{1}{A}\left[\alpha - \frac{\alpha}{A}\left[Ai \times 2^{-15} + \left\{B - \frac{Ai^2}{A}\right\} \times 2^{-30}\right]\right]$$

Any terms beginning with 2^{-45} are ignored because only 39 bits of the quotient are retained. The following computational steps are performed:

a. Form $-Ai^2$ giving a double length product.

b. Divide $-Ai^2$ {as a double length dividend} by A.

c. Form $B - \frac{Ai^2}{A}$ {rounded to single length}.

d. Form the double length dividend

$$Ai \times 2^{-15} + \left[\left\{B - \frac{Ai^2}{A}\right\} \times 2^{-30}\right]$$

{The sign of the second term makes this tricky}.

e. Divide the double length dividend by A and multiply the result by α. The multiply is accomplished by shifting the result of the divide.

f. To obtain the second half of the double length quotient, the remainder resulting from the division in the previous step must now be divided by A.

g. Form α {the most significant bits of 1/2} and the result obtained from step e as a double length dividend.

h. Divide the double length dividend by A. The result is the most significant bits of the quotient.

i. Form the remainder of step h and the result of step f as a double length dividend.

j. Divide the double length dividend by A. The result is the intermediate significant bits of the quotient.

k. Divide the remainder obtained in step j by A. The result is the least significant bits of the quotient.

l. Next, round and normalize the 3 word quotient using {29} and the procedure of step 5. Any adjustment in the exponent which is necessary because of normalization of $\left|\frac{\frac{1}{2}}{f}\right|$ must be performed.

m. The 3 word quotient is then multiplied by $|g|$.

n. Next, round and normalize the product. Any adjustment in the exponent $\beta+\delta$ which is necessary due to the normalization must be performed.

o. Finally pack the 39 bits of the normalized quotient and the 9 bits representing the sign and the exponent into three 16 bit words (in the standard way). If (Sgn F x G) is negative, the three words must then be complemented to give the correct sign to the quotient.

REFERENCES

Gregory, Robert T. and Raney, James L.; Floating Point Arithmetic with 84-Bit Numbers, Communications of the ACM, Volume 1/ Number 1/ January, 1964.

12.3 Low Core Storage Location and Temporary or Volatile Storage

DFLOT sets the low core locations $C5 through $C8; they are assigned as follows:

Location Name	Location	Use
G G+1 G+2	$C5 $C6 $C7	Pseudo accumulator - the three words of the double precision floating point number are placed here.
ERRORS	$C8	Error bits to indicate exponent overflow, divide fault, and exponent underflow. These error bits are used by IFALT to check for these floating point errors.

There are two versions of DFLOT. The two packages are called by the same name (DFLOT), but one is reentrant and the other non-re-entrant. Both packages are usable by run-anywhere programs. The re-entrant package must operate in protected core. The non re-entrant package may operate anywhere.

The non-reentrant version of DFLOT utilizes temporary storage to perform its computations. The re-entrant version of DFLOT utilizes. The volatile storage concept for temporary storag

The non-re-entrant version of DFLOT requires the deck labeled DUMVOL which contains entry points AVOLA and AVOLR. This deck provides the simulated volatile storage or temporary storage required for DFLOT in the background.

DFLOT uses 30 volatile or temporary storage locations; they are assigned as follows:

Location Name	Volatile or Temporary Storage Location	Use
	0,I	Contents of Q register
	1,I	Contents of A register
	2,I	Contents of I register
G G+1 G+2	3,I 4,I 5,I	Pseudo Accumulator - the three words of the double precision floating point number are placed here. The pseudo accumulator is broken up into C, CI, D, and DELTA
SIGN	6,I	Sign register - sign double precision floating point number
ERRORS	7,I	Error bits Bit 15 indicates exponent overflow Bit 14 indicates divide fault (divide by zero) Bit 13 indicates exponent underflow. These are used by IFALT to check for these floating point errors.
F	8,I	1. Address of parameter or 2. First word of a (three word) double precision floating point number. It is used when two operands are needed, e.g. for an add. In this case F, F+1, and F+2 are broken up into A, AI, B, BETA.

Location Name	Volatile or Temporary Storage Location	Use
F+1	9,I	The second word of a {three word} double precision floating point number.
F+2	10,I	The third word of a {three word} double precision floating point number.
A	11,I	The most significant bits of the coefficient of the double precision floating point number.
AI	12,I	The intermediate significant bits of the coefficient of the double precision floating point number.
B	13,I	The least significant bits of the coefficient of the double precision floating point number.
BETA	14,I	Exponent of the double precision floating point number.
C	15,I	Most significant bits of the coefficient of the double precision floating point number.
CI	16,I	Intermediate significant bits of the double precision floating point number.
D	17,I	Least significant bits of the coefficient of the double precision floating point number.
DELTA	18,I	Exponent of double precision floating point number.
SHIFCT	19,I	Shift counter

Location Name	Volatile or Temporary Storage Location	Use
P	20,I	Pseudo program address counter which is used by DFLOT to know where it is in the users program when it wants to pick up more parameters from the variable calling sequence or return to the calling program.
RELADR	21,I	Absolute/relative indication RELADR { = 1 relative address = 0 absolute address
OPCNT	22,I	Operation counter - keeps track of which operation code is being processed.
INDEX	23,I	Effective address of parameter
OPCODE	24,I	Operation code holder - holds the operation (op) codes contained within the call.
@S	25,I	Saves @ register upon entry into DFLOT
TEMP@	26,I	Temporary Storage for many purposes
TEMGP2	27,I	Temporary Storage for may purposes
T1		Temporary Storage for MSB of 1/Divisor
T2		Temporary Storage for ISB of 1/Divisor
T3		Temporary Storage for LSB of 1/Divisor
MULDIV		Multiply/Divide Flag; = 1 for Divide; = 0 for multiply.

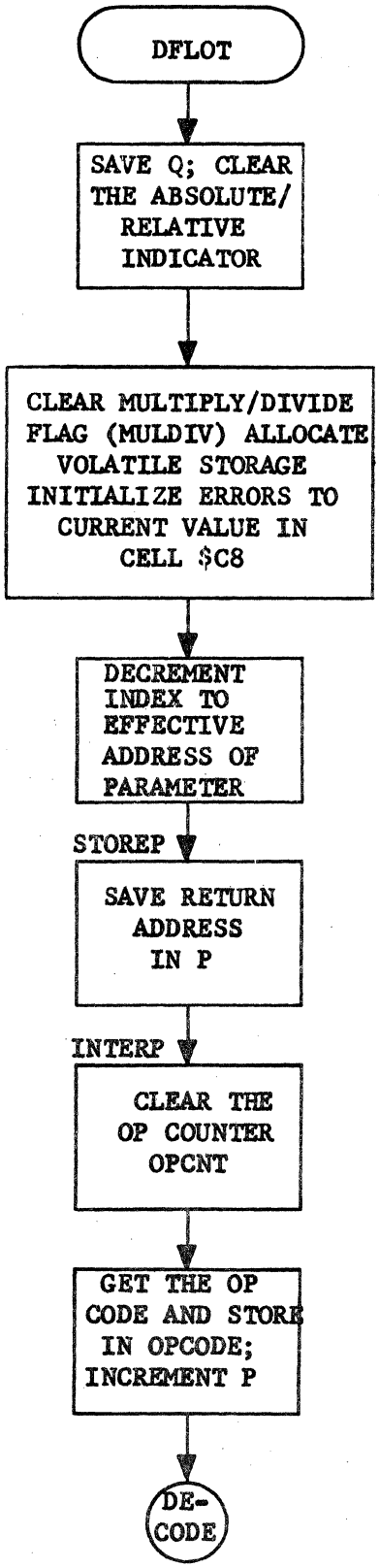
12.4 Entry Points and Externals

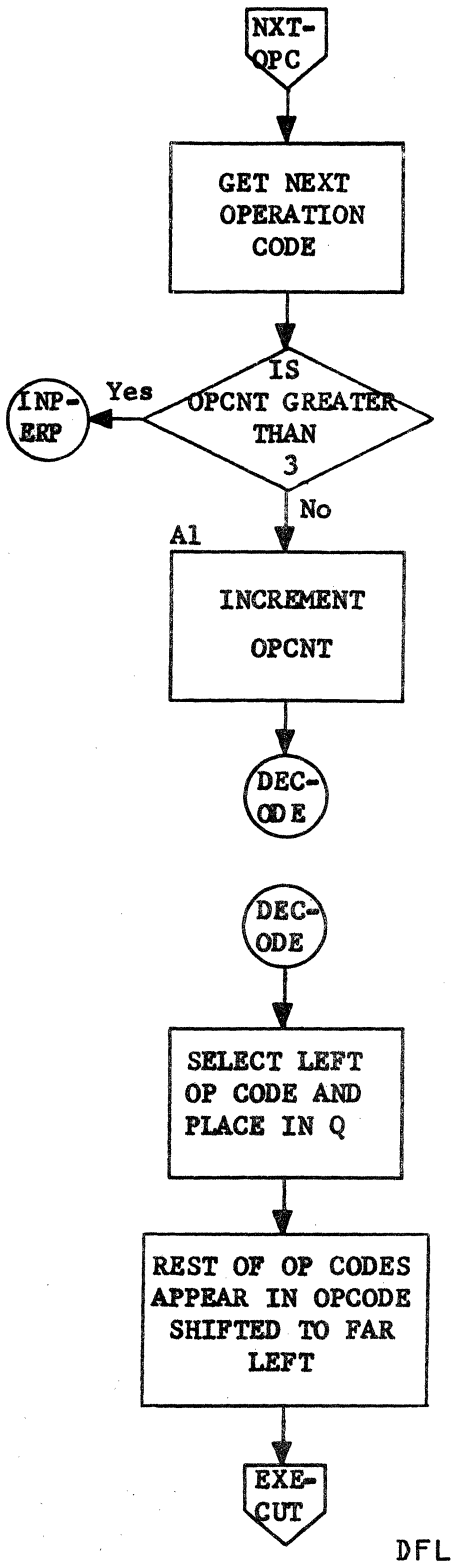
The routine DFLOT has one entry point, DFLOT. DFLOT has two externals: AVOLA and AVOLR.

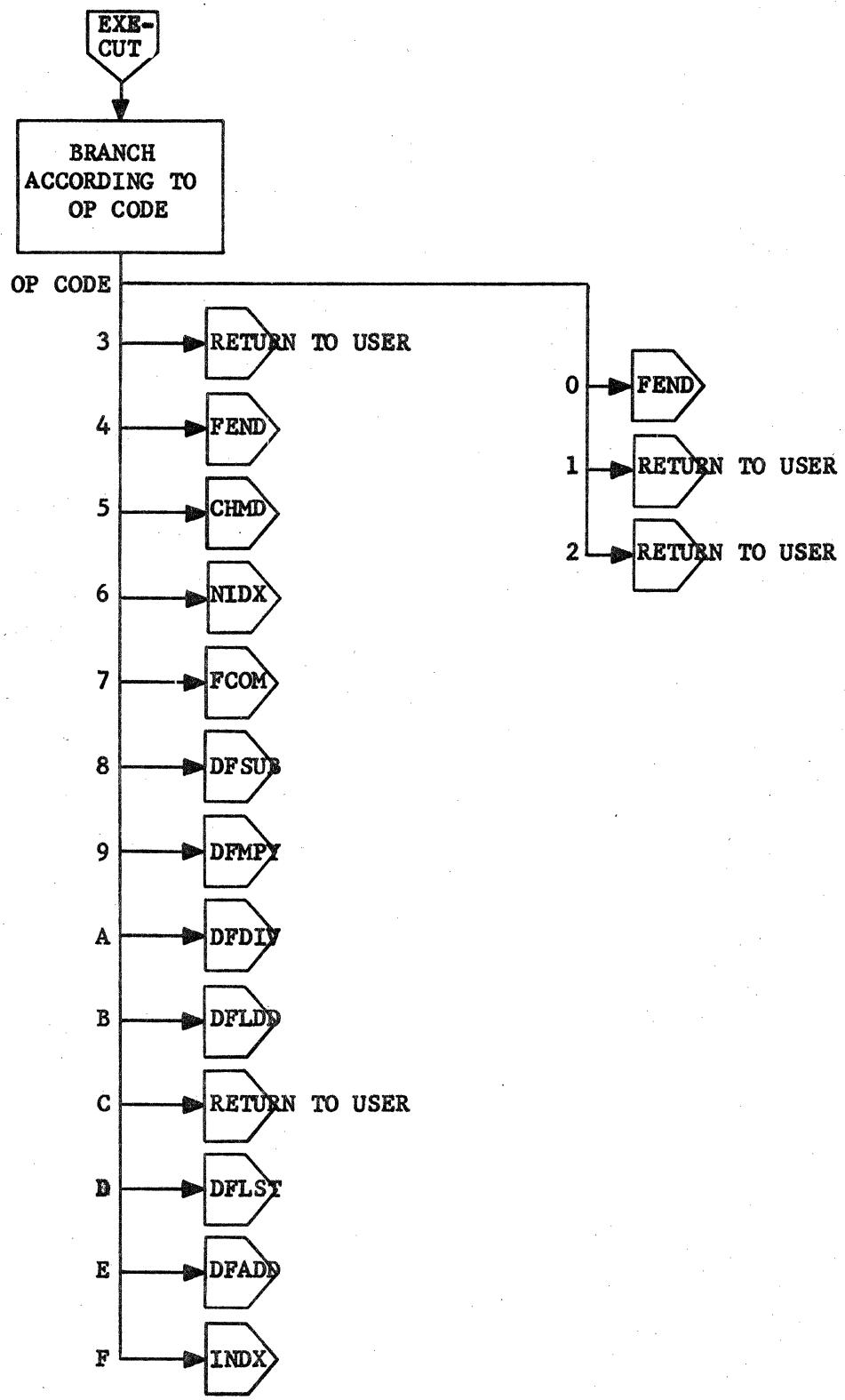
12.5 Flowchart of DFLOT

The following abbreviations are used:

MSB = most significant bits
ISB = intermediate significant bits
LSB = least significant bits


DFLOT
SAVE Q; CLEAR THE ABSOLUTE/ RELATIVE INDICATOR
CLEAR MULTIPLY/DIVIDE FLAG (MULDIV) ALLOCATE VOLATILE STORAGE INITIALIZE ERRORS TO CURRENT VALUE IN CELL $C8
DECREMENT INDEX TO EFFECTIVE ADDRESS OF PARAMETER
STOREP
SAVE RETURN ADDRESS IN P
INTERP
CLEAR THE OP COUNTER OPCNT
GET THE OP CODE AND STORE IN OPCODE; INCREMENT P
DE-CODE

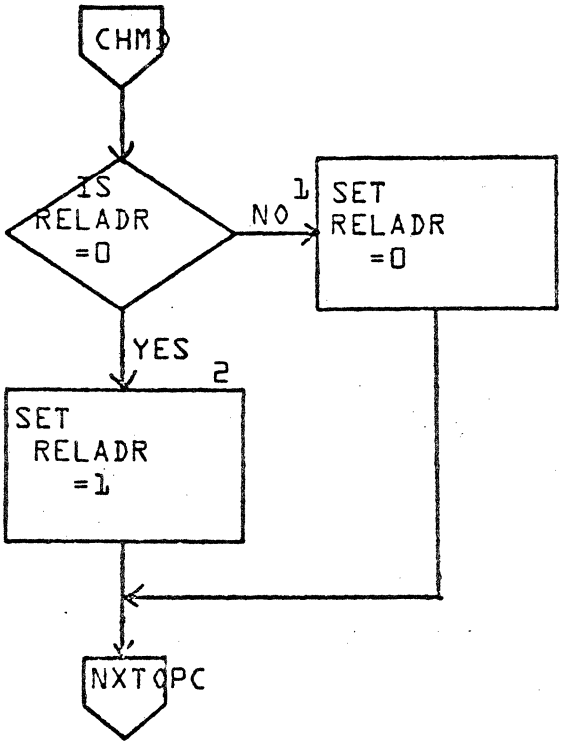

NXT-OPC
GET NEXT OPERATION CODE
IS OPCNT GREATER THAN 3
Yes
INP-ERP
No
A1
INCREMENT OPCNT
DEC-ODE
DEC-ODE
SELECT LEFT OP CODE AND PLACE IN Q
REST OF OP CODES APPEAR IN OPCODE SHIFTED TO FAR LEFT
EXE-CUT


EXE-CUT
BRANCH ACCORDING TO OP CODE
OP CODE
3
RETURN TO USER
4
FEND
5
CHMD
6
NIDX
7
FCOM
8
DFSUB
9
DFMPY
A
DFDIV
B
DFLDD
C
RETURN TO USER
D
DFLST
E
DFADD
F
INDX
0
FEND
1
RETURN TO USER
2
RETURN TO USER

OPERATION CODE =5
CHANGE MODE OF OPERATION (CHMD)

1. RELATIVE ADDRESS

2. ABSOLUTE ADDRESS



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE ☒
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

