DOT-TSC-OST-71-2

PROGRAMMER'S REFERENCE MANUAL
for
DYNAMIC DISPLAY
SOFTWARE SYSTEM

January 1971

PROGRAMMER'S MANUAL

Prepared for

OFFICE OF THE SECRETARY

TRANSPORTATION SYSTEMS CENTER
55 Broadway
Cambridge, Mass. 02142

DOT-TSC-OST-71-2

PROGRAMMER'S REFERENCE MANUAL
for
DYNAMIC DISPLAY
SOFTWARE SYSTEM

January 1971

Transportation Systems Center
Department of Transportation
Cambridge, Mass. 02142

Table of Contents

I Introduction

In 1968, the display systems group of the Systems Laboratory of the NASA/Electronics Research Center undertook a research task in the area of computer controlled flight information systems for aerospace application. The display laboratory of the Transportation Systems Division of the Transportation Systems Center for the Department of Transportation is the direct descendant of the above display laboratory. The vehicle for conducting this research consists of a Honeywell DDP-516 computer interfaced with a Sanders Associates ADDS/900 graphical display generator that controls various CRT devices. Input devices for indicating user response, including light pen, tablet, track ball and others can be interfaced with the computer and monitored by a simulation program.

With the Dynamic Display Software System, the user can interactively create the geometric characteristics of the desired display and specify dynamic linkages with a program to simulate the display environment. After the desired set of indicators is specified, the system serves as a simple real-time simulator to evaluate the usefulness of the displays. In addition, the system may run on-line with PDP-10, which allows display of conditions in a more complicated simulation environment. The user has the capability of making changes to previously created indicators to provide an evolutionary means of developing environment indicator systems. The system is described in this manual.

The section "SYSTEM CONCEPTS" provides a brief general description of the three phases of system operation and interaction of the phases.

The section "DATA STRUCTURE" shows how the display data is arranged in core and describes auxiliary blocks used for dynamic updating of the displayed data.

The final section "PROGRAM DESCRIPTIONS AND FLOWCHARTS" gives a detailed description of all of the programs and subroutines used in the system.

For a detailed discussion on system use see the document, "A GUIDE TO THE USE OF THE DYNAMIC DISPLAY SOFTWARE SYSTEM".

II. SYSTEM CONCEPTS

The system is divided into two large functional parts and one smaller part to allow efficient use of core. These parts are the create/edit phase, where display lists are created and matched via dynamic expressions to the outputs of a simulation; the simulation phase, where display and update of the data structure and operation of the simulation are under control of the real time monitor; and the input/output phase, which supervises transfer of display files between the create/edit and simulation phases, and changes all user entered specifications of dynamics into executable object code via a small compiler subprogram.

1. Create/Edit Phase

The first phase of system operation is shown functionally in Figure II. 1. This phase consists of: (a) a user language, which provides the interface between the user and the system, and preliminary conversion of alphanumeric input into display elements; (b) data structuring routines, which handle storage allocation and the creation and maintenance of the data structure; and (c) connector programs, which handle overlay of the other system phases upon user request.

Graphic components specified via the user language are displayed immediately. Also specified utilizing the user language are dynamic expressions that will modify the graphic components using the update program in the simulation phase. The language, however, does not generate object code for dynamic expressions but stores the expressions as packed characters for later processing by the mini-compiler in the input/output phase.

2. Input/Output - Mini-compiler

This phase of operation services input and output of the data structure and free storage ring onto paper tape or disc. Prior to saving the data structure, a mini-compiler using pointers internal to the data structure, compiles each dynamic expression of FORTRAN-like source code entered during the immediately prceeding create/edit phase. Once saved, the data structure may be loaded back into create/edit phase for additions or deletions or into the simulation phase. The input/output phase of operation is functionally shown in Figure II. 2.

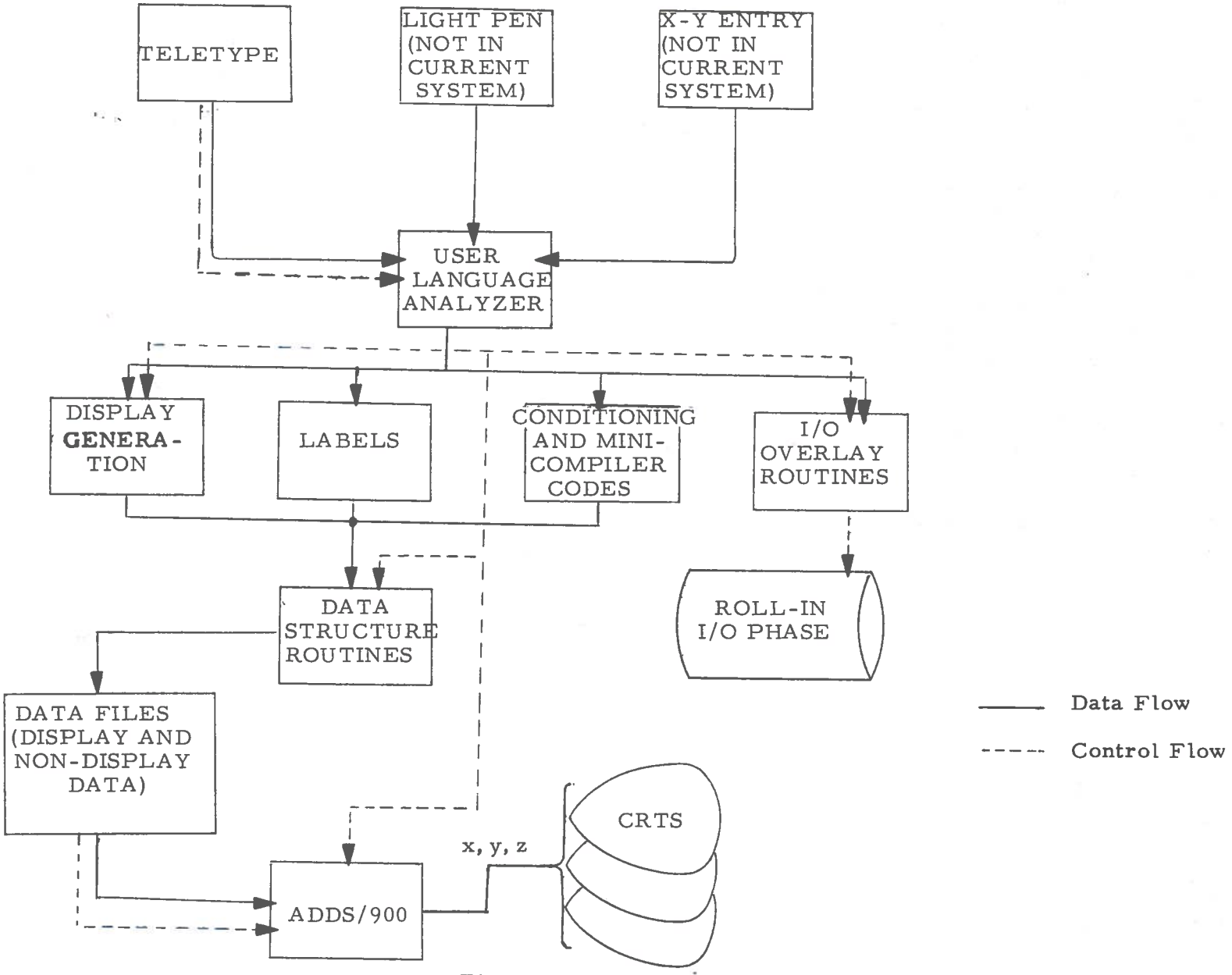


Figure II. 1
Functional Design of Create/Edit Phase

3. Simulation Phase

The Simulation Phase provides an environment whereby the displayed data may operate dynamically. Provision is made, under user option, for input from or output to any external device. Inputs may be used directly as dynamic parameters, or they may be used by a model and/or background program which may alter predefined registers. The update program uses this data and mini-compiler code to make changes to the display list. It is at this point in the display system that data provided by a simulation in the PDP-10 would be used to alter the display. See Figure II. 3.

The Monitor supervises the scheduling of the various routines and allows the user to modify the update cycle time if the system overruns the time constraint.

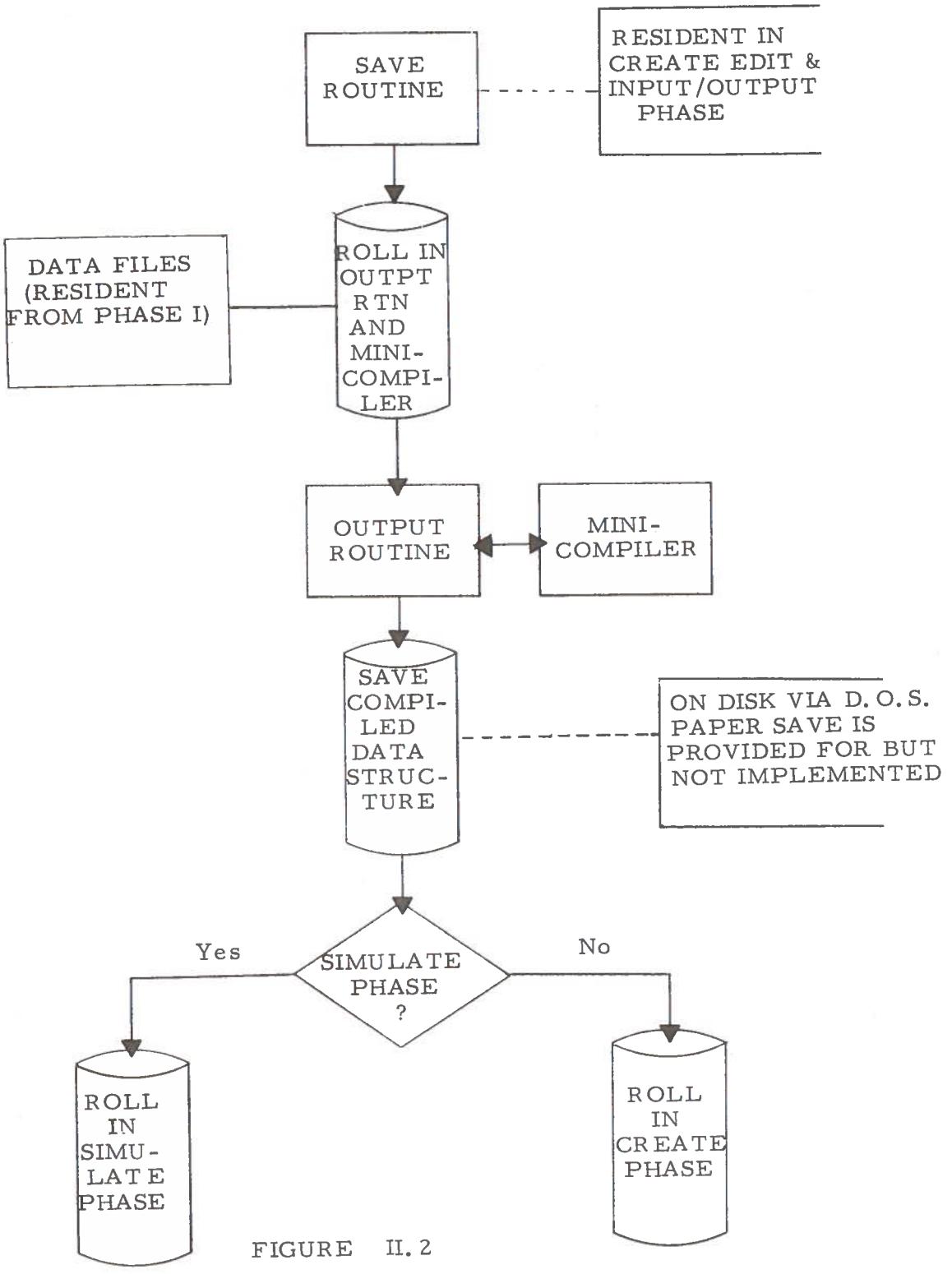


FIGURE II.2

FUNCTIONAL DESIGN OF INPUT-OUTPUT PHASE

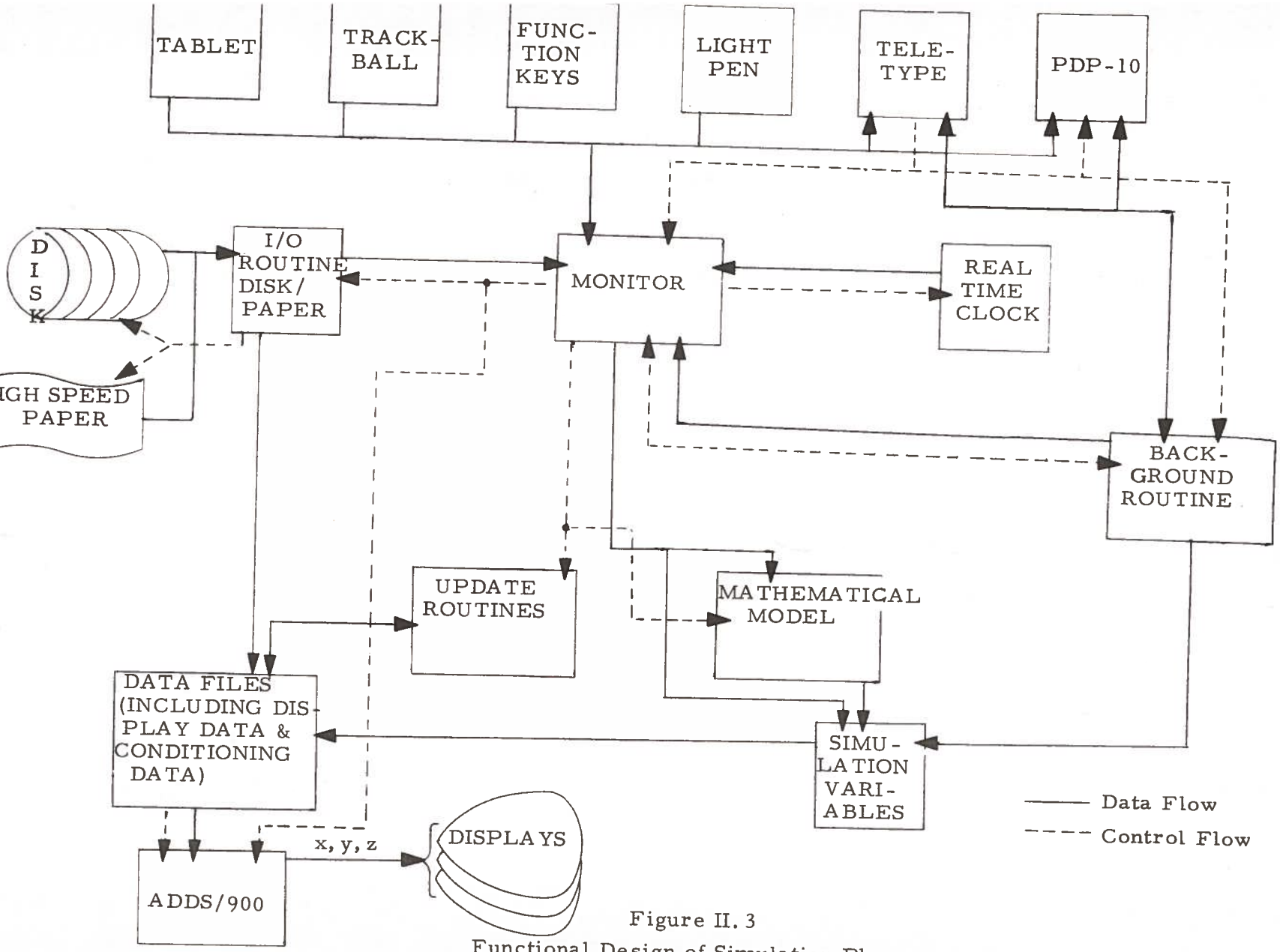


Figure II. 3
Functional Design of Simulation Phase

III DATA STRUCTURE

1. The display data format is a hierarchical ring structure with auxiliary blocks which describe the parameters to be used for dynamic updating of the data.

The highest level in the hierarchy is the frame ring which is a ring of frame blocks. Attached to a frame block is an indicator ring. There may also be an auxiliary block for register definition attached to a frame block.

An indicator ring consists of indicator blocks which belong to one particular frame. Each indicator block has an entity ring associated with it. The entity ring consists of entity blocks which all belong to the same indicator. An indicator block may have an auxiliary conditioning block attached to it.

Entity blocks have component blocks and conditioning blocks attached.

The entity blocks and the component blocks basically contain display data; that is data which is sent to the display unit. This includes vectors, characters, and coordinate converter information. These blocks are tied together using display jumps to allow automatic display of a frame.

The conditioning blocks contain information which is used by the user language processor to direct dynamic changes of the display according to the state of the simulation model. These blocks are tied to the ringed display data by pointers. See Figure III. 1.

2. Descriptions of Data Formats

A. Frame Block

A frame block is accessed by reference to a permanently assigned word (FRHD) containing the start address of the first frame on the frame ring.

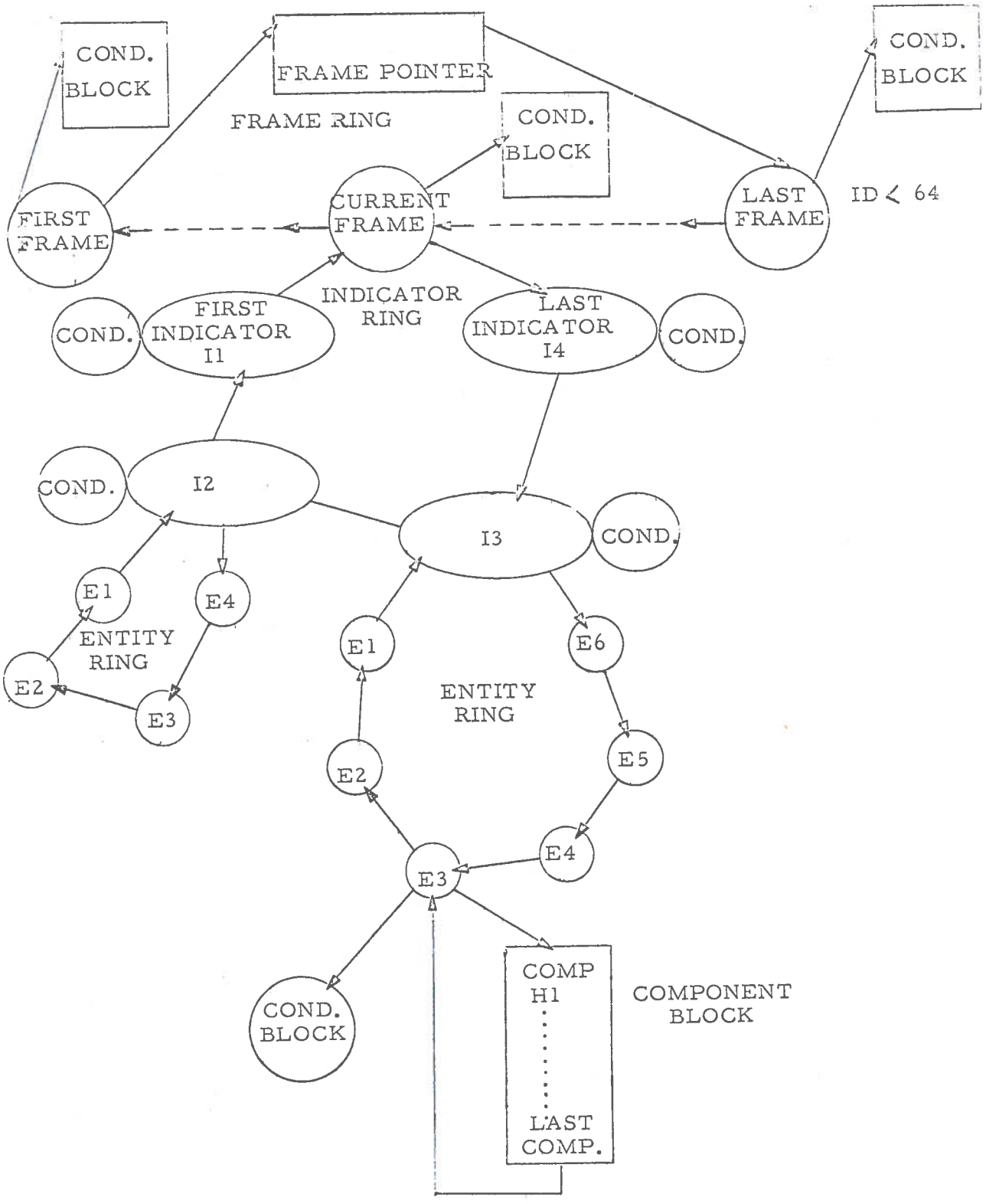


Figure III. 1
Ring Organization of the Data Structure

Format: See Figure III. 2.

Word 1: word 1 contains the storage block type (1) and the ID of the frame.

Word 2: word 2 contains a pointer to the next frame in the ring. If it is the last frame in the ring, then the pointer is to the frame header (FRHD).

Word 3: the third word contains a pointer to a register definition conditioning block if any auxiliary registers have been defined. If not, it contains zeroes.

Word 4 and 5: these two words comprise a 'display jump' to word 3 of the last created indicator on this frame. If there are no indicators, then the jump is to word 6 ("WAIT"). As indicators are added, the contents of word 5 replace the contents of NEXT INDICATOR (word 10) of the new indicator block, and the address of the new indicator replaces the contents of word 5.

Word 6: contains a 'wait' to allow refresh synchronization.

Word 7 and 8: Jump back to word 4 for refresh.

B. Indicator Block

Indicators are accessed through word 5 (jump address) of a frame block.

Format: See Figure III. 3.

Word 1: word 1 contains the block type (2) and the ID of the indicator.

Word 2: pointer to indicator conditioning block. This word has a value of zero if there are no dynamic conditions.

Word 3: NOOP (Display Command)

Word 4: Enter Register mode for display indicator select.

Word 5: assignment of channel on which indicator is to be displayed.

Word 6 and 7: Display jump to last attached entity of this indicator. If there are no entities then jump is to word 8 of the same block.

Word 8: NOOP

Word 9 and 10: display jump to next indicator. If this is the last indicator then jump is to word 6 of the frame block.

C. Entity Block

Format: See Figure III. 4.

Word 1: contains block type (3) and entity ID.

Word 2: pointer to entity conditioning block. If no such block then value is zero.

Word 3: Enter Register Mode.

Word 4 - 9: Rotation, Translation Registers.

Word 10: Intensity, Texture, Blink control.

Word 11: enter coordinate mode.

Word 12 and 13: Make blanked beam move to the 0, 0 coordinate.

Word 14 and 15: Display jump to components. If there are no components

Word 16: NOOP

Word 17 and 18: Display jump to next entity. If this is last entity on ring, then jump is to word 8 of Indicator Block.

Word 19 and 20: Pointers to conditioning block end and component block end respectively.

Word 21 and 22: Spare

D. Conditioning Blocks

Conditioning blocks contain information which is used by the language processor in the Simulation Phase of system operation to control the display dynamics. Indicators and entities have associated conditioning blocks.

The information in a conditioning block is in the form of individual actions to be performed by the language processor. These action commands are inserted into the conditioning block as the user inputs them via the keyboard. The initial entry into the conditioning block for an indicator or an entity causes a pointer to the conditioning block to be entered into the appropriate indicator or entity block. An 'end' command code is placed into the conditioning block whenever the user changes the level of operations (i.e., goes from indicator to entity or entity to indicator or frame).

In the event that the user's conditioning information includes calculations, mini-compiler source code (a subset of FORTRAN), is stored directly in the conditioning block. Subsequently, the mini-compiler produces object code which is stored in areas of core not necessarily related to the conditioning block. This code is then accessed via pointers from the conditioning block, which are located at the position where the code should be executed. Excess storage from the already compiled source code is then released to the free storage ring via RELF, and the source code is destroyed.

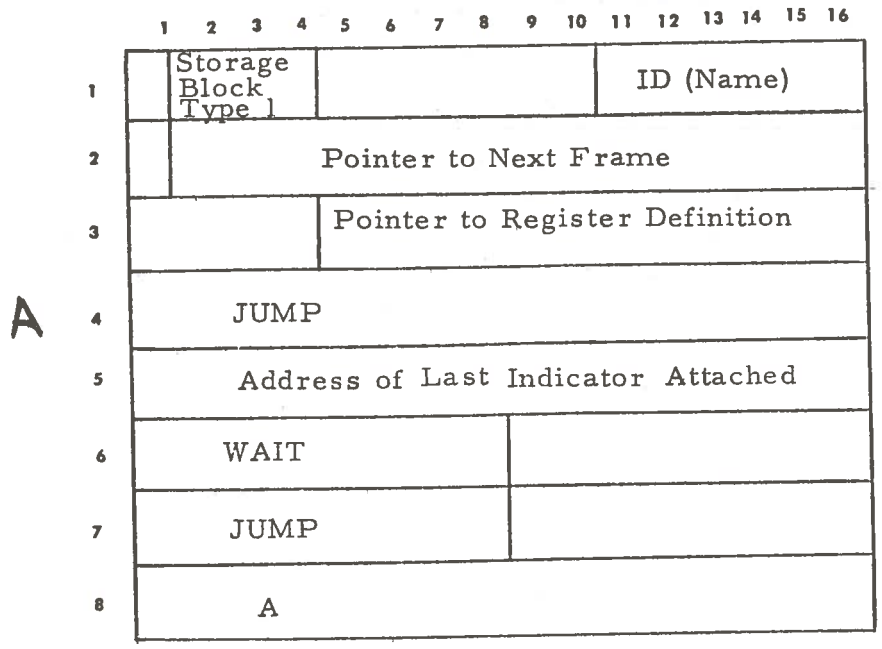


FIGURE III. 2

FRAME BLOCK

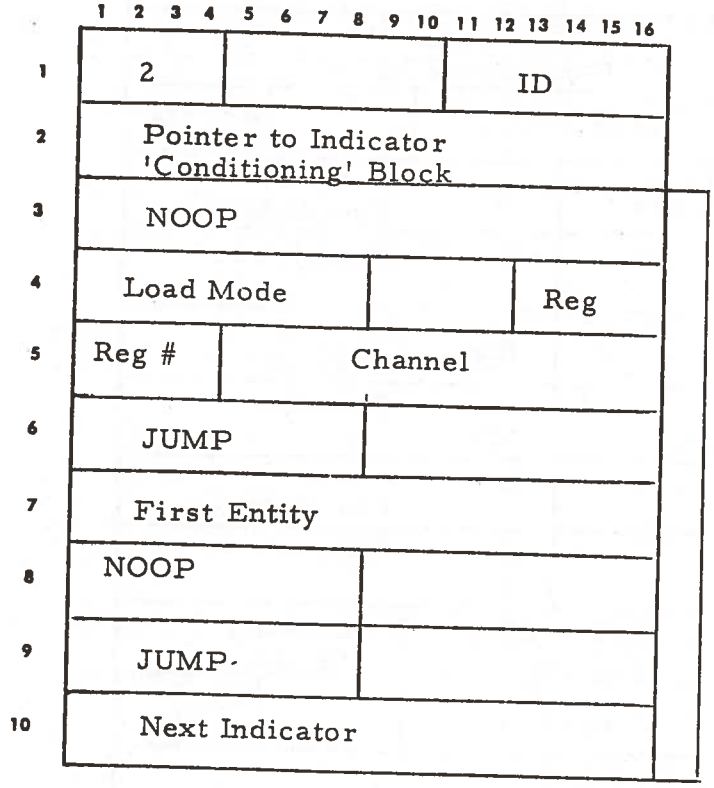


FIGURE III.3
INDICATOR BLOCK

ADDS/900
EXECUTABLE
COMMANDS

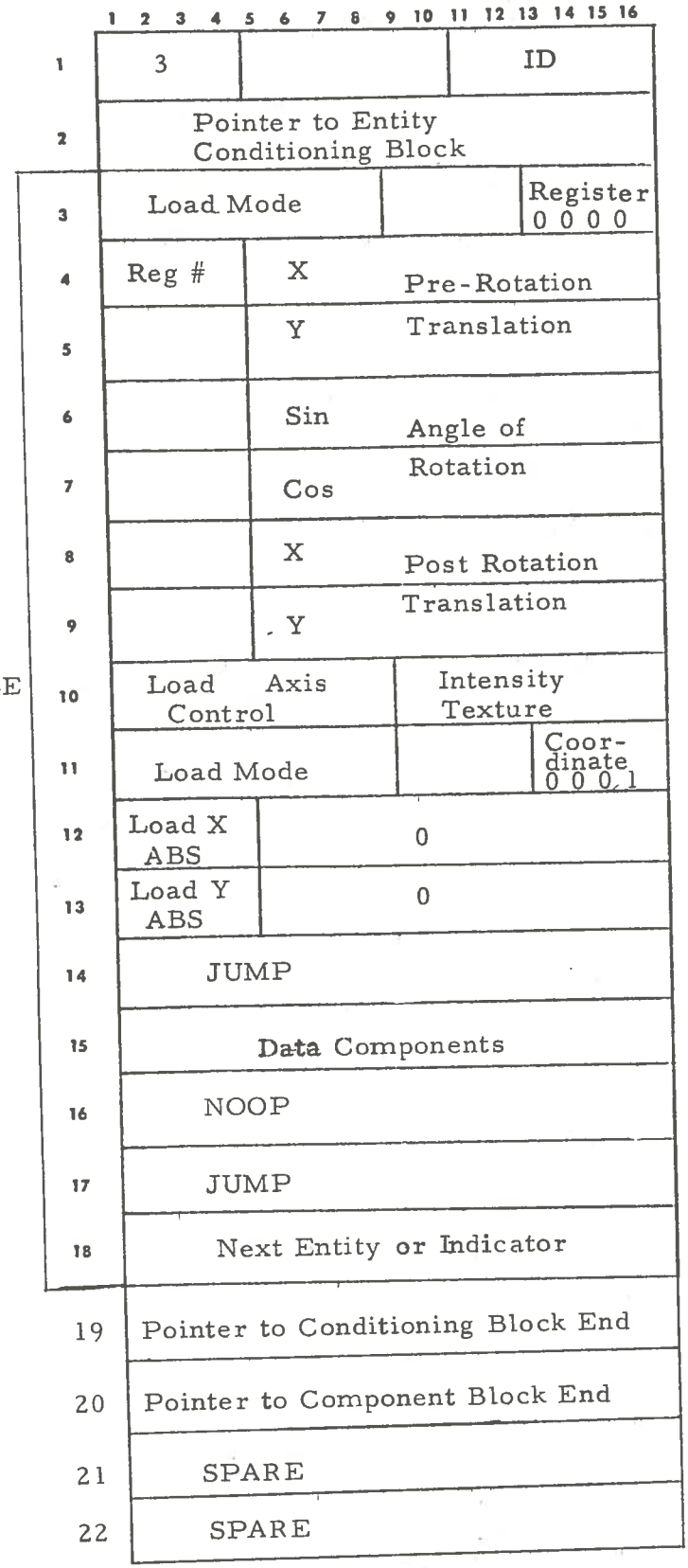


FIGURE III. 4
ENTITY BLOCK

E. Components

Components contain the beam moving commands for the display to draw lines, characters, etc. The components are attached to an entity with display jumps. They are entered into the data structure when the user gives the appropriate keyboard signals to create picture parts. If a component has expressions describing dynamic changes associated with it, then this information is entered into the conditioning block with a pointer to the entity so that the entity can be appropriately modified by the language processor during the simulation phase.

The first user's language drawing command of each entity causes a component block to be opened. Additional component blocks are chained to the first component block as needed.

Display Command Formats for Component Blocks

1. Blank Line

```
LOAD X REL
MOVE Y REL
```

2. Point

```
LOAD X REL
MOVE Y REL
DRAW  POINT
```

3. Line

```
LOAD X REL
DRAW Y REL
```

4. Arc

```
LOAD X REL
MOVE Y REL      Blank move from center
LOAD X REL
DRAW Y REL      One Segment
```

.
.
. The rest of the segments
.
.

LOAD X REL
MOVE Y REL — Blanked move back to center

5. Alphanumerics

LOAD X INDEX REG - For char. spacing
LOAD MODE CHAR
LOAD CHAR SIZE
CHAR1 CHAR2
CHAR3 CHAR4
CHAR5 CHAR6
CHAR7 CHAR8
LOAD MODE COORD

6. Tick Marks for Lines

LOAD X REL
MOVE Y REL — Position
LOAD X INDEX REG - Set up counter
N - No. of tick marks
.
A LOAD X REL
DRAW Y REL — Draw the mark
LOAD X REL
MOVE Y REL — Set for next
DSKZ - DECR. INDEX AND SKIP 2 WORDS ON ZERO
JUMP TO A (2 WORDS)

7. Scales for Lines

LOAD X REL
MOVE Y REL — Position X_i, Y_i
LOAD CHAR SIZE
LOAD X INDEX REG - For char. spacing

```
LOAD X REL
MOVE Y REL
LOAD MODE CHAR
RANDOM CHAR
.                          As needed
LOAD MODE COORD
.
LOAD X REL
MOVE Y REL                 Set for next position
REPEAT
```

8. Tick marks for Arcs

```
LOAD X REL
MOVE Y REL                 Position tick
LOAD X REL
DRAW Y REL                 Draw tick mark
LOAD X REL
MOVE Y REL                 Position to next tick
REPEAT
LOAD X REL
MOVE Y REL                 Move beam back to center
```

9. Scales for Arcs

Same as 7 except reposition beam to center of arc at end of sequence with LOAD X REL
MOVE Y REL

10. Digital Readout

```
LOAD X REL
MOVE Y REL                 Position
RANDOM CHARS               As needed, no more than six
.
.
.
```

F. Dynamic Components Definitions

The following components may be dynamically defined:

1) Point
2) Line
3) Blank Line
4) Digital Readout

In addition, certain dynamic parameters may be specified to alter the entire entity. These are:

1) Rotate
2) Translate
3) Blank/Unblank, Intensity, Blink, and Texture

These commands cause a conditioning command much like that for the dynamic components, but do not affect words in the component block. Rather, words in the entity block header itself are changed. These commands should not be confused with dynamic components, which are actually displayed.

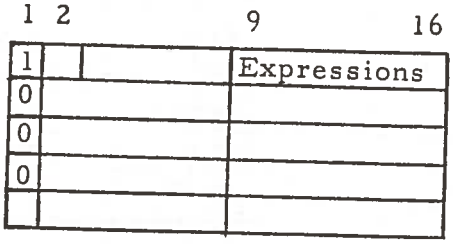
Specifications for dynamic components are written following the conditions in the conditioning block and each has a pointer to the display data for the component.

See Figure III. 5 for the format of dynamic entry in the conditioning block.

G. Register Definition Block

A register definition block is an auxiliary block attached by a pointer to a frame block. It contains expressions which define new registers in terms of model parameter registers.

Bit 1 = 1 means start of expression if bit 2 = 0



NOTE: Expressions replaced by pointer after compilation.