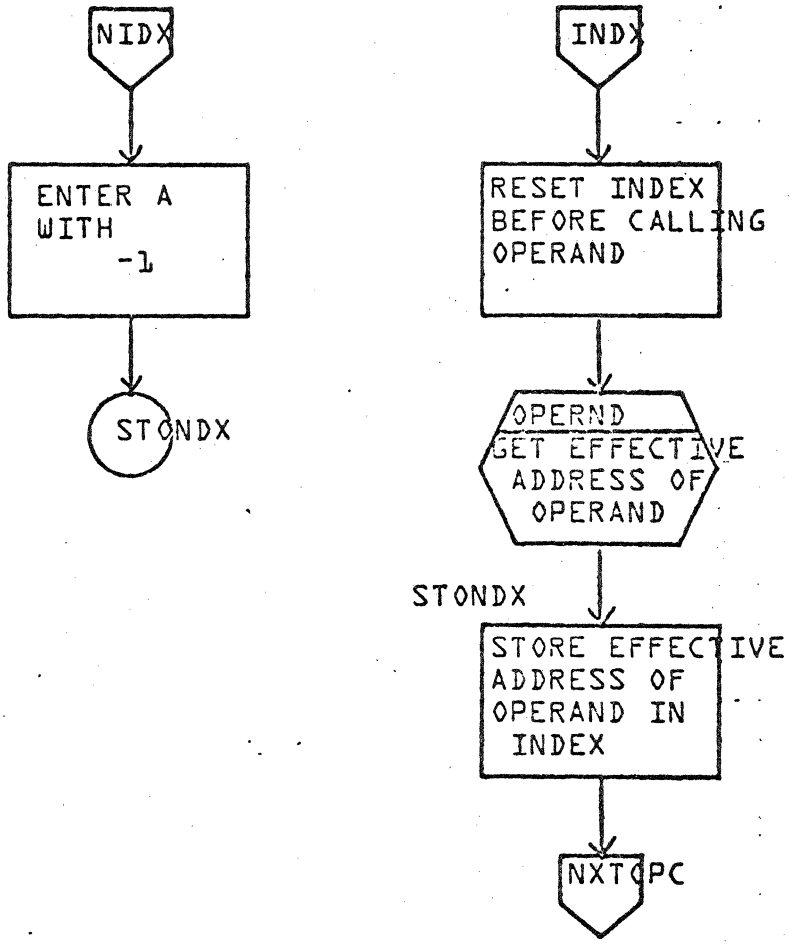
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: DFLOT
PAGE 3 OF 31
NUMBER
ISSUE DATE
DRAWN BY
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
REV
APPROVED
DATE

OPERATION CODE = 6

NO INDEX (NIDX)

OPERATION CODE = F

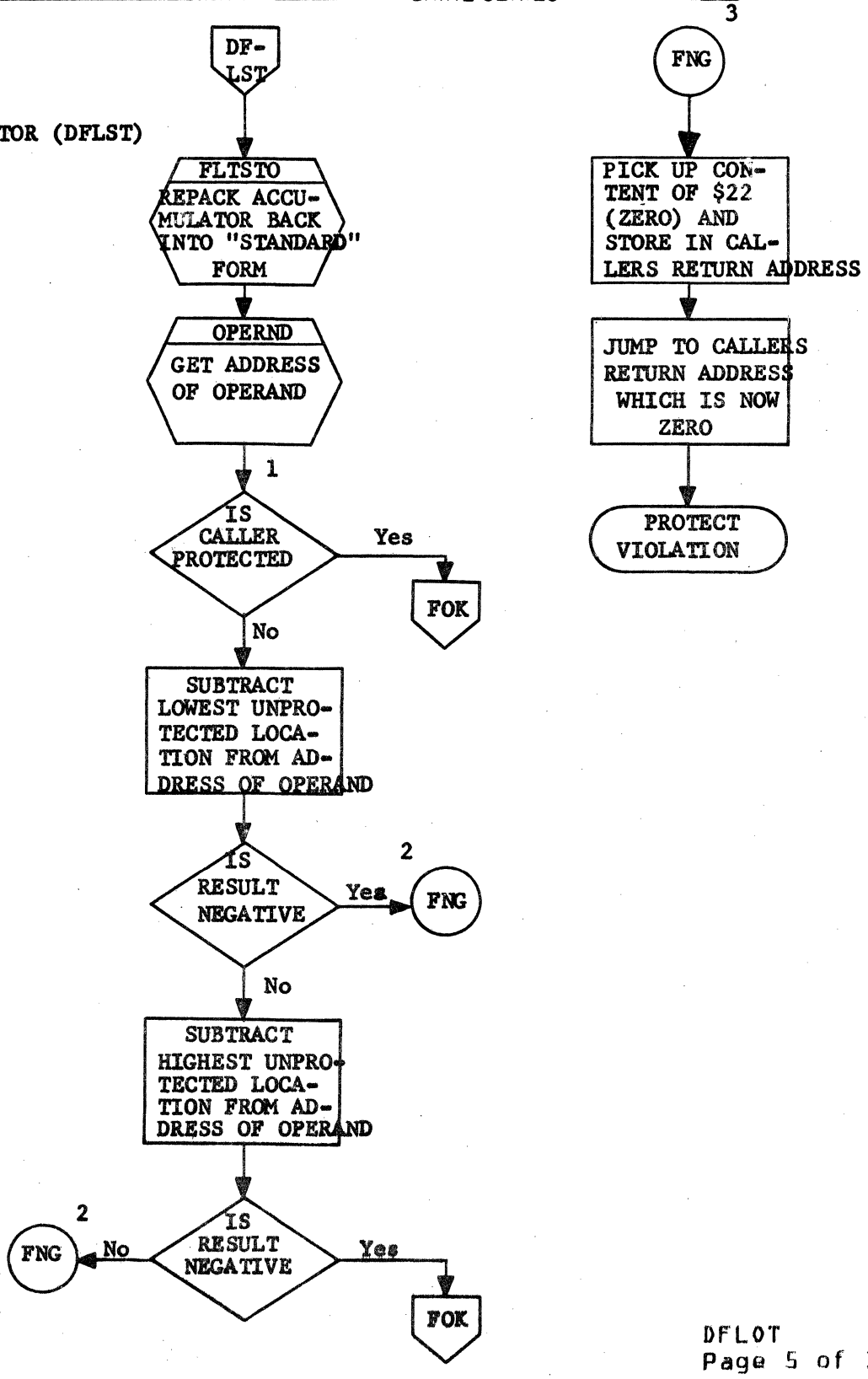
INDEX (INDX)

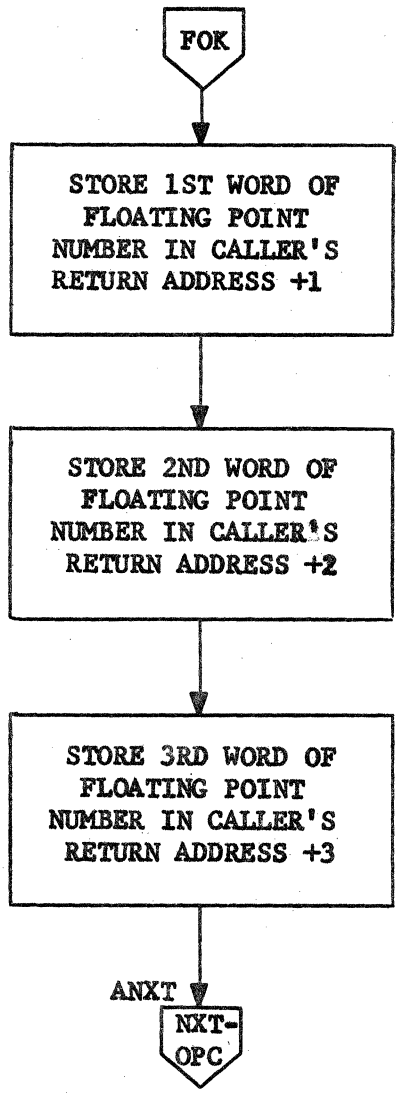


OPERATION CODE = D

DOUBLE STORE ACCUMULATOR (DFLST)

1. TEST FOR PROTECT VIOLATION.
2. STORAGE ADDRESS IS PROTECTED.
3. SIMULATE PROTECT VIOLATION.

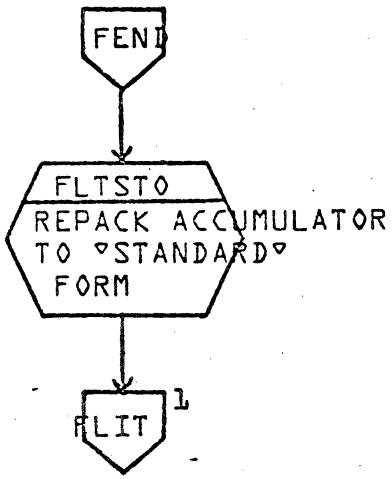



FOK
STORE 1ST WORD OF FLOATING POINT NUMBER IN CALLER'S RETURN ADDRESS +1
STORE 2ND WORD OF FLOATING POINT NUMBER IN CALLER'S RETURN ADDRESS +2
STORE 3RD WORD OF FLOATING POINT NUMBER IN CALLER'S RETURN ADDRESS +3
ANXT
NXT-OPC

OPERATION CODE = 0,4

END OF CALLING SEQUENCE (FEND)

1. RESTORE Q AND
   RETURN TO USER.



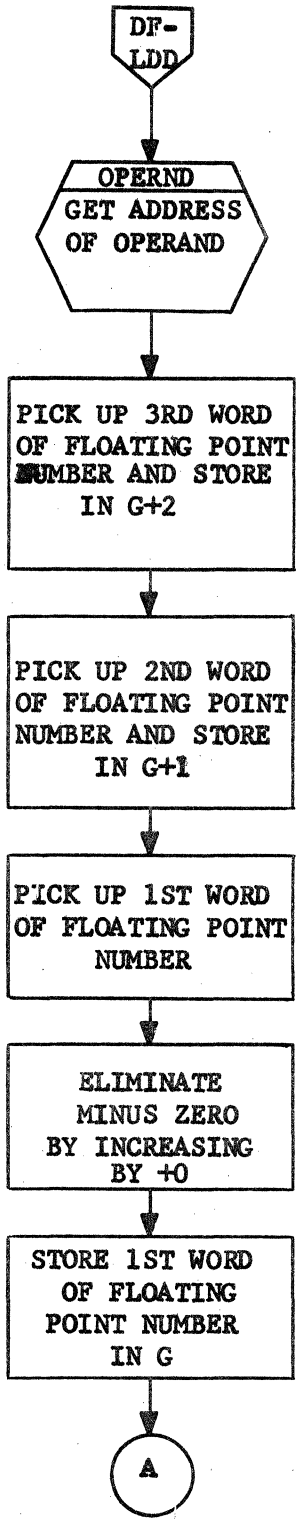
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE [X]
FLOWCHART [ ]
DECISION TABLE [ ]
OTHER [ ]

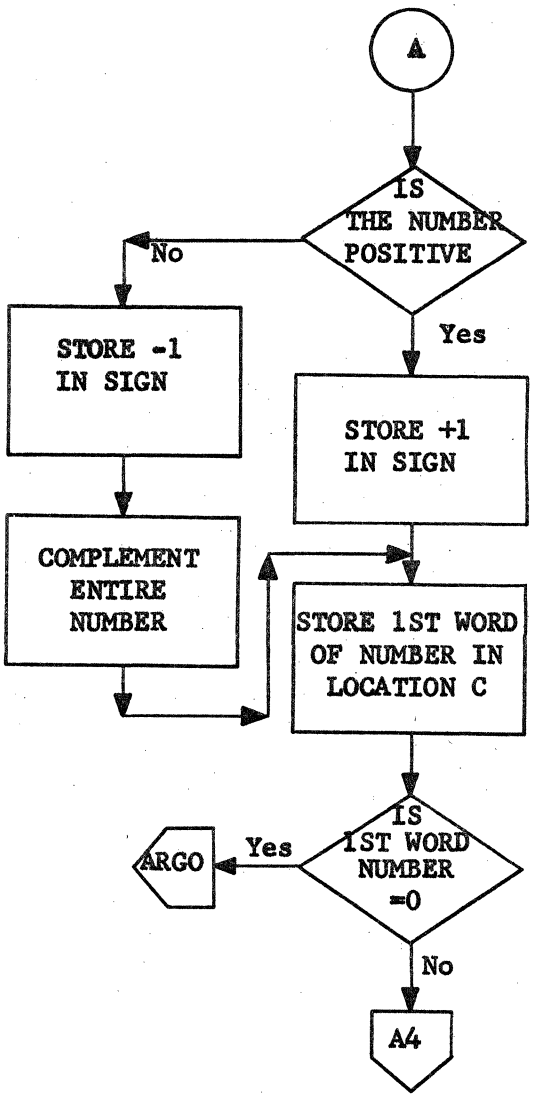
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: DFLOT
PAGE 70 OF 79
NUMBER
ISSUE DATE
DRAWN BY
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
REV
APPROVED
DATE

OPERATION CODE = B

DOUBLE FLOATING LOAD (DFLOD)

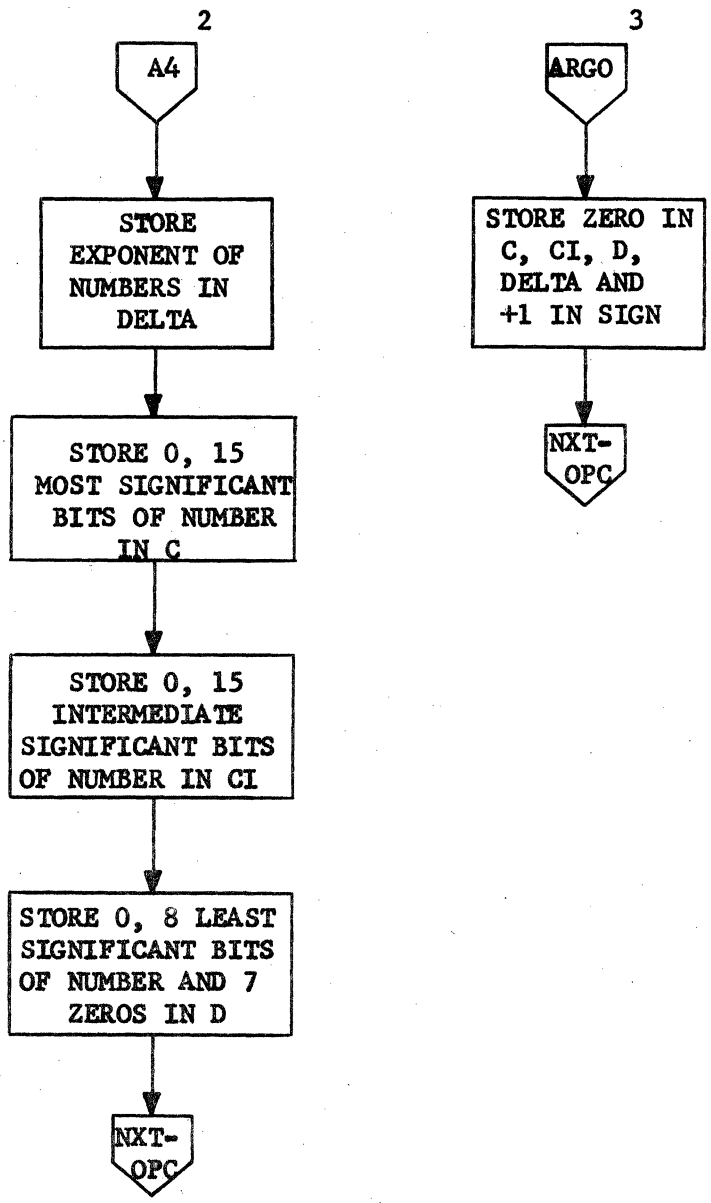
1. FOLLOWING FIVE BOXES LOAD THE OPERAND INTO G, G+1, AND G+2.





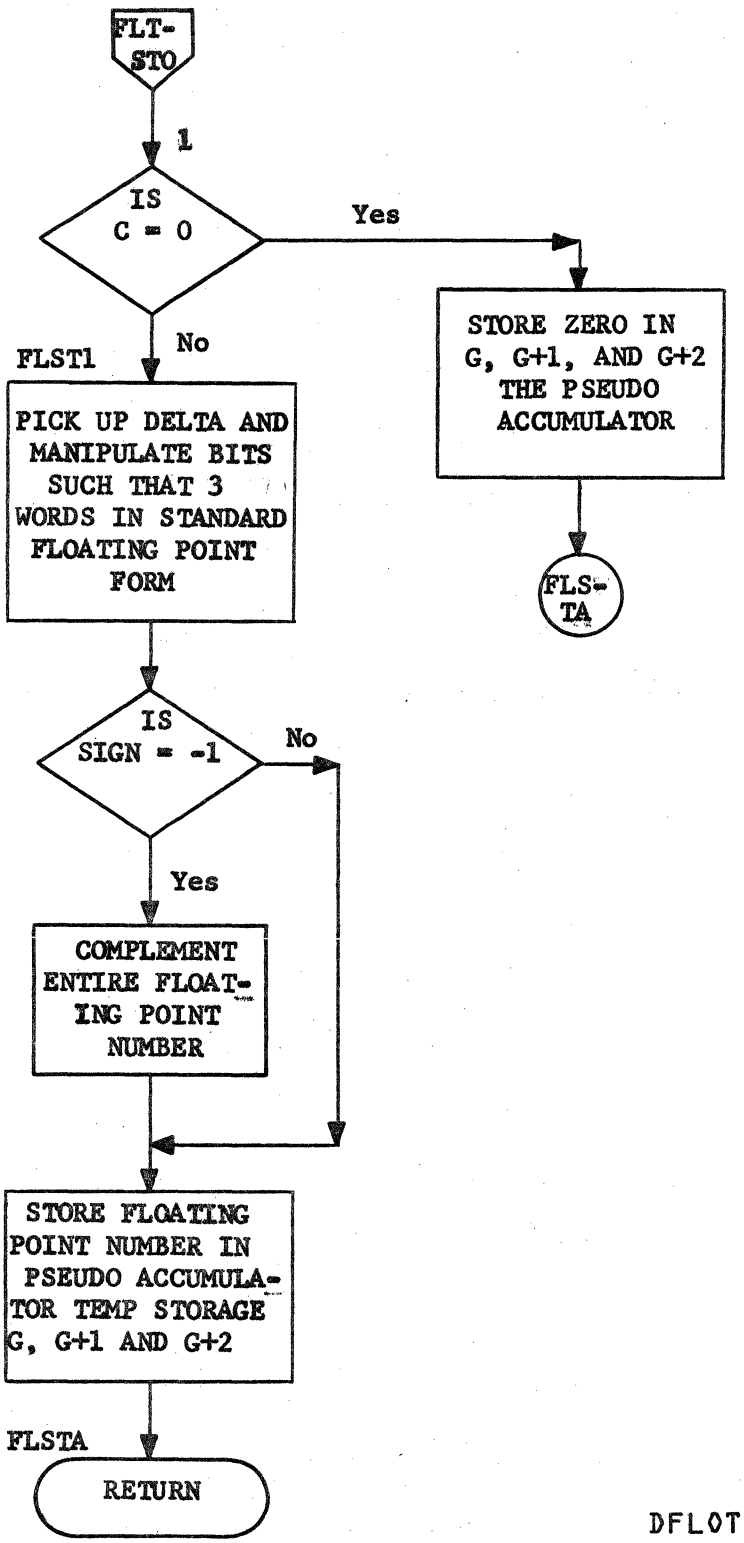
2. Unpack OPERAND into C, CI. D and DELTA.

3. Floating Point number is assumed to be zero since first word is zero.



REPACKING ACCUMULATOR

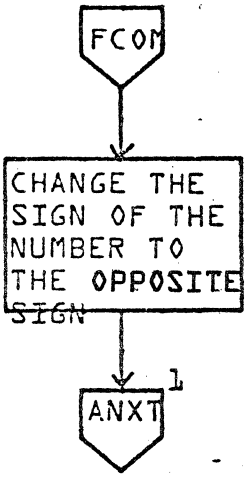
1. **If C is zero, then the PSEUDO Accumulator is zero.**



OPERATION CODE = 7

FLOATING COMPLEMENT {FCOM}

1. A JUMP TO THE LABEL ANXT TAKES YOU TO A JUMP TO THE LABEL NXTOPC-NEXT OP CODE

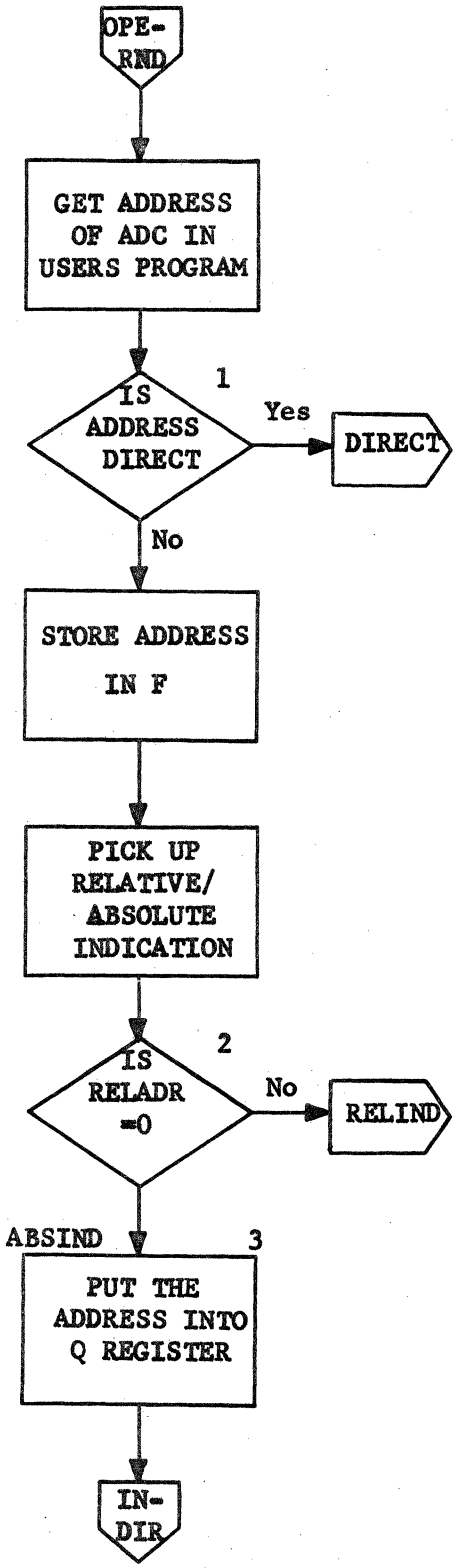


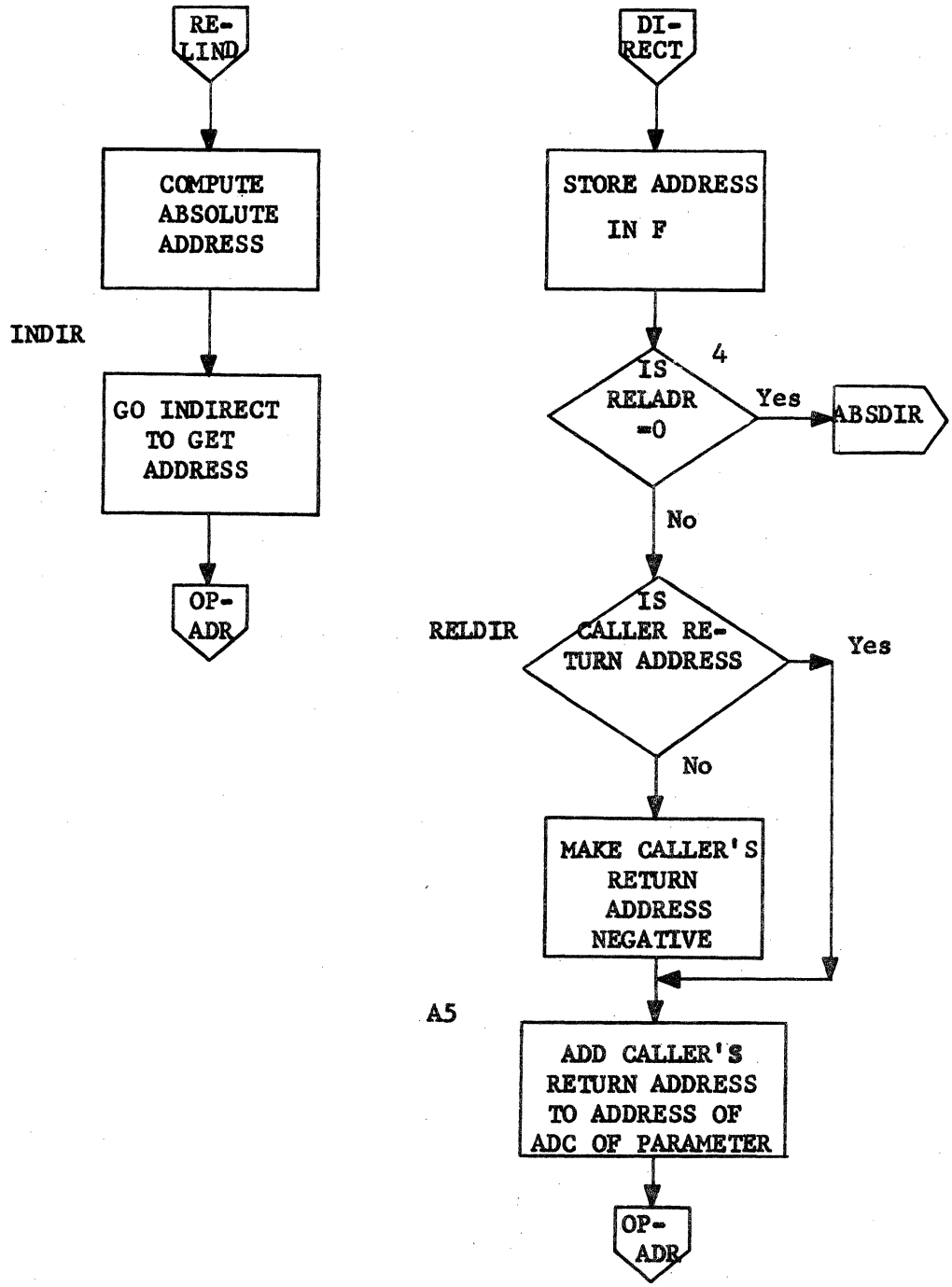
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE [X]
FLOWCHART [ ]
DECISION TABLE [ ]
OTHER [ ]

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: DFLOT
PAGE 11 of 39
NUMBER
ISSUE DATE
DRAWN BY
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
REV
APPROVED
DATE

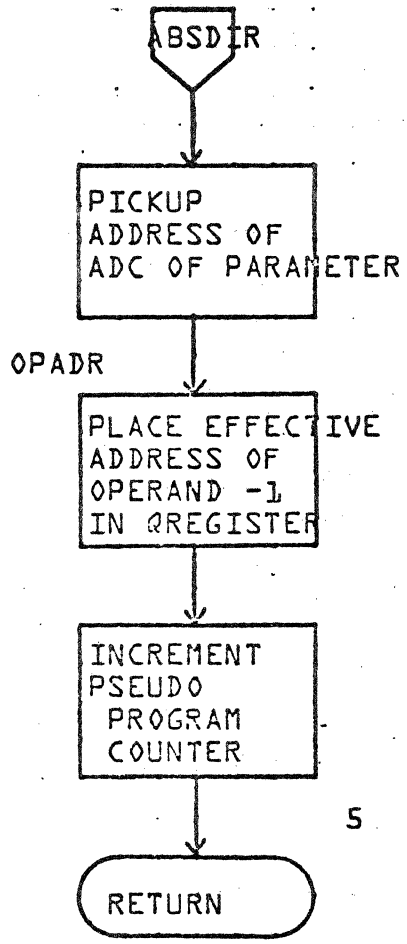
GET EFFECTIVE ADDRESS OF NEXT OPERAND

1. Address is direct if indirect bit (Bit 15) is not set.
2. RELADR=1 if relative address and=0 if absolute address.
3. Absolute and Indirect address.




RE-
LIND
COMPUTE
ABSOLUTE
ADDRESS
INDIR
GO INDIRECT
TO GET
ADDRESS
OP-
ADR
DI-
RECT
STORE ADDRESS
IN F
4
IS
RELADR
=0
Yes
ABSDIR
No
RELDIR
IS
CALLER RE-
TURN ADDRESS
Yes
No
MAKE CALLER'S
RETURN
ADDRESS
NEGATIVE
A5
ADD CALLER'S
RETURN ADDRESS
TO ADDRESS OF
ADC OF PARAMETER
OP-
ADR

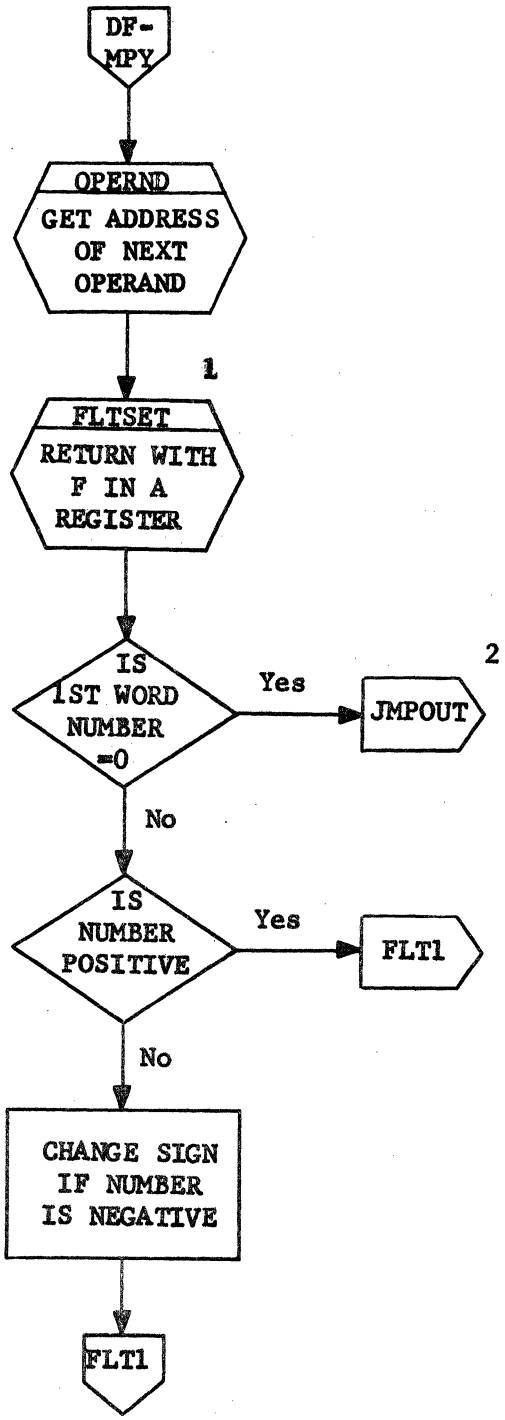
5. RETURN WITH EFFECTIVE ADDRESS OF OPERAND-1 IN QREGISTER.



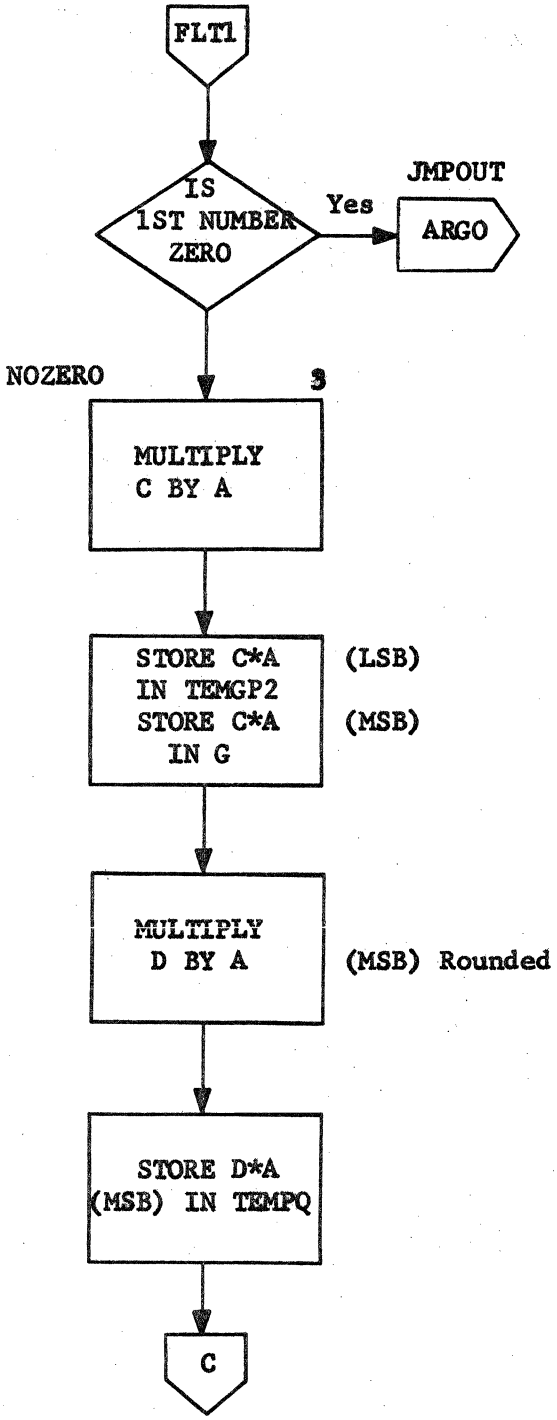
OPERATION CODE = 9

DOUBLE FLOATING MULTIPLY (DFMPY)

1. Gets the floating point number and stores in F, F+1 and F+2 then unpacks the number into
   A=0, 15MSB
   AI=0, 15ISB
   B=0, 8LSB, 7 ZEROS
   BETA=EXPONENT
2. A jump to JMPOUT causes a jump to ARGO.