FIGURE III. 5
REGISTER DEFINITION BLOCK

0 = Data
1 = New Cond. Command

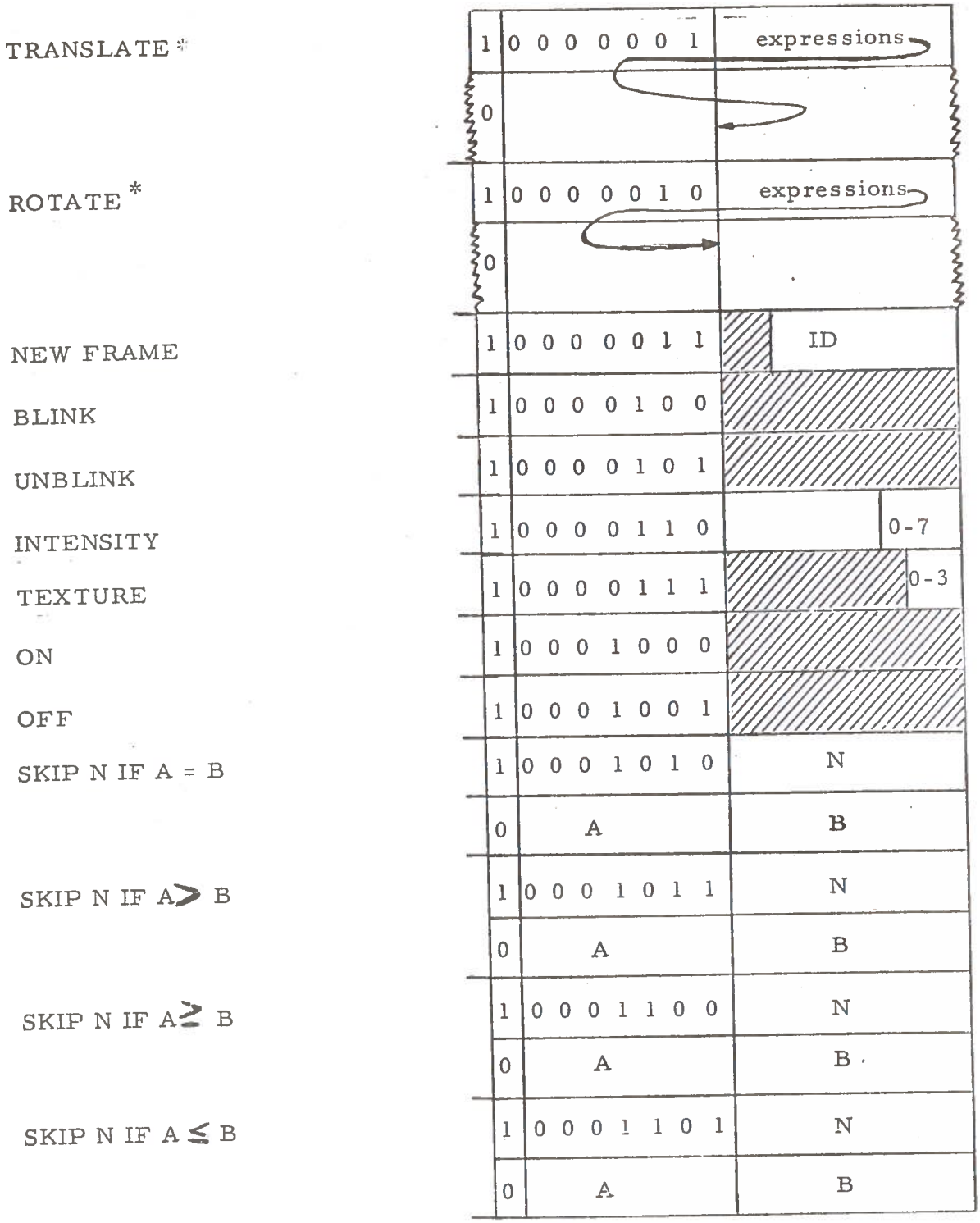


FIGURE III.6
CONDITIONING BLOCK FORMATS

SKIP N IF A ≤ B

1	0 0 0 1 1 1 0	N
0	A	B

SKIP N

1	0 0 0 1 1 1 1	N

END OF BLOCK

1	0 0 1 0 0 0 1	

CHANNEL

1	0 0 1 0 0 0 0	Channel Designation

BLANK, LINE AND POINT *

1	0 0 1 0 0 1 0	expressions
0		

DIGITAL READOUT

1	0 0 1 0 0 1 1	N
0	Readout Expression	

*NOTE: Expressions replaced by pointer after compilation

FIGURE III.6 (Con't)
CONDITIONING BLOCK FORMATS

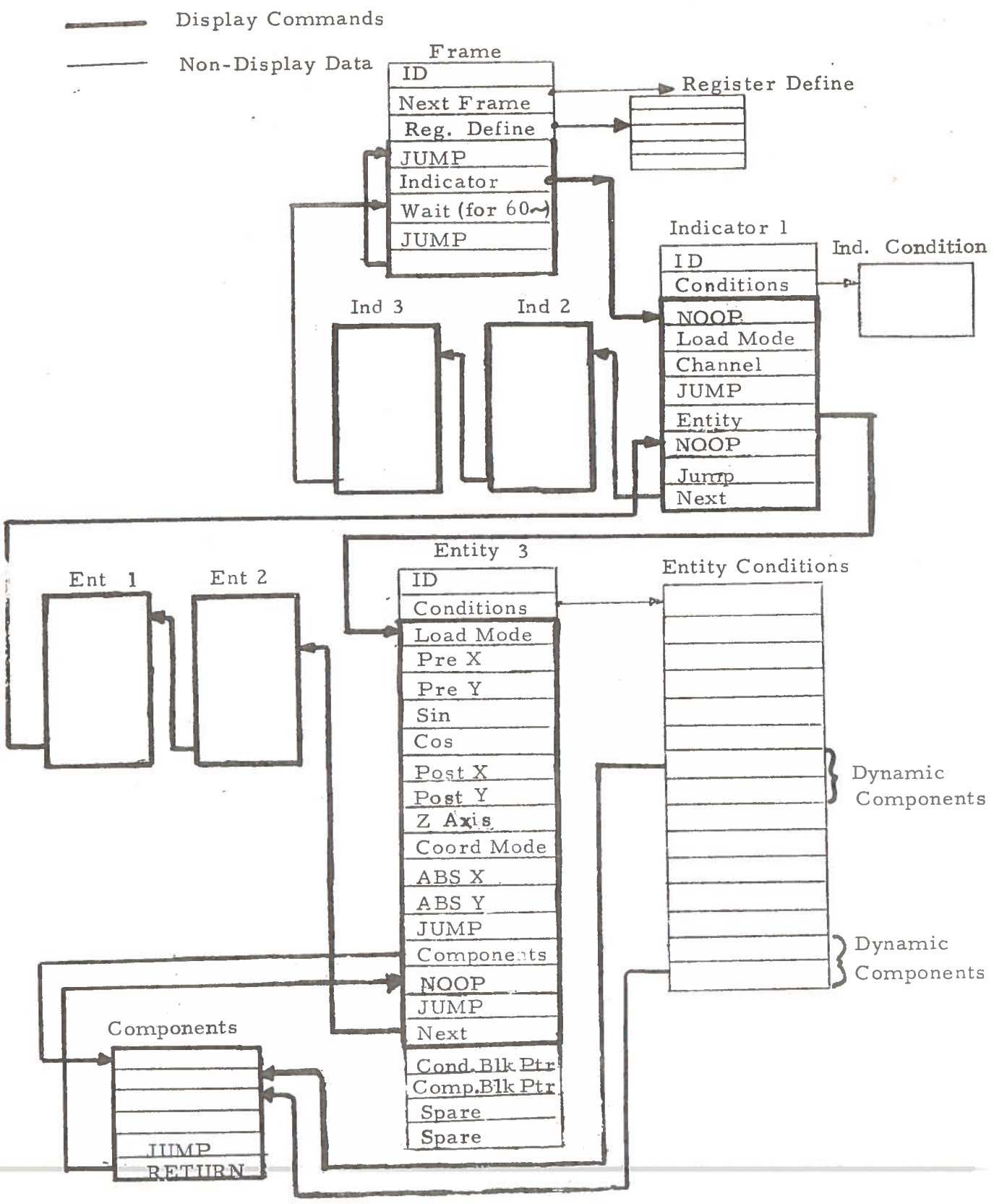
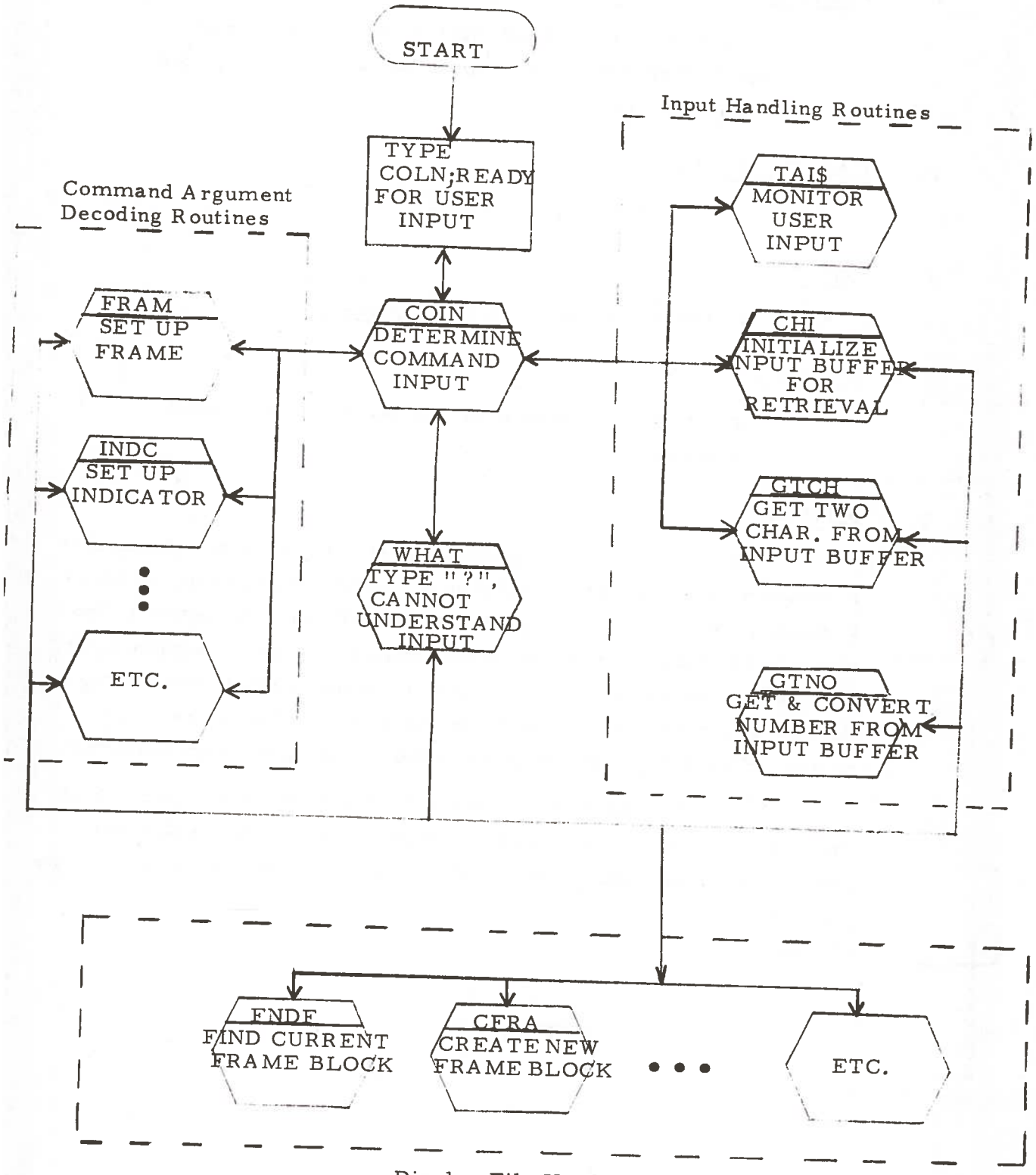


FIGURE III. 7
HIERARCHICAL DATA STRUCTURE

USER LANGUAGE ANALYZER

Command Input Flowchart



SUBROUTINE COIN

1. PURPOSE:

To determine command input from teletype and call appropriate routine to analyze and act on the inputted argument list.

2. CALLING SEQUENCE:

CALL COIN

3. INPUT:

70 character buffer with teletype input.

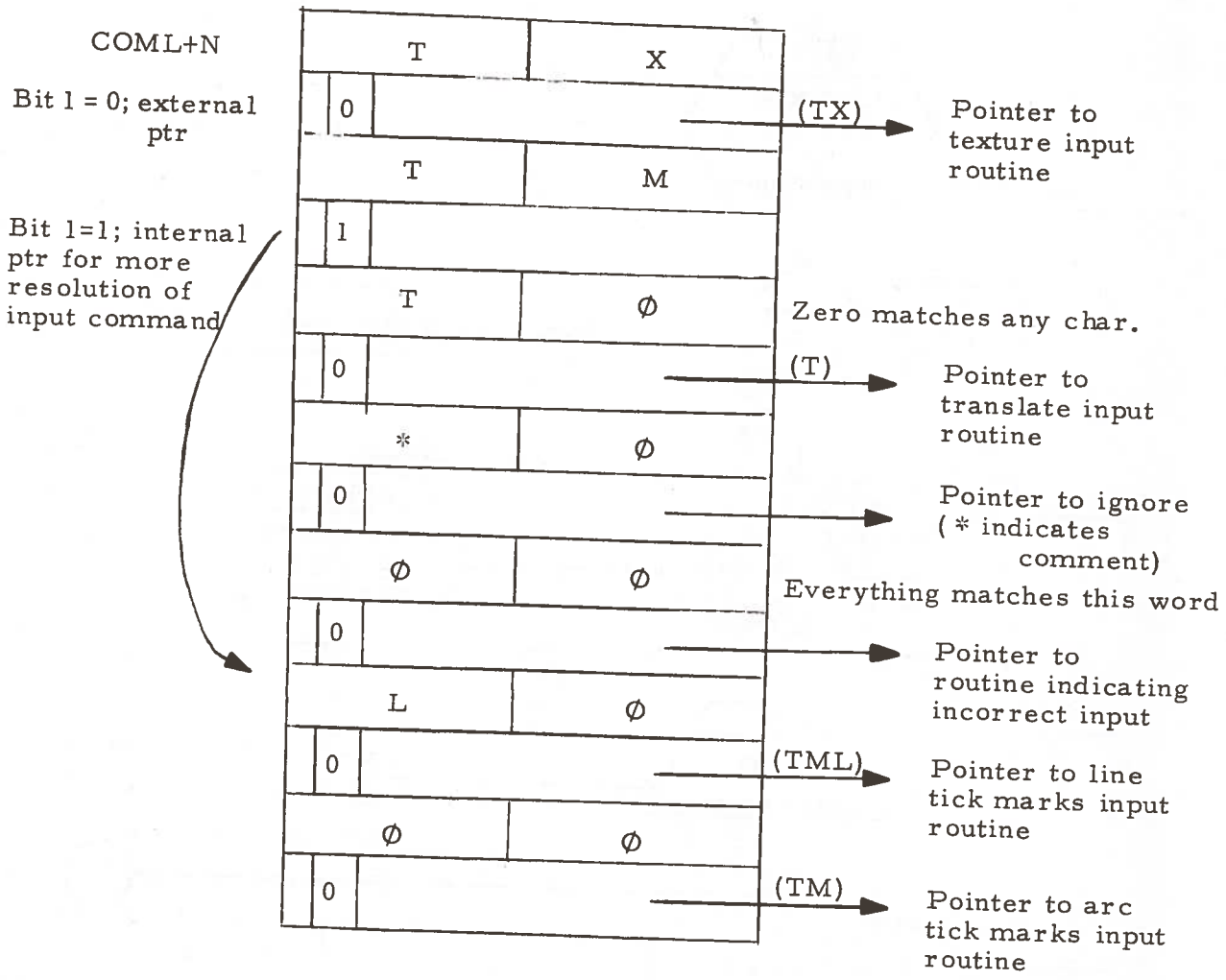
4. OUTPUT:

Calls to routine to analyze argument list of inputted command.

5. ACTION:

Call routine to get input from teletype. Gets two characters from buffer and compares until a match in the command list (COML) is made. (A half-word of zeroes matches with any character.) The next word in the command list after a matched word is examined. If bit 2 is set, further resolution of the command is made by checking the next two characters in the buffer against the characters in the portion of the list pointed to by the word following the matched word.

If bit 2 of the word following the matched word is zero, then the word is a pointer to an external routine, and control calls that routine, after locating a space, tab, comma, or carriage return in the input string.



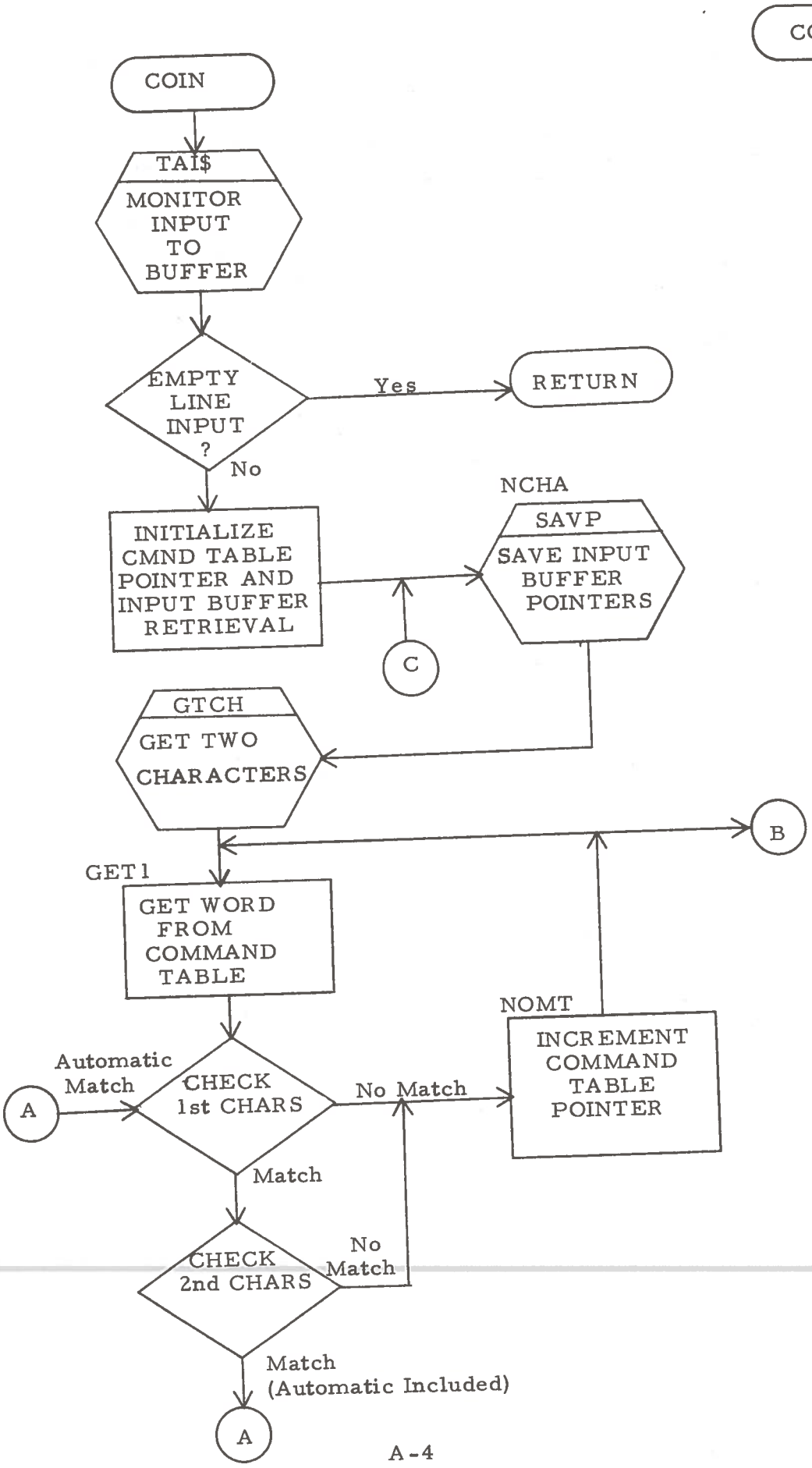
Example of what COML will look like

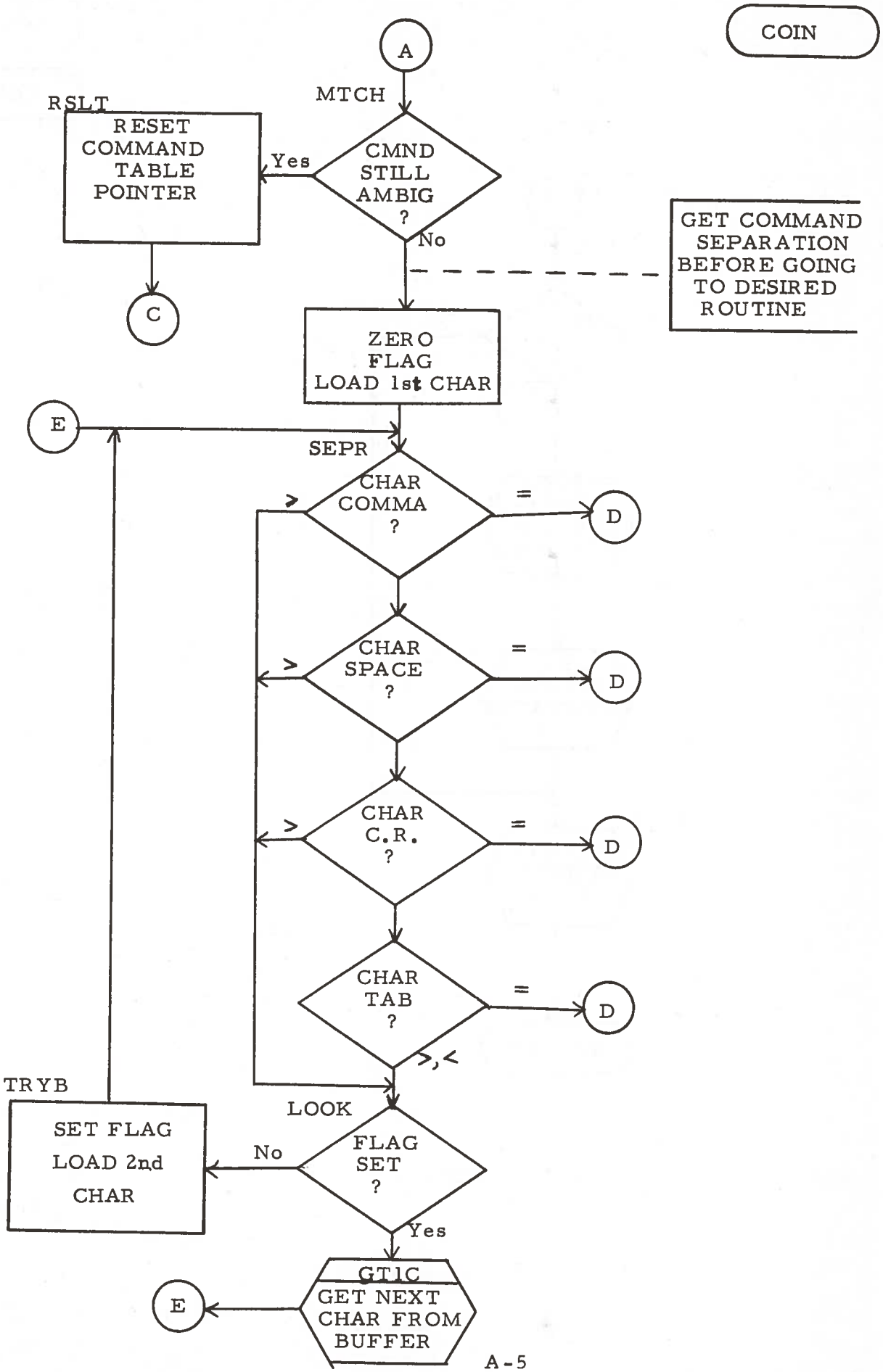
Figure 1

6. EXTERNAL REFERENCES:

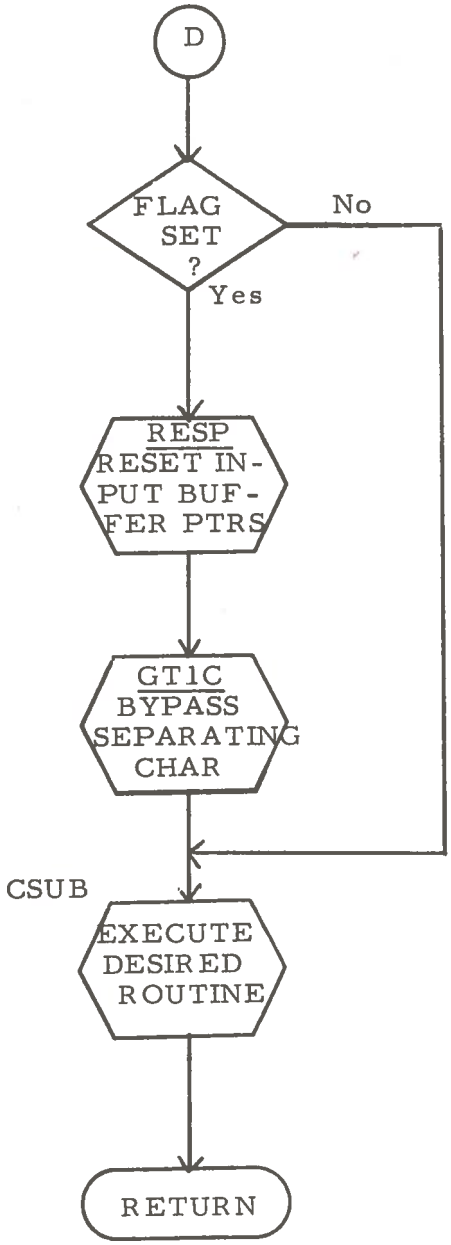
TAI$
CHBF
CHI
GTCH
COML
GT1C

7. CORE USED: 122_8


COIN
TAI$
MONITOR INPUT TO BUFFER
EMPTY LINE INPUT ?
Yes
RETURN
No
NCHA
SAVP
SAVE INPUT BUFFER POINTERS
INITIALIZE CMND TABLE POINTER AND INPUT BUFFER RETRIEVAL
C
GTCH
GET TWO CHARACTERS
B
GET1
GET WORD FROM COMMAND TABLE
NOMT
INCREMENT COMMAND TABLE POINTER
Automatic Match
A
CHECK 1st CHARS
No Match
Match
CHECK 2nd CHARS
No Match
Match (Automatic Included)
A


COIN
A
MTCH
RSLT
RESET COMMAND TABLE POINTER
Yes
CMND STILL AMBIG ?
No
GET COMMAND SEPARATION BEFORE GOING TO DESIRED ROUTINE
C
ZERO FLAG LOAD 1st CHAR
E
SEPR
CHAR COMMA ?
=
D
>
CHAR SPACE ?
CHAR C.R. ?
CHAR TAB ?
>,<
TRYB
LOOK
SET FLAG LOAD 2nd CHAR
No
FLAG SET ?
Yes
GT1C
GET NEXT CHAR FROM BUFFER

COIN


D
FLAG SET ?
No
Yes
RESP
RESET INPUT BUFFER PTRS
GT1C
BYPASS SEPARATING CHAR
CSUB
EXECUTE DESIRED ROUTINE
RETURN

SUBROUTINE COML

Command List

1. PURPOSE:

Buffer containing characters to be matched against inputted commands and containing pointers to subroutines to analyze arguments. (Description of buffer structure is write-up for Subroutine COIN.)

2. ENTRY POINT:

COML

3. INPUT:

None

4. OUTPUT:

None

5. ACTION:

None

6. EXTERNAL REFERENCES:

Pointers to subroutines to be called when command is interpreted.

7. CORE USED:

172_8

SUBROUTINE COLN

1. PURPOSE:

Types colon to indicate it is ready for user input.

2. CALLING SEQUENCE:

Not normally called. Main executive R+N.

3. INPUT:

None.

4. OUTPUT:

Outputs colon on teletype.

5. ACTION:

Outputs colon on teletype, turns on ADDS if off and CLVL 70, then calls routine to interpret input commands.

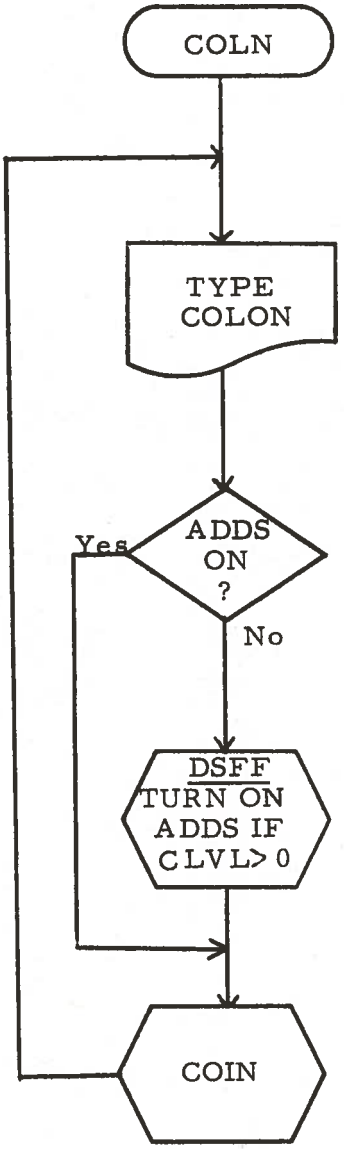
6. EXTERNAL REFERENCES:

TAO1
COIN
DSFF

7. CORE USED:

11_8

COLN



SUBROUTINE: CENT

1. PURPOSE:

Create an entity block and tie it to the appropriate entity ring.

2. CALLING SEQUENCE:

Call CENT

3. INPUT:

Indicator address in INAD entity ID in ENID

4. OUTPUT:

Start address of frame block in AC/zero in AC if no storage.

5. ACTION:

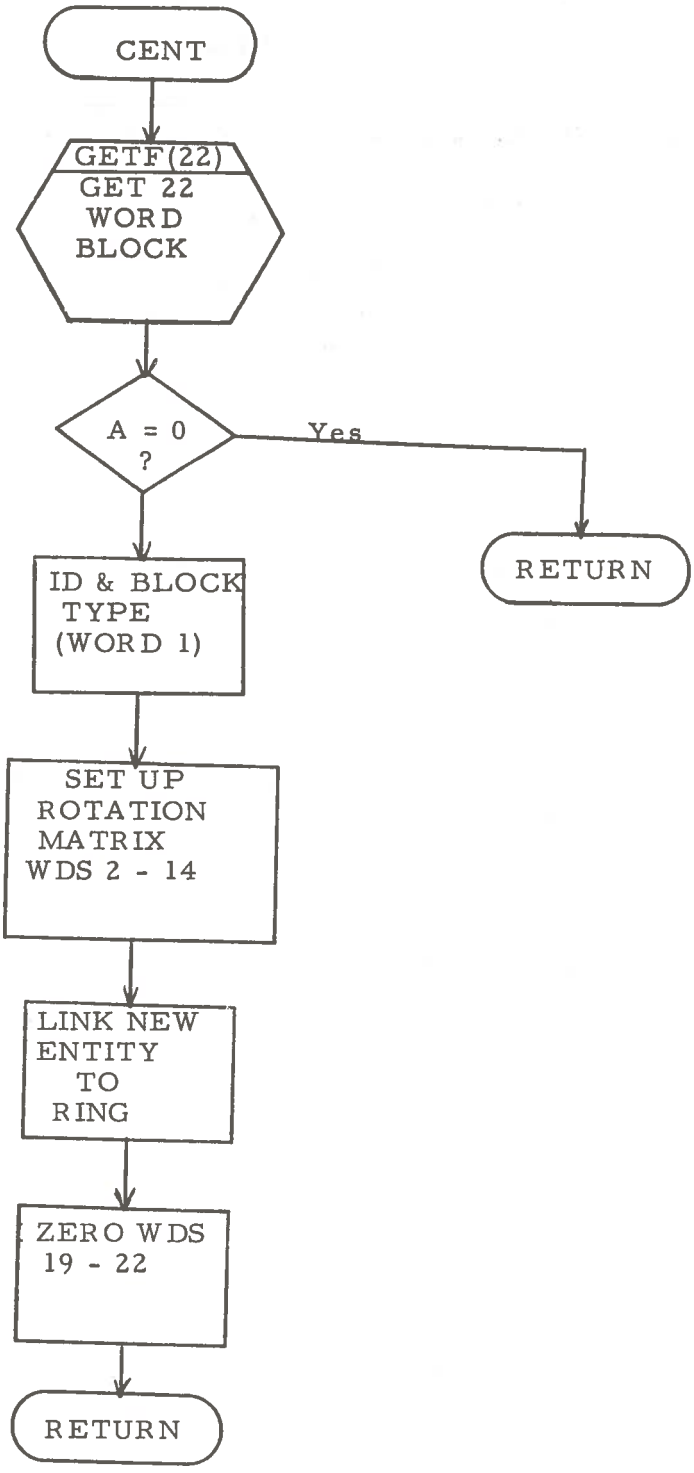
A block is retrieved from free storage, set up as an entity, attached to the appropriate entity ring and the entity address is returned in the AC.

6. EXTERNAL REFERENCES:

INAD
ENID

7. CORE USED:

117_8


CENT
GETF(22)
GET 22
WORD
BLOCK
A = 0
?
Yes
RETURN
ID & BLOCK
TYPE
(WORD 1)
SET UP
ROTATION
MATRIX
WDS 2 - 14
LINK NEW
ENTITY
TO
RING
ZERO WDS
19 - 22
RETURN

SUBROUTINE CFRM

1. PURPOSE:

Create a frame block and tie it to frame ring.

3. CALLING SEQUENCE:

Call CFRM

3. INPUT:

Frame ID in FRID

4. OUTPUT:

Start address of frame block in A / zero in A if no storage.

5. ACTION :

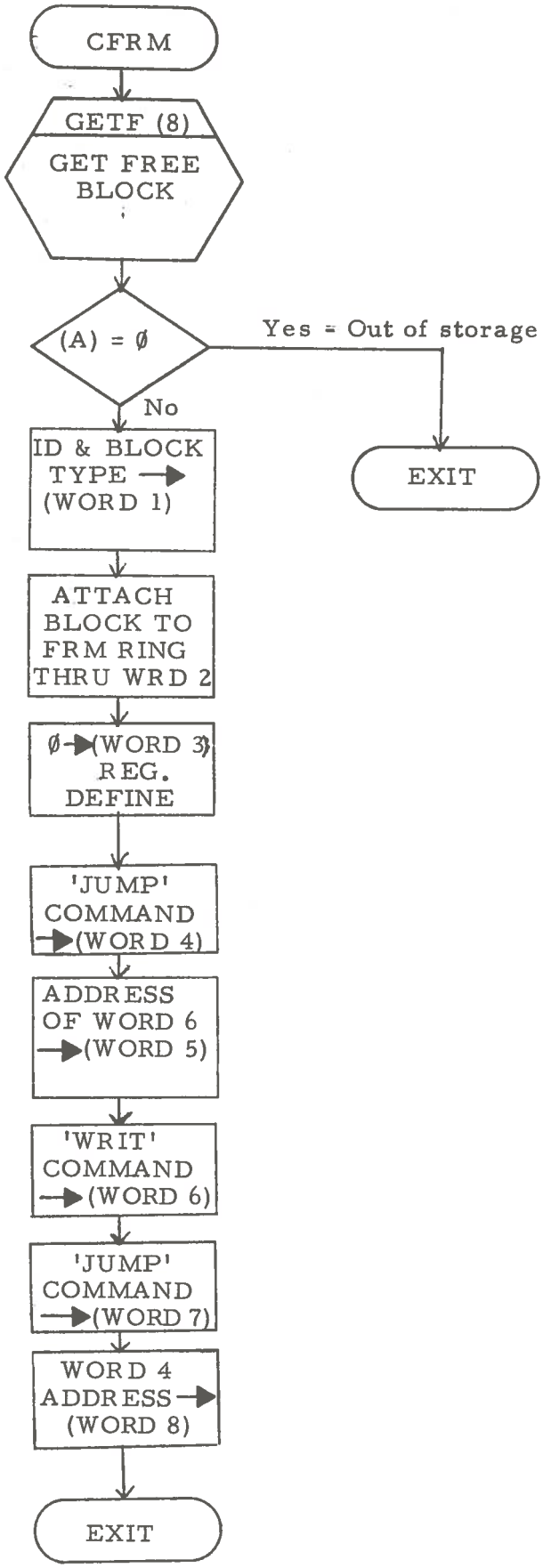
An eight word block is retrieved from free storage, set up as a frame block and attached to the frame ring. The starting address of the frame is returned in the A Register.

6. EXTERNAL REFERENCES:

FRID Frame ID
FRHD Frame Header

7. CORE USED:

57_8


CFRM
GETF (8)
GET FREE BLOCK
(A) = Ø
Yes = Out of storage
No
EXIT
ID & BLOCK TYPE → (WORD 1)
ATTACH BLOCK TO FRM RING THRU WRD 2
Ø → (WORD 3) REG. DEFINE
'JUMP' COMMAND → (WORD 4)
ADDRESS OF WORD 6 → (WORD 5)
'WRIT' COMMAND → (WORD 6)
'JUMP' COMMAND → (WORD 7)
WORD 4 ADDRESS → (WORD 8)
EXIT

SUBROUTINE CHAD

1. PURPOSE:

Converts 16-bit word to left adjusted table of Decimal characters.

2. CALLING SEQUENCE:

```
LDA     NUM     / 16-bit word to be converted
CALL    CHAD
LOC     TABL    / Pointer to table where chars
                  are to be packed
```

Returns with the number at chars in the A-register

3. INPUT:

a) An octal integer in the A-register
b) Pointer to 4-word table following call statement

4. OUTPUT:

Table of packed decimal characters. Character count in A-register. If odd number of characters, right half of last word contains the null character.

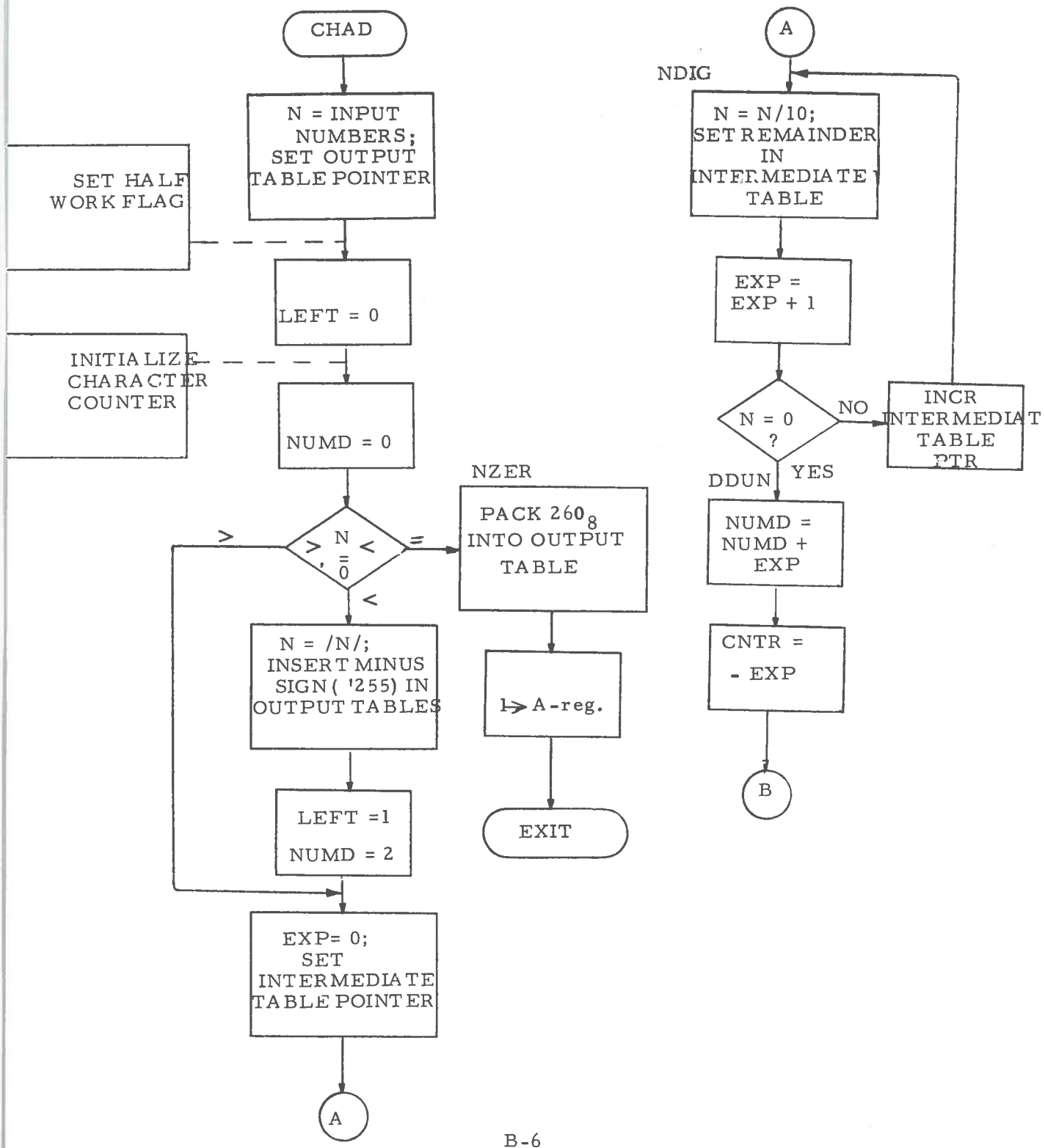
5. ACTION:

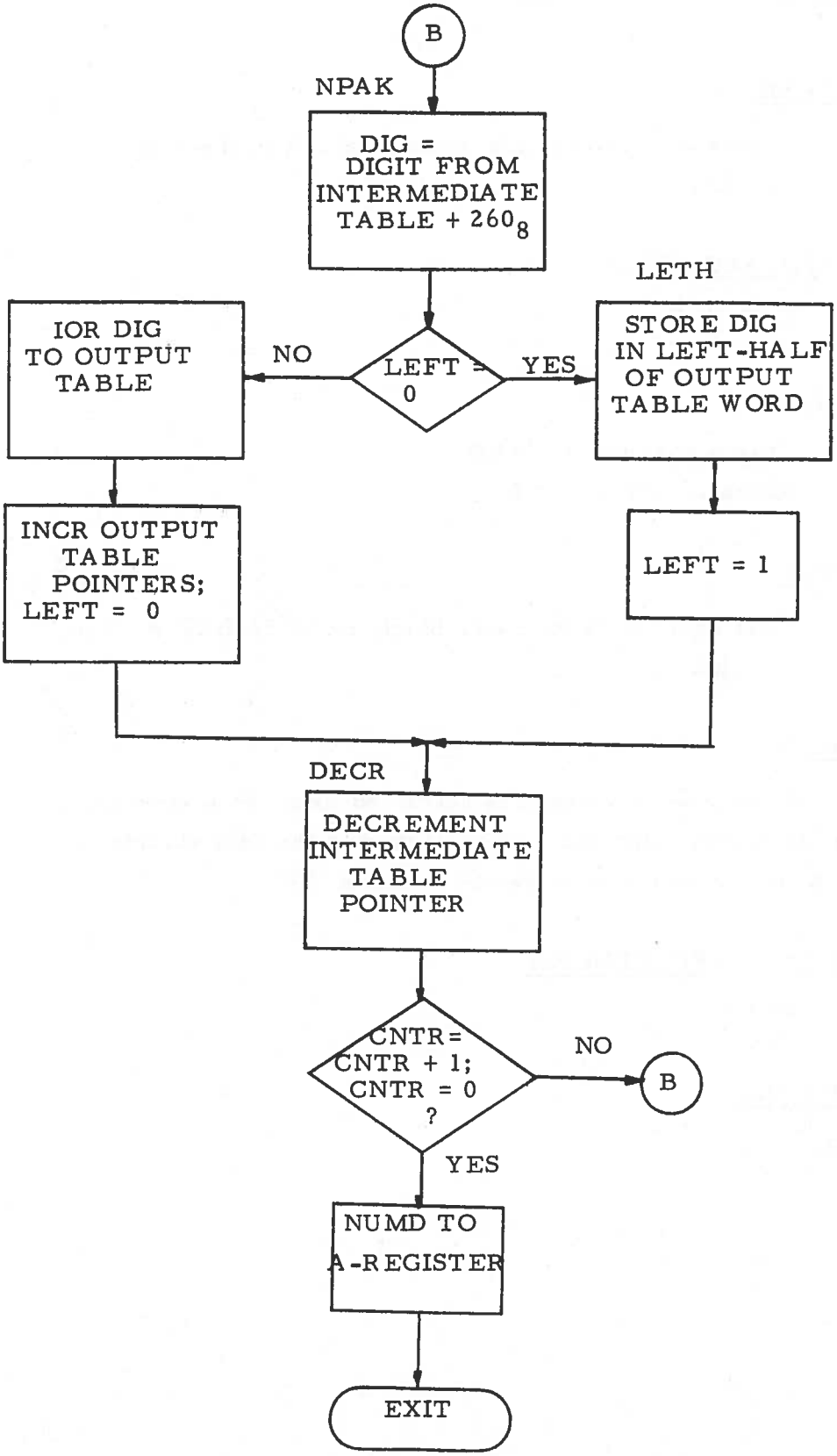
Places minus sign in table if number is negative and complements number. If number is zero, packs zero character and returns. Otherwise breaks number down into base ten digits in an intermediate table. Then, backing through intermediate table, the proper char for each digit is packed into the given table. When finished it returns to the calling program.

6. EXTERNAL REFERENCES: None

7. CORE USED: 132_8

CHAD Flowchart




B
NPAK
DIG = DIGIT FROM INTERMEDIATE TABLE + 260₈
LETH
IOR DIG TO OUTPUT TABLE
NO
LEFT = 0
YES
STORE DIG IN LEFT-HALF OF OUTPUT TABLE WORD
INCR OUTPUT TABLE POINTERS; LEFT = 0
LEFT = 1
DECR
DECREMENT INTERMEDIATE TABLE POINTER
CNTR = CNTR + 1; CNTR = 0 ?
NO
B
YES
NUMD TO A-REGISTER
EXIT

SUBROUTINE: CIND

1. PURPOSE:

Create an indicator block and attach it to the data structure.

2. CALLING SEQUENCE:

Call CIND

3. INPUT:

Frame address in FRAD
Indicator ID in INID

4. OUTPUT:

Start address of indicator block in A / zero in A if no storage.

5. ACTION:

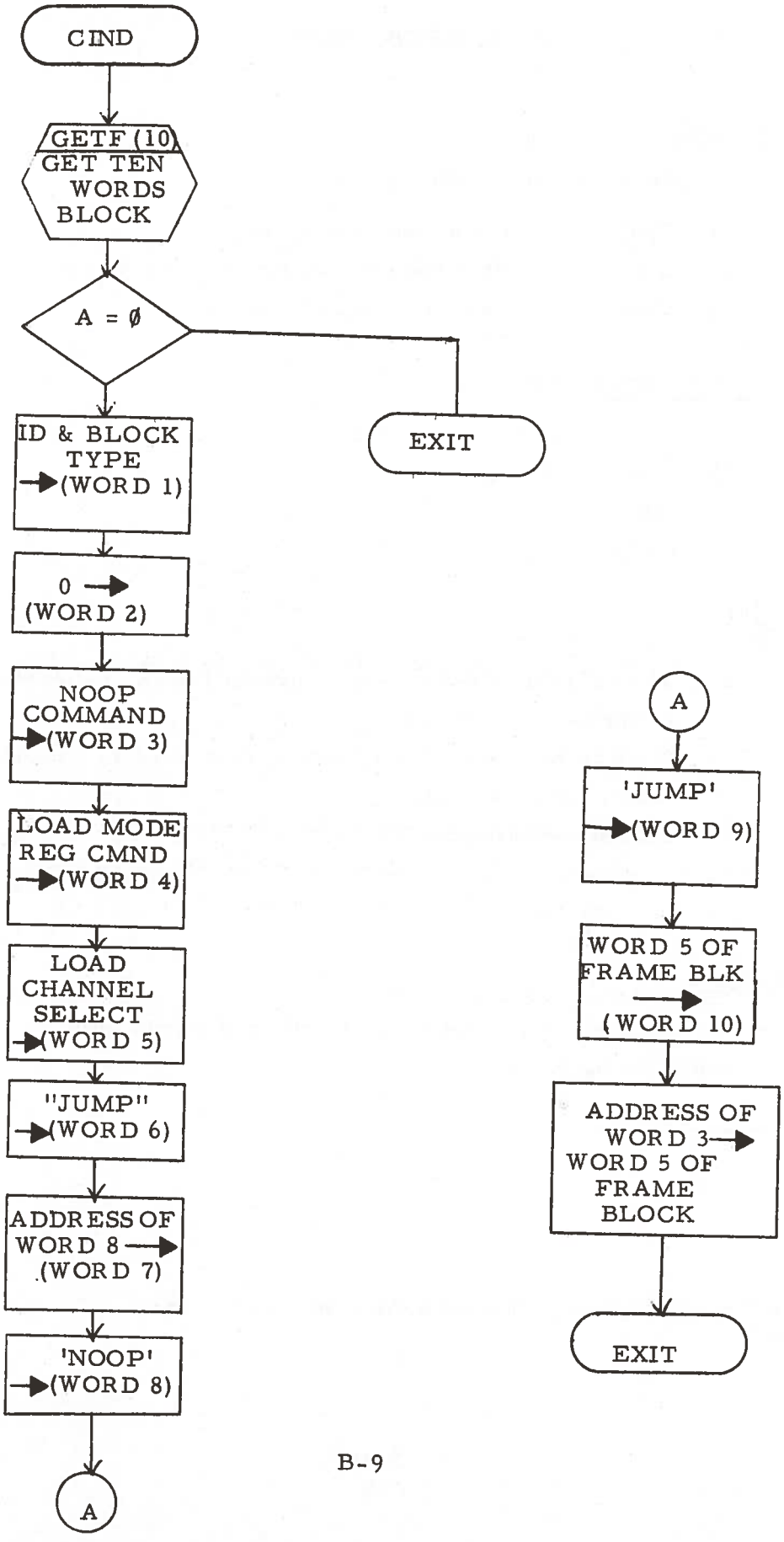
A block of 10 words is retreived from free storage, set up as an indicator block and attached to the data structure. The SA of the indicator is returned in the AC.

6. EXTERNAL REFERENCES:

FRAD
INID

7. CORE USED:

64_8


CIND
GETF (10)
GET TEN WORDS BLOCK
A = ∅
EXIT
ID & BLOCK TYPE (WORD 1)
0 (WORD 2)
NOOP COMMAND (WORD 3)
LOAD MODE REG CMND (WORD 4)
LOAD CHANNEL SELECT (WORD 5)
"JUMP" (WORD 6)
ADDRESS OF WORD 8 (WORD 7)
'NOOP' (WORD 8)
A
A
'JUMP' (WORD 9)
WORD 5 OF FRAME BLK (WORD 10)
ADDRESS OF WORD 3 WORD 5 OF FRAME BLOCK
EXIT

SUBROUTINE: CLCB, OPCB, INCB

1. PURPOSE:

Handlers for conditioning blocks. *

a) OPCB - Opens a conditioning block
b) INCB - Inserts a word into conditioning block
c) CLCB - closes a conditioning block

2. CALLING SEQUENCE:

```
a)  CALL   OPCB
b)  LDA    WORD
    CALL   INCB
c)  CALL   CLCB
```

3. INPUT:

a) The current working level, current frame, indicator or entity block, free storage
b) Word to be inserted, location of next word in conditioning block. Free storage
c) Current working level.

4. OUTPUT:

Set or reset parameters for insertion of words into conditioning block.

* Define Register block is treated as a conditioning block in Phase 1.

5. ACTION:

a) OPCB - If no conditioning block yet assigned (i.e., conditioning block pointer or register definition block pointer is zero) a 40_8 word block is retrieved from free storage and a pointer to it is inserted into the appropriate header. If a previous conditioning block exists, the 40_8 word block is linked to the last block.

b) INCB - Store word in A-Reg. in next available conditioning block word. If the block is full (except for two words used for return or linking), a new block is retrieved from free storage and linked to the old by inserting an "end" command and calling OPCB.

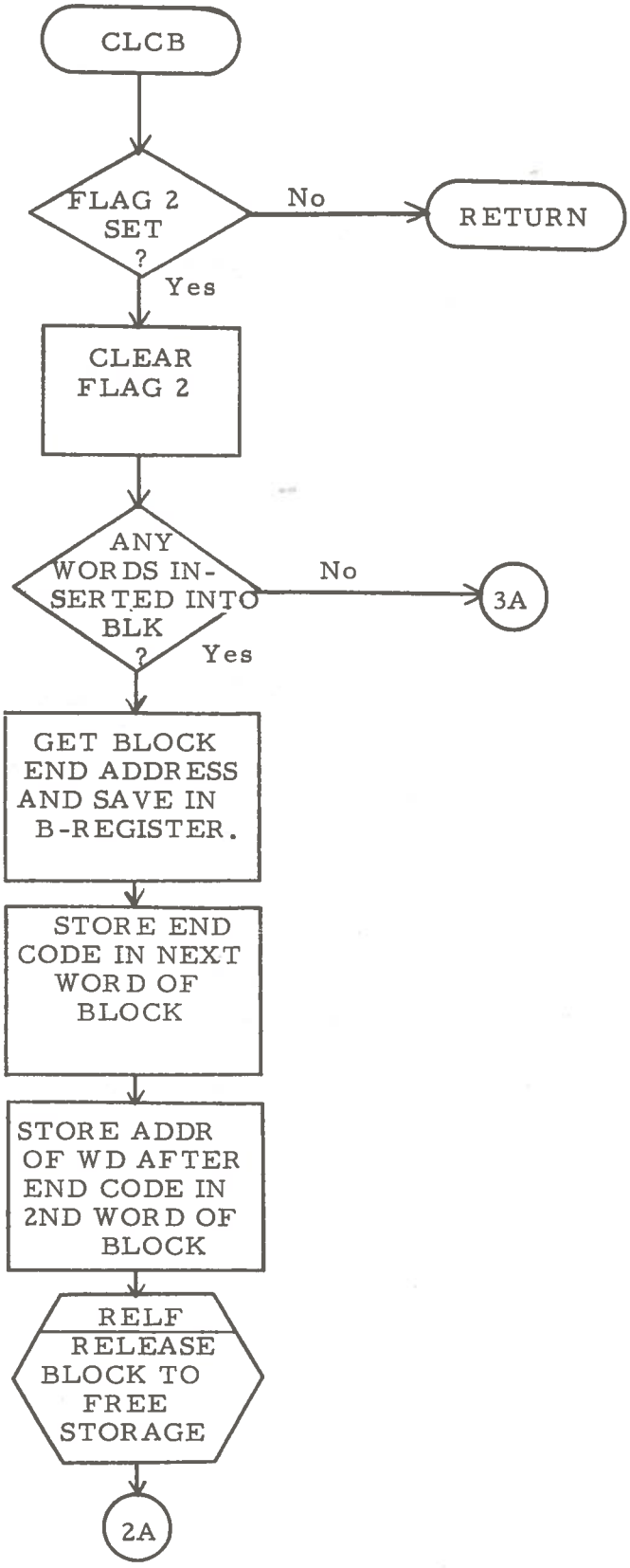
c) CLCB - If no words have been inserted into the conditioning block, the entire block is returned to free storage. Otherwise the end code is inserted into the next available word followed by a zero and this address is stored in word 2 of the conditioning block. The remainder of the block is then returned to free storage.

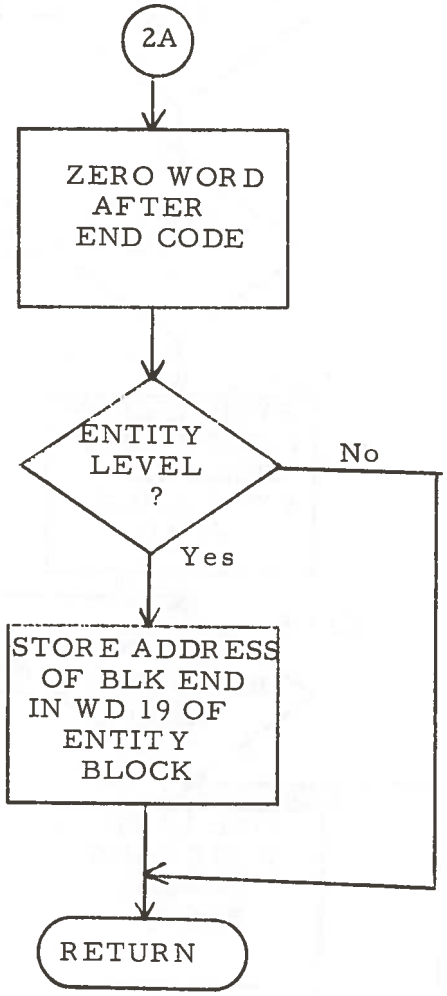
6. EXTERNAL REFERENCES:

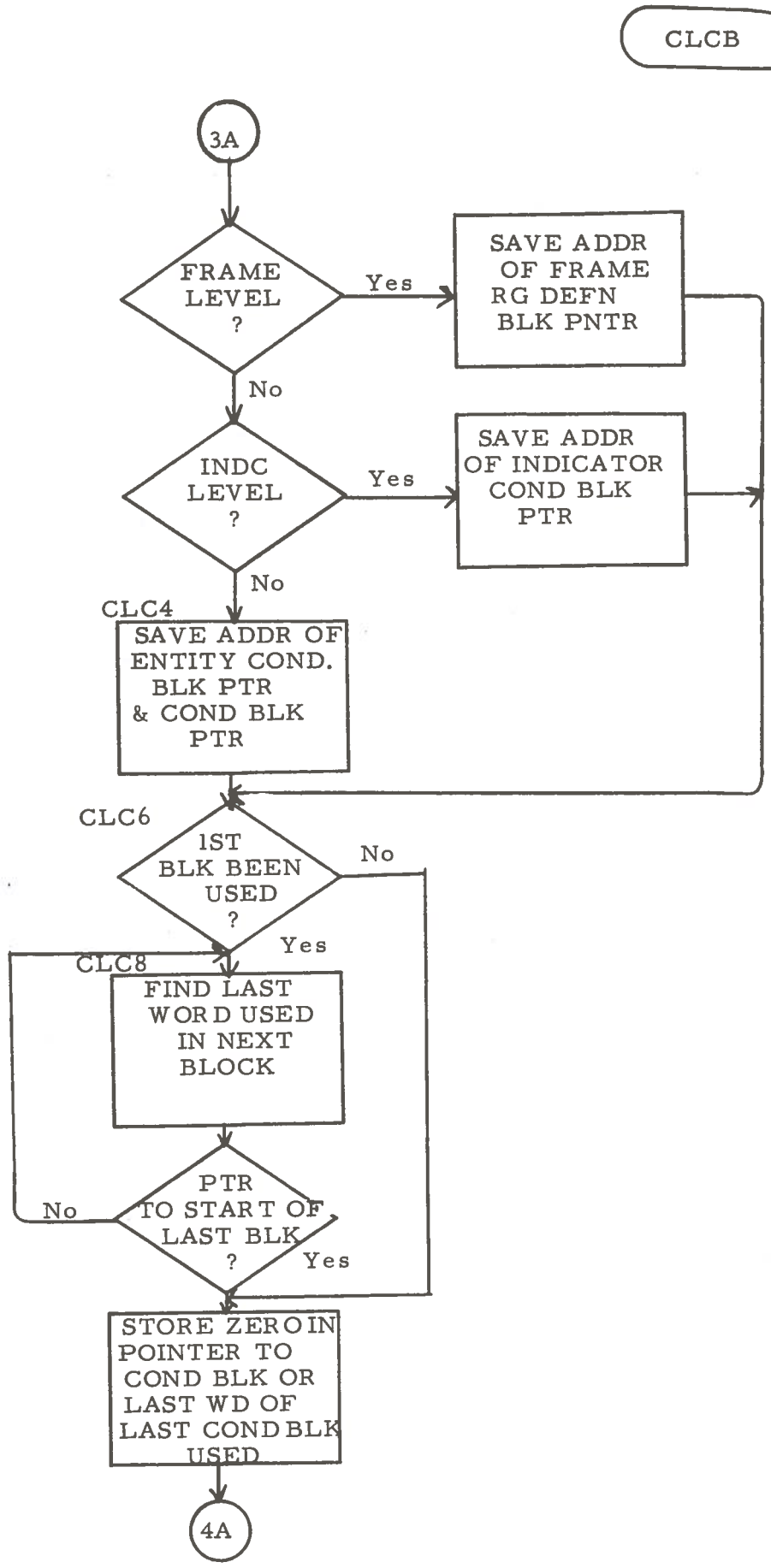
FLG2 - Set if Conditioning Block Open
CLVL - Current Level
FRAD - Frame Address
INAD - Indicator Address
ENAD - Entity Address

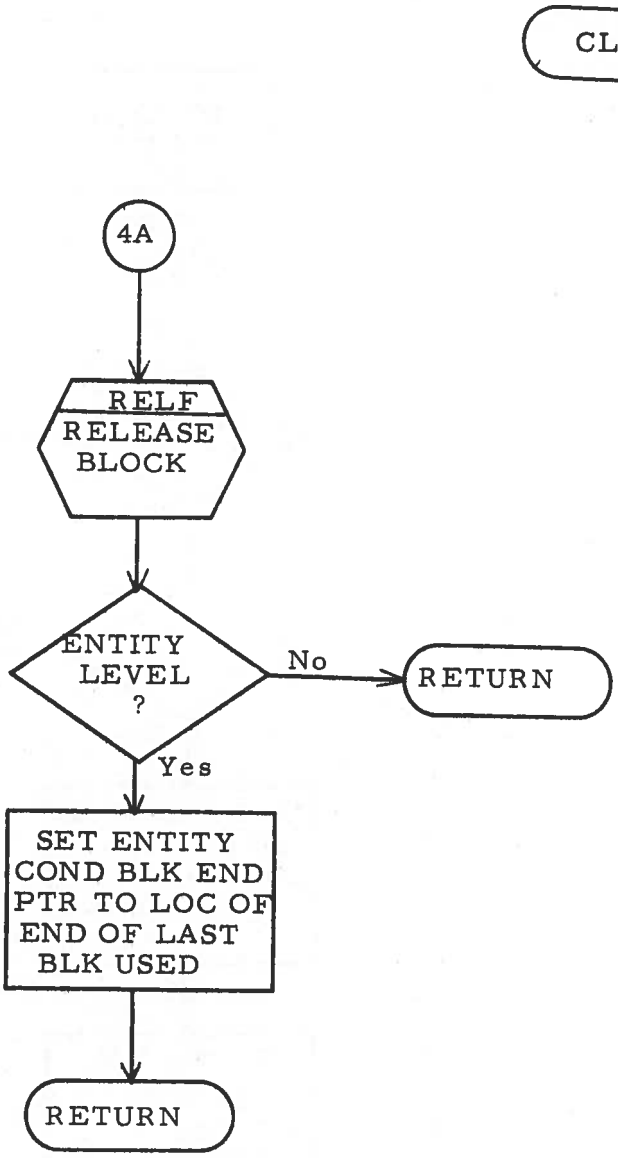
7. CORE USED:

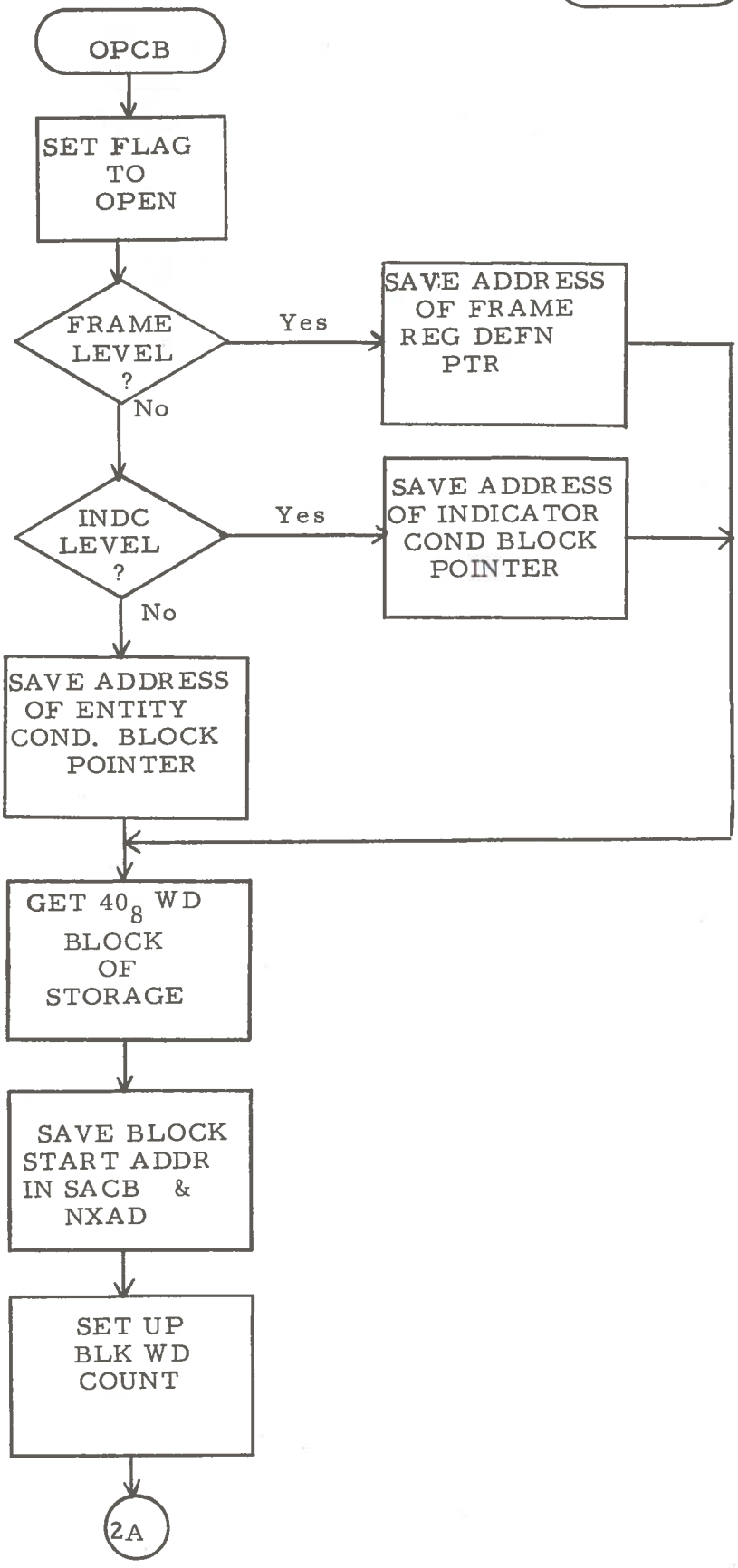
261_8

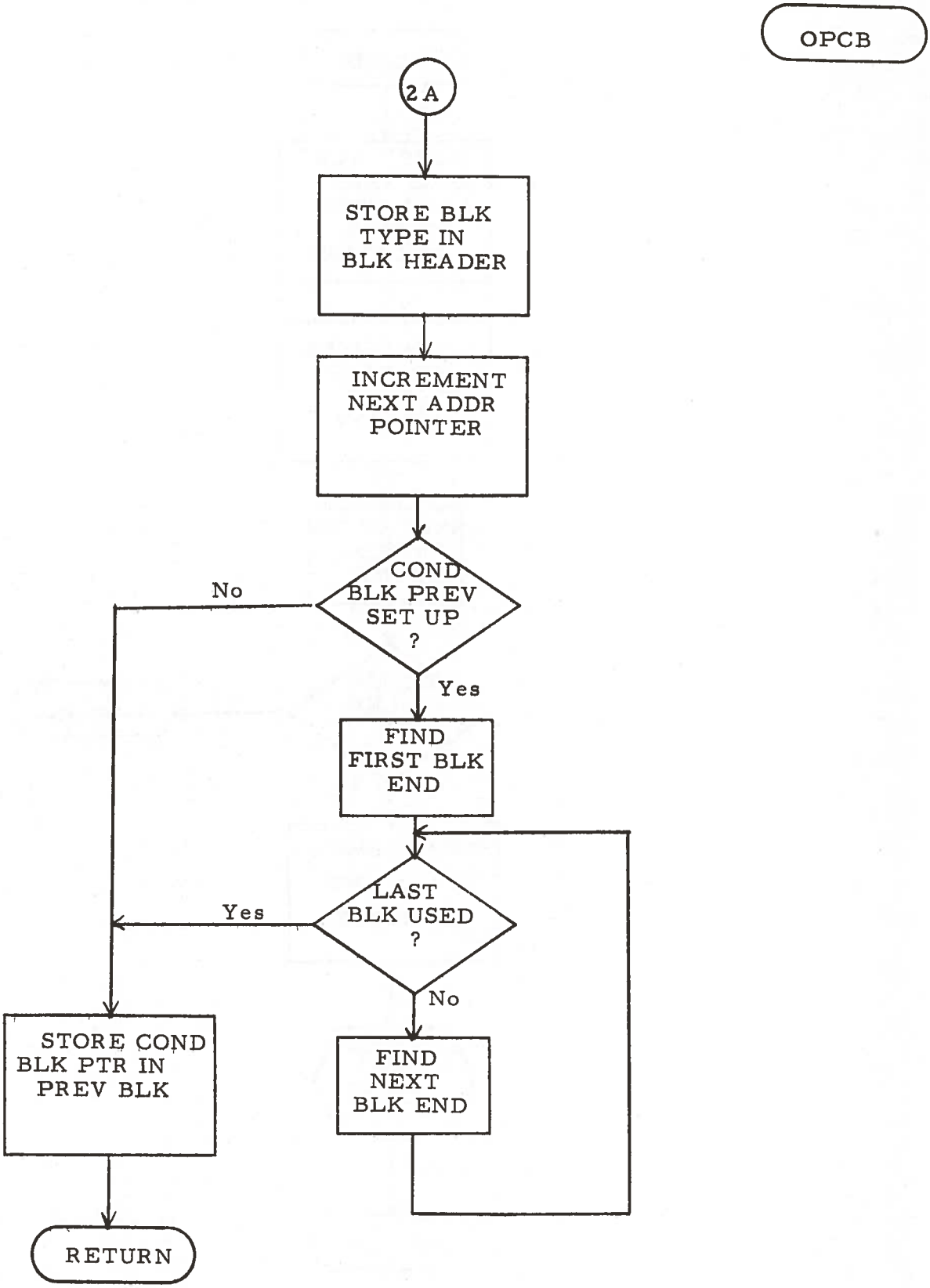

CLCB
FLAG 2 SET ?
No
RETURN
Yes
CLEAR FLAG 2
ANY WORDS IN-SERTED INTO BLK ?
No
3A
Yes
GET BLOCK END ADDRESS AND SAVE IN B-REGISTER.
STORE END CODE IN NEXT WORD OF BLOCK
STORE ADDR OF WD AFTER END CODE IN 2ND WORD OF BLOCK
RELF
RELEASE BLOCK TO FREE STORAGE
2A

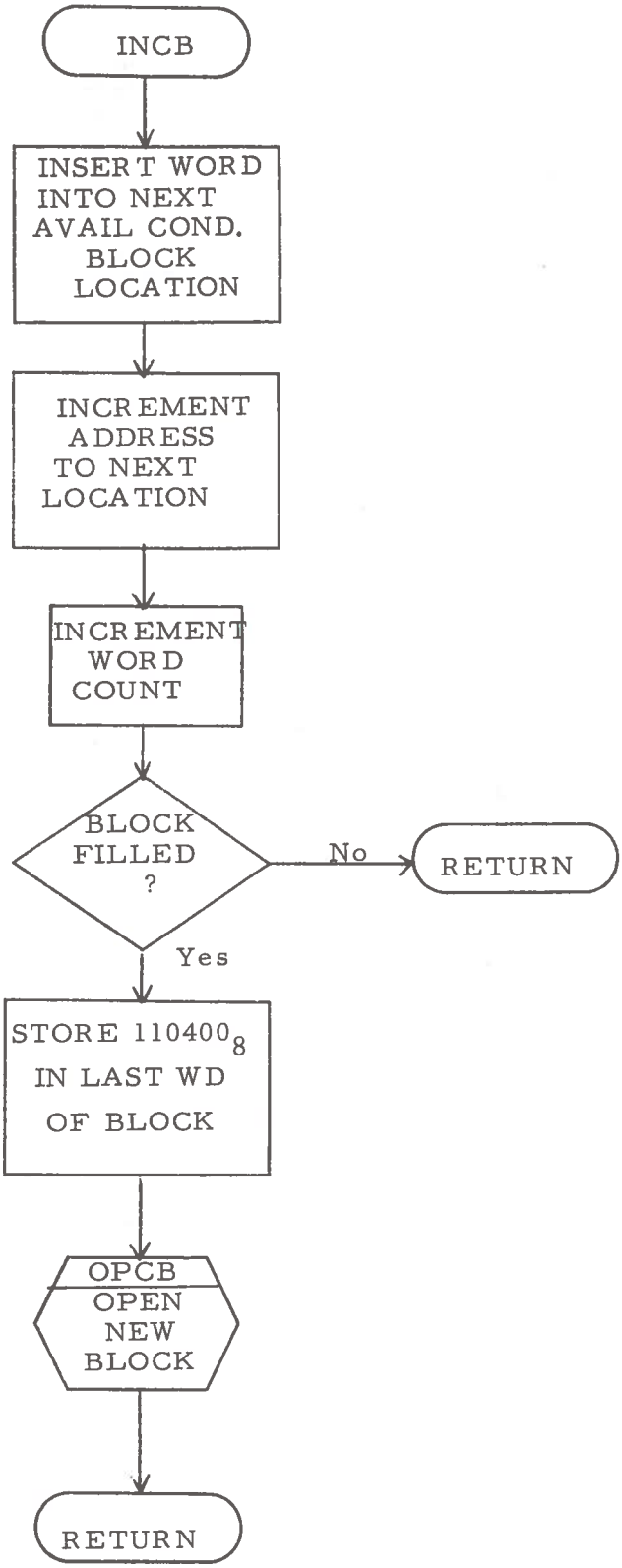

2A
ZERO WORD AFTER END CODE
ENTITY LEVEL ?
No
Yes
STORE ADDRESS OF BLK END IN WD 19 OF ENTITY BLOCK
RETURN


CLCB
3A
FRAME LEVEL ?
Yes
SAVE ADDR OF FRAME RG DEFN BLK PNTR
No
INDC LEVEL ?
Yes
SAVE ADDR OF INDICATOR COND BLK PTR
No
CLC4
SAVE ADDR OF ENTITY COND. BLK PTR & COND BLK PTR
CLC6
1ST BLK BEEN USED ?
No
Yes
CLC8
FIND LAST WORD USED IN NEXT BLOCK
PTR TO START OF LAST BLK ?
No
Yes
STORE ZERO IN POINTER TO COND BLK OR LAST WD OF LAST COND BLK USED
4A


CLCB
4A
RELF
RELEASE
BLOCK
ENTITY
LEVEL
?
No
RETURN
Yes
SET ENTITY
COND BLK END
PTR TO LOC OF
END OF LAST
BLK USED
RETURN


OPCB
SET FLAG TO OPEN
FRAME LEVEL ?
Yes
SAVE ADDRESS OF FRAME REG DEFN PTR
No
INDC LEVEL ?
Yes
SAVE ADDRESS OF INDICATOR COND BLOCK POINTER
No
SAVE ADDRESS OF ENTITY COND. BLOCK POINTER
GET 40_8 WD BLOCK OF STORAGE
SAVE BLOCK START ADDR IN SACB & NXAD
SET UP BLK WD COUNT
2A


OPCB
2A
STORE BLK TYPE IN BLK HEADER
INCREMENT NEXT ADDR POINTER
COND BLK PREV SET UP ?
No
Yes
FIND FIRST BLK END
LAST BLK USED ?
Yes
No
STORE COND BLK PTR IN PREV BLK
FIND NEXT BLK END
RETURN


INCB
INSERT WORD INTO NEXT AVAIL COND. BLOCK LOCATION
INCREMENT ADDRESS TO NEXT LOCATION
INCREMENT WORD COUNT
BLOCK FILLED ?
No
RETURN
Yes
STORE 110400_8 IN LAST WD OF BLOCK
OPCB
OPEN NEW BLOCK
RETURN

SUBROUTINE CMOD

1. PURPOSE:

Check working level.