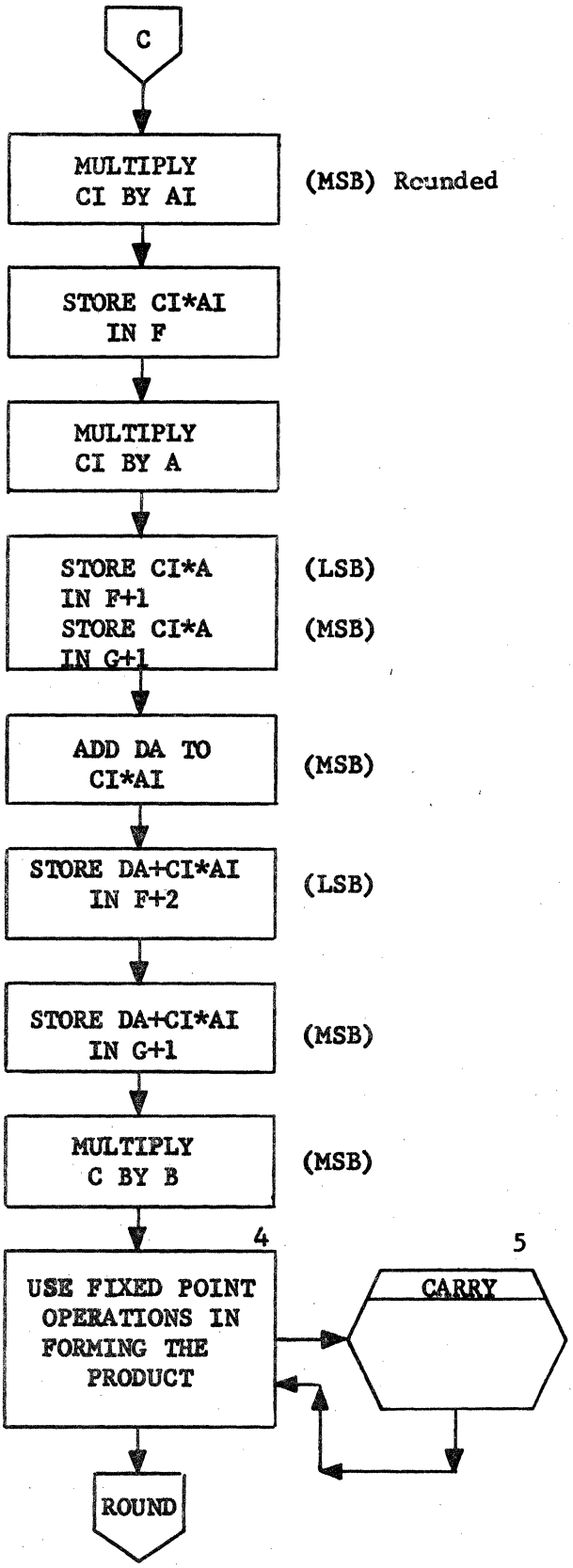


3. Multiply the MSB of one number by the MSB of the other number.



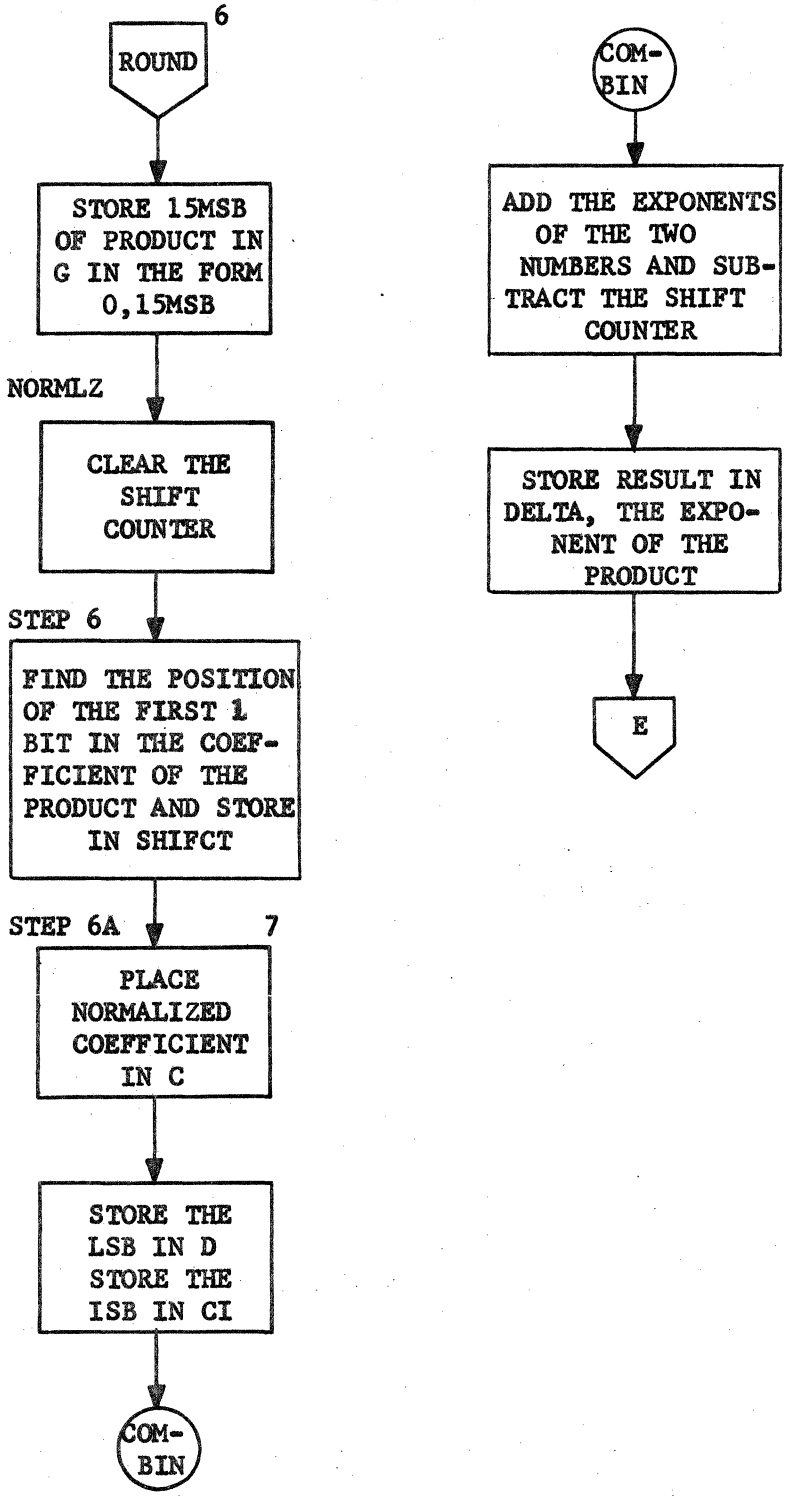
4. Product = $C*A+(C*AI+CI*A)X2^{-15}$
$+ (D*A+CI*AI+C*B)X2^{-30}$

5. This routine checks for a carry in the A and Q registers. If A is negative, carry into Q. If Q is negative, end around.

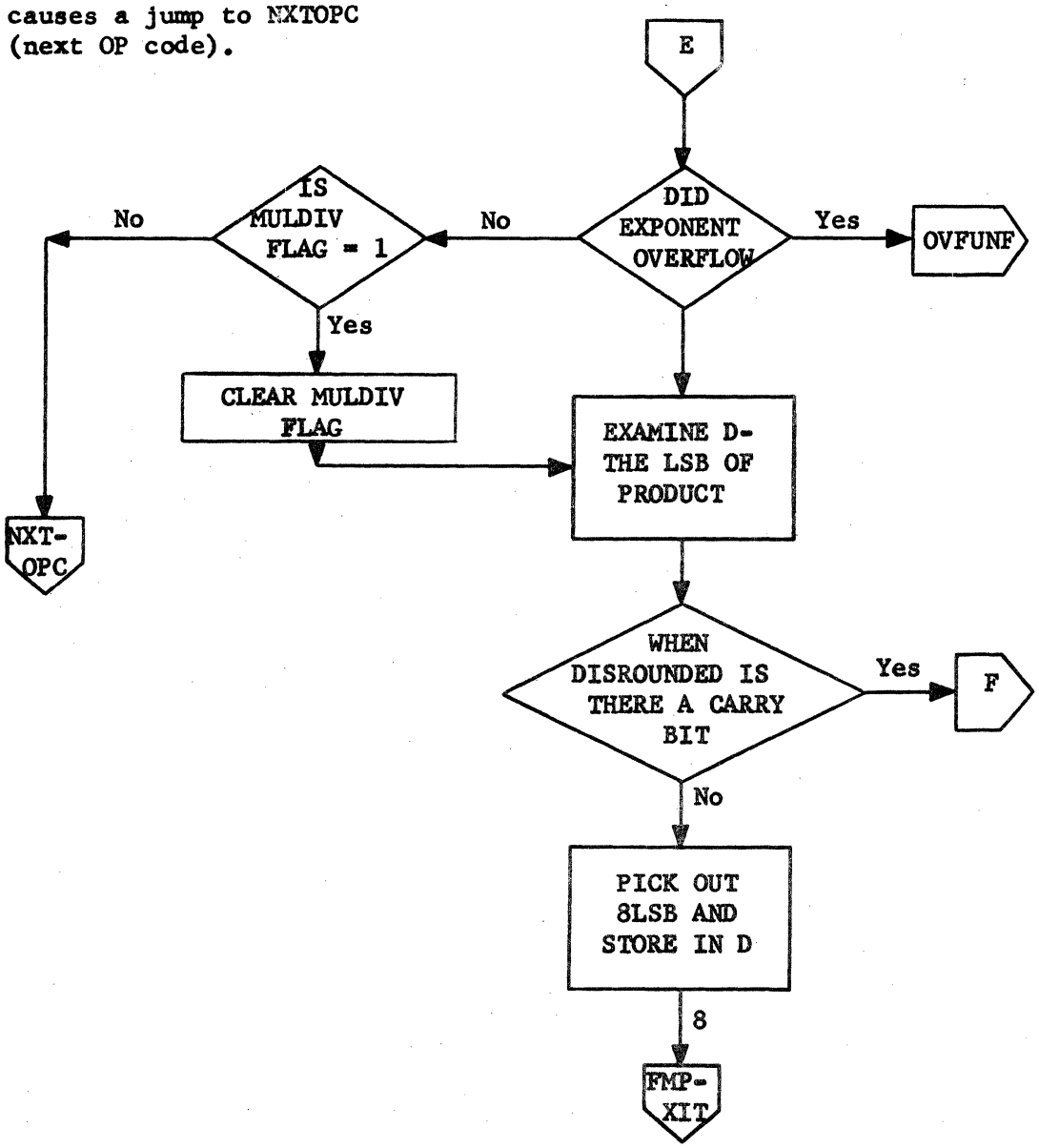


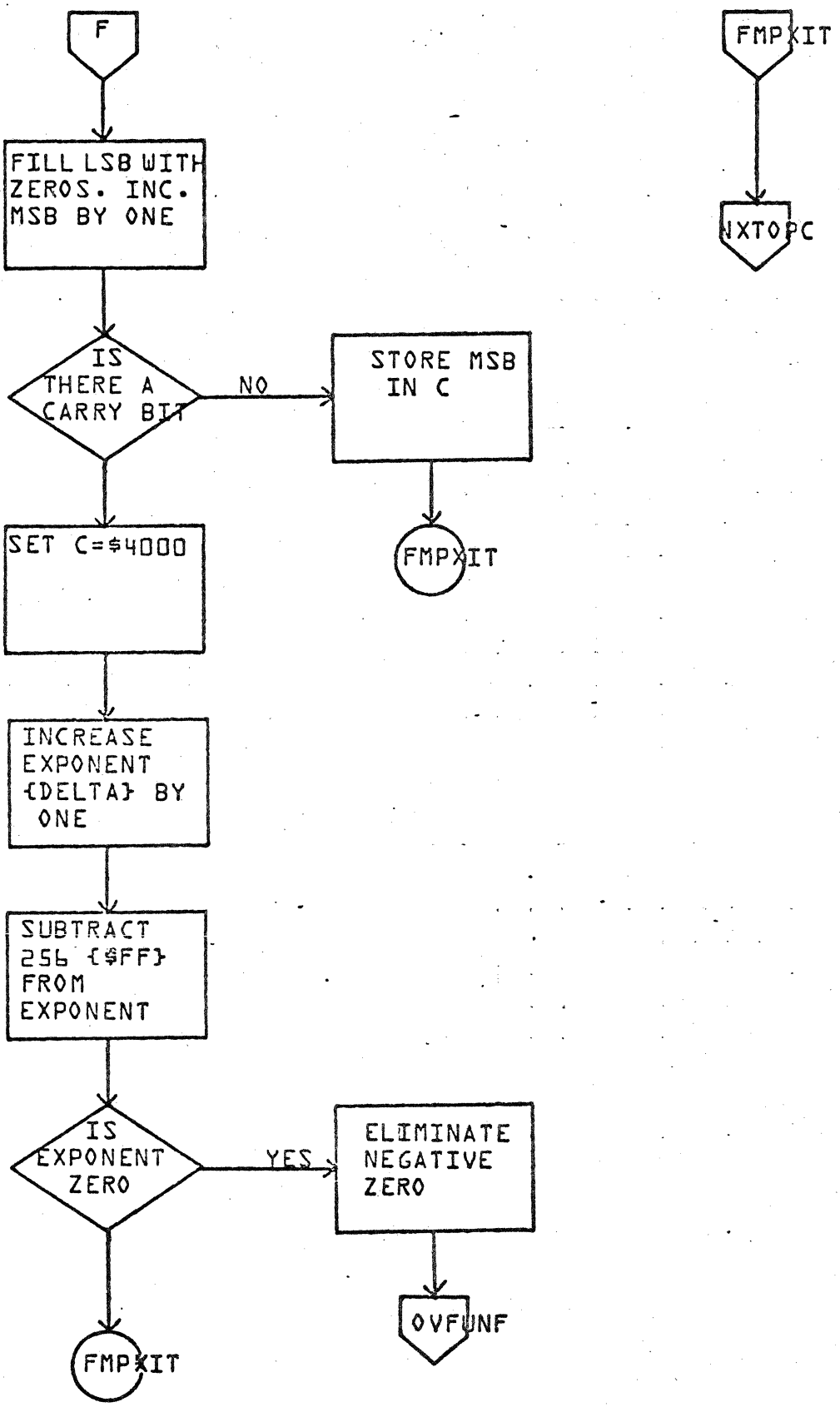
6. This section rounds and truncates to 39 bits.

7. Coefficient is of the form 0. 15MSB. but the 15MSB are normalized such that a one appears in the first location of the MSB with SHIFCT keeping track of how many places have been shifted.



8. A jump to FMPXIT then causes a jump to NXTOPC (next OP code).





LA JOLLA RESOURCE CENTER
IMS Page 12-42
1700 MASS STORAGE FORTRAN
C005*3.1 A/B

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

REV	APPROVED	DATE

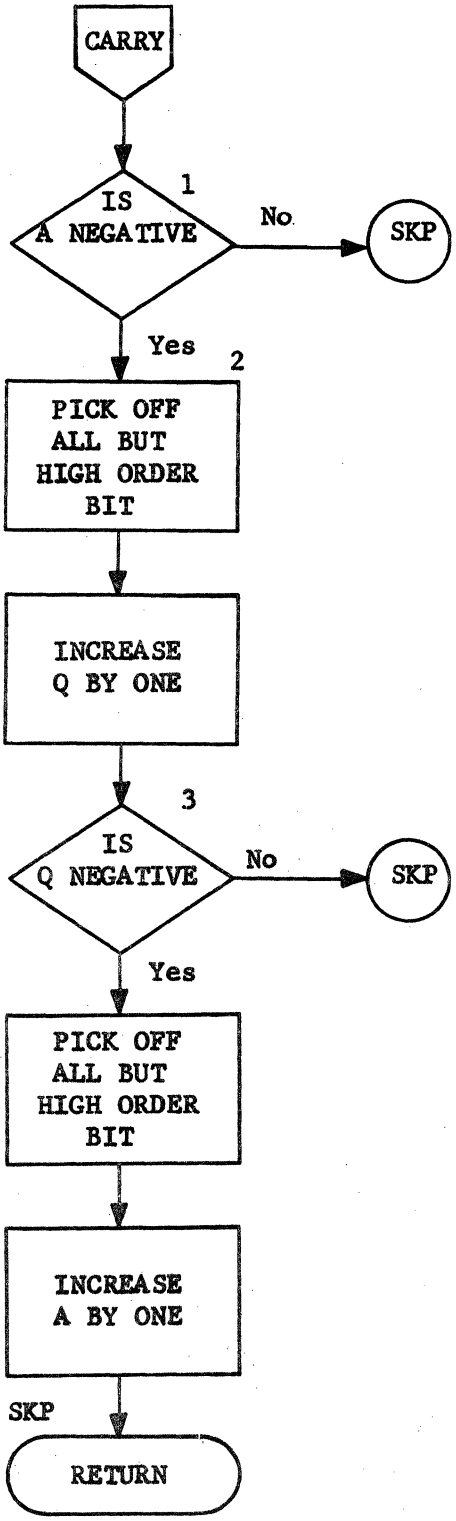
DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: DFLOT
PAGE 0089
ISSUE DATE
NUMBER
DRAWN BY
DATE

PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

PRINTED IN U.S.A.

CA127-11

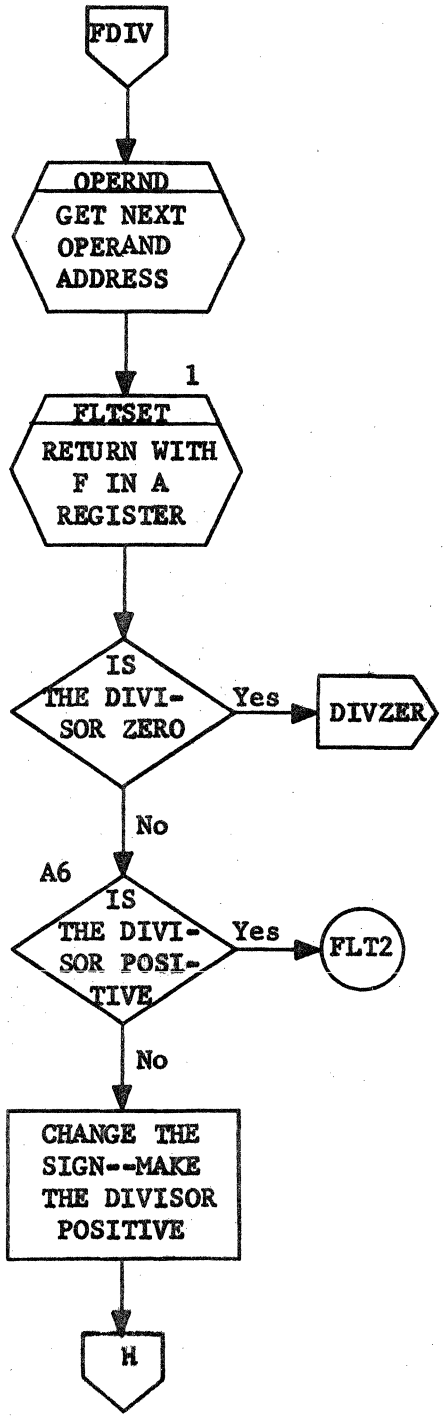
1. A contains the sum of the first word of one argument times the second word of the other argument and vice versa.
2. If A is negative, carry into Q.
3. If Q is negative, end around.