2. CALLING SEQUENCE:

```
LDA     MODE
CALL    CMOD
 ***                    / Error Return
 ***                    / Normal Return
```

3. INPUT:

Mode desired is indicated by bits 14-16 of A register.

Bit 14 - 0, Entity level illegal
1, Entity level legal

Bit 15 - 0, Indicator level illegal
1, Indicator level legal

Bit 16 - 0, Frame level illegal
1, Frame level legal

4. OUTPUT:

If illegal level, error message is typed.

5. ACTION:

The mode indicator (input) is shifted right n times, where n is the current level. If bit 16 is one, after shift, level is correct and a normal return is executed. Otherwise an error message is typed and an error return is made.

6. EXTERNAL REFERENCES:

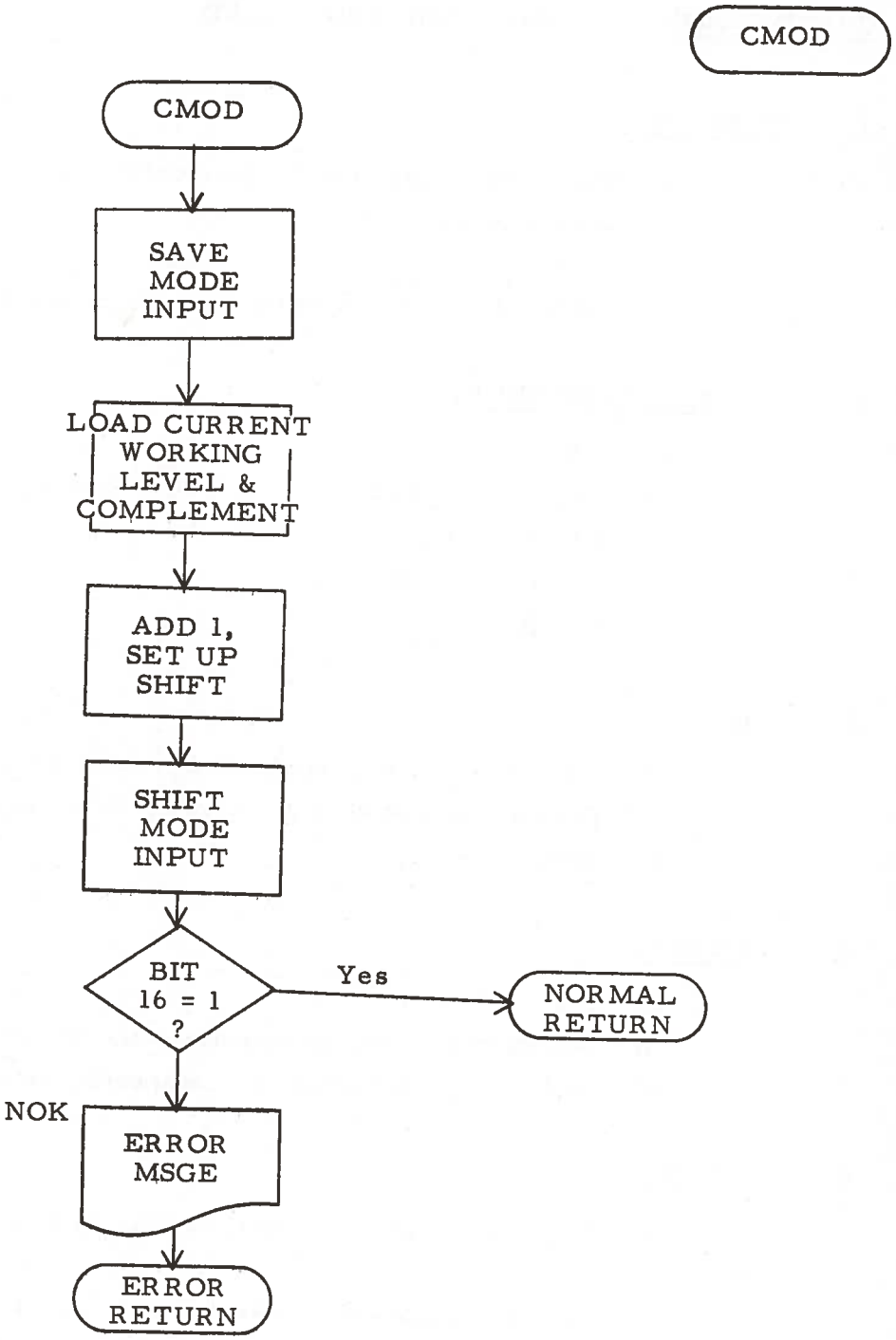
TAO$
CLVL

7. NOTE:

CLVL = 1 - Frame level
2 - Indicator level
4 - Entity level

8. CORE USED:

42_8


CMOD
CMOD
SAVE MODE INPUT
LOAD CURRENT WORKING LEVEL & COMPLEMENT
ADD 1, SET UP SHIFT
SHIFT MODE INPUT
BIT 16 = 1 ?
Yes
NORMAL RETURN
NOK
ERROR MSGE
ERROR RETURN

SUBROUTINE: COP, CIN, CLO, CPAD

1. PURPOSE:

a) Setup for display command insertion into component block
b) Insert display command
c) Reset
d) Address of next available component word

2. CALLING SEQUENCE:

```
a)  CALL    COP
b)  LDA     COMN        / load display command
    CALL    CIN
c)  CALL    CLO
d)  CPAD -
```

3. INPUT:

a) Address of next available component word
b) Display command to be inserted in A-register
c) None

4. OUTPUT:

a) None
b) Display command is inserted into component block
c) Address of next available component word is reset

5. ACTION:

a) External pointer, ACWD, is retrieved and set up internally

b) Display command is inserted into next available component word. Internal pointer is incremented. Component word counter is incremented. If needed, a new component block is created.

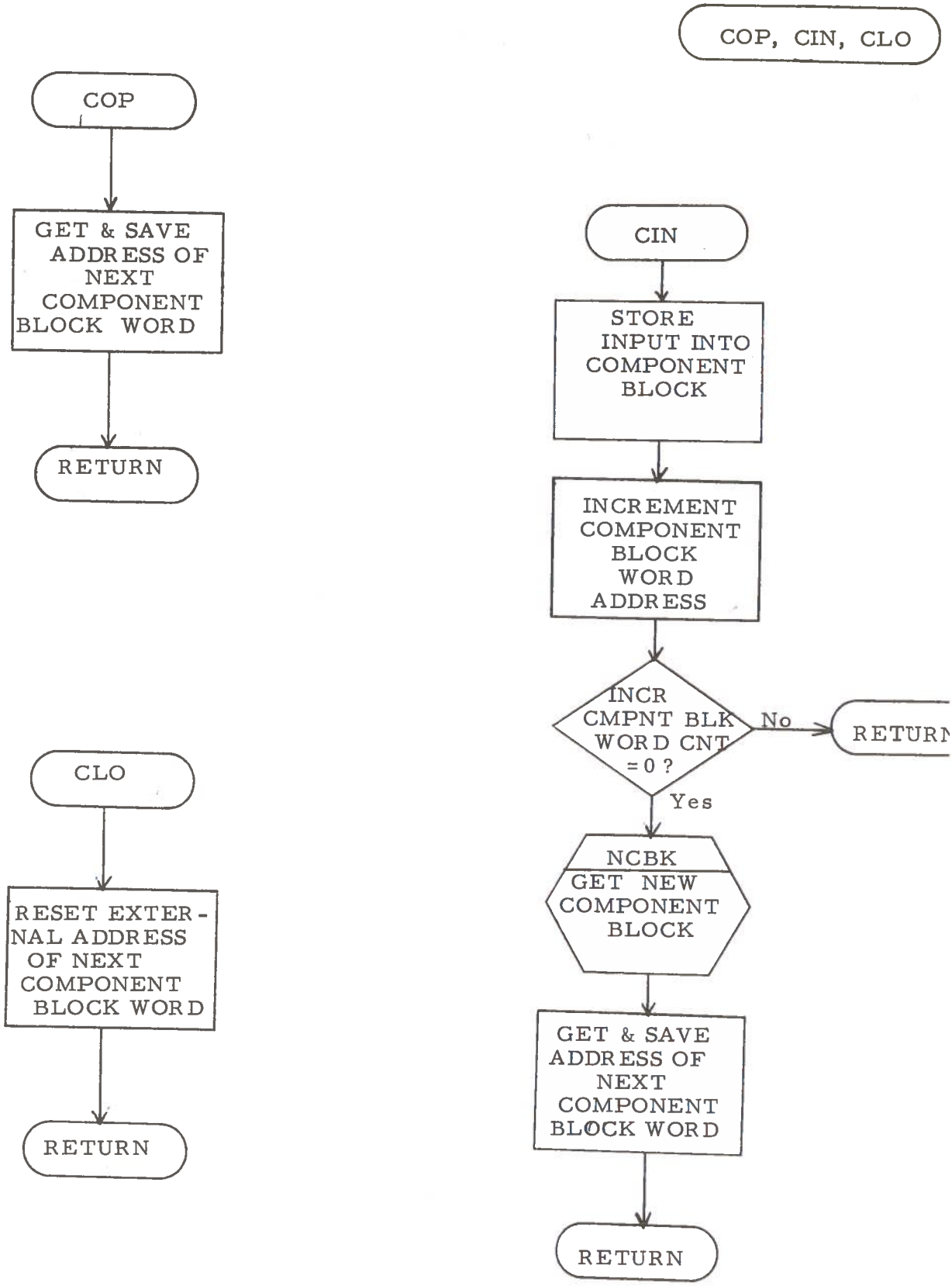
c) External pointer, ACWD, is updated.

6. EXTERNAL REFERENCES:

NCBK	Create new component block
ACWD	Next available component word address
NCWD	Two's complement of words remaining in component block

7. CORE USED:

27_8


COP, CIN, CLO
COP
GET & SAVE ADDRESS OF NEXT COMPONENT BLOCK WORD
RETURN
CIN
STORE INPUT INTO COMPONENT BLOCK
INCREMENT COMPONENT BLOCK WORD ADDRESS
INCR CMPNT BLK WORD CNT = 0 ?
No
RETURN
Yes
NCBK
GET NEW COMPONENT BLOCK
GET & SAVE ADDRESS OF NEXT COMPONENT BLOCK WORD
RETURN
CLO
RESET EXTERNAL ADDRESS OF NEXT COMPONENT BLOCK WORD
RETURN

SUBROUTINE: FNDE

1. PURPOSE:

Locate a particular entity on an indicator

2. CALLING SEQUENCE:

Call FNDE

3. INPUT:

Entity ID in ENID
Indicator ID in INID
Frame ID in FRID

4. OUTPUT:

Starting address of entity block in A; $\emptyset$ in A if block not found.

5. ACTION:

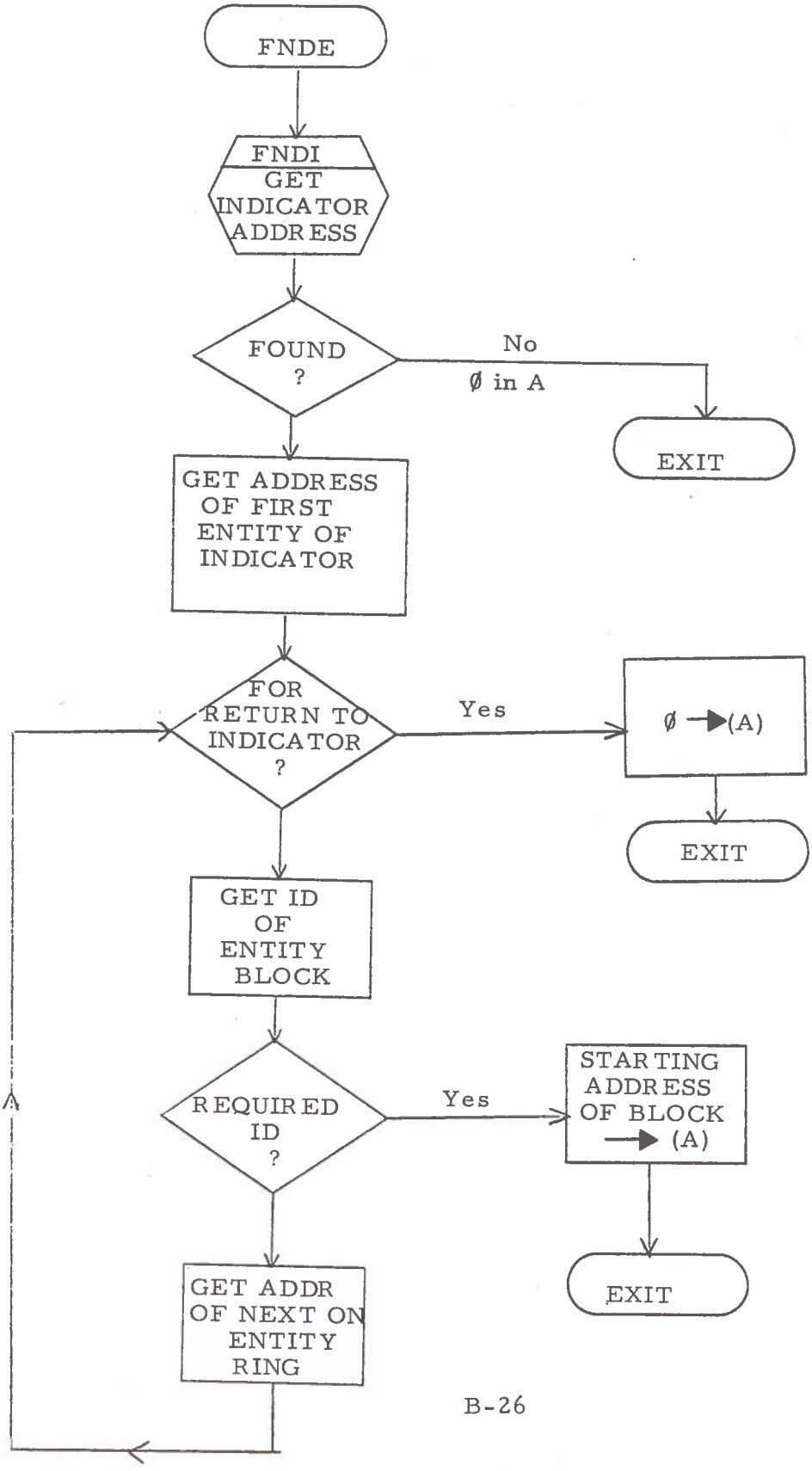
FNDI is called to get the indicator address. The entities of the indicators are searched for one with the given ID. The start address is returned in the A, if the indicator is found; if the indicator is not found then $\emptyset$ is returned in the A.

6. EXTERNAL REFERENCES

ENID, COLN, LEAD

7. CORE USED

55_8


FNDE
FNDI
GET INDICATOR ADDRESS
FOUND ?
No
Ø in A
EXIT
GET ADDRESS OF FIRST ENTITY OF INDICATOR
FOR RETURN TO INDICATOR ?
Yes
Ø → (A)
EXIT
GET ID OF ENTITY BLOCK
REQUIRED ID ?
Yes
STARTING ADDRESS OF BLOCK → (A)
EXIT
GET ADDR OF NEXT ON ENTITY RING

SUBROUTINE: FNDF

1. PURPOSE:

Locate a particular block on the frame ring.

2. CALLING SEQUENCE:

Call FNDF

3. INPUT:

ID of frame in FRID

4. OUTPUT:

Starting address of frame block in A; Ø in A if block not found

5. ACTION:

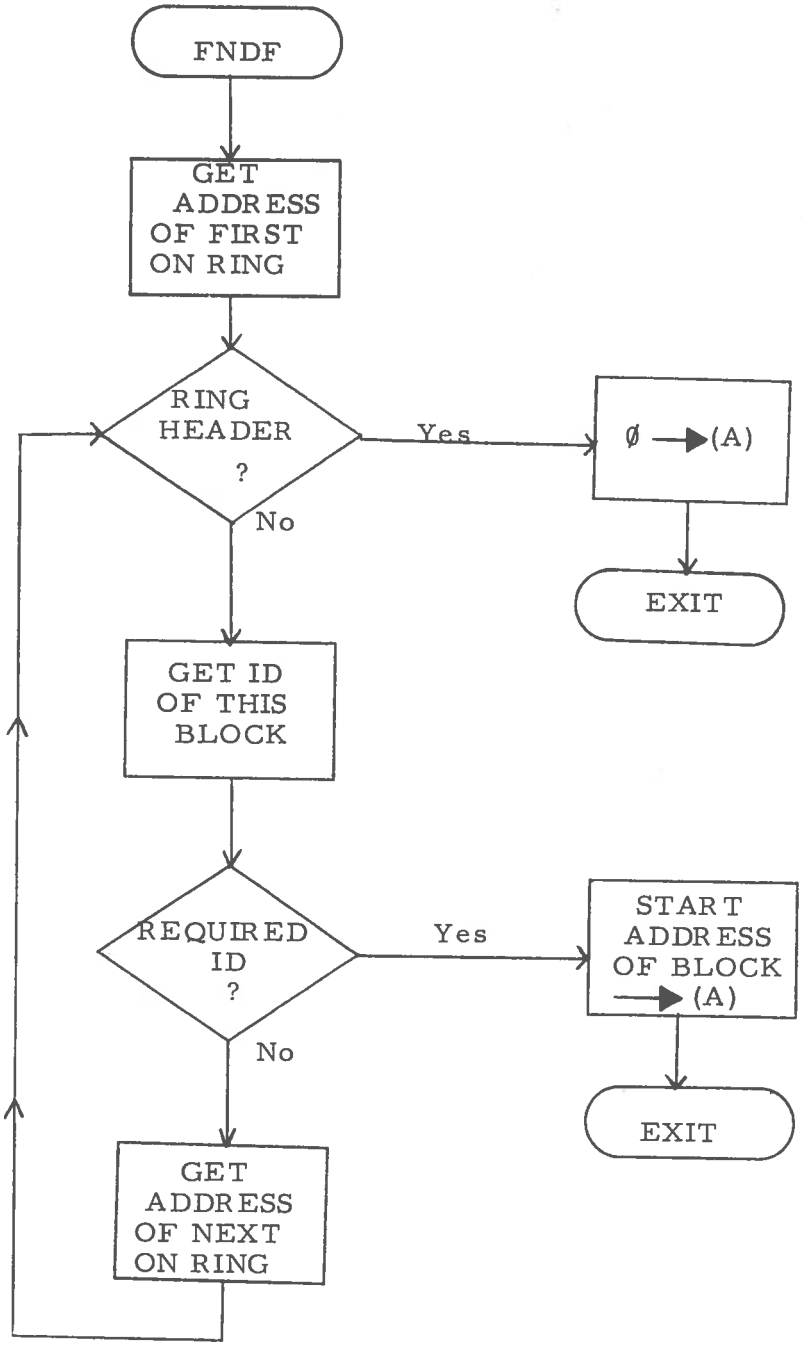
The frame ring is searched for a block with the given ID. If the block is found, the starting address is returned in the A. If the block is not found then Ø is returned in the A register.

6. EXTERNAL REFERENCES:

FRHD, FPNT, FRID, LFAD

7. CORE USED:

40_8


FNDF
GET ADDRESS OF FIRST ON RING
RING HEADER ?
Yes
Ø → (A)
EXIT
No
GET ID OF THIS BLOCK
REQUIRED ID ?
Yes
START ADDRESS OF BLOCK → (A)
EXIT
No
GET ADDRESS OF NEXT ON RING

SUBROUTINE: FNDI

1. PURPOSE:

Locate a particular indicator block on a frame.

2. CALLING SEQUENCE:

Call FNDI

3. INPUT:

Indicator ID in INID

4. OUTPUT:

Starting address of indicator block in A; ϕ in A if block not found.

5. ACTION:

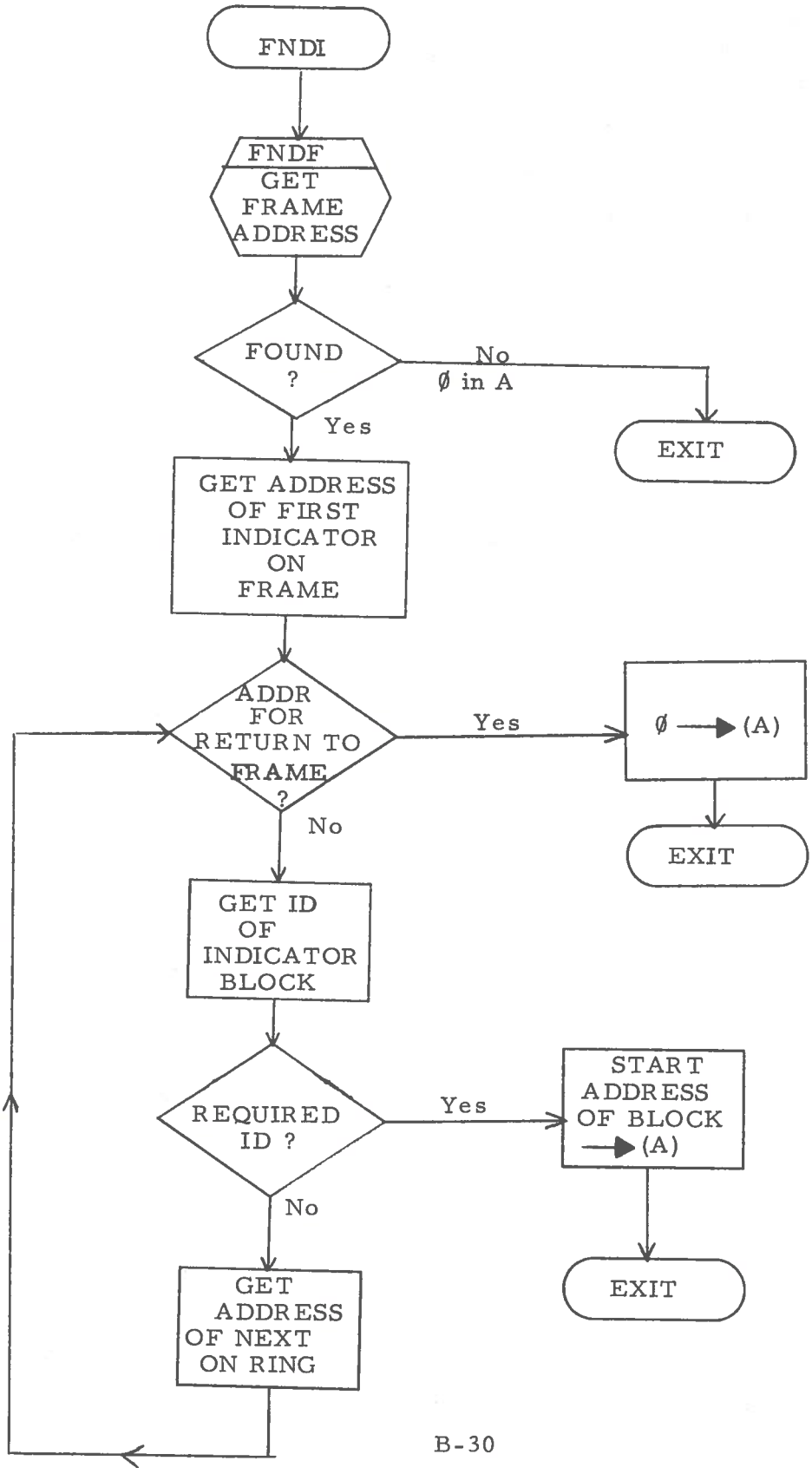
FNDF is called to get the frame address. The indicators of the frame are searched for one with the given ID. If not found ϕ is returned in the AC, otherwise the start address is returned in the AC.

6. EXTERNAL REFERENCES

INID, COLN, LIAD

7 CORE USED:

55_8


FNDI
FNDF
GET
FRAME
ADDRESS
FOUND
?
No
Ø in A
EXIT
Yes
GET ADDRESS
OF FIRST
INDICATOR
ON
FRAME
ADDR
FOR
RETURN TO
FRAME
?
Yes
Ø → (A)
EXIT
No
GET ID
OF
INDICATOR
BLOCK
REQUIRED
ID ?
Yes
START
ADDRESS
OF BLOCK
→ (A)
EXIT
No
GET
ADDRESS
OF NEXT
ON RING

SUBROUTINE GABN, SKER

1. PURPOSE:

Get Registers A, B and number of words to skip and insert into conditional block.

2. CALLING SEQUENCE:

```
a)      LDA     CODE    /load skip code
        CALL    GABN

b)      CALL    SKER
```

3. INPUT:

a) Skip code in A-Reg.;
Reg. A, B and skip number N from input buffer.

b) None

4. OUTPUT:

a) Skip instruction inserted into conditional block.

b) Error message.

5. ACTION:

a) Gets Registers A, B. Pack and save. Get skip number, if less than or equal to zero, output error message and return; otherwise, add to skip instruction. Insert skip command and A, B into conditional block. Exit with normal return.

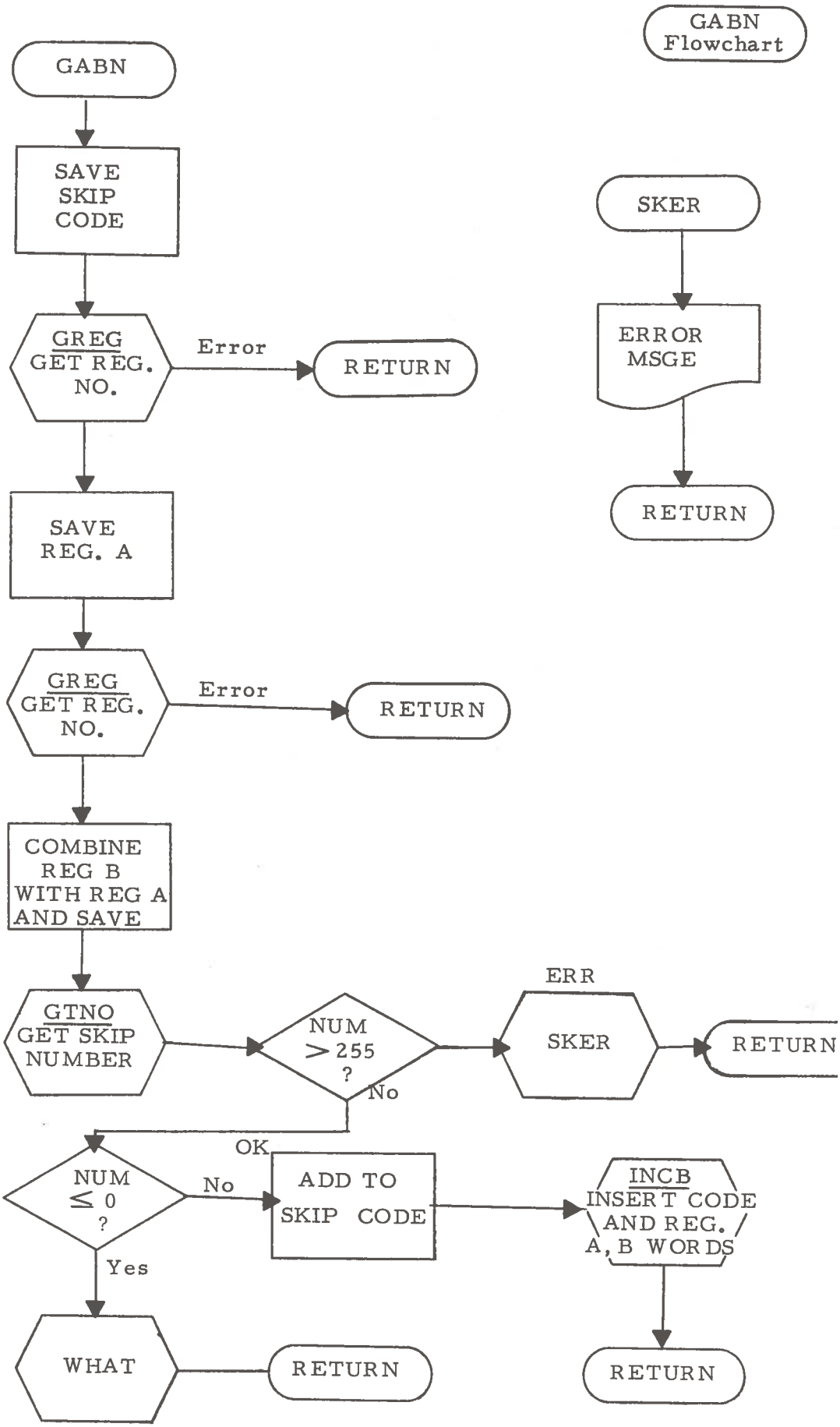
b) Output error message.

6. EXTERNAL REFERENCES:

GREG
GTNO
WHAT
INCB
TAO$

7. CORE USED:

62_8


GABN Flowchart
GABN
SAVE SKIP CODE
GREG GET REG. NO.
Error
RETURN
SAVE REG. A
GREG GET REG. NO.
Error
RETURN
COMBINE REG B WITH REG A AND SAVE
GTNO GET SKIP NUMBER
NUM > 255 ?
ERR
SKER
RETURN
No
OK
NUM ≤ 0 ?
No
ADD TO SKIP CODE
INCB INSERT CODE AND REG. A, B WORDS
Yes
WHAT
RETURN
RETURN
SKER
ERROR MSGE
RETURN

1. PURPOSE:

Get address of next entity on indicator ring. Jump out when ring completed.

2. CALLING SEQUENCE:

```
SSM/SSP                          /initialize or next
CALL      GENT
JMP       OUT                    /ring completed
```

Returns here with entity addr in A-Reg.

3. INPUT:

If bit 1 of A-Reg is set (SSM) get the address of the first entity of ring. If bit 1 of A-Reg is zero the address of the next entity of the ring.

4. OUTPUT:

If no next entity (end of ring) return to calling location + 1. Otherwise returnto calling location + 2 with address of first word of entity block in A-Reg.

5. EXTERNAL REFERENCES:

INAD - INDICATOR ADDRESS

6. CORE USED:

45_8

SUBROUTINE: GETF

1. PURPOSE:

Get blocks of storage from the free ring.

2. CALLING SEQUENCE:

Call GETF

3. INPUT:

A = block size required, or
A = Ø if largest available block is desired

4. OUTPUT:

A = starting address of block, or
A = Ø if out of storage
B = last address of block

5. ACTION:

If no size is specified then the ring is searched for the largest block. The block is detached from the ring and the appropriate output is returned.

If size is specified then the ring is searched for a block of the specified length or the smallest block greater in length if only a larger block is found then the unneeded portion is retained on the ring.