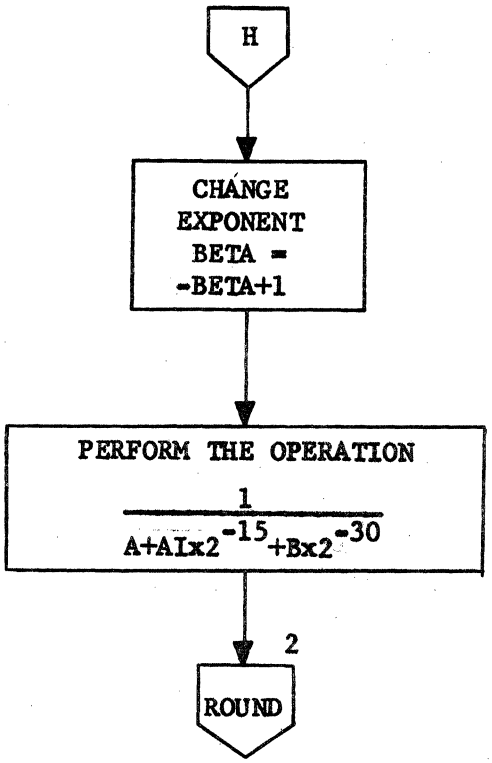
OPERATION CODE = A

FLOATING DIVIDE (FDIV)

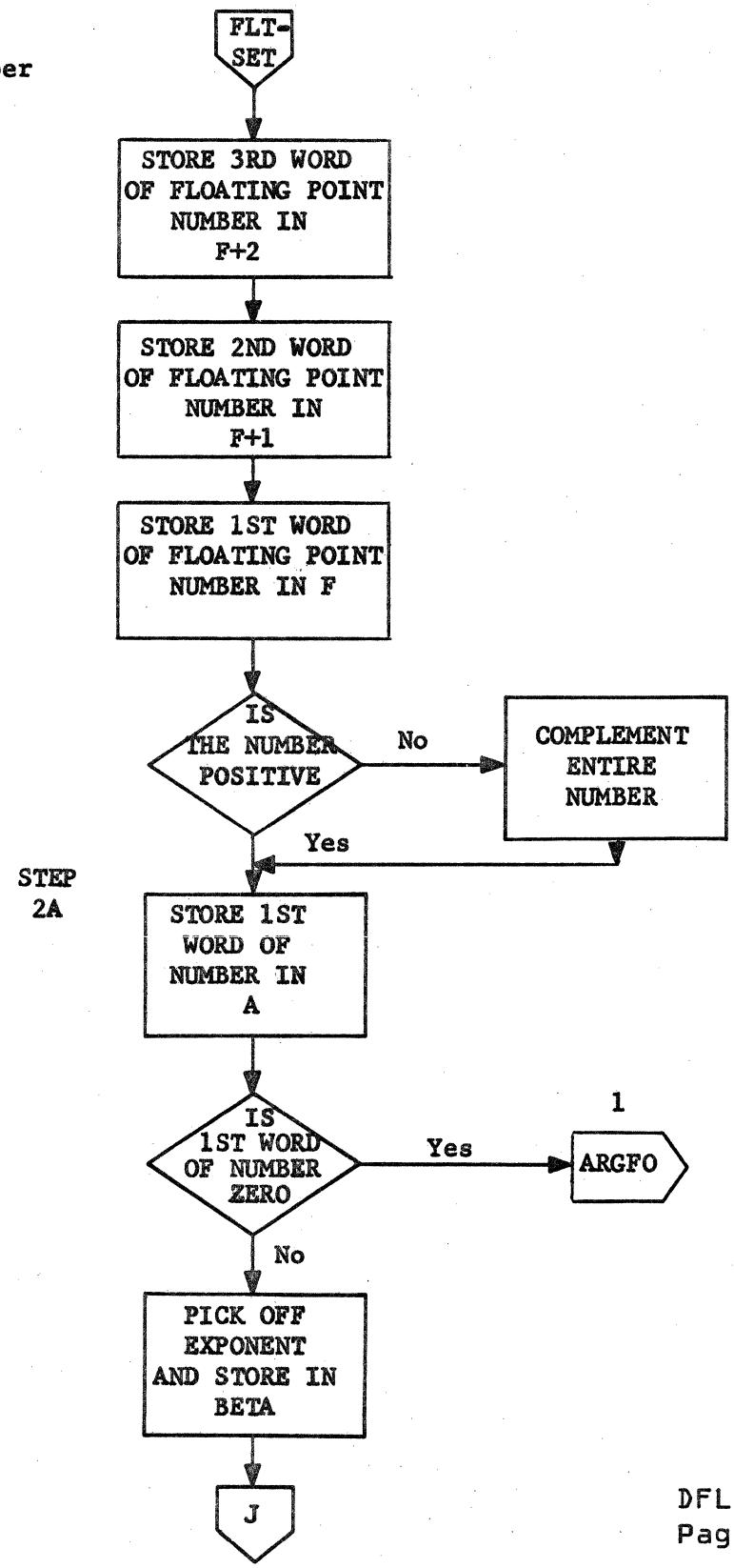
1. Gets the floating point number and stores in F, F+1 and F+2, then unpacks the number into A=0,15MSB
   AI=0,15ISB
   B=0,8LSB, 7 zeros
   BETA=EXPONENT

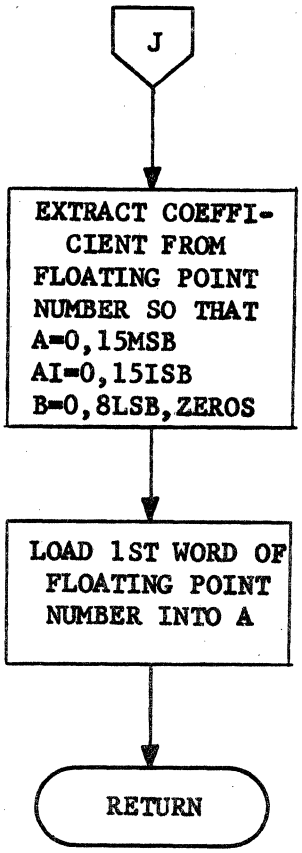


2. **Round, Truncate and Normalize the results.**

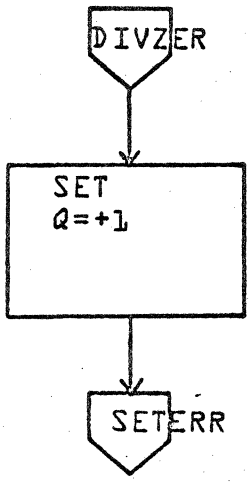


1. At ARGFO, the entire number is set equal to +0
   A=+0
   B=+0
   BETA=+1




J
EXTRACT COEFFI-
CIENT FROM
FLOATING POINT
NUMBER SO THAT
A=0,15MSB
AI=0,15ISB
B=0,8LSB,ZEROS
LOAD 1ST WORD OF
FLOATING POINT
NUMBER INTO A
RETURN

DIVIDE CHECK WHEN DIVIDING BY ZERO



LA JOLLA RESOURCE CENTER
IMS Page 12-48
1700 MASS STORAGE FORTRAN
C005*3.1 A/B

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER

DOCUMENT CLASS: IMS
MACH. TYPE: 1700
DOCUMENT TITLE: DFLOT
PAGE 26 OF 37
NUMBER
ISSUE DATE
DRAWN BY
DATE

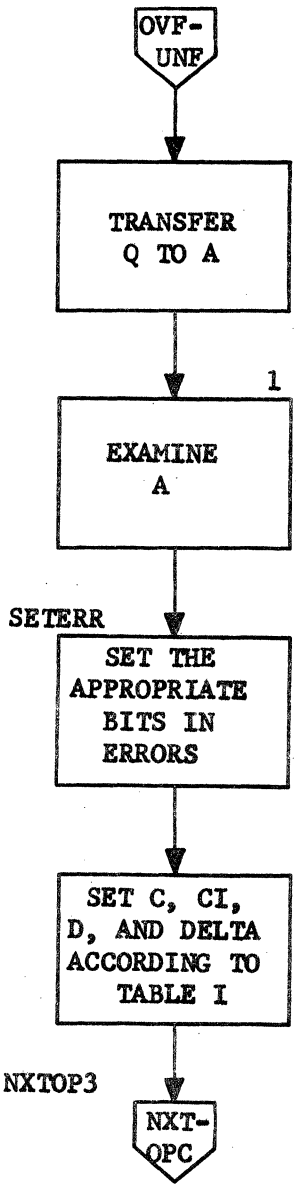
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME

REV | APPROVED | DATE

OVERFLOW OR UNDERFLOW

1. If A is positive, an overflow occurred.

   If A is not positive, an underflow occurred.



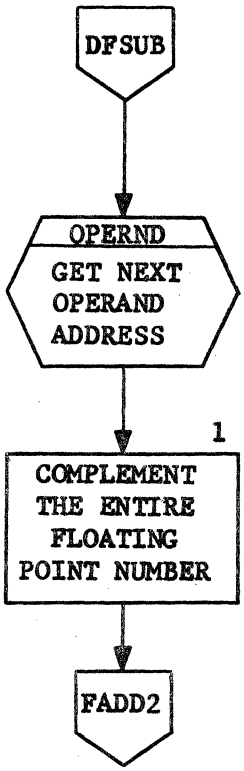
	C	CI	D	DELTA
EXPONENT OVERFLOW	$7FFF	$7FFF	$7FC0	$7F
DIVIDE FAULT	$7FFF	$7FFF	$7FC0	$7F
EXPONENT UNDERFLOW	$4000	$0000	$0000	-$7F

TABLE I

OPERATION CODE = 8

DOUBLE FLOATING SUBTRACT (DFSUB)

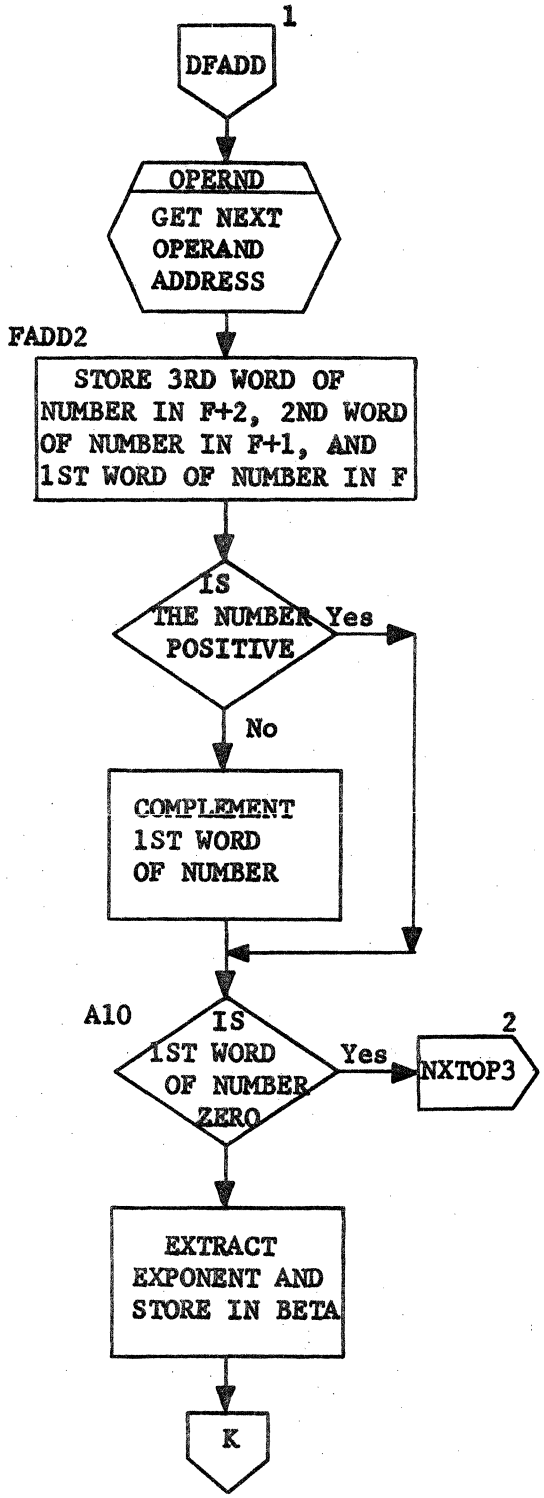
1. Change the sign before entering the Add routine.

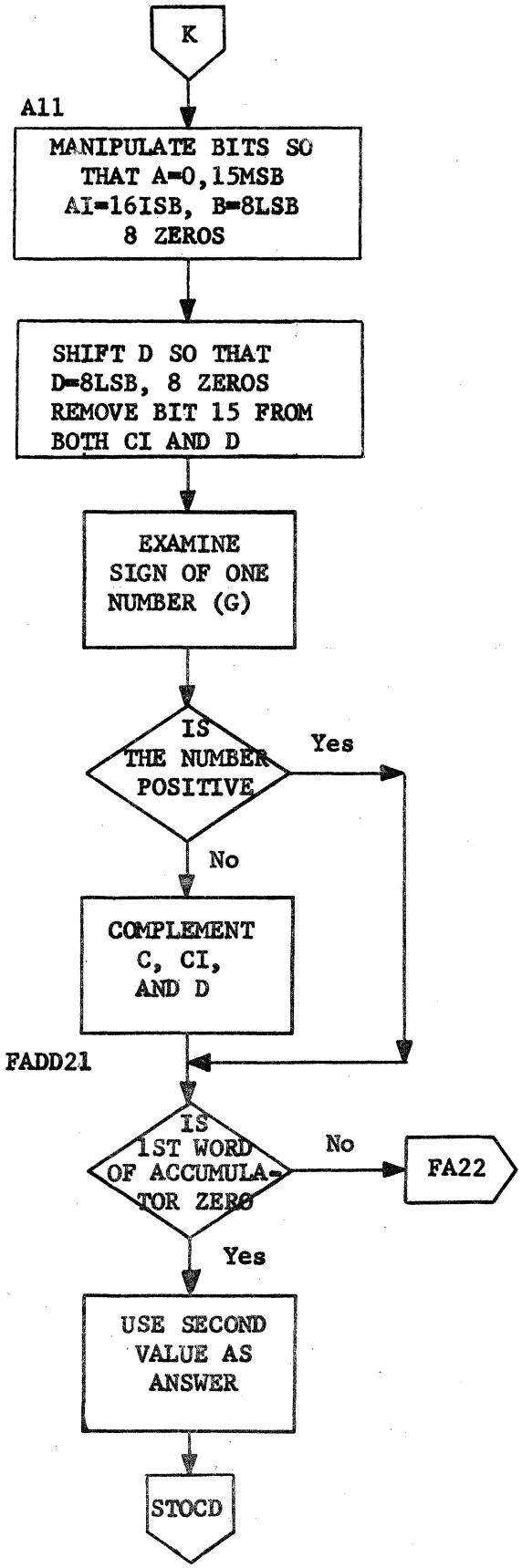


OPERATION CODE = E

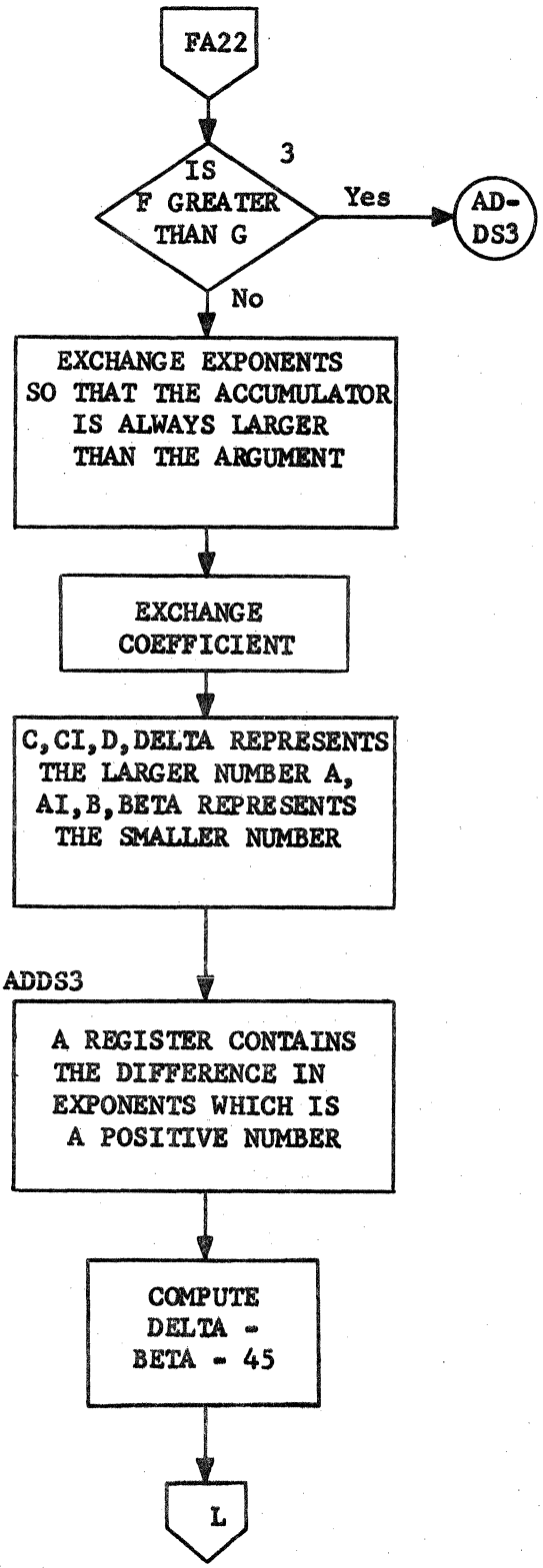
DOUBLE FLOATING ADD (DFADD)

1. This section adds the floating point numbers in G and F and places the result in G.
2. Jump out - back to NXTOPC, get next OP code.



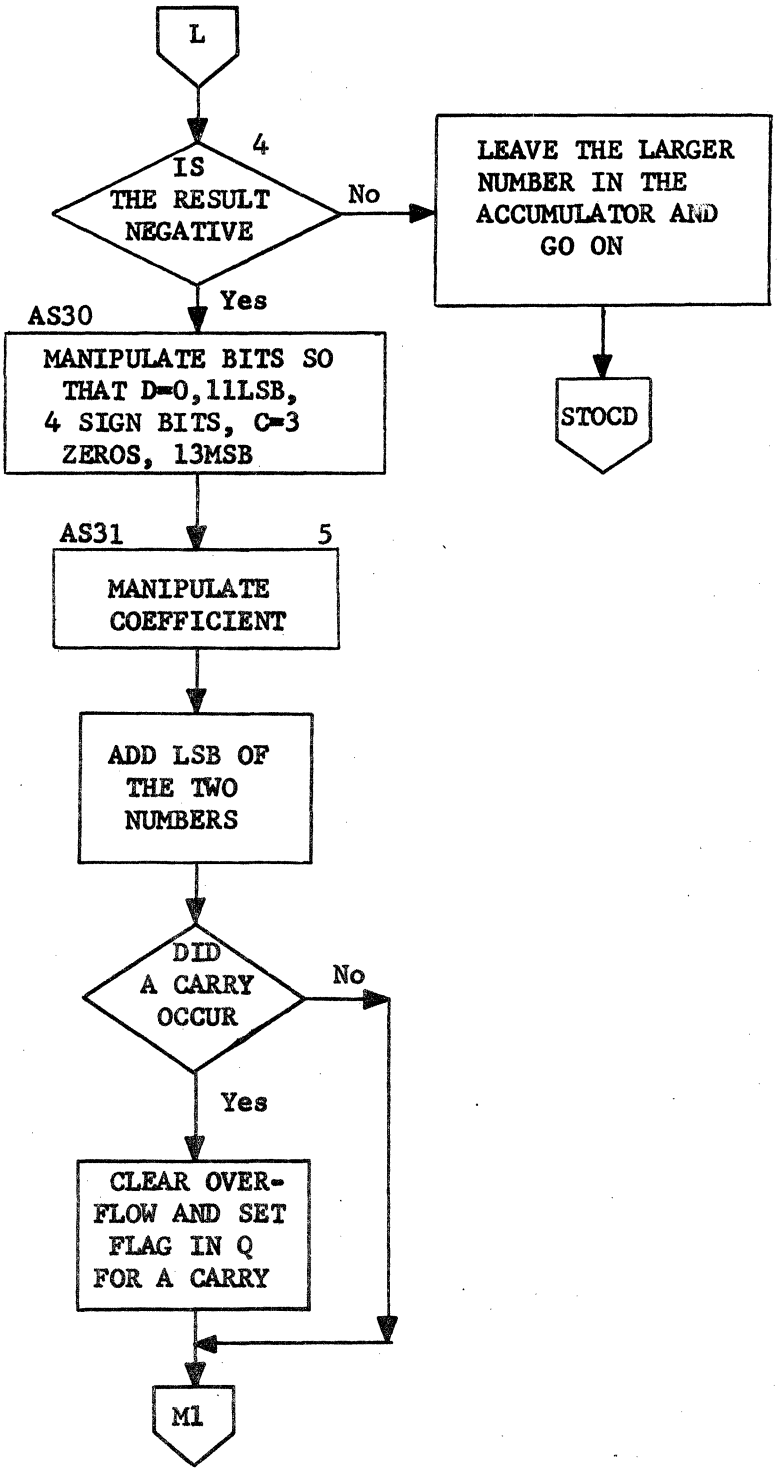

K
A11
MANIPULATE BITS SO THAT A=0,15MSB AI=16ISB, B=8LSB 8 ZEROS
SHIFT D SO THAT D=8LSB, 8 ZEROS REMOVE BIT 15 FROM BOTH CI AND D
EXAMINE SIGN OF ONE NUMBER (G)
IS THE NUMBER POSITIVE
Yes
No
COMPLEMENT C, CI, AND D
FADD21
IS 1ST WORD OF ACCUMULATOR ZERO
No
FA22
Yes
USE SECOND VALUE AS ANSWER
STOCD

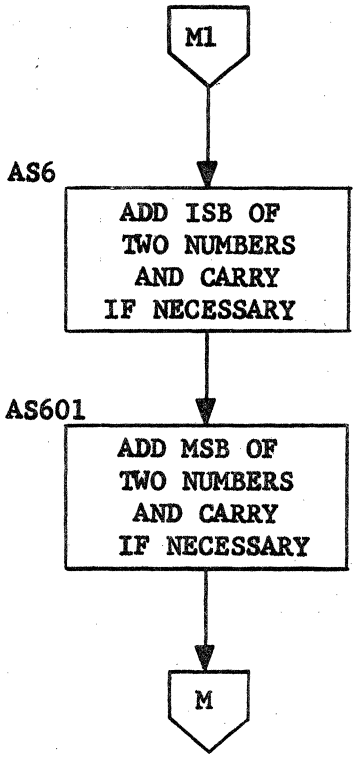
3. This check is made by examining the difference of the exponents of the two numbers.



4. RESULT is result of DELTA-BETA-45; is there "TOO" great a difference between the numbers.

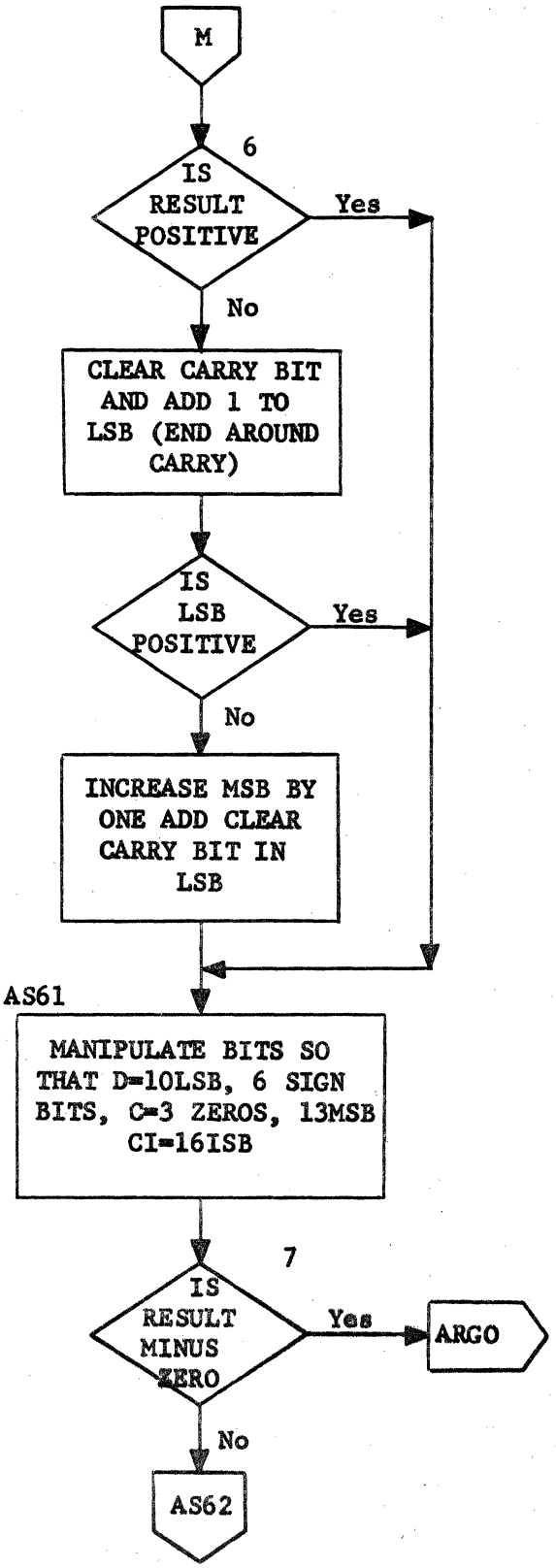
5. Shift smaller number right (DELTA-BETA) and set the sign of MSB to positive. Shift LSB right 1 and set the sign of LSB positive. Clear the carry to bit 15 and add 1 to MSB positive the smaller number.



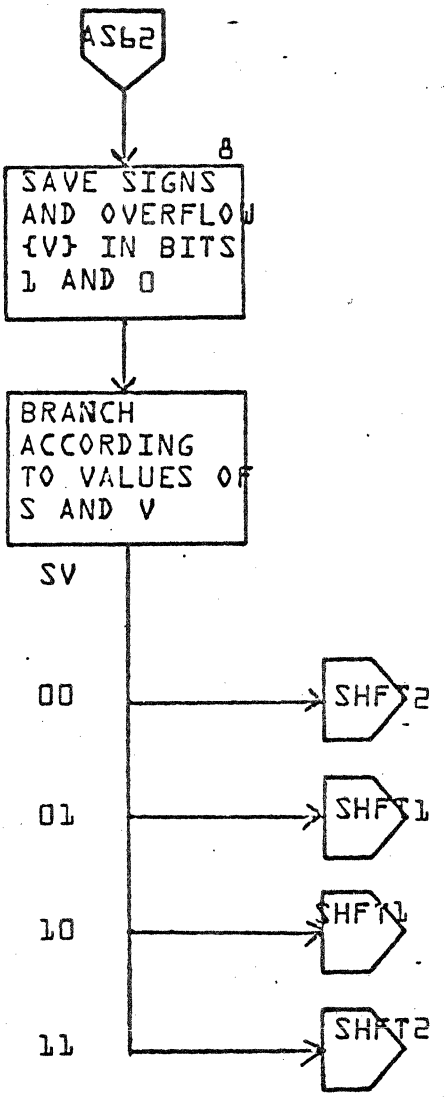

M1
AS6
ADD ISB OF TWO NUMBERS AND CARRY IF NECESSARY
AS601
ADD MSB OF TWO NUMBERS AND CARRY IF NECESSARY
M

6. Result is result of addition of MSB and carry bit.

7. Minus zero is
   Q=7FFF
   A=FFFE



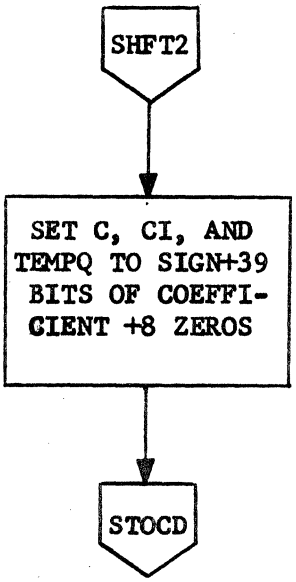
8. IF S.EQ.V, THEN V IS SIGN EXTENSION.
IF S.NE.V, THEN RESULT OVERFLOWED INTO V.



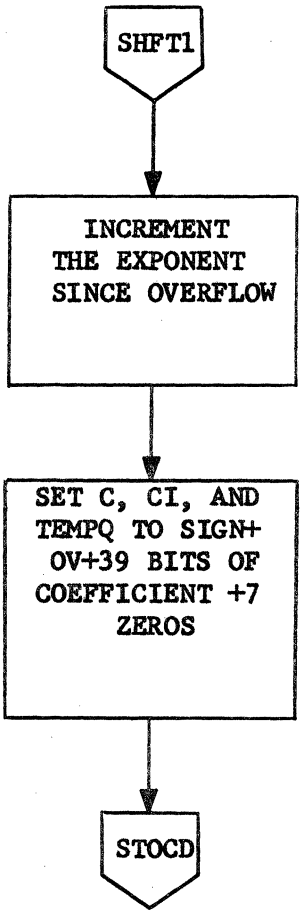
LA JOLLA RESOURCE ENTER
IMS Page 12-58
1700 MASS STORAGE FORTRAN
C005*3.1 A/B

CONTROL DATA CORPORATION
SOFTWARE DOCUMENT
SAMPLE CODE
FLOWCHART
DECISION TABLE
OTHER
DOCUMENT CLASS IMS
MACH. TYPE 1700
DOCUMENT TITLE DFLOT
PAGE 36 OF 39
NUMBER
ISSUE DATE
DRAWN BY
DATE
PROJECT NO.
PROJECT MGR.
PROJECT NAME
TASK NO.
TASK NAME
REV
APPROVED
DATE
PRINTED

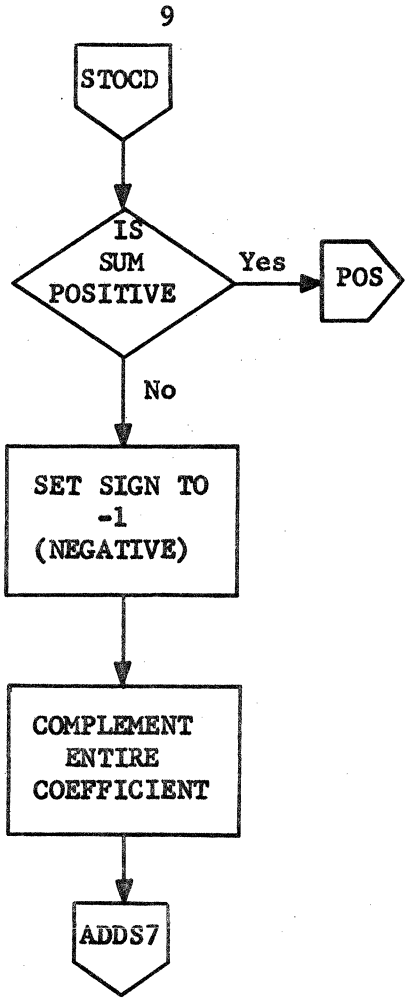
NO OVERFLOW



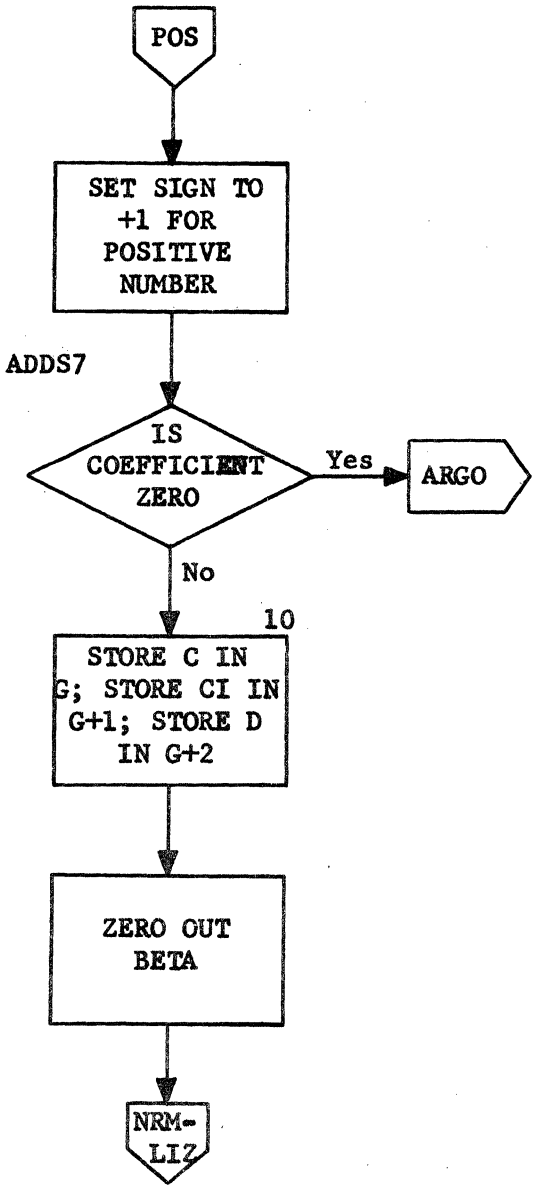
OVERFLOW OCCURRED



9. Save the sign and magnitude of the results.
On entry
C=SIGN+15MSB
CI=16ISB
D=8LSB+8ZEROS



10. C=0+15MSB
CI=16ISB
D=8LSB+8ZEROS



CHAPTER 13

T A B L E O F C O N T E N T S

13.0 OBJECT TIME DOUBLE PRECISION INTRINSIC FUNCTIONS

13.1 General Description

The following Object Time Intrinsic Functions are included in this chapter:

Intrinsic Functions	I/O Routine
DABS(X)	Q8DAB
DSIGN(X,Y)	DSIGN
DFIX(X)	FXFL (See Chapter 10 for Description)
DFLT(I)	Q8DFLT
SNGL	SNGL
DBLE	Q8DBLE

13.2 Q8DAB Routine

This routine, which is written in 1700 Assembly Language, computes the absolute value of a double precision floating point number and leaves the result in the pseudo accumulator.

The calling sequence is:

```
RTJ  DABS
1.  address of argument
```

The entry points are:

```
Q8DAB
DABS   (which is equated to Q8DAB)
```

The external declared is:

```
DFLOT
```

The low core locations used by this routine are:

```
DFLACC   ($C5,$C6,$C7) Double Precision pseudo
         accumulator
MASKSB   ($11) Mask $7FFF
```

13.3 DSIGN Routine

This routine, which is written in 1700 Assembly Language, computes the sign of the second argument times the absolute value of the first argument and leaves the result in the pseudo accumulator.

The calling sequence is:

```
RTJ  DSIGN

1.  address of first argument
2.  address of second argument
```

The entry points are:

```
@8DSG
DSIGN   {which is equated to @8DSG}
```

The declared external is:

```
DFLOT
```

The low core locations used by this routine are:

```
A1   {$D8}
A2   {$D9}
FF   {$E1}  used to save I register
@S   {$E2}  used to save @ register
```

13.4 @8DFLT Routine

Entry DFLT

This routine, which is written in 1700 Assembly Language, performs the conversion of an integer number into a double precision floating point number.

The calling sequence is:

```
RTJ  DFLT

1.  address of argument
```

The external declared is:

```
FLOAT
```

The low core locations used by this routine are:

MASK {$11} MASK bit pattern $7FFF
@S {$E2} used to save @ register

Entry @8DFLT

This routine, which is written in Assembly Language, performs the conversion {across the equal sign} of an integer into a double precision Floating Point Number {DP=I}.

The calling sequence is:

```
LDA  I   {the integer value to be converted is in
          the A register upon entry}
RTJ  @8DFLT
```

The external declared is:

@8@FLT

The low core location used by this routine is:

@S {$E2} used to save @ register

13.5 SNGL Routine

This routine, which is written in 1700 Assembly Language, obtains the most significant 32 bits of a double precision argument.

The calling sequence is:

```
RTJ  SNGL

1.  address of argument
```

The entry points are:

SNGL
@8SNGL {which is equated to SNGL}

The external declared is:

FLOT

The low core location used by this routine is:

MASK {$11} MASK word $7FFF

13.6 Q8DBLE Routine

This routine, which is written in 1700 Assembly Language, expresses a single precision number (the 2-word argument) in double precision form (3-word result).

The calling sequence is:

```
RTJ  DBLE
1.   address of argument
```

The entry points are:

```
Q8DBLE
DBLE   (which is equated to Q8DBLE)
```

The low core locations used by this routine are:

DFLACC	{$C5,$C6,$C7}	Double Precision Floating Point Pseudo Accumulator
MASK	{$11}	Mask utilizing the following bit pattern $7FFF
ZERO	{$22}	Low Core cell containing Zero

13.7 DRSTOR Subroutine - Double Precision Compiler Support Object Time Routine

This subroutine is called by the FORTRAN compiler to perform double precision and single precision pseudo-accumulator stores which are necessary to provide efficient generated code. This subroutine is coded in 1700 Assembly Language. It contains the following entry points: DSTOR1, RSTOR1, and DSTOR2.