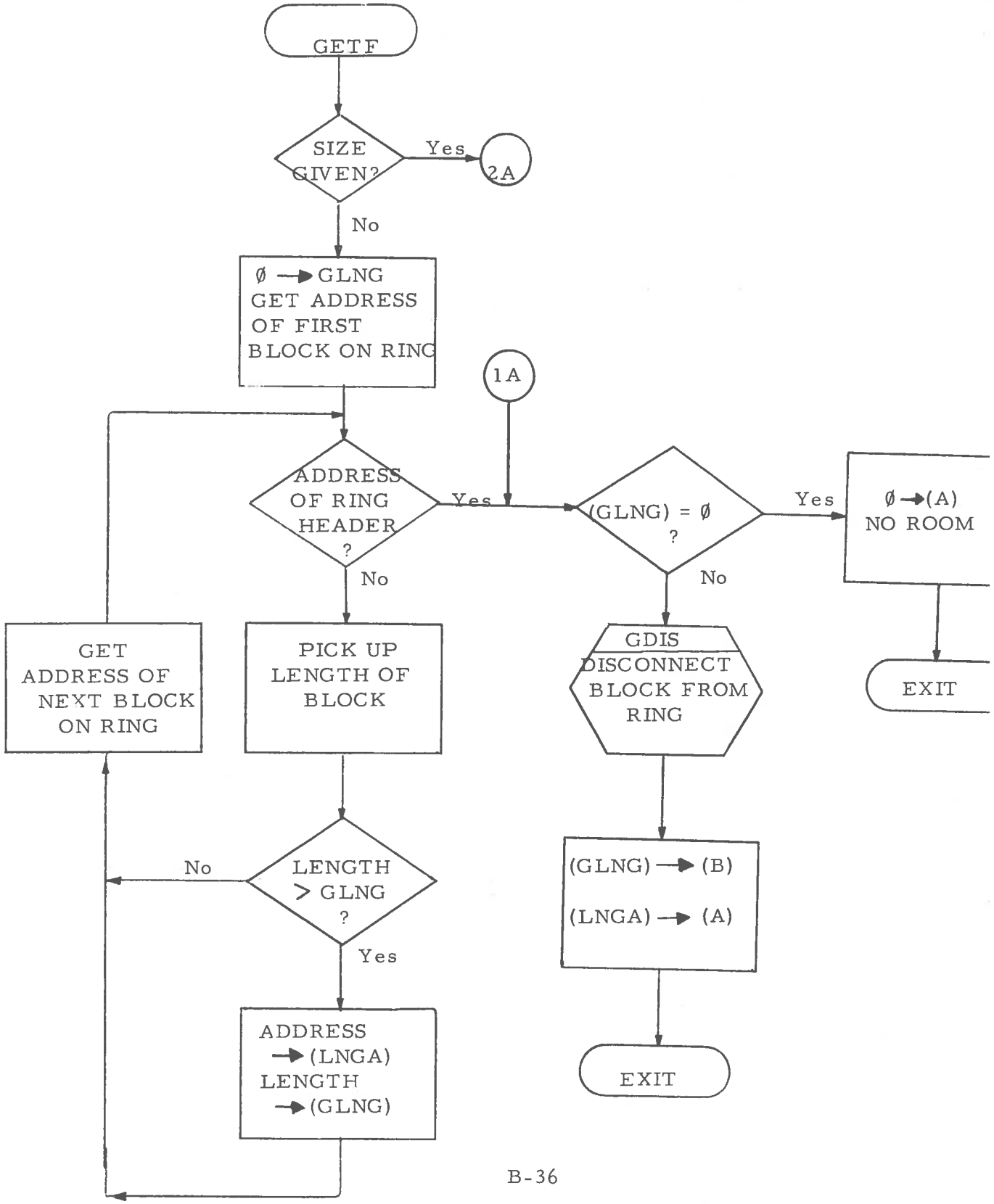
6. EXTERNAL REFERENCES:

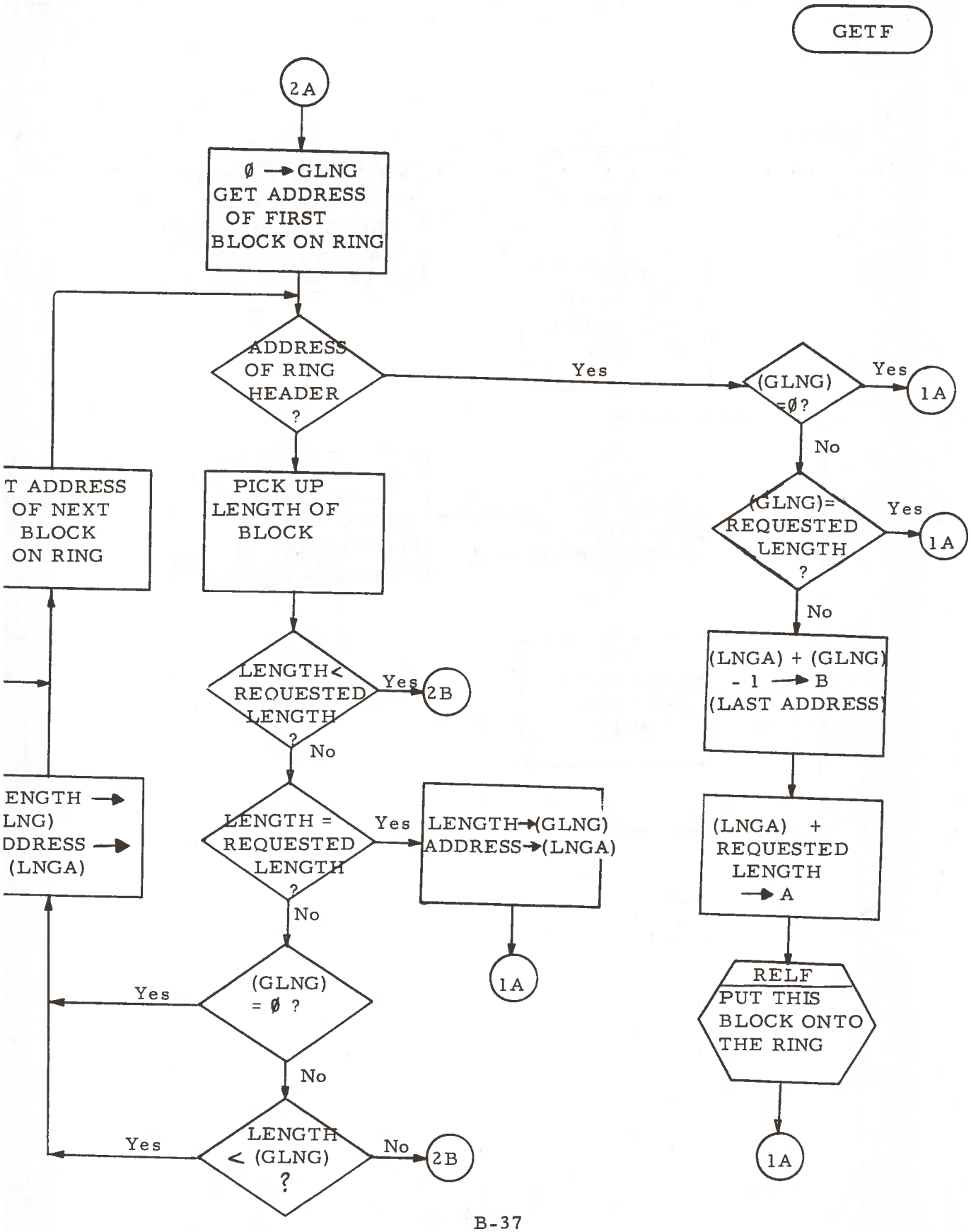
FREE, COLN

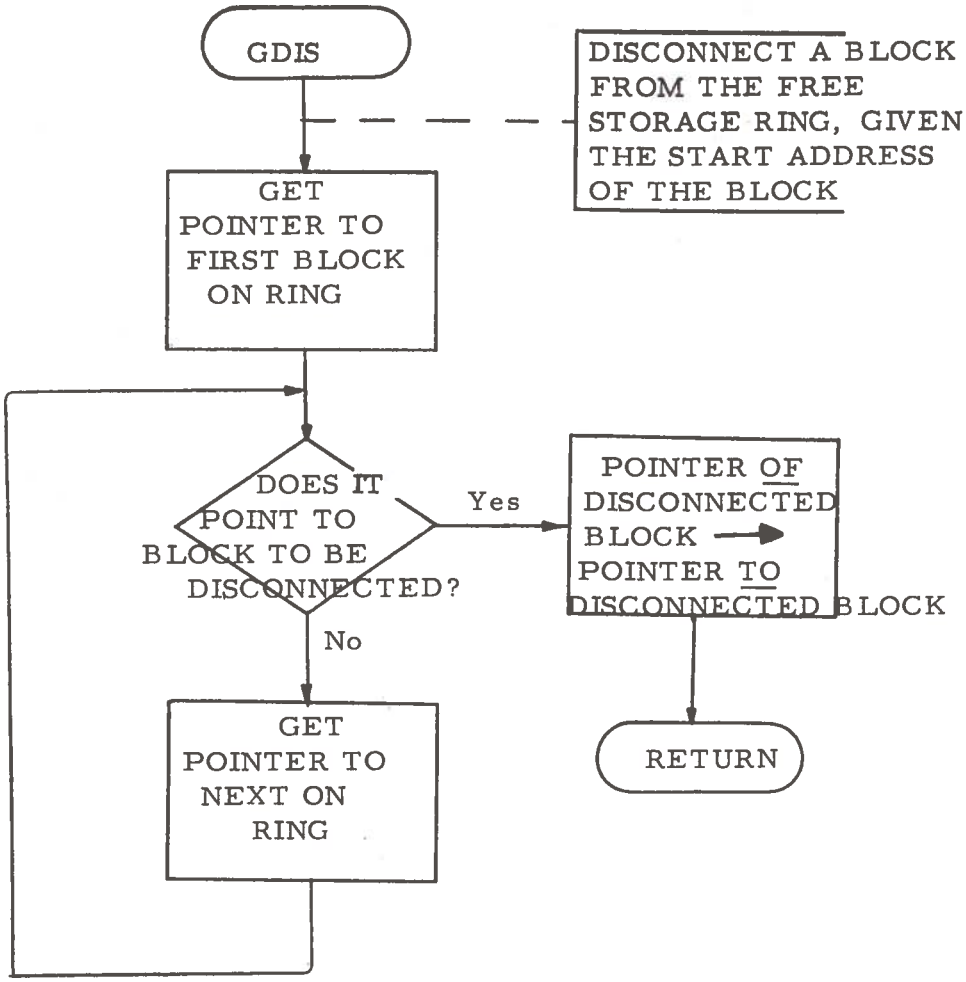
7. CORE USED:

154_8

GETF: Get Free Storage Blocks, given size of block.
If size not given, get largest.




GETF
2A
Ø → GLNG
GET ADDRESS
OF FIRST
BLOCK ON RING
ADDRESS
OF RING
HEADER
?
Yes
(GLNG)
=Ø?
Yes
1A
No
(GLNG)=
REQUESTED
LENGTH
?
Yes
1A
No
(LNGA) + (GLNG)
- 1 → B
(LAST ADDRESS)
(LNGA) +
REQUESTED
LENGTH
→ A
RELF
PUT THIS
BLOCK ONTO
THE RING
1A
T ADDRESS
OF NEXT
BLOCK
ON RING
PICK UP
LENGTH OF
BLOCK
LENGTH<
REQUESTED
LENGTH
?
Yes
2B
No
ENGTH →
LNG)
DDRESS →
(LNGA)
LENGTH =
REQUESTED
LENGTH
?
Yes
LENGTH→(GLNG)
ADDRESS→(LNGA)
1A
No
(GLNG)
= Ø ?
Yes
No
LENGTH
< (GLNG)
?
Yes
No
2B


GDIS
DISCONNECT A BLOCK FROM THE FREE STORAGE RING, GIVEN THE START ADDRESS OF THE BLOCK
GET POINTER TO FIRST BLOCK ON RING
DOES IT POINT TO BLOCK TO BE DISCONNECTED?
Yes
POINTER OF DISCONNECTED BLOCK → POINTER TO DISCONNECTED BLOCK
No
GET POINTER TO NEXT ON RING
RETURN

MEXP

1. PURPOSE:

Move characters into conditioning block

2. CALLING SEQUENCE:

CALL MEXP

3. INPUT:

Character string from input buffer

4. OUTPUT:

Character string in conditioning block;
A-Reg = 1 if null word added; = 0 no null word added

5. ACTION:

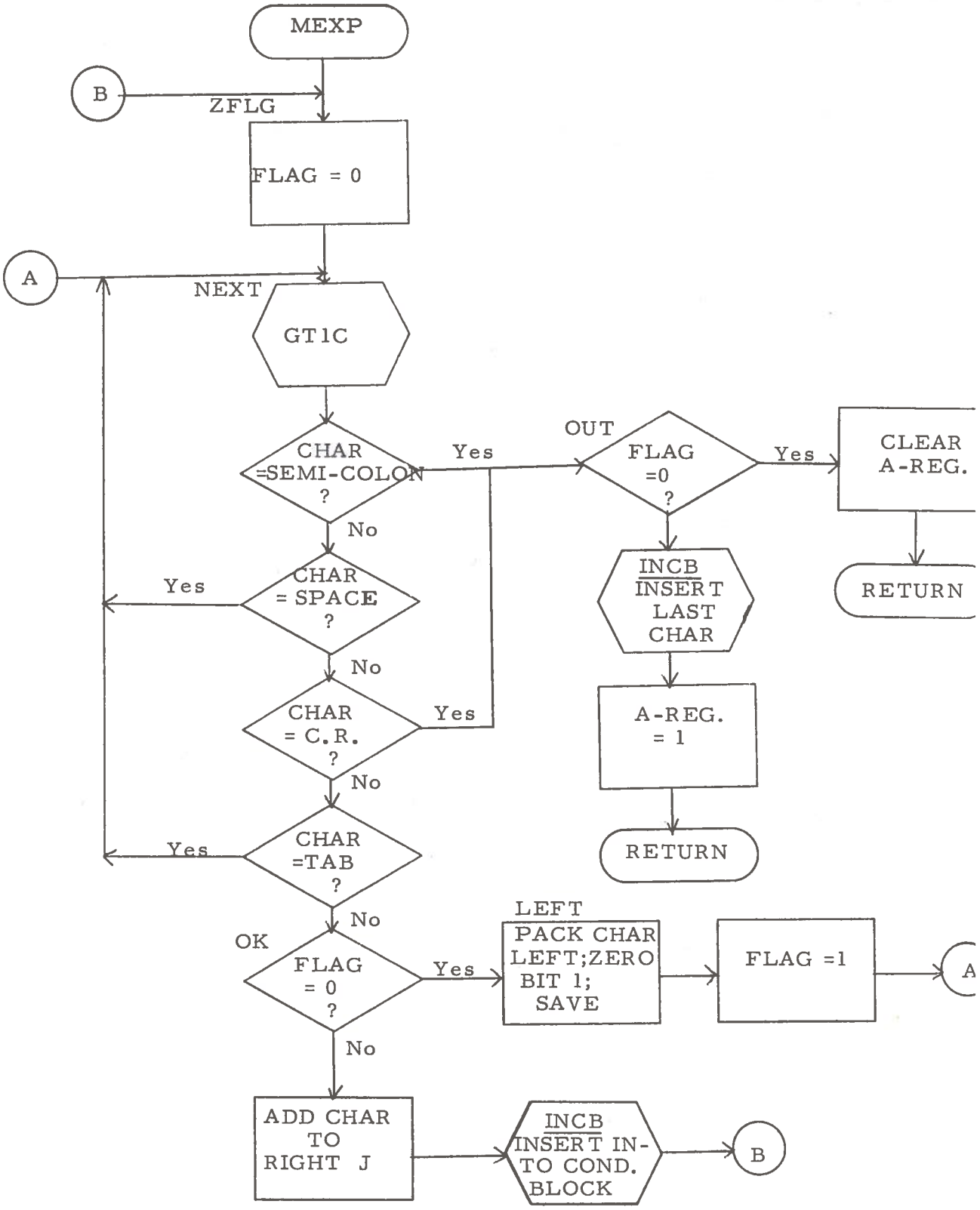
Retrieves chars. and packs, spaces and tabs are ignored. Chars. are inserted into conditioning block, two at a time. Move is terminated when a semicolon or a carriage return is encountered.

6. EXTERNAL REFERENCES:

GT1C
INCB

7. CORE USED:

54_8


MEXP
B
ZFLG
FLAG = 0
A
NEXT
GT1C
CHAR =SEMI-COLON ?
Yes
OUT
FLAG =0 ?
Yes
CLEAR A-REG.
RETURN
No
CHAR = SPACE ?
Yes
INCB INSERT LAST CHAR
A-REG. = 1
RETURN
No
CHAR = C.R. ?
Yes
No
CHAR =TAB ?
Yes
No
OK
FLAG = 0 ?
Yes
LEFT
PACK CHAR LEFT;ZERO BIT 1; SAVE
FLAG =1
A
No
ADD CHAR TO RIGHT J
INCB INSERT IN-TO COND. BLOCK
B

SUBROUTINE OCBK, CCBK, NCBK, RCBK

1. PURPOSE

Open, close, create (continue) and reopen component blocks.

2. CALLING SEQUENCE:

a. Call OCBK
b. Call CCBK
c. Call NCBK
d. Call RCBK

3. INPUT

Free storage blocks

4. OUTPUT

Entity component blocks

5. ACTION

a) The component blocks of the current entity will be traced to find the end of the components. (location with return to entity). This (location) will be saved. A free block will be retrieved from free storage. The address of this new block will be saved by the program and also placed in the system variable ACWD. The number of words in the new block minus four will be two's complemented and saved in system variable NCWD.

b) CCBK - The component block of the current entity will be closed in the following manner. If ACWD contains the address +2 as the saved address of the new block - SAB, then it is assumed that no components were inserted. Then the new block is returned to free storage, and the CCBK exits to the calling routine.

If the addresses are different, then a component has been inserted. The return command is inserted into the next available location of the new block. Any portion of the new block still unused is returned to free storage, and CCBK exits to the calling routine.

c) NCBK - NCBK inserts the return to entity command in the last location of the new block and updates the word saving the return to entity command location. Next a block from free storage is retrieved, and the address to its first available word is saved by NCBK (same word as in OCBK) in ACWD. The number of words in the new block minus four is two's complemented and saved in NCWD. The new block is filled with noops and linked to both the previous componen block and the entity block. Then NCBK exists to the calling routine.

d) RCBK - RCBK reopens a component block for a previously existing entity. The address of the last word of the previous block is saved and control transfers to NCB4 in subroutine NCBK.

6. EXTERNALS REFERENCED

ACWD - Address of next available component word

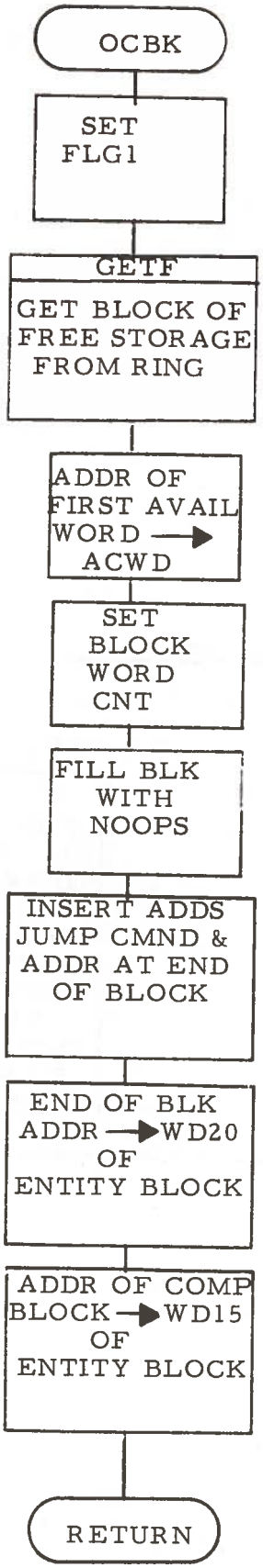
NCWD - Counter of words left in new component block

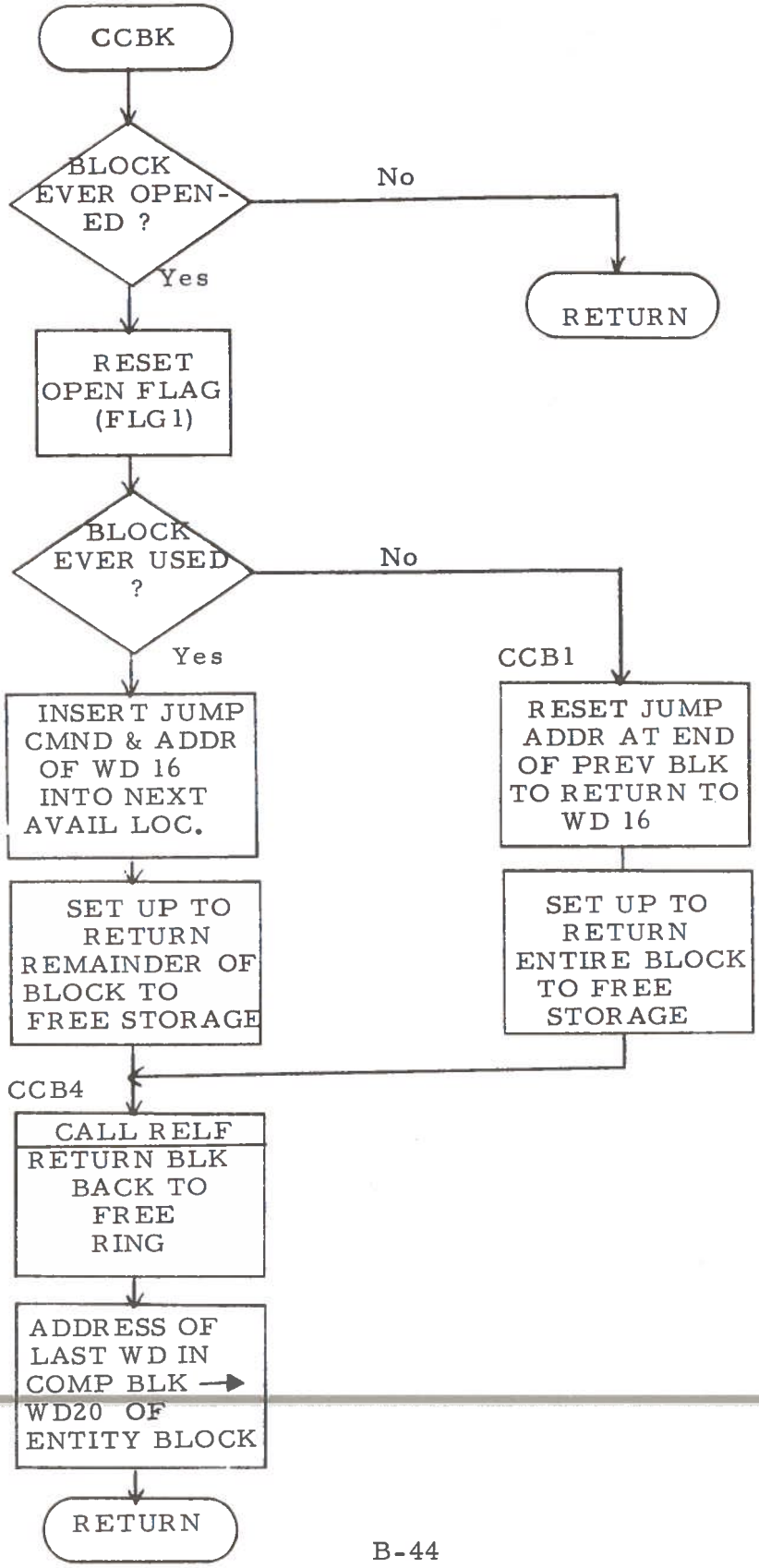
ENAD - Entity address

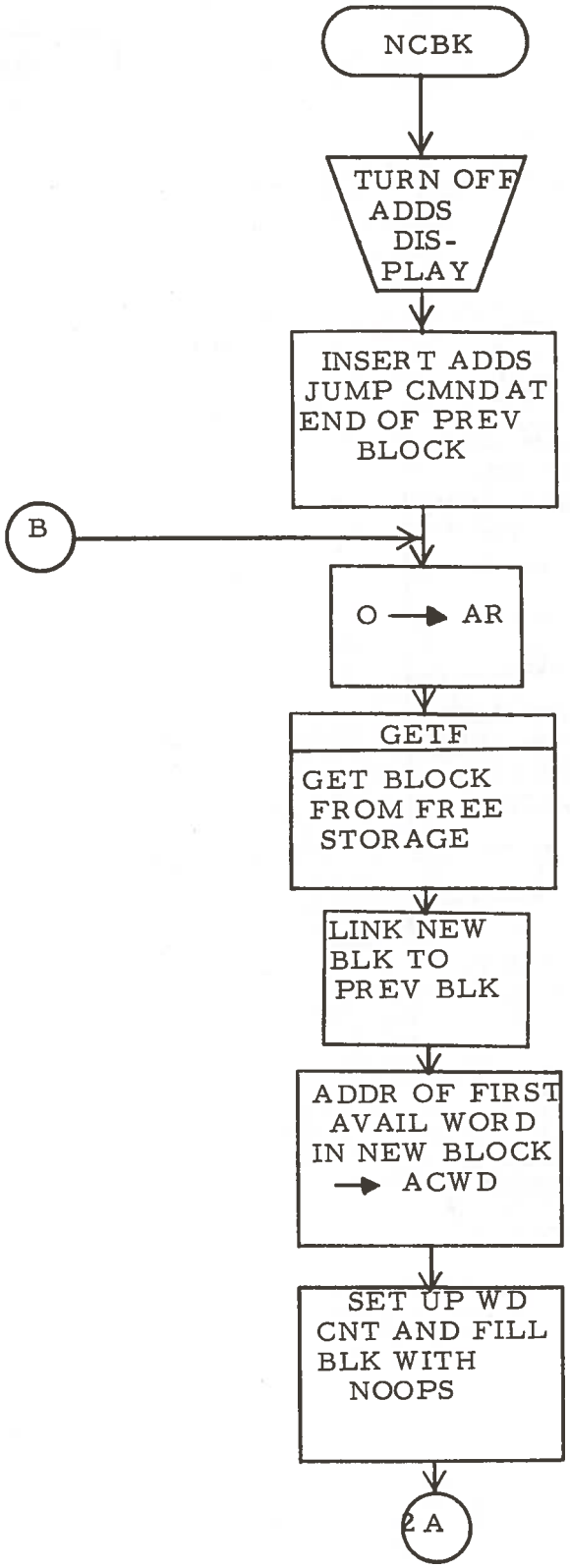
FLG1 - Component block flag (set if open)

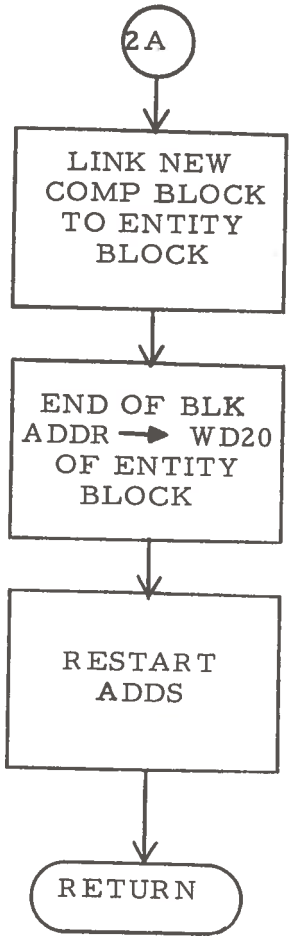
7. CORE USED

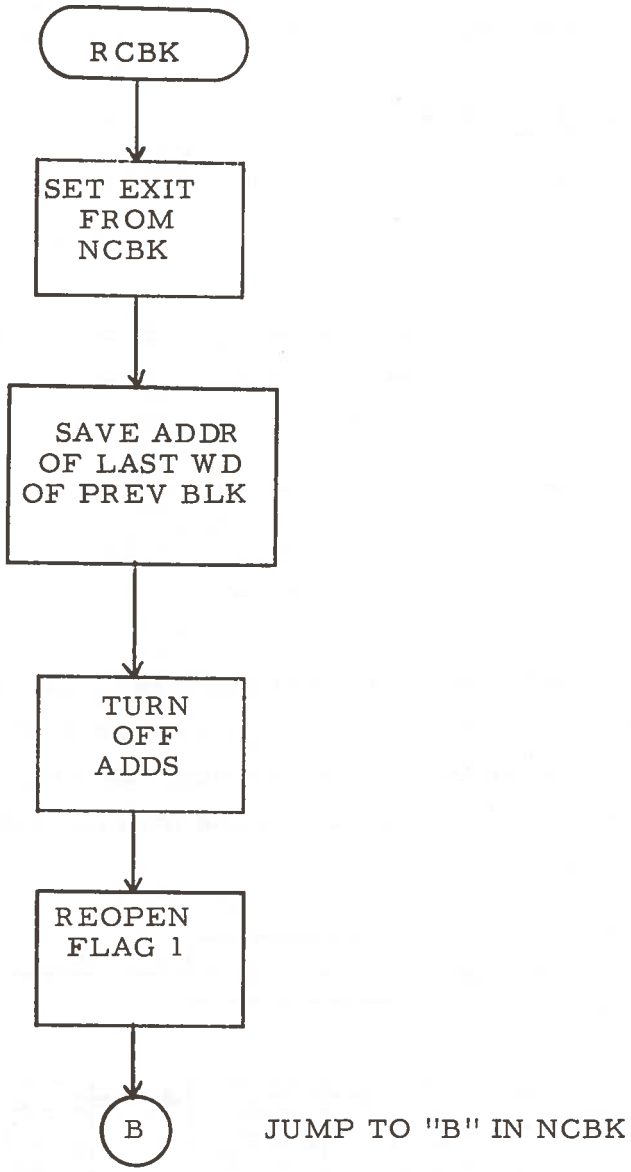
230_8


OCBK
SET FLG1
GETF
GET BLOCK OF FREE STORAGE FROM RING
ADDR OF FIRST AVAIL WORD → ACWD
SET BLOCK WORD CNT
FILL BLK WITH NOOPS
INSERT ADDS JUMP CMND & ADDR AT END OF BLOCK
END OF BLK ADDR → WD20 OF ENTITY BLOCK
ADDR OF COMP BLOCK → WD15 OF ENTITY BLOCK
RETURN


CCBK
BLOCK EVER OPEN-ED ?
No
Yes
RETURN
RESET OPEN FLAG (FLG1)
BLOCK EVER USED ?
No
Yes
CCB1
INSERT JUMP CMND & ADDR OF WD 16 INTO NEXT AVAIL LOC.
RESET JUMP ADDR AT END OF PREV BLK TO RETURN TO WD 16
SET UP TO RETURN REMAINDER OF BLOCK TO FREE STORAGE
SET UP TO RETURN ENTIRE BLOCK TO FREE STORAGE
CCB4
CALL RELF
RETURN BLK BACK TO FREE RING
ADDRESS OF LAST WD IN COMP BLK → WD20 OF ENTITY BLOCK
RETURN


NCBK
TURN OFF ADDS DIS-PLAY
INSERT ADDS JUMP CMND AT END OF PREV BLOCK
B
0 → AR
GETF
GET BLOCK FROM FREE STORAGE
LINK NEW BLK TO PREV BLK
ADDR OF FIRST AVAIL WORD IN NEW BLOCK → ACWD
SET UP WD CNT AND FILL BLK WITH NOOPS
2 A


2A
LINK NEW COMP BLOCK TO ENTITY BLOCK
END OF BLK ADDR → WD20 OF ENTITY BLOCK
RESTART ADDS
RETURN


RCBK
SET EXIT FROM NCBK
SAVE ADDR OF LAST WD OF PREV BLK
TURN OFF ADDS
REOPEN FLAG 1
B
JUMP TO "B" IN NCBK

SUBROUTINE: RELF

1. PURPOSE:

Release blocks of storage no longer needed back to the free ring.

2. CALLING SEQUENCE:

Call RELF

3. INPUT:

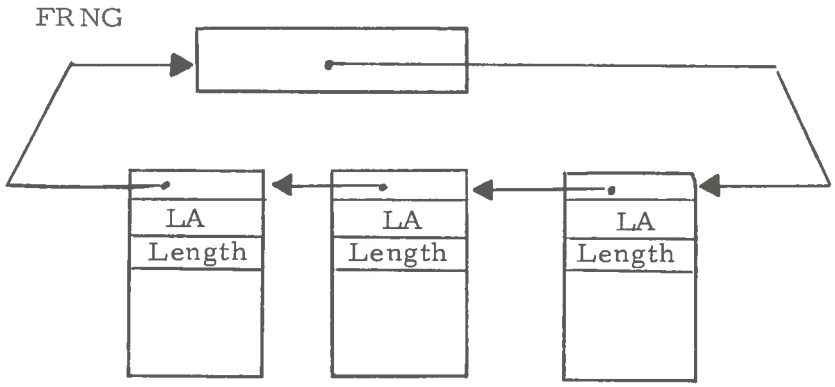
A = first address of block to be released
B = last address of block to be released

4. OUTPUT:

None

5. ACTION:

The released block is attached to the beginning of the free ring. The contents of the first word of the block contain the pointer to the next on the ring; the second word contains the last address of the block and the third word the length of the block.

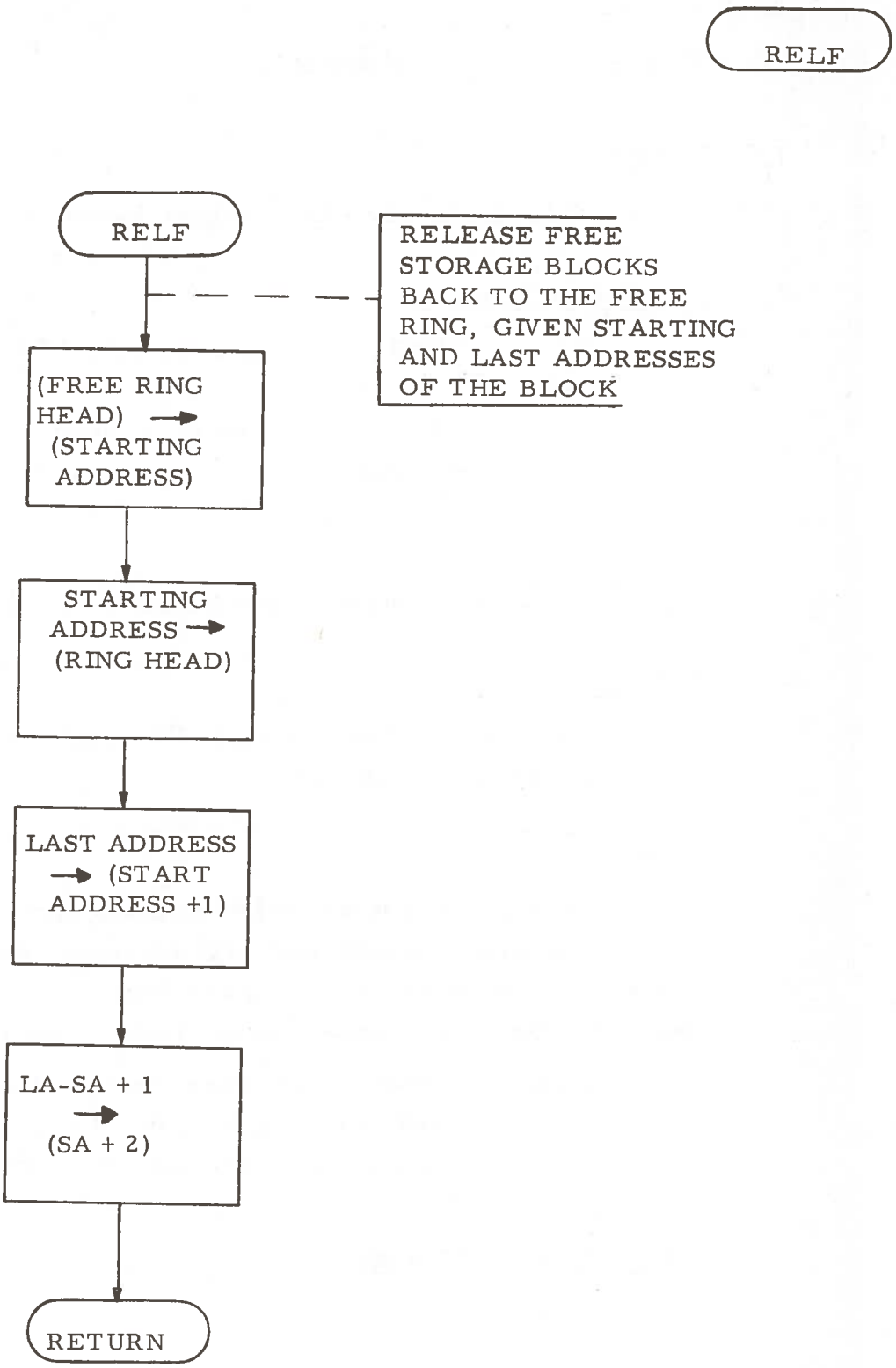


6. EXTERNAL REFERENCES:

None

7. CORE USED:

42_8


RELF
RELF
RELEASE FREE STORAGE BLOCKS BACK TO THE FREE RING, GIVEN STARTING AND LAST ADDRESSES OF THE BLOCK
(FREE RING HEAD) → (STARTING ADDRESS)
STARTING ADDRESS → (RING HEAD)
LAST ADDRESS → (START ADDRESS +1)
LA-SA + 1 → (SA + 2)
RETURN

SUBROUTINE: SQRJAM

1. PURPOSE:

Calculate square root of integer double precision

2. CALLING SEQUENCE:

```
LDA   LARG    /Load least SIGNIF Bits
IAB
LDA   MARG    /Load most SIGNIF Bits
CALL  SQRJAM
```

3. INPUT:

DDP-516 double precision integer in A and B-Registers

4. OUTPUT:

Square root of input rounded off to single precision integer (in A-Register)

5. ACTION:

Converts input to a positive integer if necessary. Zeros root, then squares root and checks against input. Root is incremented, squared, and compared until its square either exceeds input or equals input. If square of root equals inputs, root is returned.