The low core locations used by this routine are:

CZERO	{$22}	Low Core cell containing Zero
DFLACC	{$C5,$C6,$C7}	Double precision floating point pseudo accumulator
MASK	{$11}	MASK bit Pattern $7FFF
Q$	{$E2}	Used to save Q register

Entry Point DSTOR1

This routine obtains values stored in $C5, $C6, and $C7 and stores them in the parameter argument.

The calling sequence is:

```
RTJ  DSTOR1
1.  address of double precision argument
```

The input to the routine is in cells $C5, $C6 and $C7. The output from the routine is the contents of cells $C5 and $C6 along with the value zero stored in the parameter argument.

Entry Point RSTOR1

This routine obtains values stored in $C5, $C6, and $C7 and stores only $C5 and $C6 in the parameter argument which is single precision argument.

The calling sequence is:

```
RTJ  RSTOR1
1.  address of single precision argument
```

The input to the routine is stored in cells $C5, $C6 and $C7.

The output from the routine is the contents of cells $C5 and $C6 into the parameter argument.

Entry Point DSTOR2

This routine obtains values stored in the pseudo floating accumulator {cells $C5 and $C6} and stores them in the parameter argument. The routine also stores a zero into cell $C7 as well as into the third word of the parameter argument.

The calling sequence is:

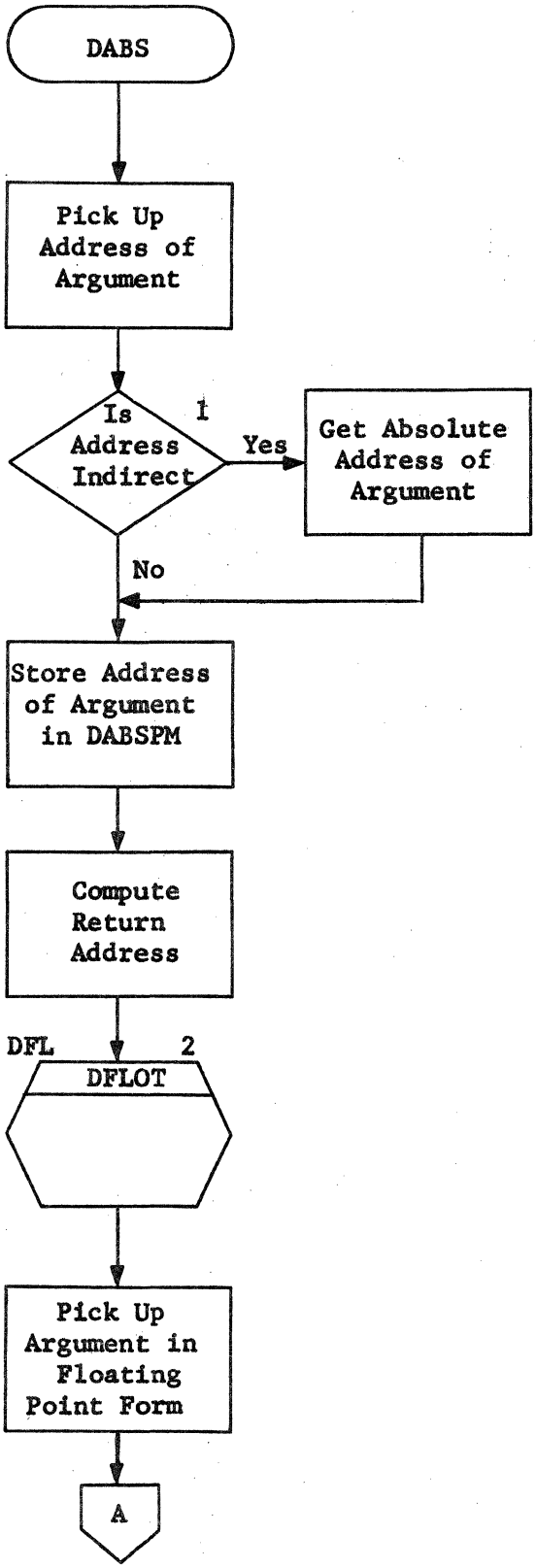
```
RTJ  DSTOR2
1.  address of double precision argument
```

The input to the routine is stored in cells $C5, $C6 and $C7.

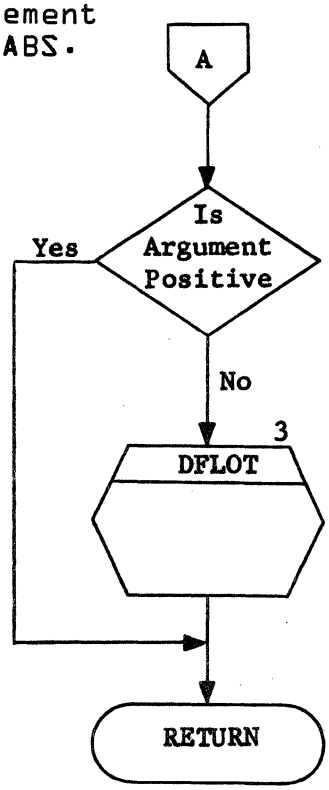
The output from the routine is the contents of cells $C5 and $C6 and the value zero all stored in the parameter argument. Cell $C7 is also set to zero.

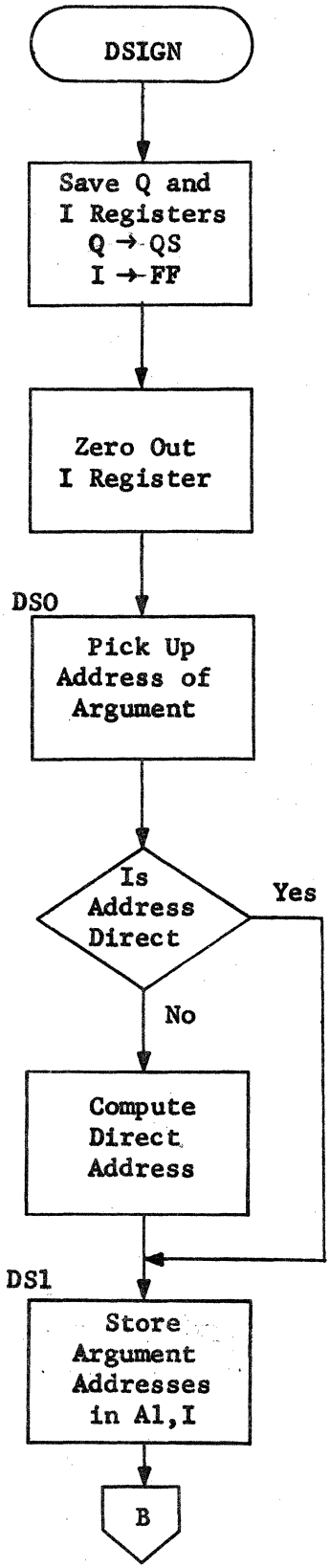
1. Is address negative?

2. Do a double precision floating point load of argument into pseudo accumulator.



3. Do a double precision floating point complement of the argument of DABS.

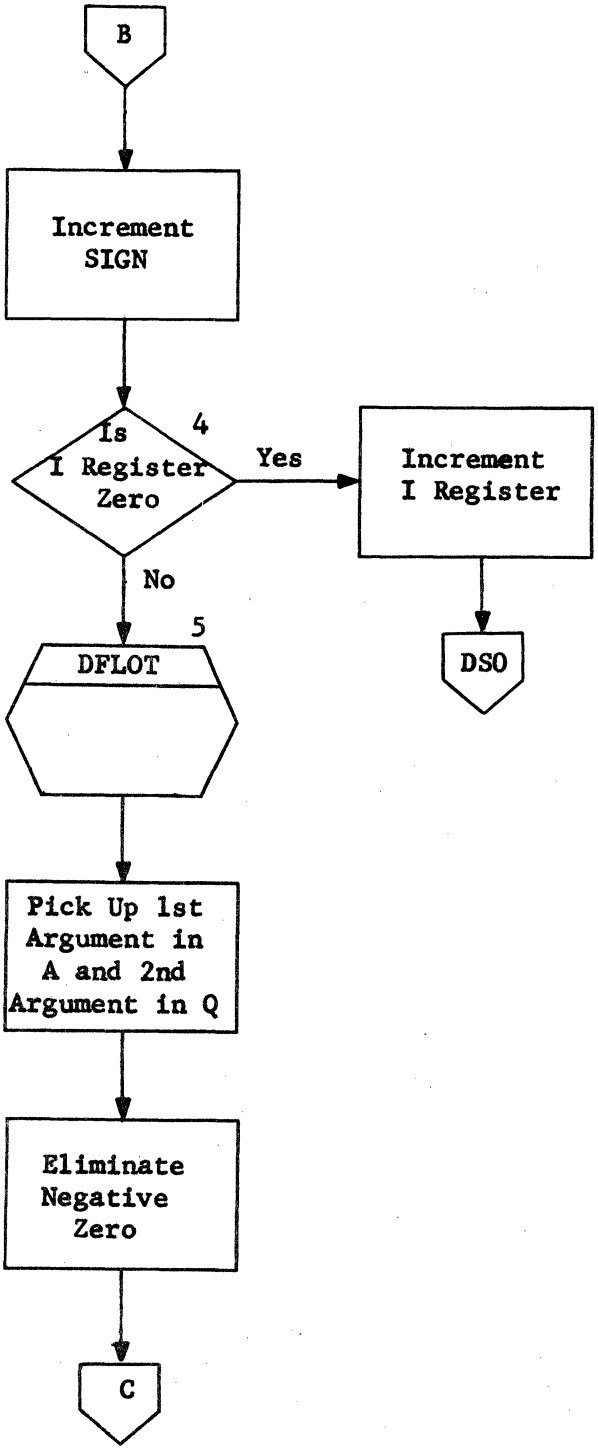


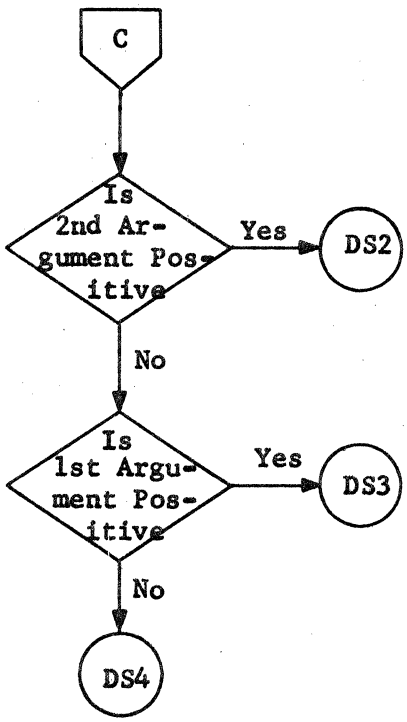


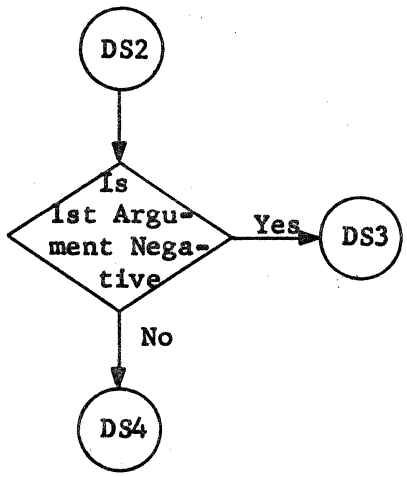
DSIGN
Page 1 of 3

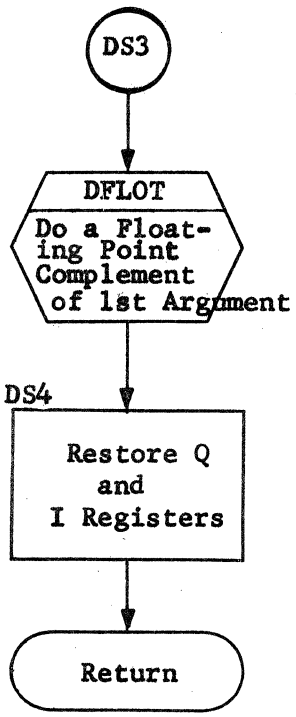
4. Have we just processed the first argument's address.

5. Do a Double Precision floating point load of 1st argument.

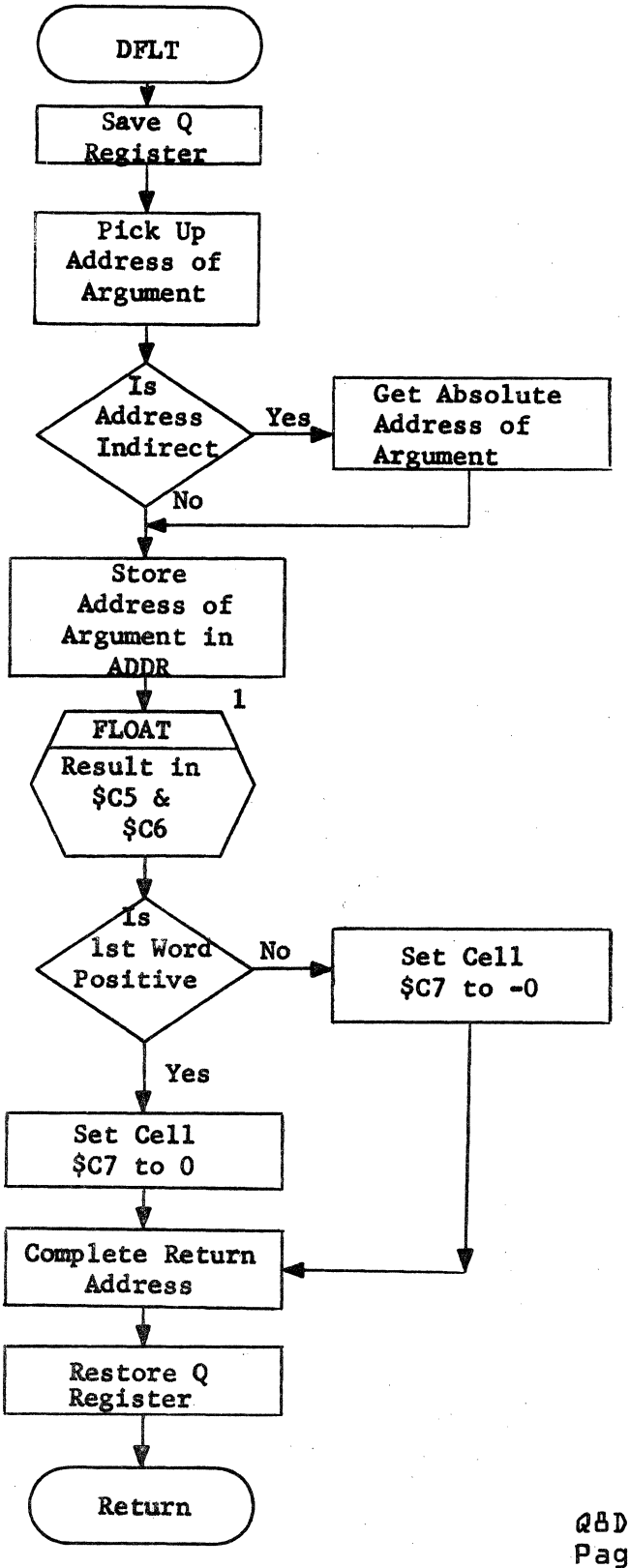



C
Is 2nd Argument Positive
Yes
DS2
No
Is 1st Argument Positive
Yes
DS3
No
DS4


DS2
Is 1st Argument Negative
Yes
DS3
No
DS4

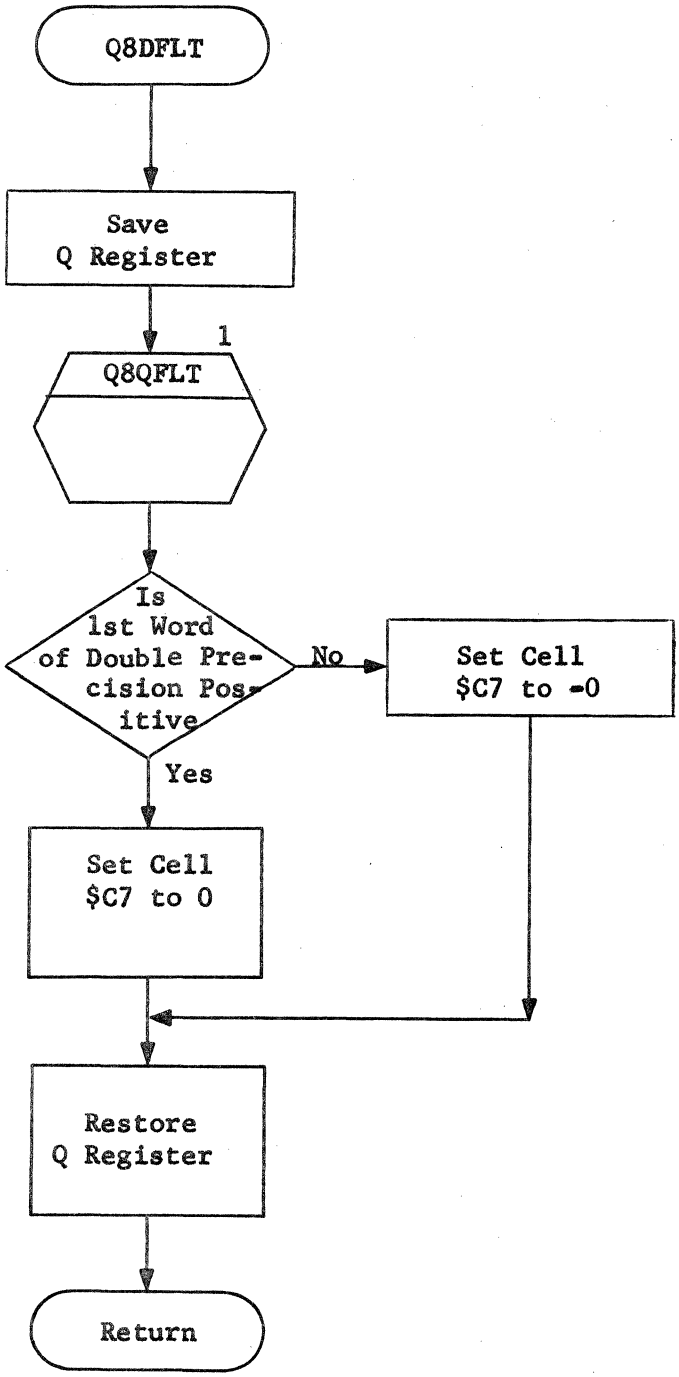

DS3
DFLOT
Do a Floating Point Complement of 1st Argument
DS4
Restore Q and I Registers
Return

1. FLOAT the integer parameter argument.

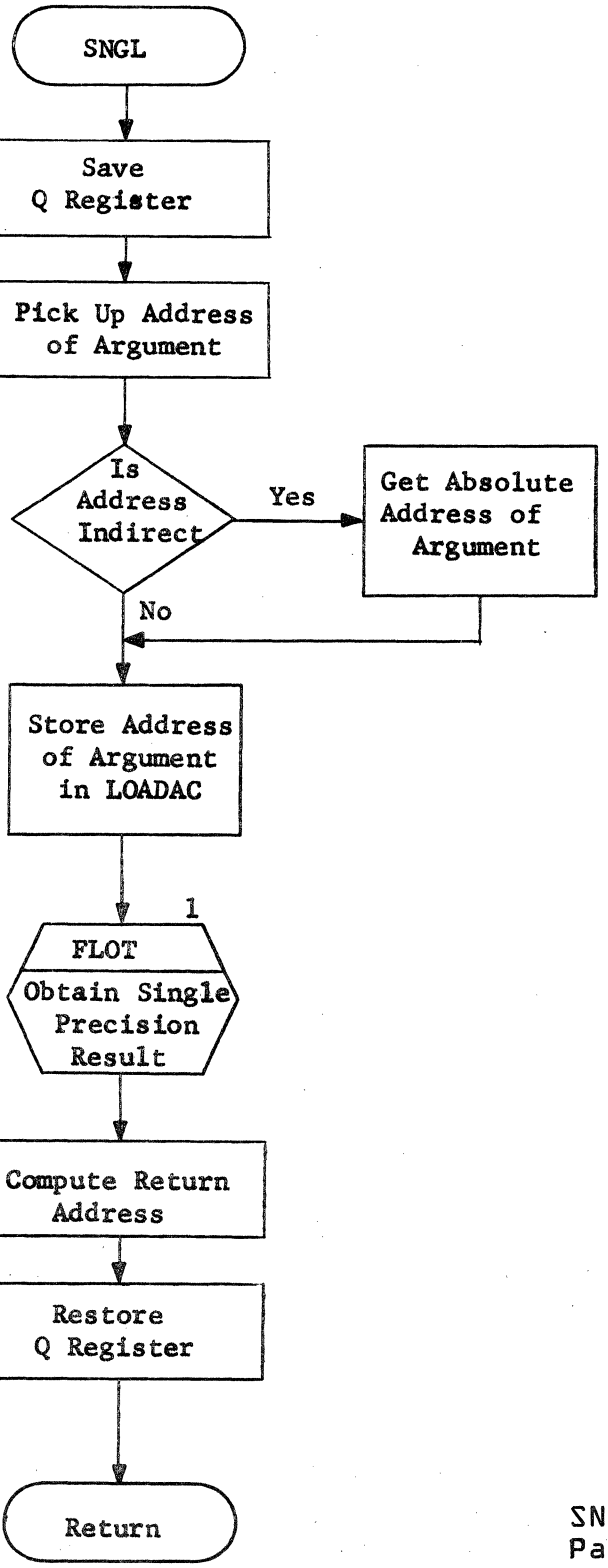


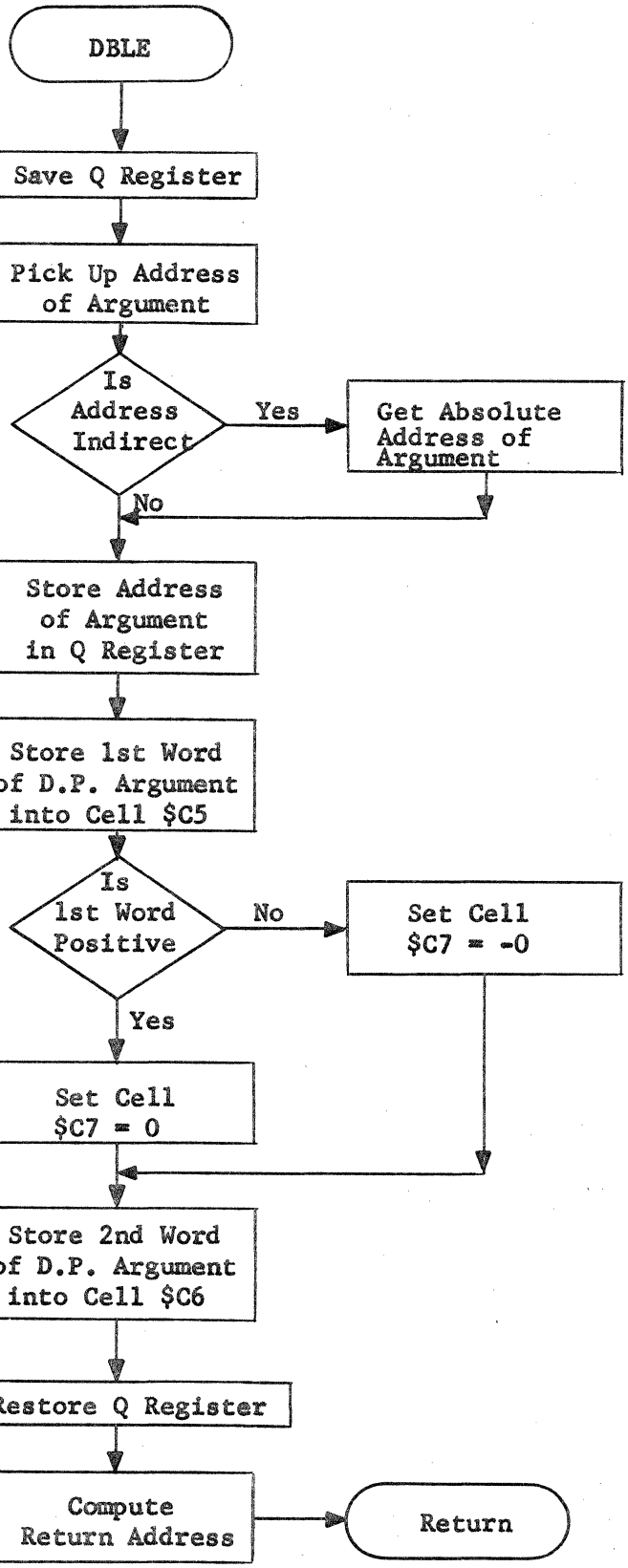
Q8DFLT
Page 1 of 2

1. Perform conversion of an integer to a floating point number. Input is in A register. Output is a floating point number contained in cells C5 & C6.

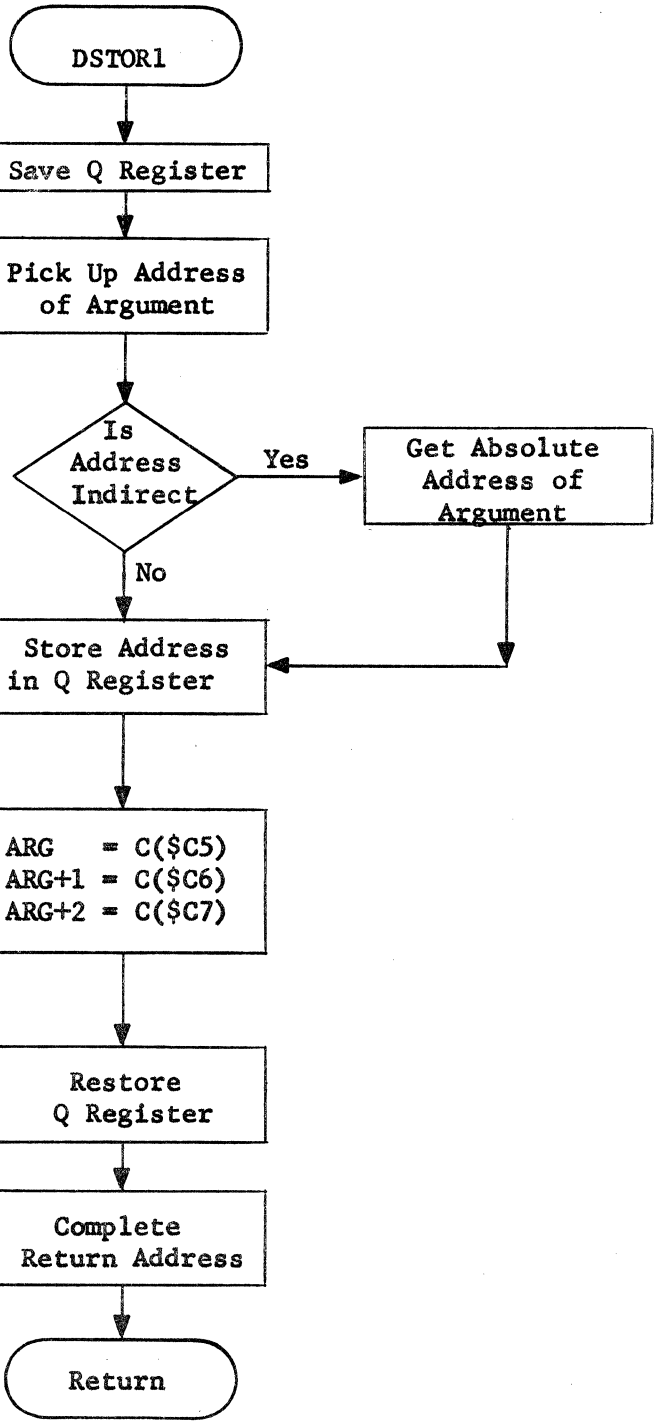


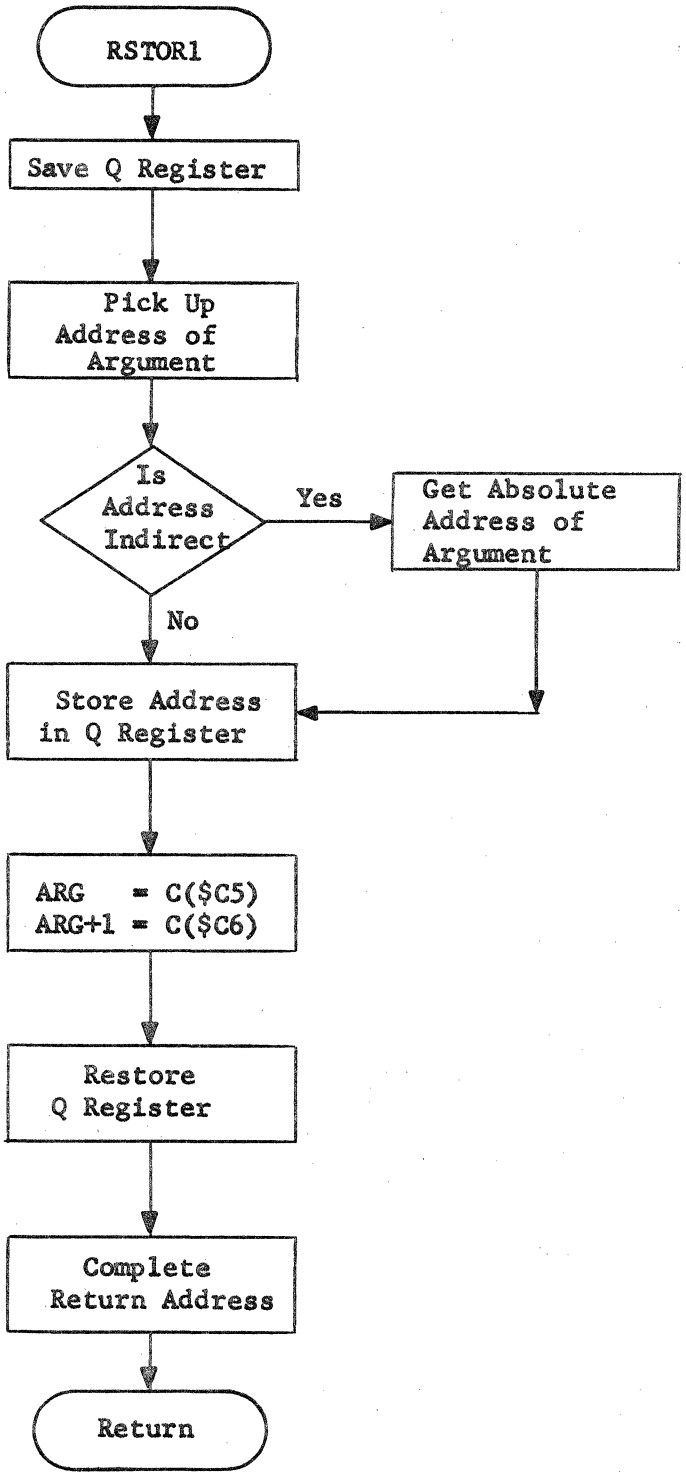
1. Do a floating point load of argument into pseudo accumulator (Cells $C5 & $C6).

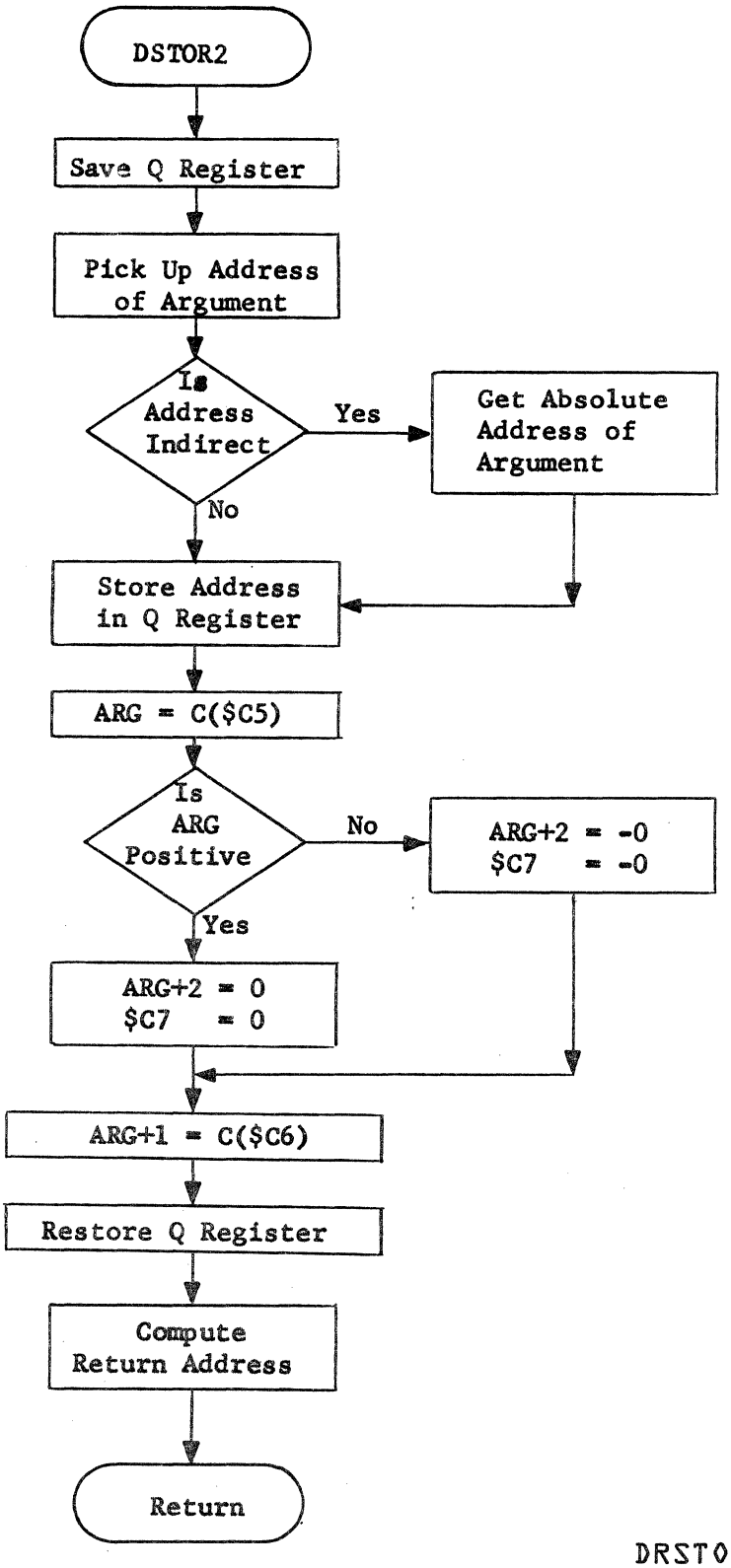




Q8DBLE
Page 1 of 1


DSTOR1
Save Q Register
Pick Up Address of Argument
Is Address Indirect
Yes
Get Absolute Address of Argument
No
Store Address in Q Register
ARG = C($C5)
ARG+1 = C($C6)
ARG+2 = C($C7)
Restore Q Register
Complete Return Address
Return


RSTOR1
Save Q Register
Pick Up Address of Argument
Is Address Indirect
Yes
Get Absolute Address of Argument
No
Store Address in Q Register
ARG = C($C5)
ARG+1 = C($C6)
Restore Q Register
Complete Return Address
Return


DSTOR2
Save Q Register
Pick Up Address of Argument
Is Address Indirect
Yes
Get Absolute Address of Argument
No
Store Address in Q Register
ARG = C($C5)
Is ARG Positive
No
ARG+2 = -0
$C7 = -0
Yes
ARG+2 = 0
$C7 = 0
ARG+1 = C($C6)
Restore Q Register
Compute Return Address
Return