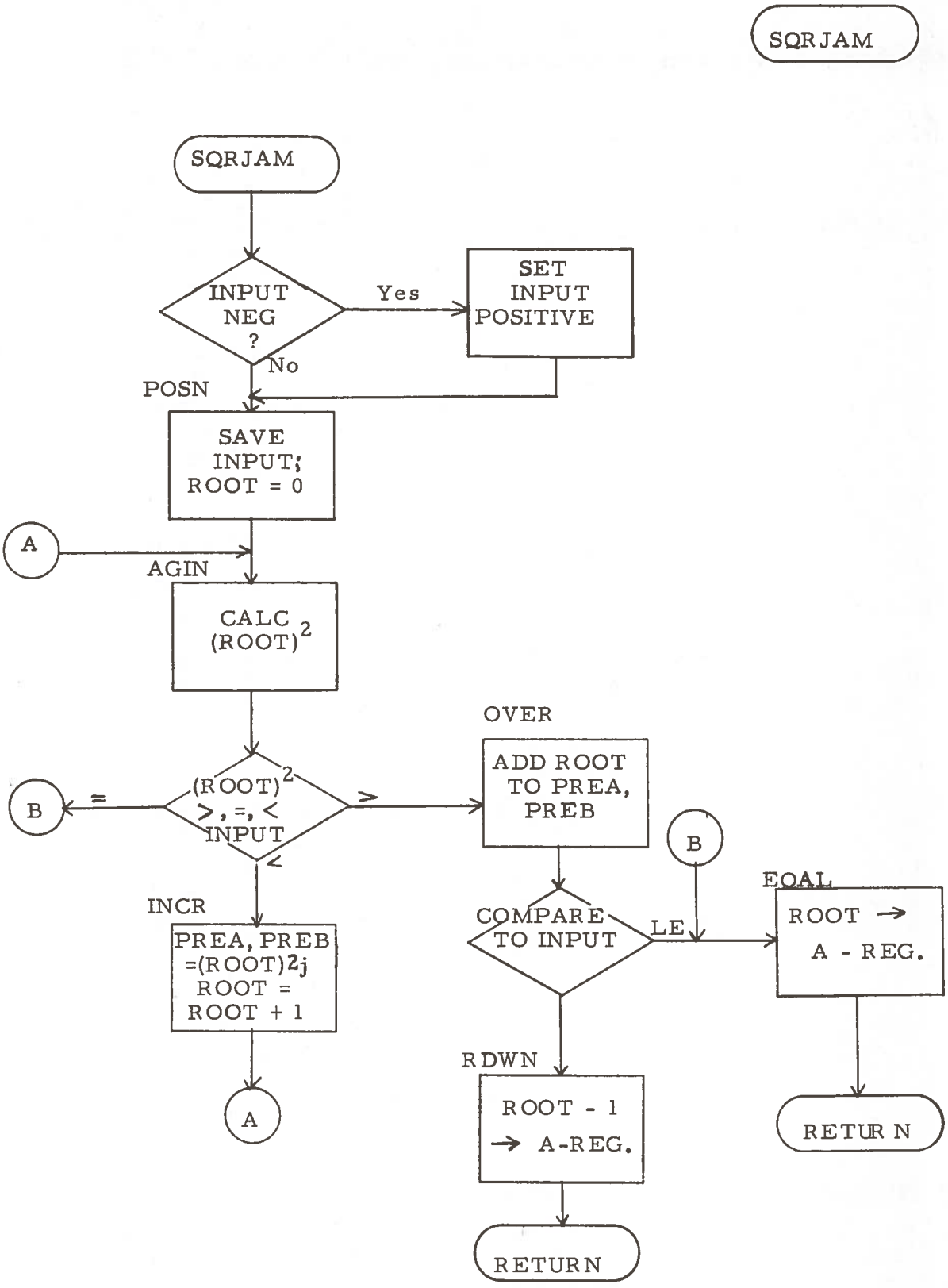
If square of root exceeds input, root is added to the square of the previous root and compared to input. If this sum is greater than the input, root minus one is returned. Otherwise, root is returned.

6. EXTERNAL REFERENCES:

None

7 CORE USED:

64_8


SQRJAM
SQRJAM
INPUT NEG ?
Yes
SET INPUT POSITIVE
No
POSN
SAVE INPUT; ROOT = 0
A
AGIN
CALC (ROOT)2
OVER
ADD ROOT TO PREA, PREB
B
=
(ROOT)2 >, =, < INPUT
>
<
B
INCR
PREA, PREB =(ROOT)2; ROOT = ROOT + 1
COMPARE TO INPUT
LE
EQAL
ROOT → A - REG.
A
RDWN
ROOT - 1 → A-REG.
RETURN
RETURN

SUBROUTINE: **ALPA**

1. PURPOSE:

Sets up character mode and size commands. Reads text from input buffer and inserts appropriate display commands into component block.

2. CALLING SEQUENCE:

```
            JST*        PTR
             :
PTR         DAC         COML+N
             :
COML+N      XAC         ALPA
```

3. INPUT:

Size (1-4) from input buffer. Smallest=1; largest=4.
Text from input buffer.

4. OUTPUT:

Display commands in component file.

5. ACTION:

Checks for entity working level. Gets size from input buffer and sets up size command. Checks terminating character of size number for comma, space, or tab. Retrieves text from input buffer. Packs and inserts characters into component block. Returns to calling program when a carriage return is encountered.

6. EXTERNAL REFERENCES:

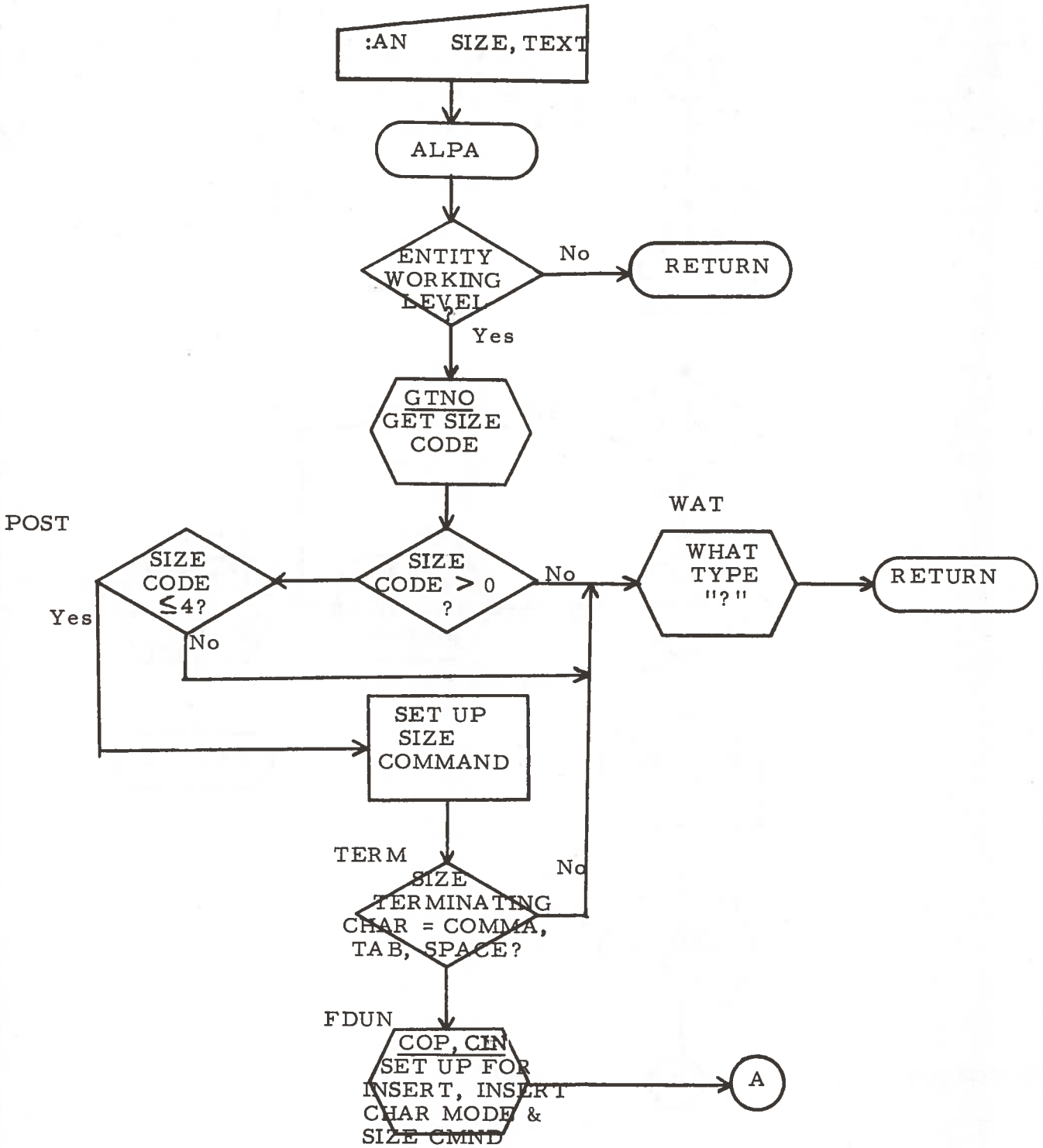
CMOD
GTNO
WHAT
SBLK
COP
CIN
GTCH
CLO

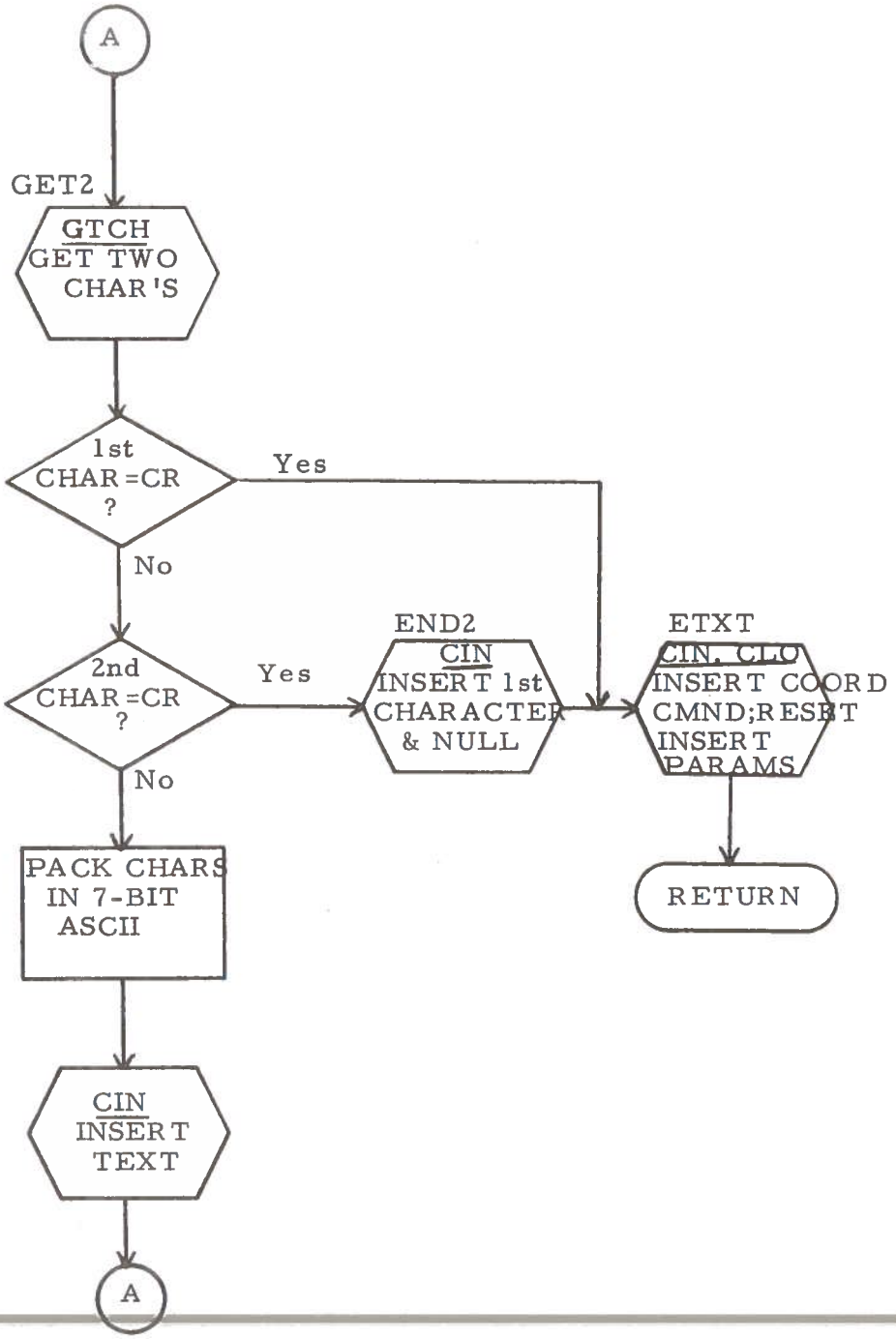
7. CORE USED:

114_8

ALPA FLOWCHART

DRAW ALPHANUMERICS




A
GET2
GTCH
GET TWO
CHAR'S
1st
CHAR=CR
?
Yes
No
2nd
CHAR=CR
?
Yes
END2
CIN
INSERT 1st
CHARACTER
& NULL
ETXT
CIN, CLO
INSERT COORD
CMND;RESET
INSERT
PARAMS
No
PACK CHARS
IN 7-BIT
ASCII
RETURN
CIN
INSERT
TEXT
A

SUBROUTINE: ARC

1. PURPOSE:

Set up component commands to draw arc given relative start point and radius, (Δx, Δy) and angle of arc, Θ.

2. CALLING SEQUENCE:

```
                JST*      PTR
                 :
PTR             DAC       COML+N
                 :
COML+N          XAC       ARC
```

3. INPUT:

From the input buffers:

a) Δx, x-component of start point, radius vector
b) Δy, y-component of start point, radius vector
c) Θ, angle of arc

NOTE: If Δy, Θ not input, Δx is assumed to be the radius and a full circle is drawn.

4. OUTPUT:

Display commands are inserted into the current component block.

5. ACTION:

Input will be checked to determine size and angle of arc. Arcs will be drawn clockwise starting at the point (Δx, Δy) relative to initial position (center of arc).

An Arc will be defined as a portion of a 60-sided regular polygon, one side for each 6^{o}. Arcs will be drawn to the nearest 6^{o} so that each arc will be an integral number of sides.* Δx, Δy will specify the radius of the polygon's circumscribed circle. Figures below show the relationship of the portions of the arc to each other.

* No arc will be drawn for 2^{o} or less.

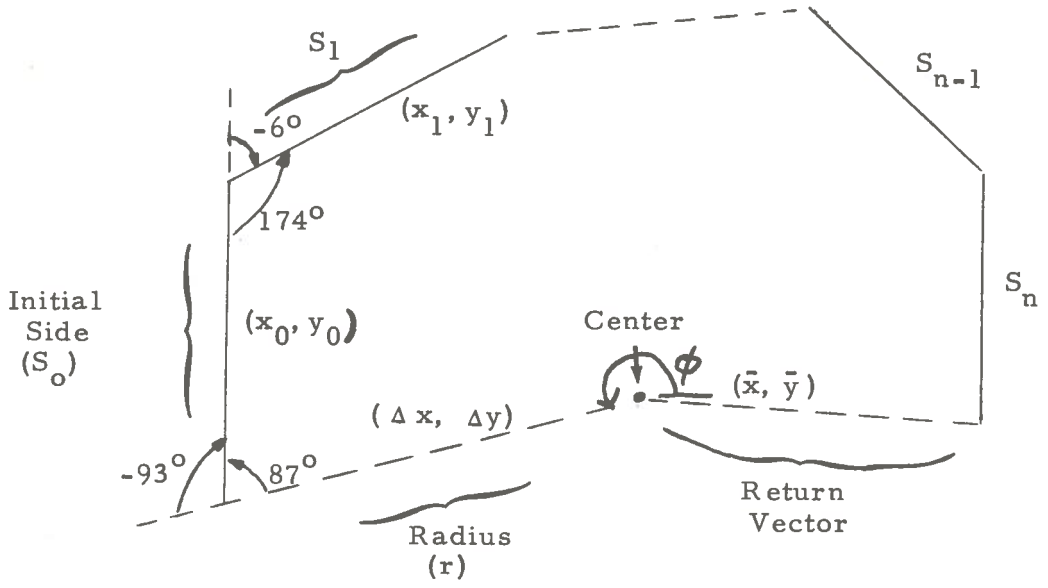
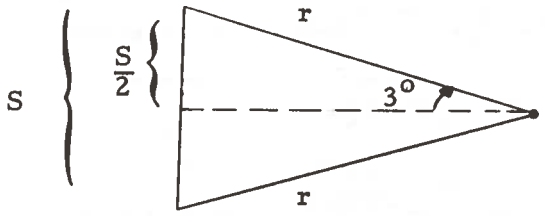


FIGURE 1.

Angular Relationship of Sides and Given Radius



$$\frac{S}{2} = r \sin 3^{o}$$

$$S = 2r \sin 3^{o}$$

FIGURE 2.

Side, Radius Length Relationship

COMPUTATION OF $(X, Y)_{15n+1}$ $(n = 0, 1, 2, 3)$

XI, YI are computed for the first chord of each quarter-circle as functions of the radius components, DELX, DELY, in raster units*16.

$$XI = [DELX*\sin(-3^{o})+DELY*\cos 3^{o} *2 \sin 3^{o}] \quad (1a)$$

$$YI = [DELY*\sin(-3^{o})-DELX*\cos 3^{o} *2 \sin 3^{o}] \quad (1b)$$

PROGRAM:

$$XI = \frac{\frac{-1715*DELX+32723*DELY}{32768} *3430+16384}{32768}$$

$$YI = \frac{\frac{-32723*DELX-1715*DELY}{32768} *3430+16384}{32768}$$

Where 16384 is the round-off summand for the double-word numerators.

COMPUTATION OF $(X, Y)_{15n+j}$ $(n = 0, \ldots, 3; j = 2, \ldots, 14)$

For the second through the fourteenth chords:

$$XI' = XI*\cos 6^{o} + YI*\sin 6^{o} \quad (2a)$$

$$YI' = XI*\sin(-6^{o}) + YI*\cos 6^{o} \quad (2b)$$

PROGRAM:

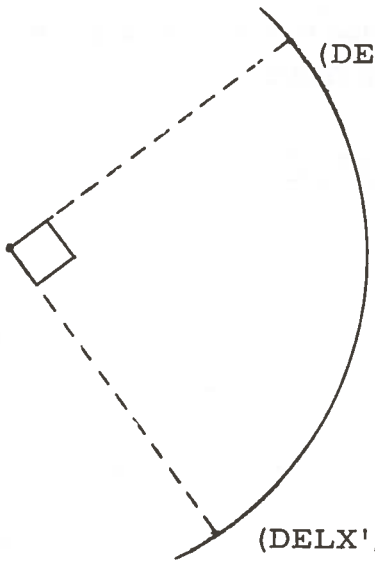
$$XI' = \frac{32588*XI+3425*YI+16384}{32768}$$

$$YI' = \frac{-3425*XI+32588*YI+16384}{32768}$$

Where XI', YI' replace XI, YI for subsequent computation.

COMPUTATION OF $(X, Y)_{15n\ (n = 1, 2, 3, 4)}$

The last chord for a quarter-circle is computed to draw a vector from the end of the previous (14th, 29th, 44th or 59th) chord to the end of the radius vector resulting from a 90^{o} clockwise transformation of the radius to the first point on the quarter-circle.



X, Y components of beam movement are summed as XBAR, YBAR, hence, we wish to compute $(X, Y)_{15n} = XI, YI$ such that

$$16\ XBAR_{14} + XI_{15} = DELY, \text{ and}$$

$$16\ YBAR_{14} + YI_{15} = -DELX$$

So, we set:

$$XI_{15} = DELY - 16\ XBAR, \text{ and} \qquad (3a)$$

$$YI_{15} = -DELX - 16YBAR \qquad (3b)$$

The transformation is applied to DELX, DELY for the next quadrant:

$$DELX' = DELY,\ DELY' = -DELX$$

We note that the updated values of XBAR and YBAR are computed properly:

$$XBAR_{15} = XBAR_{14} + (DELY - 16\ XBAR_{14})\ /16 = DELY/16$$

$$YBAR_{15} = YBAR_{14} + (-DELX - 16\ YBAR_{14})\ /16 = DELX/16$$

Last we note that the first chord of each quarter-circle cannot be computed by (2a, b) due to the propagated errors of precision - the errors which (3a, b) eliminate.

6. EXTERNAL REFERENCES

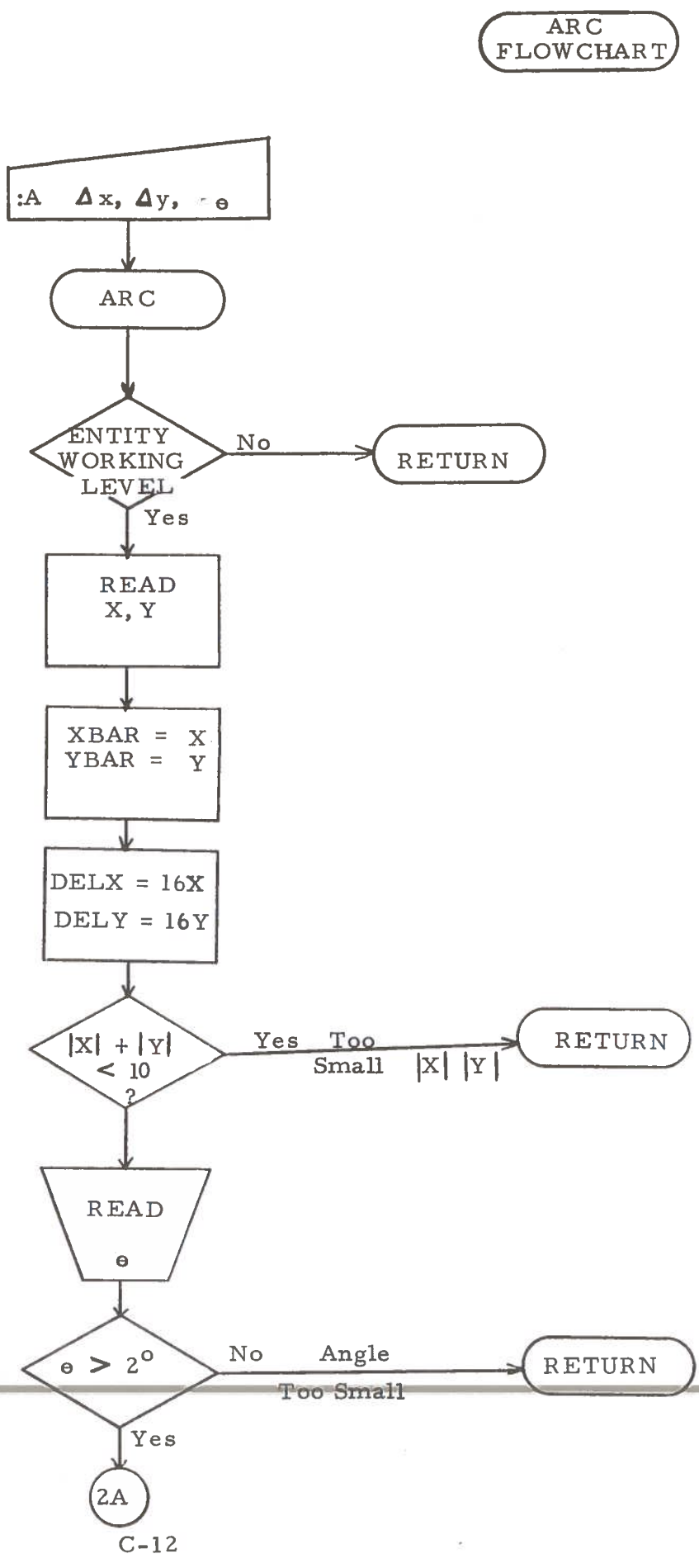
None

7. CORE USED

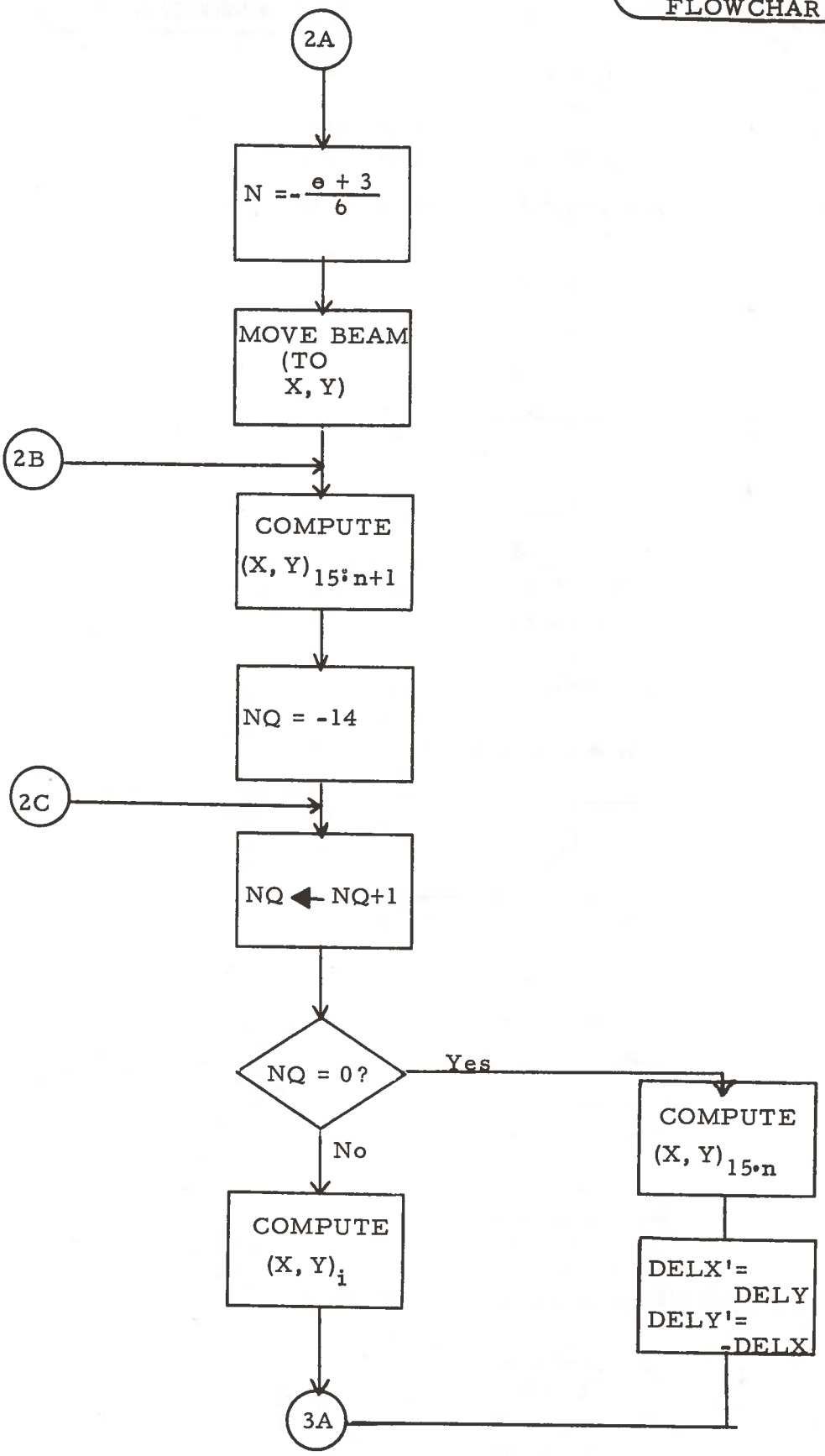
316_8

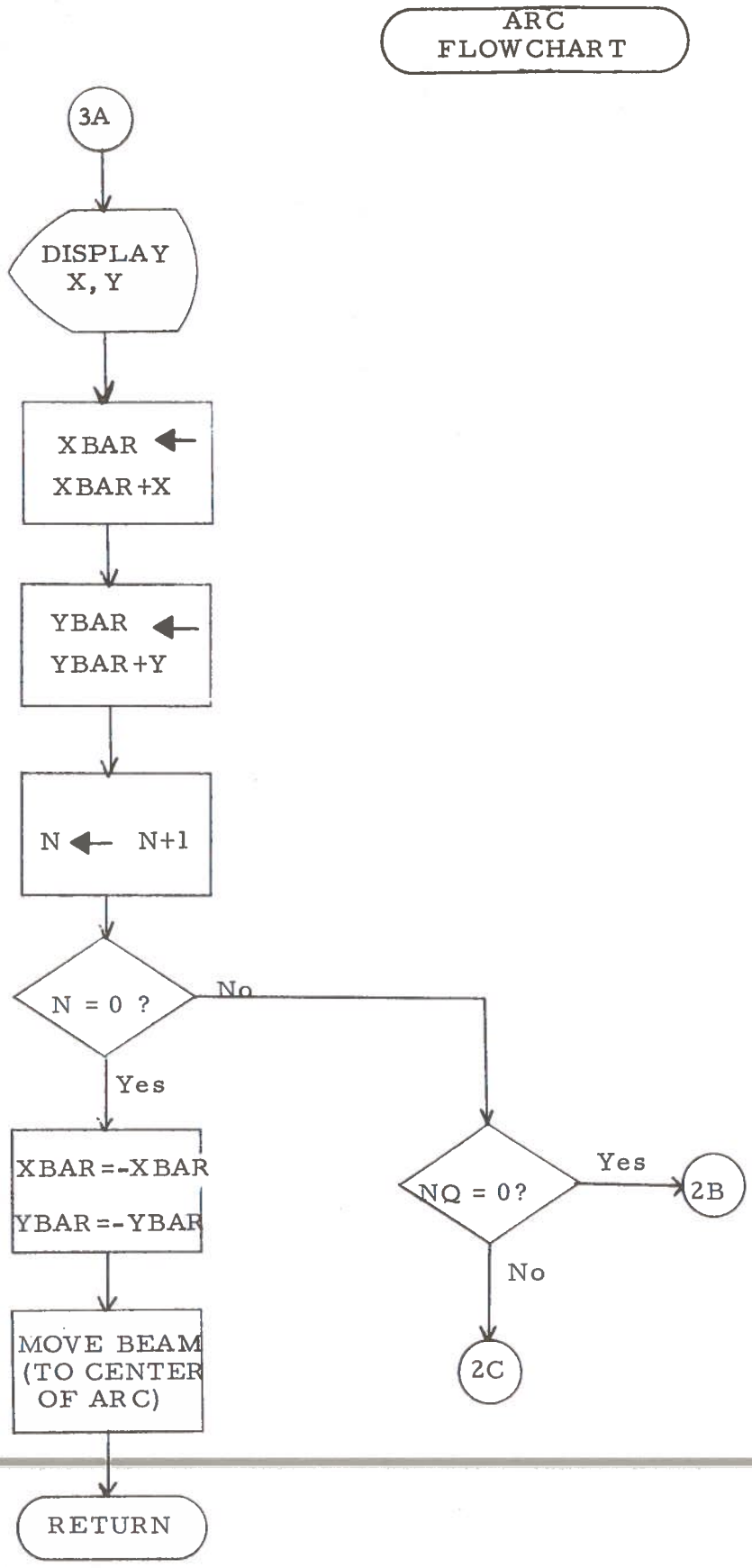
VARIABLES

YBAR:	Cumulative Y-axis (ordinate) beam travel; final value is 2's complemented for return of beam to center of arc.
XBAR:	Similarly for X-axis (abscissa) beam travel.
DELY:	16*Y - component of radius relative to the center of the arc; 90-degree clockwise transformation is applied for each quarter-circle component of the arc drawn.
DELX:	16*X - component of radius, as for DELY.
YI:	16*Y - component of chord drawn; this is the internal (to the program) computed value of the relative Y beam travel for the chord drawn.
XI:	16*X-component of chord.
YD:	YI/16 (rounded); the relative Y beam travel for a chord drawn; a summand for YBAR.
XD:	Similarly.
N:	The counter for the number of chords with subtended angle = 6^{o}, to be drawn; an additional (6^{o}) chord is drawn for input angles modulo 6 equal to 3^{o} or more.
NQ:	Internal counter for quarter-circle chords drawn; used to sense the last chord of a quarter-circle or the first chord of the subsequent quarter-circle is to be drawn.
LMXY:	Minimum -1 of sum of absolute values of raduis X and Y components; error message typed is "TOO SMALL |X| |Y| ".


ARC
FLOWCHART
:A Δx, Δy, e
ARC
ENTITY
WORKING
LEVEL
No
RETURN
Yes
READ
X, Y
XBAR = X
YBAR = Y
DELX = 16X
DELY = 16Y
|X| + |Y|
< 10
?
Yes Too
Small |X| |Y|
RETURN
READ
e
e > 2°
No Angle
Too Small
RETURN
Yes
2A

ARC FLOWCHART




ARC
FLOWCHART
3A
DISPLAY
X, Y
XBAR ← XBAR+X
YBAR ← YBAR+Y
N ← N+1
N = 0 ?
No
Yes
XBAR=-XBAR
YBAR=-YBAR
MOVE BEAM
(TO CENTER
OF ARC)
RETURN
NQ = 0?
Yes
2B
No
2C

BLIK, INTY, NEWF, OFF$, ON$$, UBLI

1. PURPOSE:

   a) BLIK — set blink command in conditioning block for indicator or entity

   b) INTY — set intensity command in conditioning block

   c) NEWF — set new frame command in conditioning block

   d) OFF$ — set command to turn off indicator or entity in conditioning block

   e) ON$$ — set command to turn on indicator or entity in conditioning block

   f) UBLI — set unblink command in conditioning block

2. CALLING SEQUENCE:

```
          JST*   PTR
           :
PTR       DAC    COML+N
           :
COML+N    XAC    Prog
```

where Prog = BLIK, INTY, NEWF, OFF$, ON$$, UBLI

3. INPUT:

   a) None
   b) Intensity from teletype buffer
   c) New frame ID from teletype buffer
   d) None
   e) None
   f) None

4. OUTPUT:

a) Blink command in conditioning block

b) Intensity command in conditioning block

c) New frame command in conditioning block

d) Command to turn off indicator or entity in conditioning block

e) Command to turn on indicator or entity in conditioning block

5. ACTION:

Each routine checks for proper working level and:

a) inserts blink command into conditioning block

b) gets intensity, checks if in range 1 through 7. If not, calls WHAT and returns. Otherwise inserts intensity into conditioning block.

c) calls IDER to get end check input ID for range. If OK, inserts new frame command into conditioning block.

d) inserts command to turn off indicator or entity into conditioning block

e) inserts command to turn-on indicator or entity into conditioning block

f) inserts unblink command into conditioning block

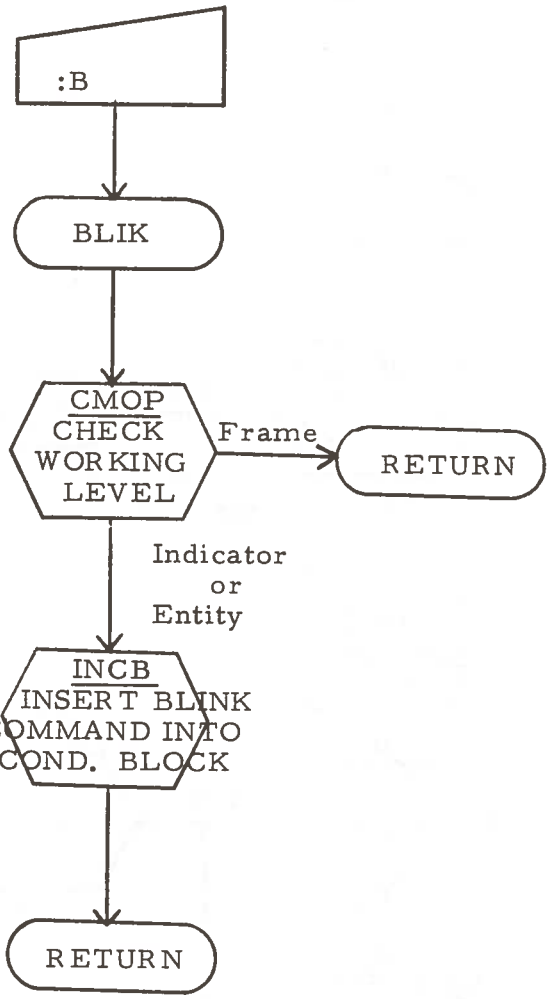
6. EXTERNAL REFERENCES:

CMOD, INCB, GTNO, WHAT, IDER

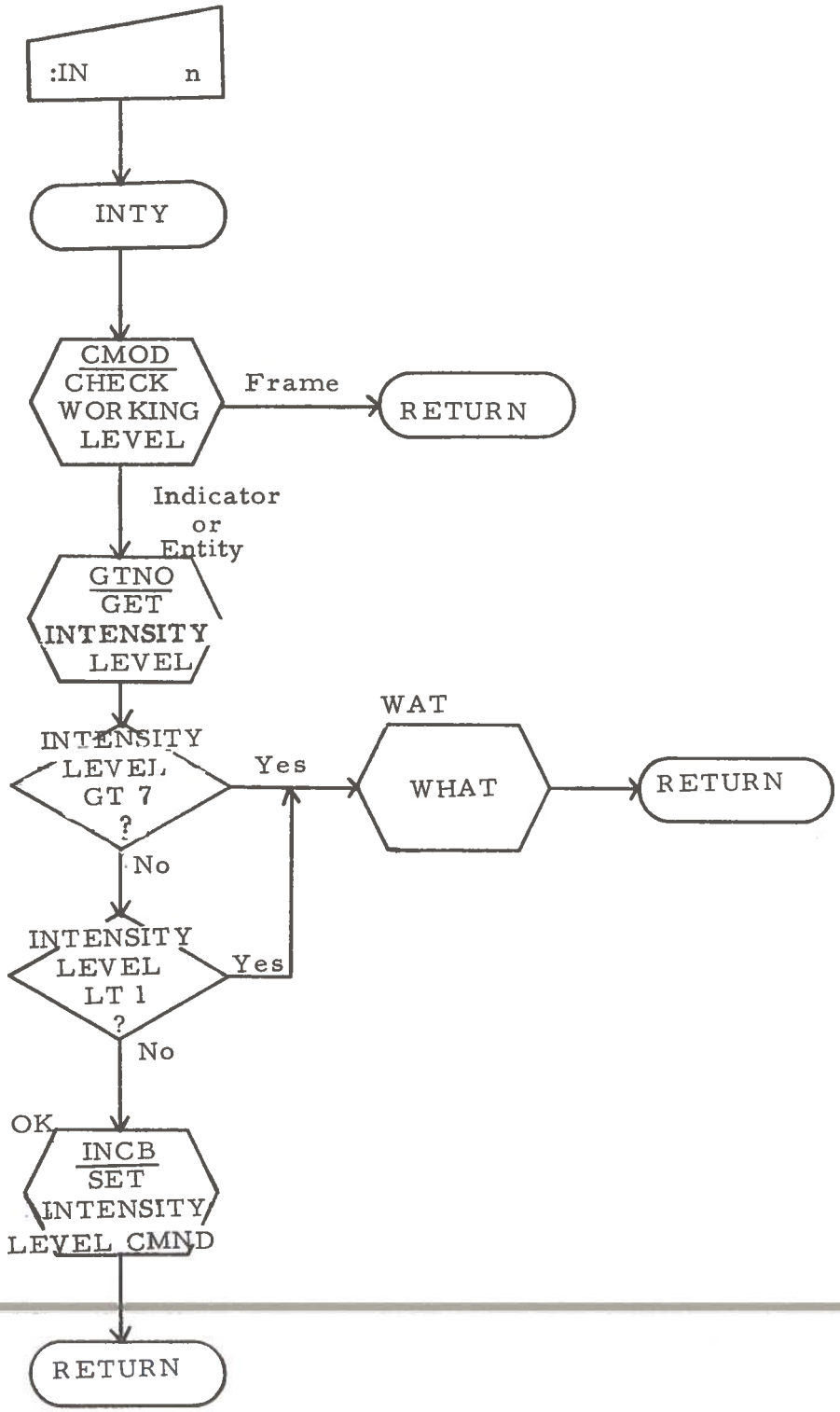
7. CORE USED:

114_8

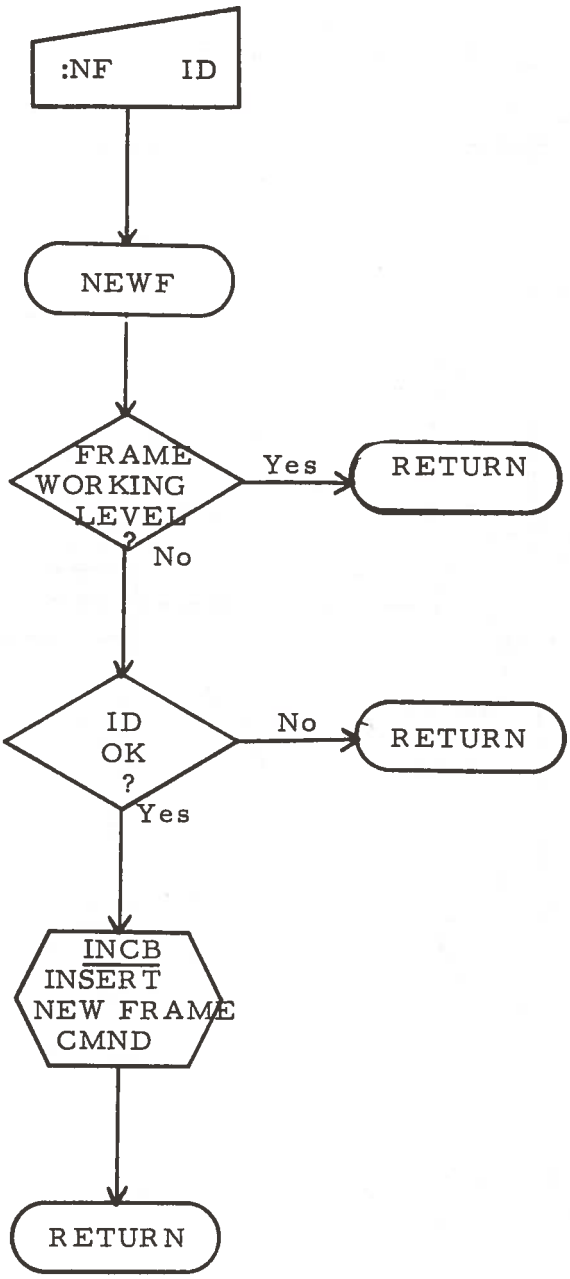
BLIK
FLOWCHART

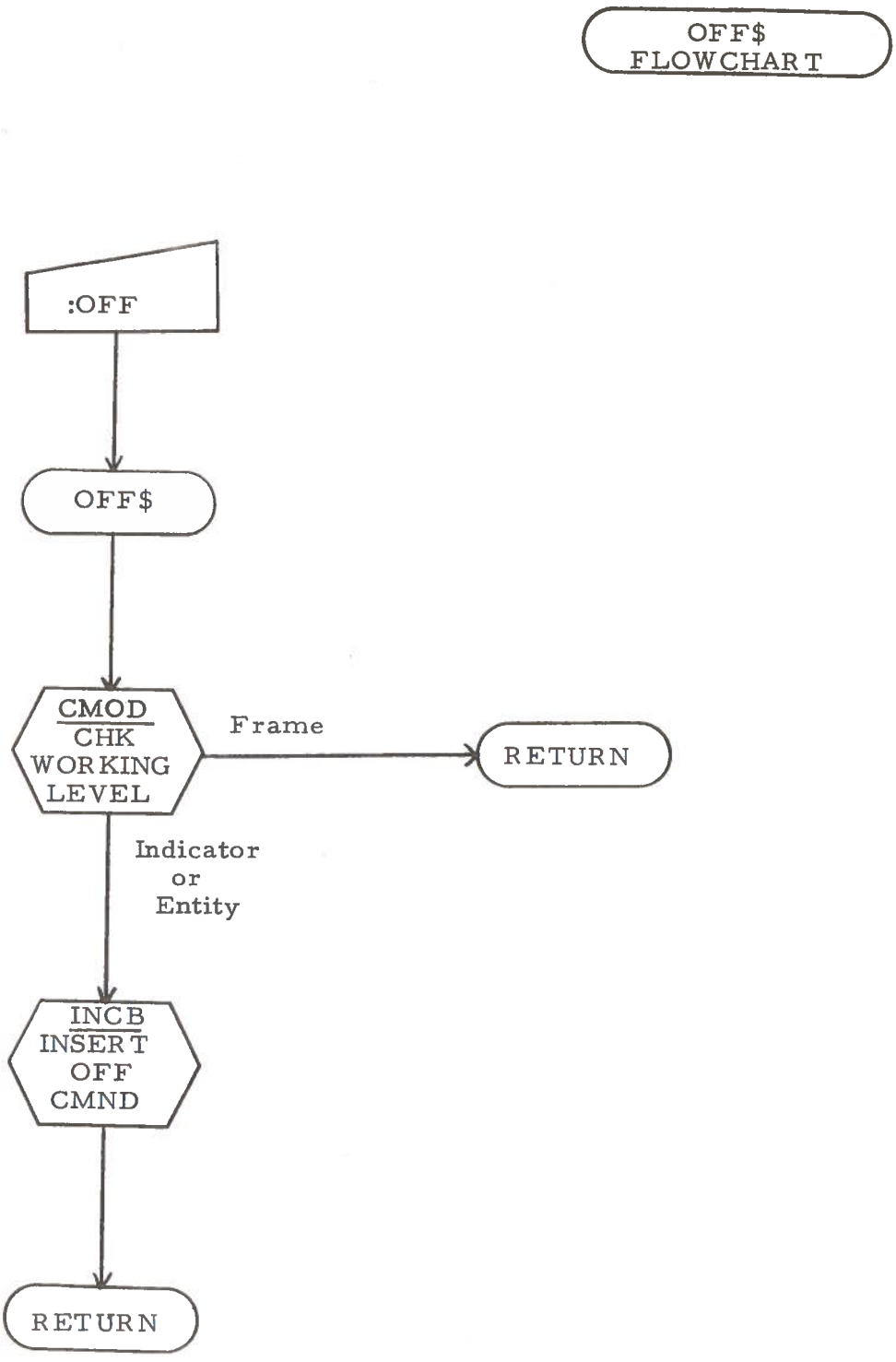


INTY
FLOWCHART

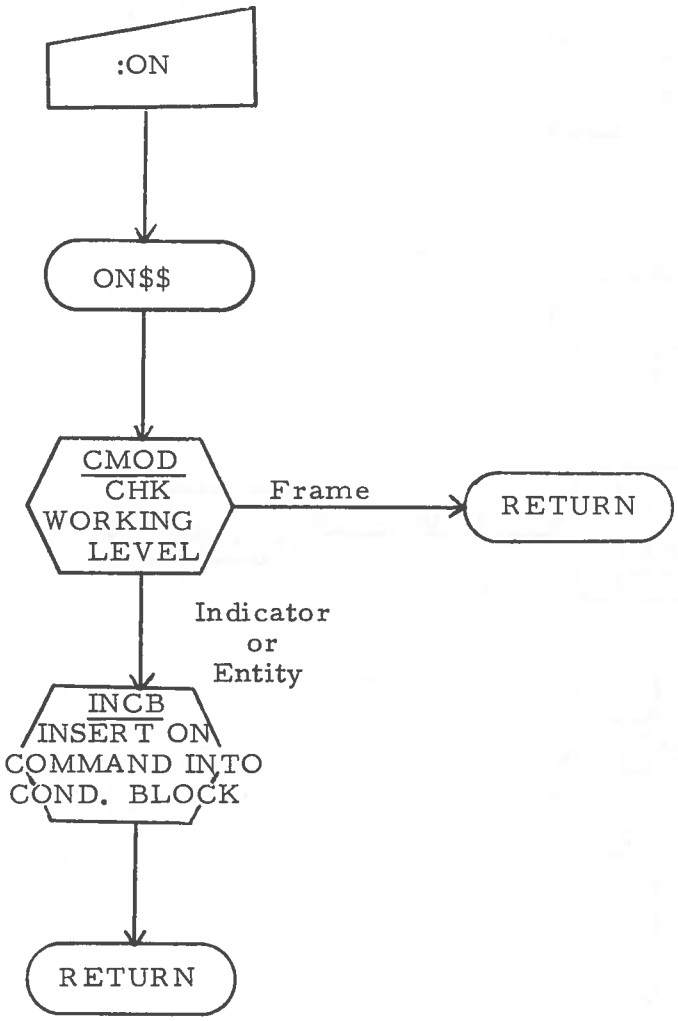


NEWF
FLOWCHART



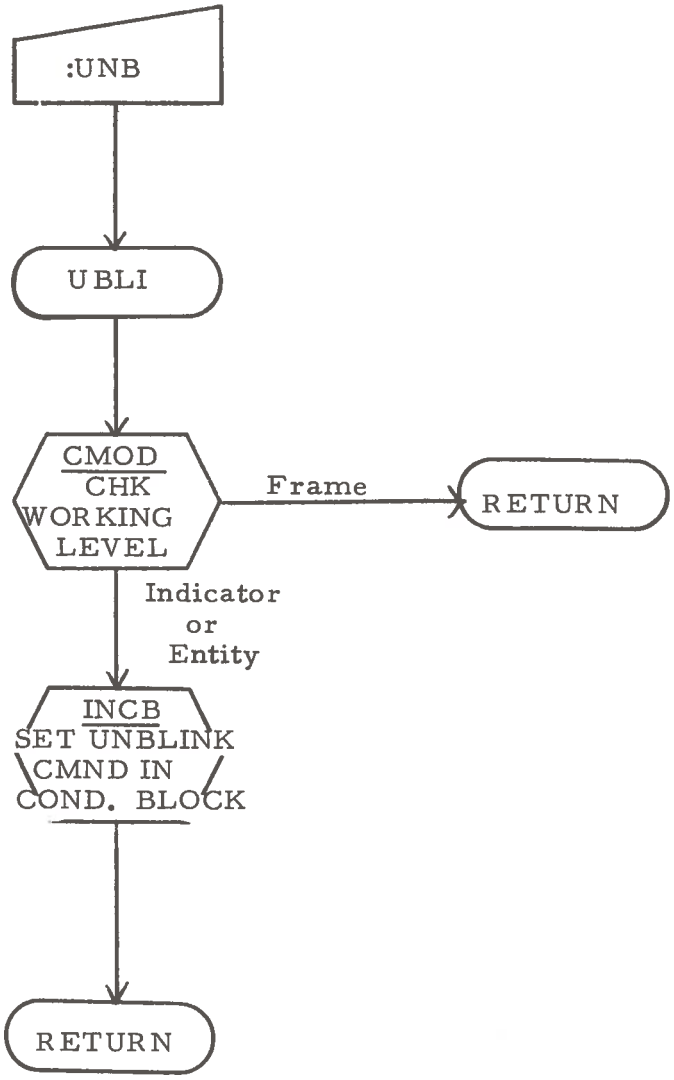

OFF$
FLOWCHART
:OFF
OFF$
CMOD
CHK
WORKING
LEVEL
Frame
RETURN
Indicator
or
Entity
INCB
INSERT
OFF
CMND
RETURN

ON$$
FLOWCHART



UBLI
FLOWCHART

UNBLINK



BLIN, LINE, PINT

1. PURPOSE:

   a) BLIN — Draw blank line
   b) LINE — Draw line
   c) PINT — Display point

2. CALLING SEQUENCE:

```
            JST*    PTR
             :
PTR         DAC     COML+N
             :
COML+N      XAC     Prog
```

where: Prog = BLIN, LINE, PINT

3. INPUT:

Δx, Δy from input buffer. Also dynamic expression if any from input buffer.

4. OUTPUT:

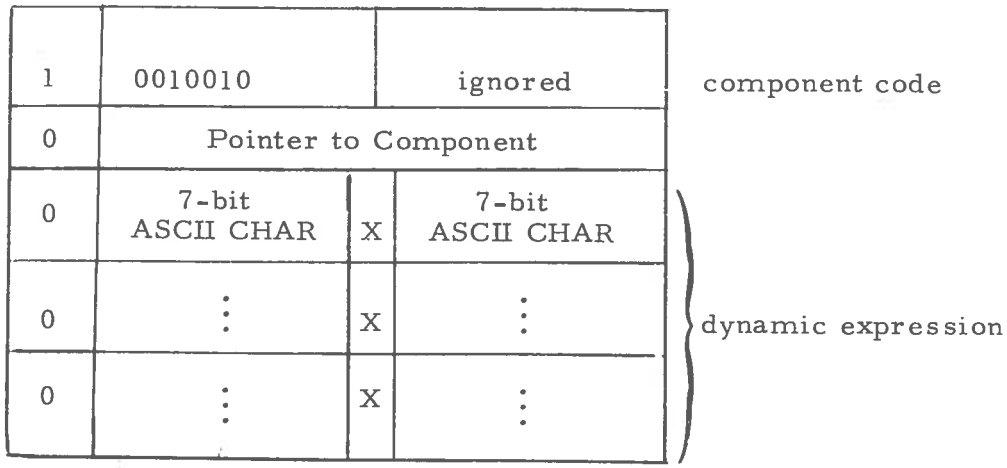
Appropriate display commands in component block and conditioning information in conditioning block if any.

5. ACTION:

Sets up return address and parameter depending upon entry point. Checks working level for entity. Gets component block address for x command and saves.

Gets Δx and inserts "LOAD ΔX REL" command into component block. Checks for number sign indicating a dynamic expression is present. If not, jumps to input Δy.

If so, the component code, a pointer to the component, and the expression are moved into the conditioning block in the following format:



Δy is processed similar to Δx. The "Draw" or "Load" command, a "draw x rel" command is given where x=0.

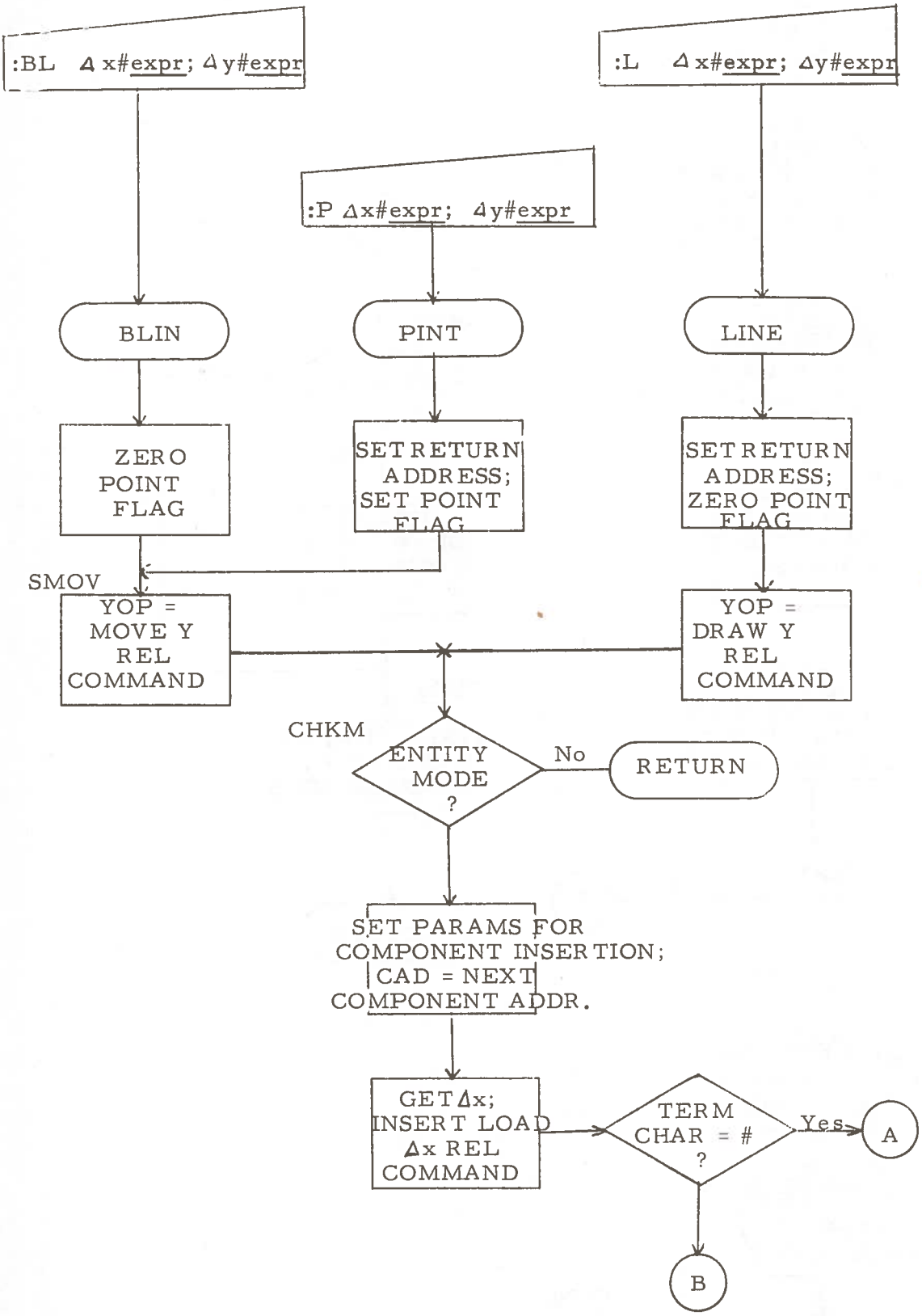
6. EXTERNAL REFERENCES:

CMOD
GTNO
COP
CIN
CLO
INCB
CPAD

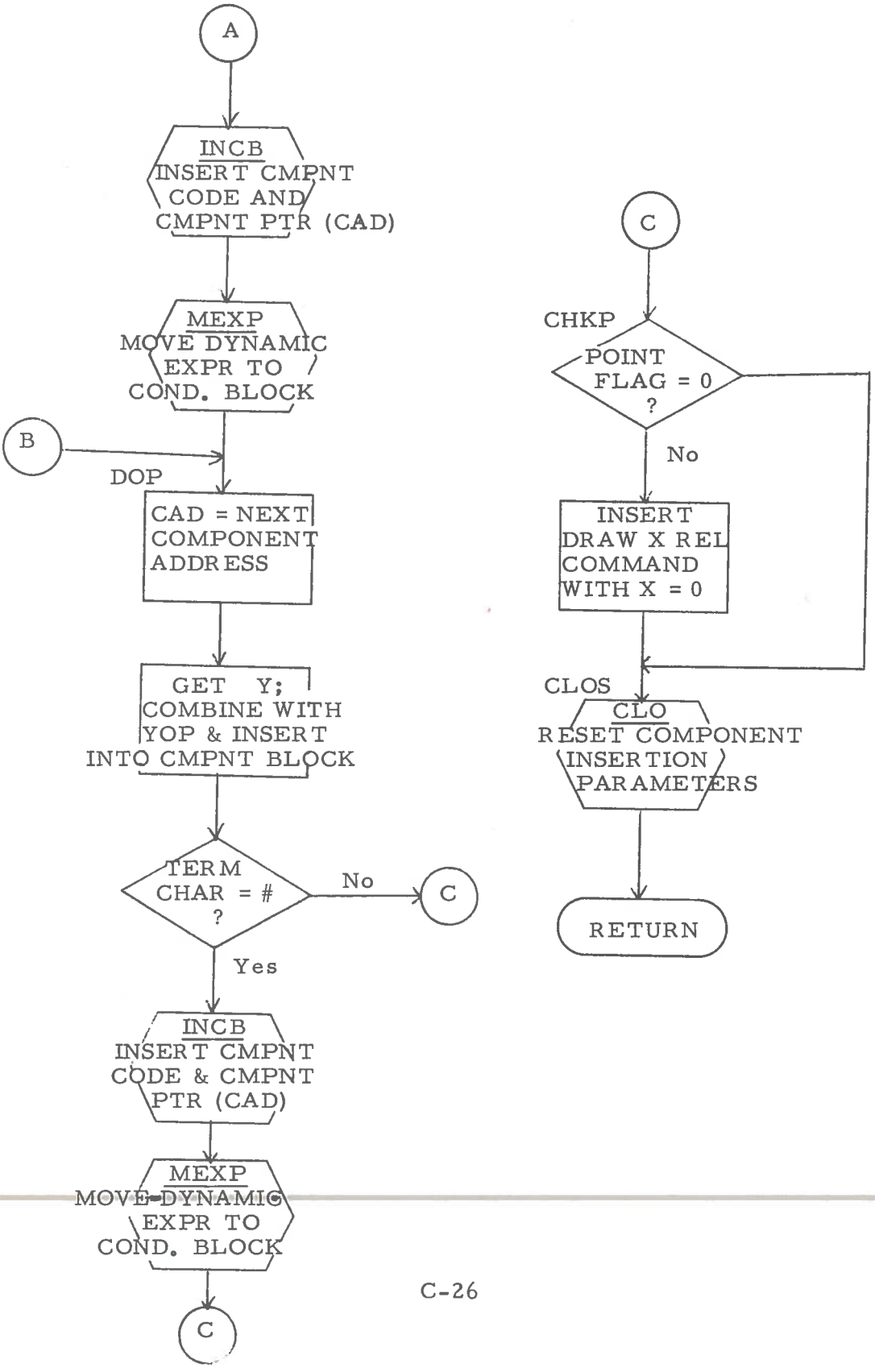
7. CORE USED:

125_8

Draw Blank Line, Draw Line, or Draw Point



BLIN, LINE, PINT FLOWCHART



DIRE

1. PURPOSE:

Set up display commands for Digital Readout

2. CALLING SEQUENCE:

```
            JST*    PTR
             :
             :
PTR         DAC     COML+N
             :
             :
COML+N      XAC     DIRE
```

3. INPUT:

Dynamic Expression, number of digits to display, and character size from input buffer

4. OUTPUT:

Character setup initialized to zero in component block. Digital Readout expression set up in conditional block.

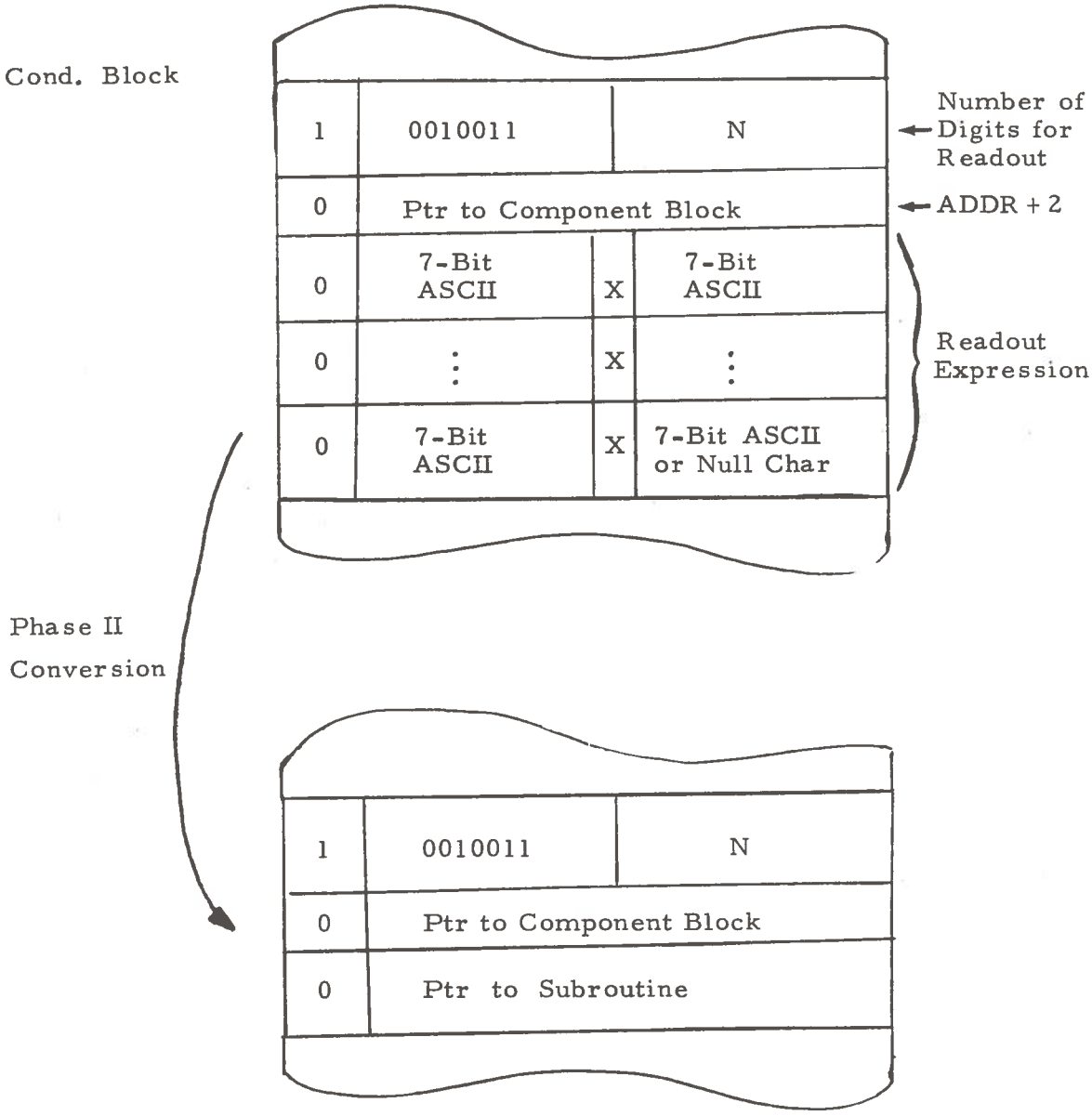
5. ACTION:

Checks for entity working level. Exits on frame and indicator level. Bypasses expression. Gets number of digits to display. Inserts "enter char mode" command into component block. Gets size and sets command into component block. Inserts Digital Readout command, component block pointer and expression in conditional block. Sets characters in component block to zero. Sets component block to return to coordinate mode and returns.

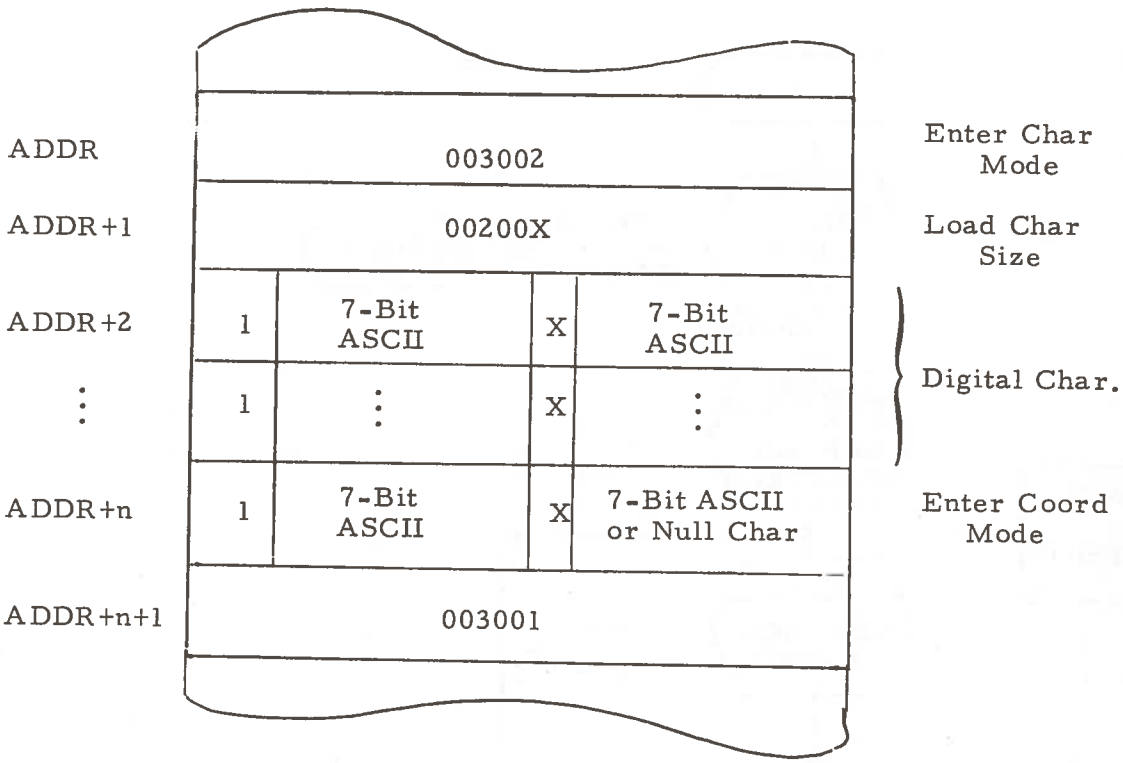
6. EXTERNAL REFERENCES:

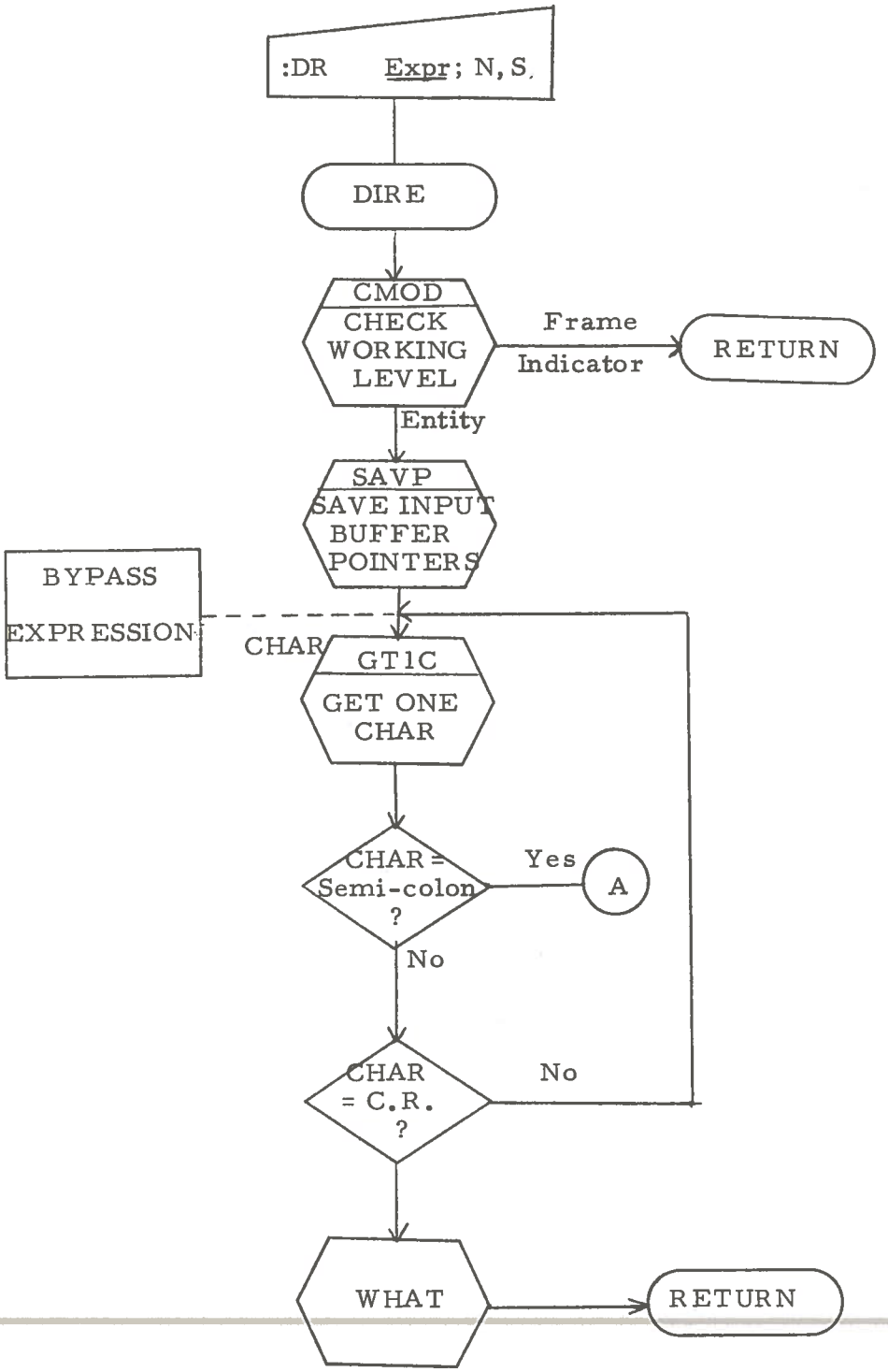
CMOD, SAVP, GI1C, WHAT, GTNO, TAO$, COP, CIN, CLO, CPAD, INCB, RESP, MEXP

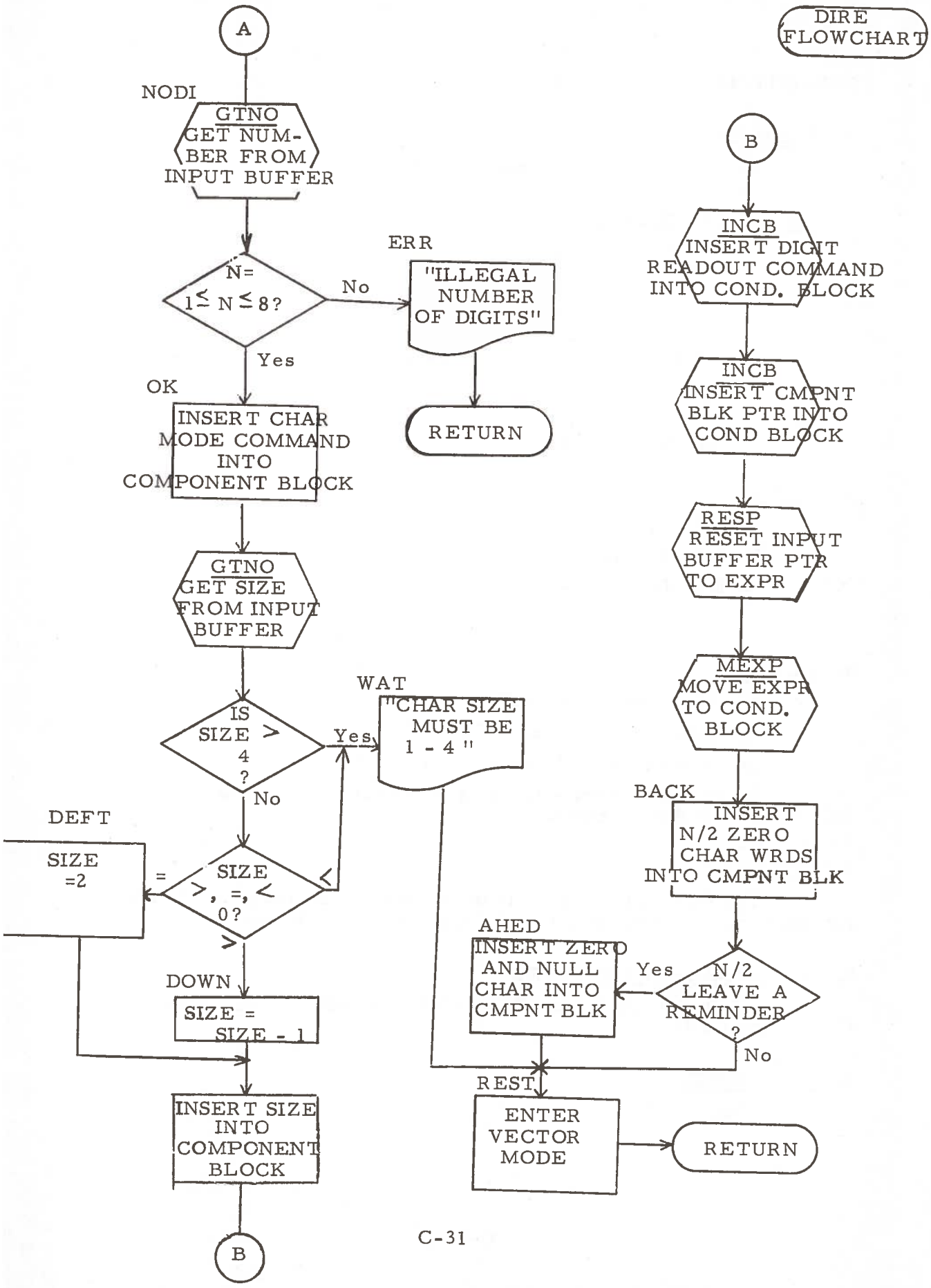
7. CORE SETUP OF DISPLAY FILE



Component Block




:DR Expr; N, S,
DIRE
CMOD
CHECK WORKING LEVEL
Frame Indicator
RETURN
Entity
SAVP
SAVE INPUT BUFFER POINTERS
BYPASS EXPRESSION
CHAR
GT1C
GET ONE CHAR
CHAR = Semi-colon ?
Yes
A
No
CHAR = C.R. ?
No
WHAT
RETURN


DIRE FLOWCHART
A
NODI
GTNO GET NUMBER FROM INPUT BUFFER
N= 1≤N≤8?
No
ERR
"ILLEGAL NUMBER OF DIGITS"
RETURN
Yes
OK
INSERT CHAR MODE COMMAND INTO COMPONENT BLOCK
GTNO GET SIZE FROM INPUT BUFFER
IS SIZE > 4 ?
Yes
WAT
"CHAR SIZE MUST BE 1 - 4 "
No
DEFT
SIZE =2
=
SIZE >, =, < 0?
<
>
DOWN
SIZE = SIZE - 1
INSERT SIZE INTO COMPONENT BLOCK
B
B
INCB INSERT DIGIT READOUT COMMAND INTO COND. BLOCK
INCB INSERT CMPNT BLK PTR INTO COND BLOCK
RESP RESET INPUT BUFFER PTR TO EXPR
MEXP MOVE EXPR TO COND. BLOCK
BACK
INSERT N/2 ZERO CHAR WRDS INTO CMPNT BLK
N/2 LEAVE A REMINDER ?
Yes
AHED
INSERT ZERO AND NULL CHAR INTO CMPNT BLK
No
REST
ENTER VECTOR MODE
RETURN

SUBROUTINE DLET

1. PURPOSE

Delete Frames, Indicators, and Entities.

2. CALLING SEQUENCE:

```
                JST          PTR
                 .
                 .
                 .
                 .
        PTR     DAC         COML+N
                 .
                 .
                 .
     COML+N     XAC          DLET
```

3. INPUT

a) D F ID, I ID, E ID

b) D F ID, I ID

c) D F ID

d) D

4. OUTPUT

a) Delete specified entity.

b) Delete specified indicator.

c) Delete specified frame.

d) Delete current entity if at entity level; otherwise type 'ILLEGAL WORKING LEVEL'.

5. ACTION

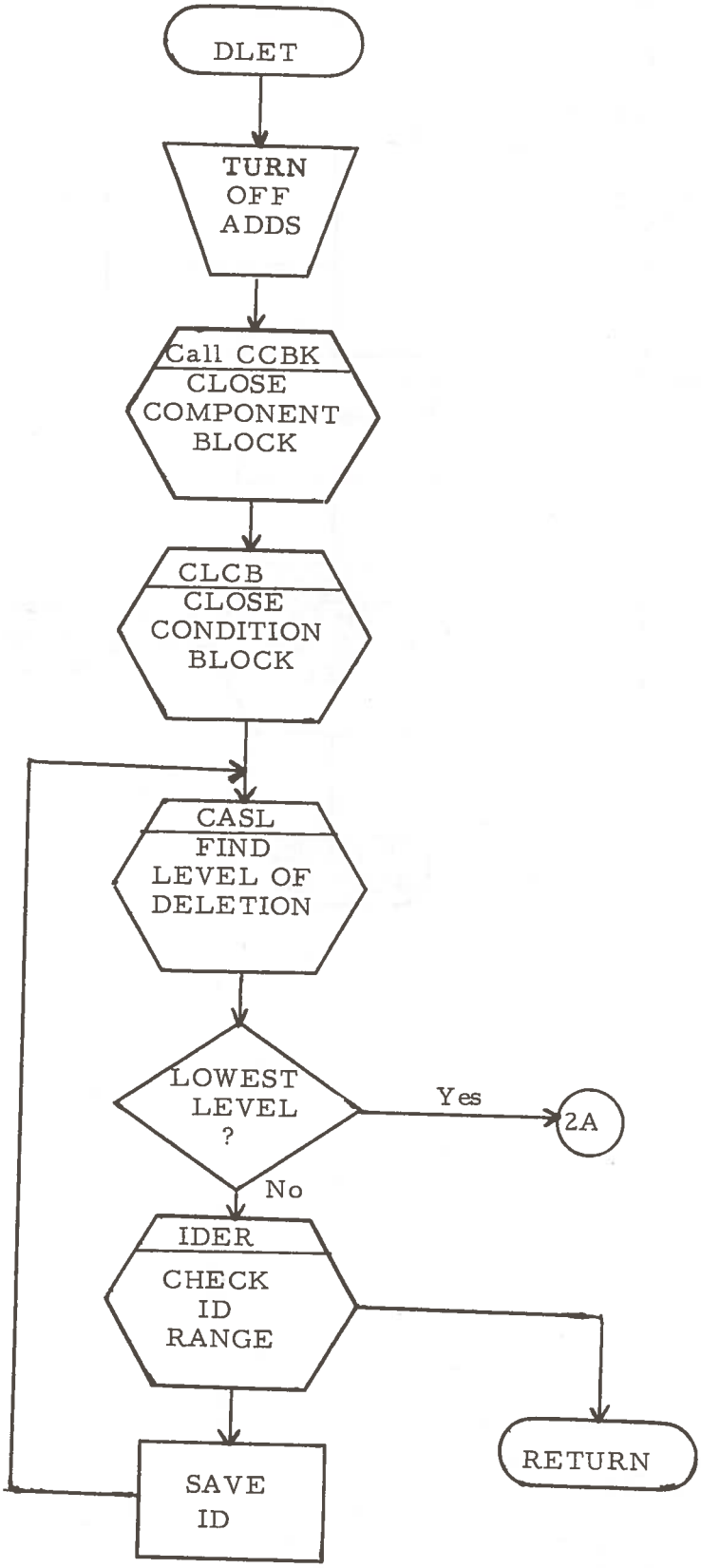
The highest level block to be deleted and all lower levels are removed from the display list and returned to free storage.

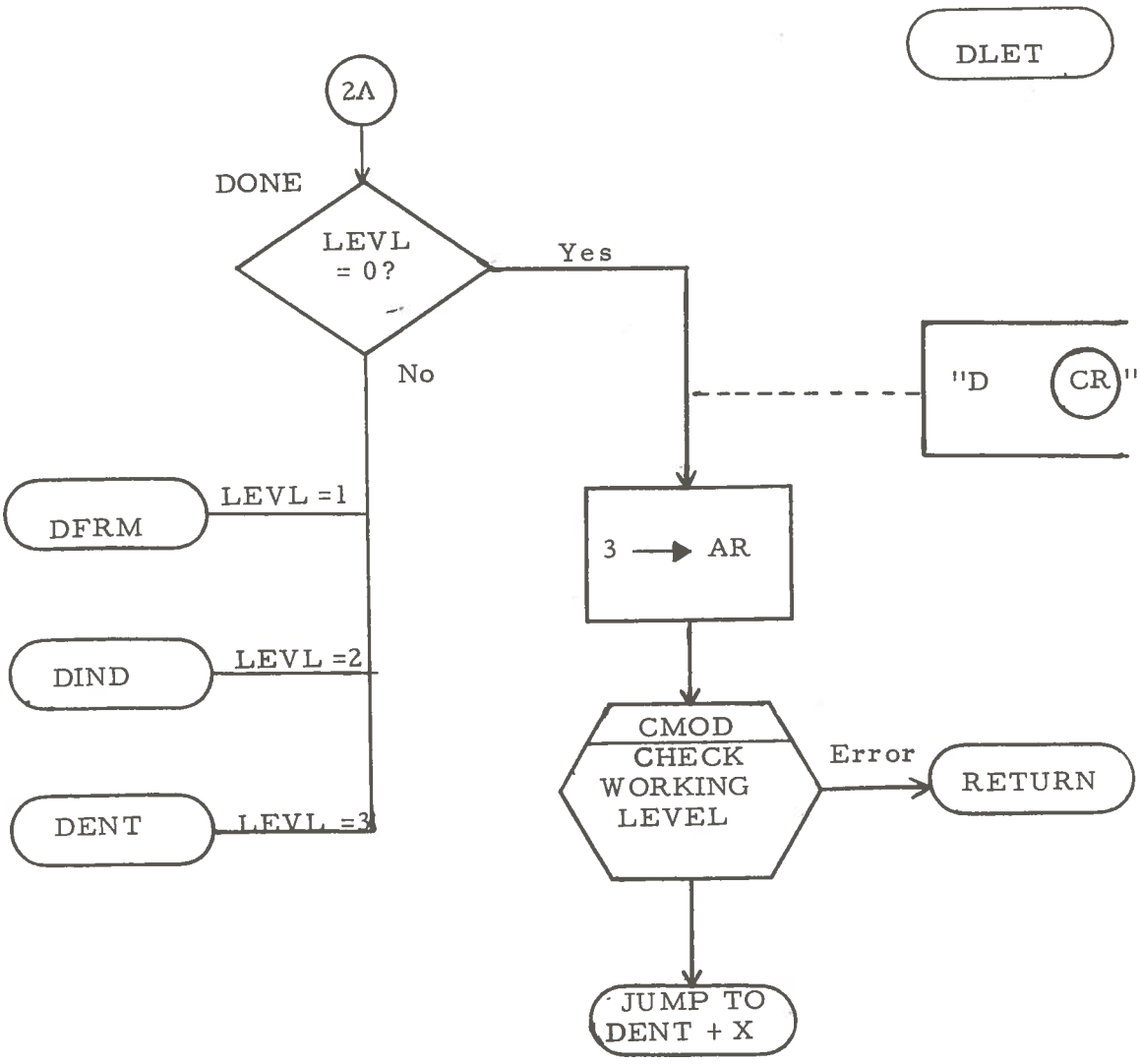
6. EXTERNALS REFERENCED

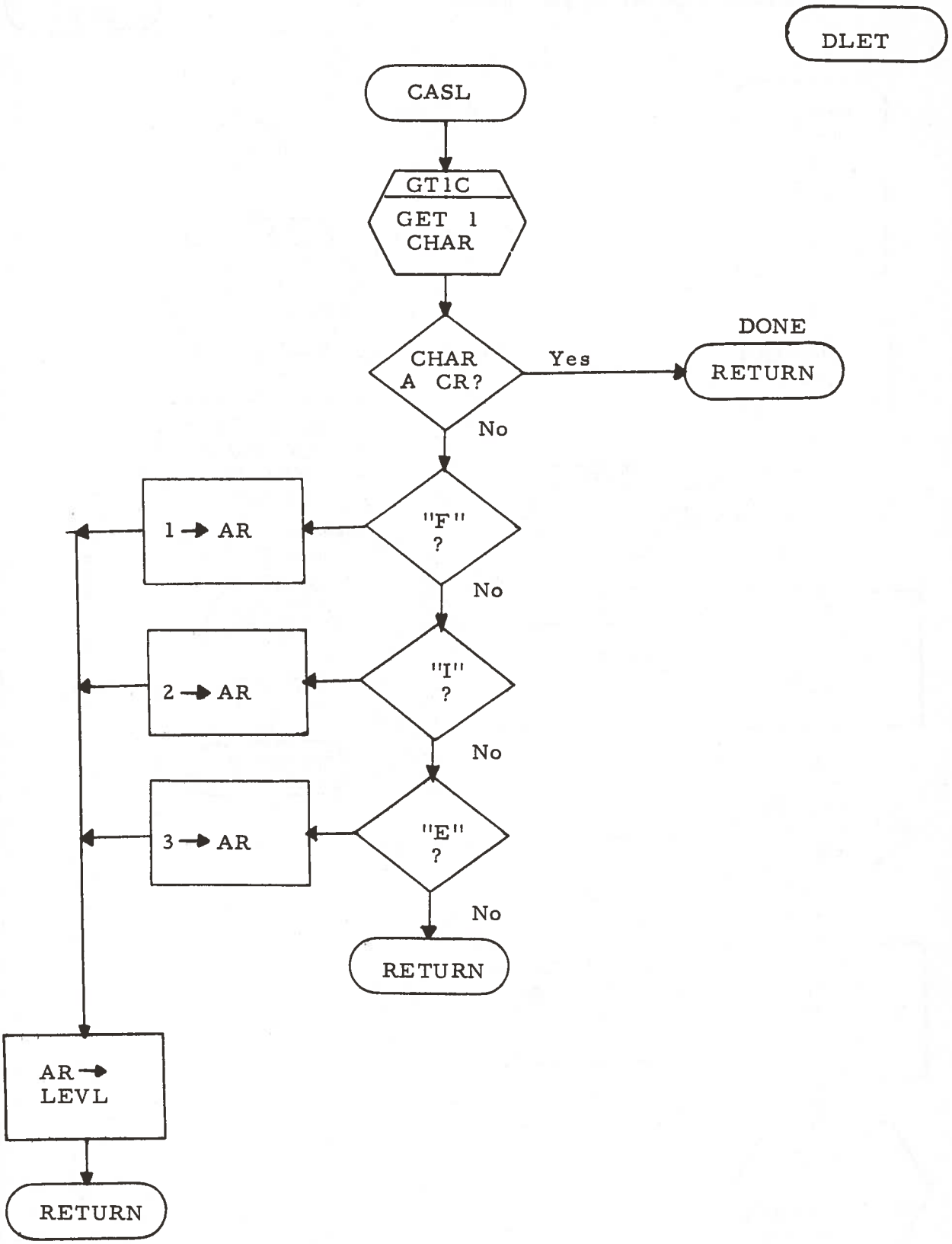
FRID, FRAD, INID, ENID, ENAD, LFAD, LIAD, LEAD, FRHD, and CLVL.

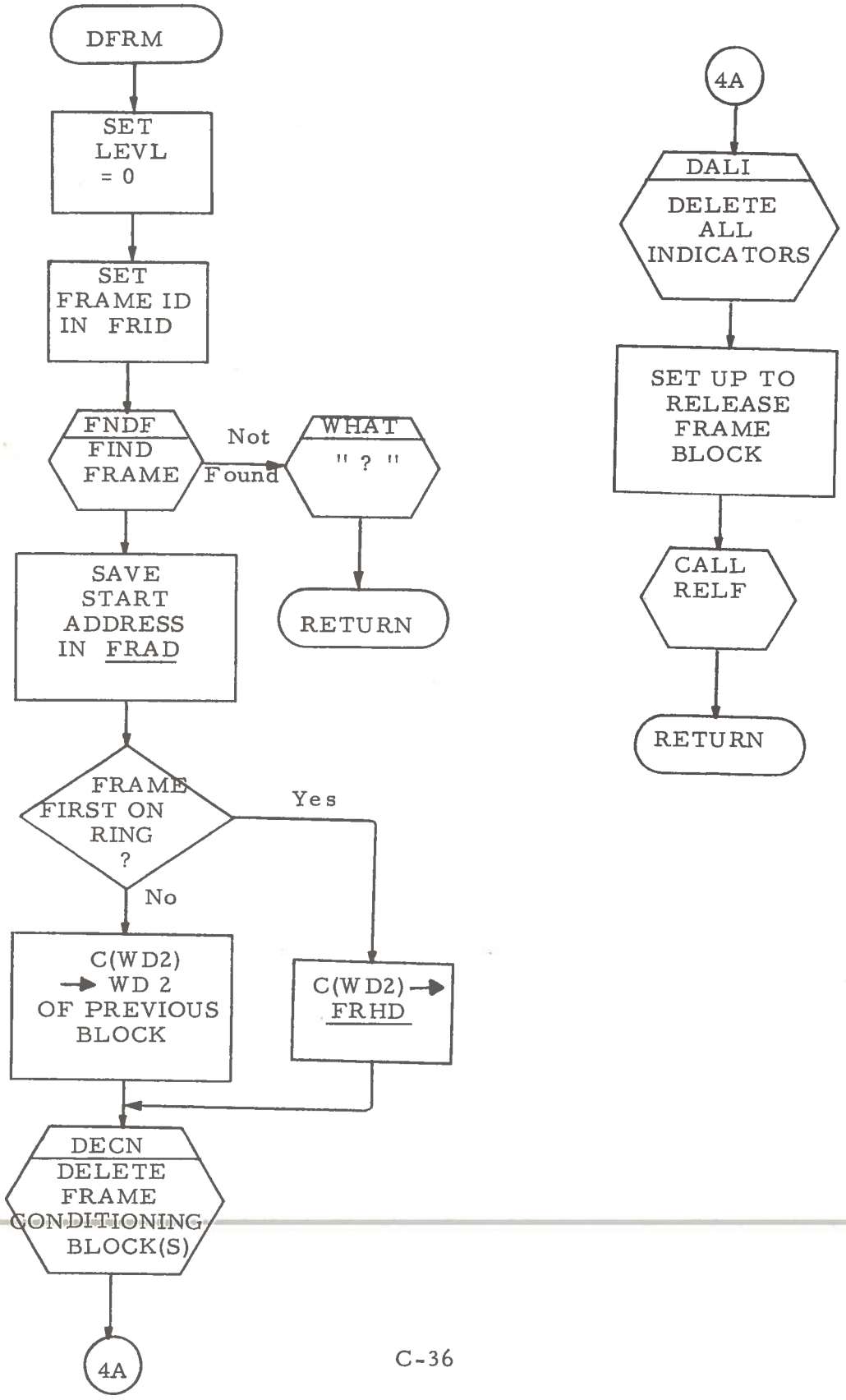
7. CORE USED

446_8

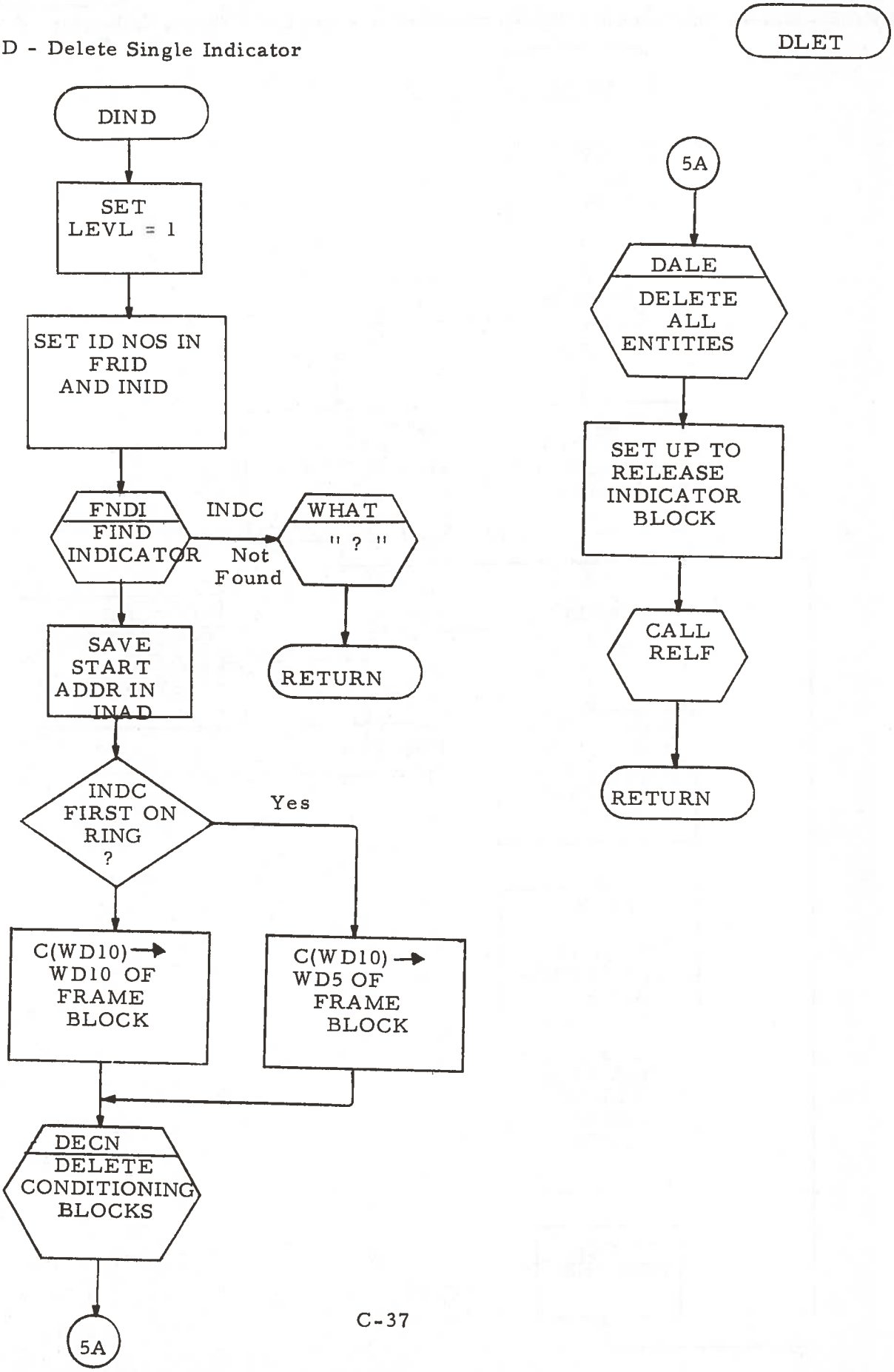

DLET
TURN OFF ADDS
Call CCBK
CLOSE COMPONENT BLOCK
CLCB
CLOSE CONDITION BLOCK
CASL
FIND LEVEL OF DELETION
LOWEST LEVEL ?
Yes
2A
No
IDER
CHECK ID RANGE
RETURN
SAVE ID


DLET
2Λ
DONE
LEVL
= 0?
Yes
No
"D
CR"
DFRM
LEVL =1
3 → AR
DIND
LEVL =2
CMOD
CHECK
WORKING
LEVEL
Error
RETURN
DENT
LEVL =3
JUMP TO
DENT + X

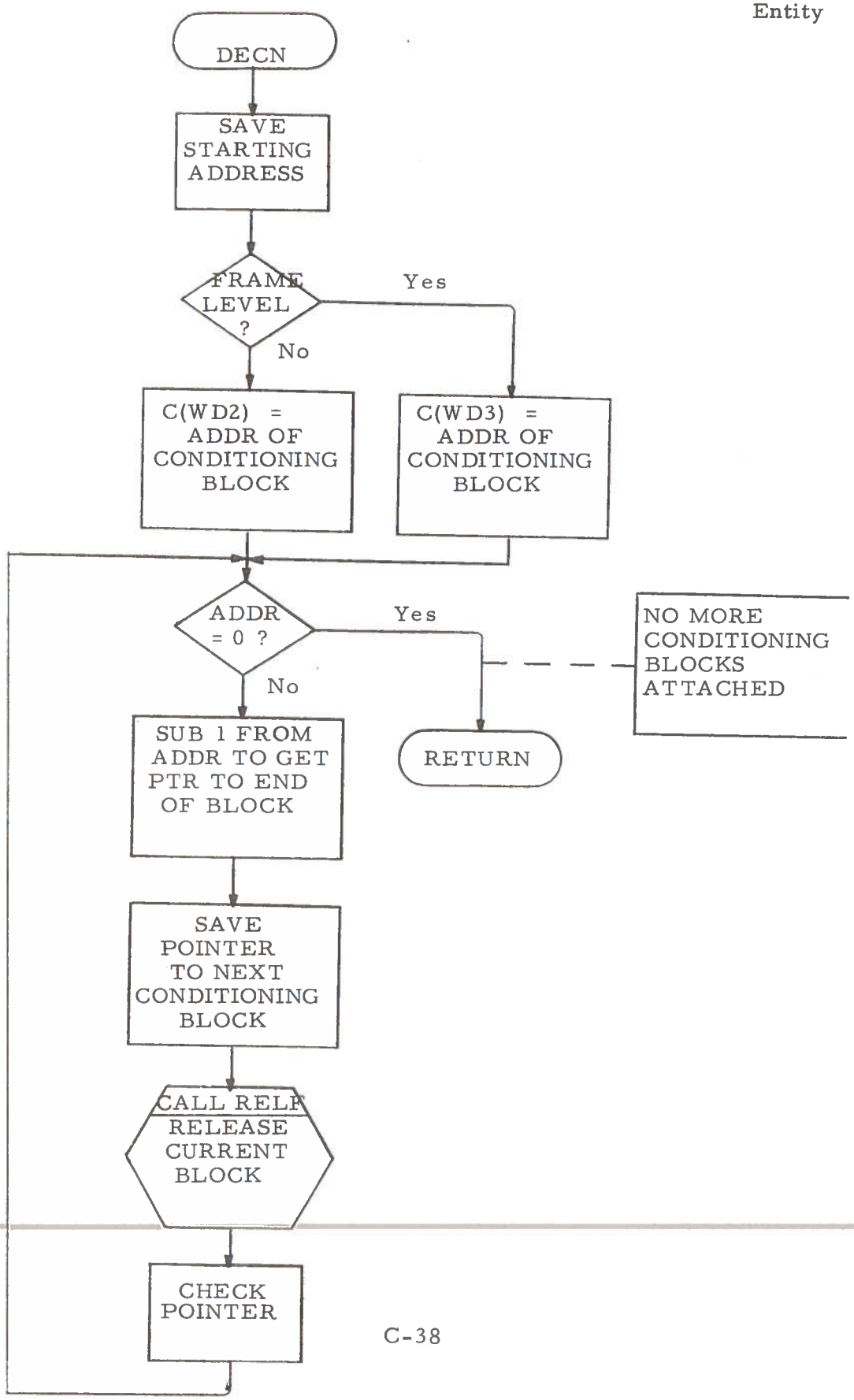

DLET
CASL
GT1C
GET 1
CHAR
CHAR
A CR?
Yes
DONE
RETURN
No
"F"
?
1 → AR
No
"I"
?
2 → AR
No
"E"
?
3 → AR
No
RETURN
AR →
LEVL
RETURN


DFRM
SET LEVL = 0
SET FRAME ID IN FRID
FNDF
FIND FRAME
Not Found
WHAT
" ? "
RETURN
SAVE START ADDRESS IN FRAD
FRAME FIRST ON RING ?
Yes
No
C(WD2) → WD 2 OF PREVIOUS BLOCK
C(WD2) → FRHD
DECN
DELETE FRAME CONDITIONING BLOCK(S)
4A
4A
DALI
DELETE ALL INDICATORS
SET UP TO RELEASE FRAME BLOCK
CALL RELF
RETURN

DIND - Delete Single Indicator

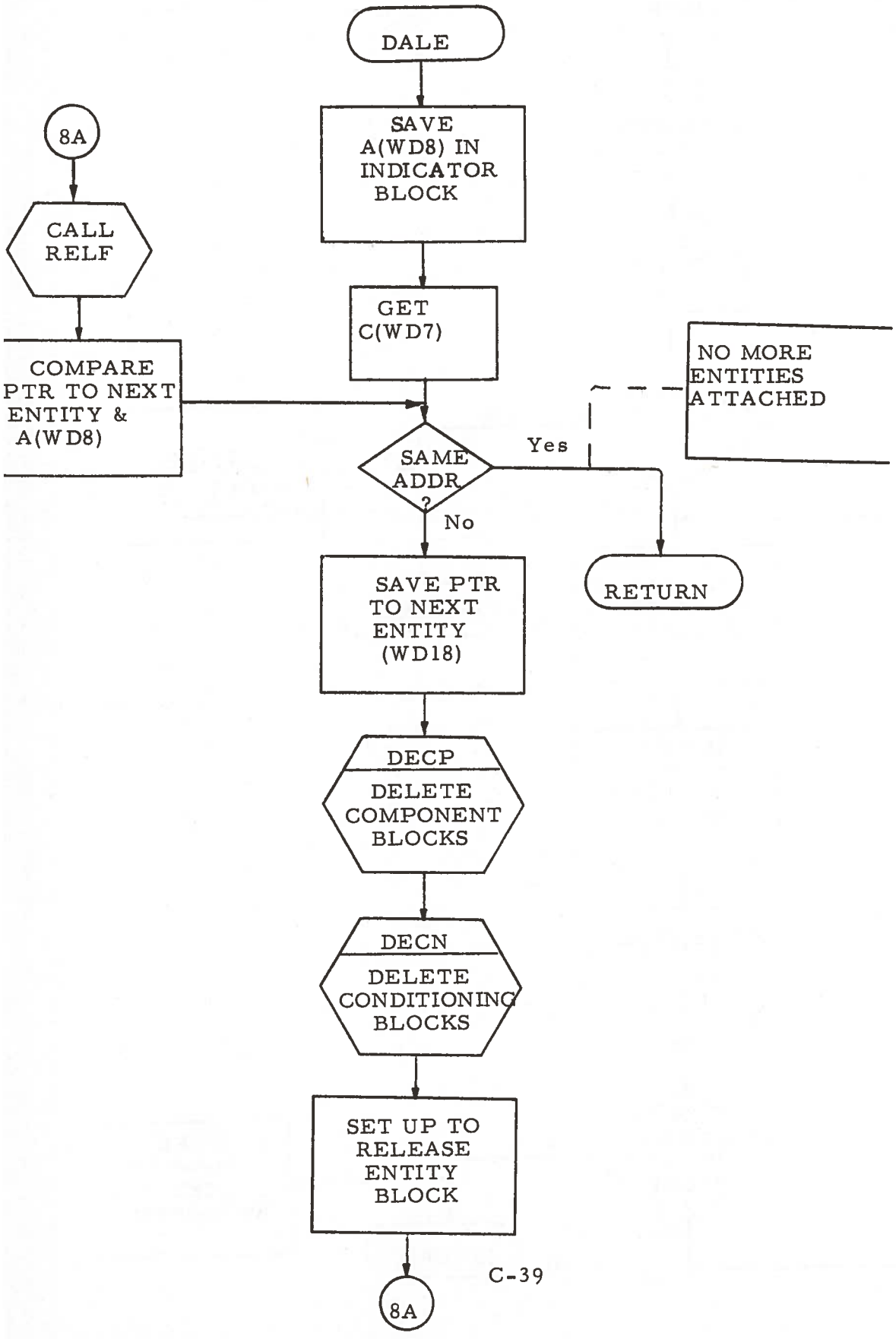


DECN - Delete Conditioning Blocks on Entry A = Addr of Frame, Indicator or Entity



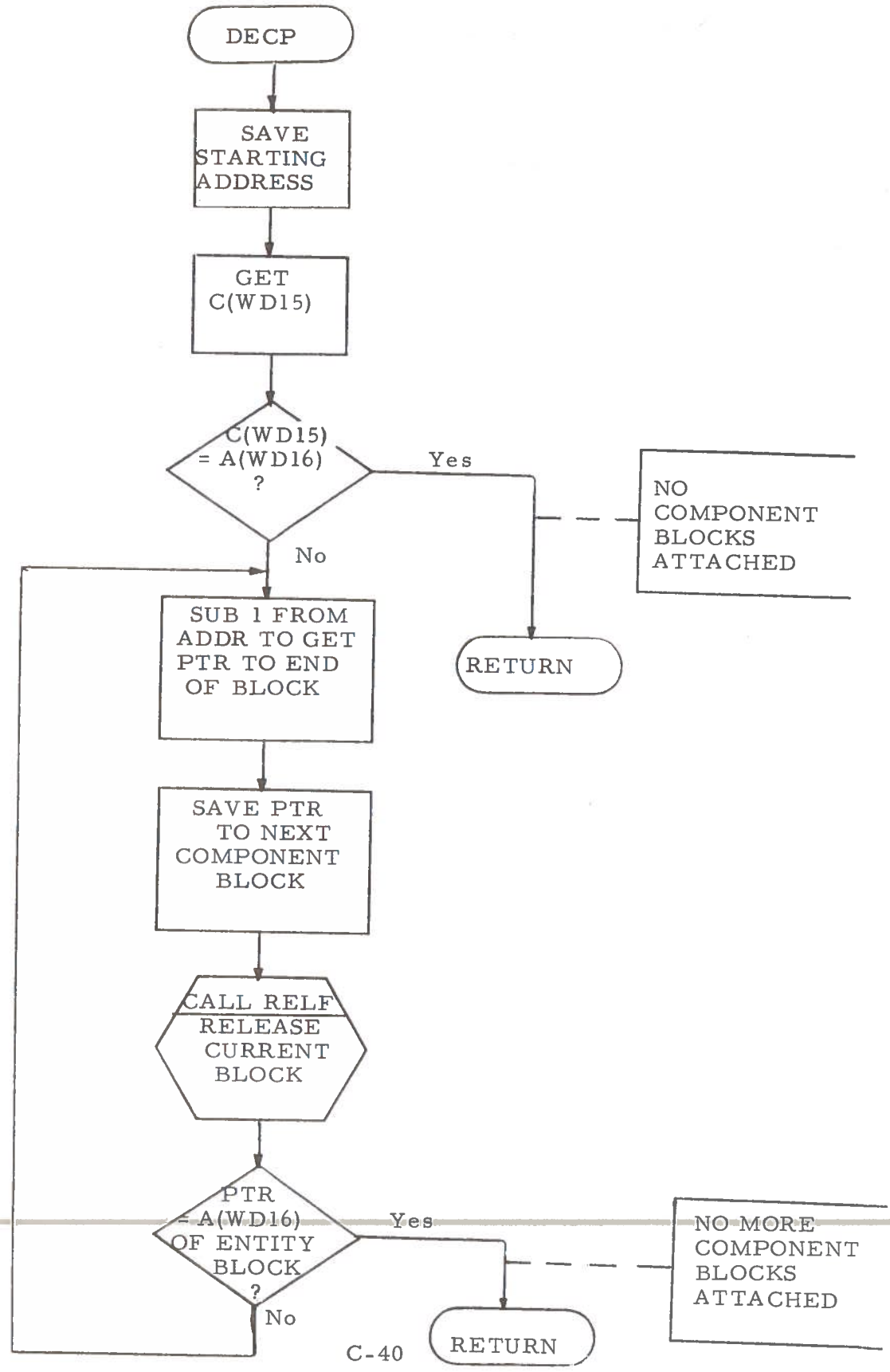
DALE - Delete all Entities
Input - Start Address of Indicator in INAD

DLET

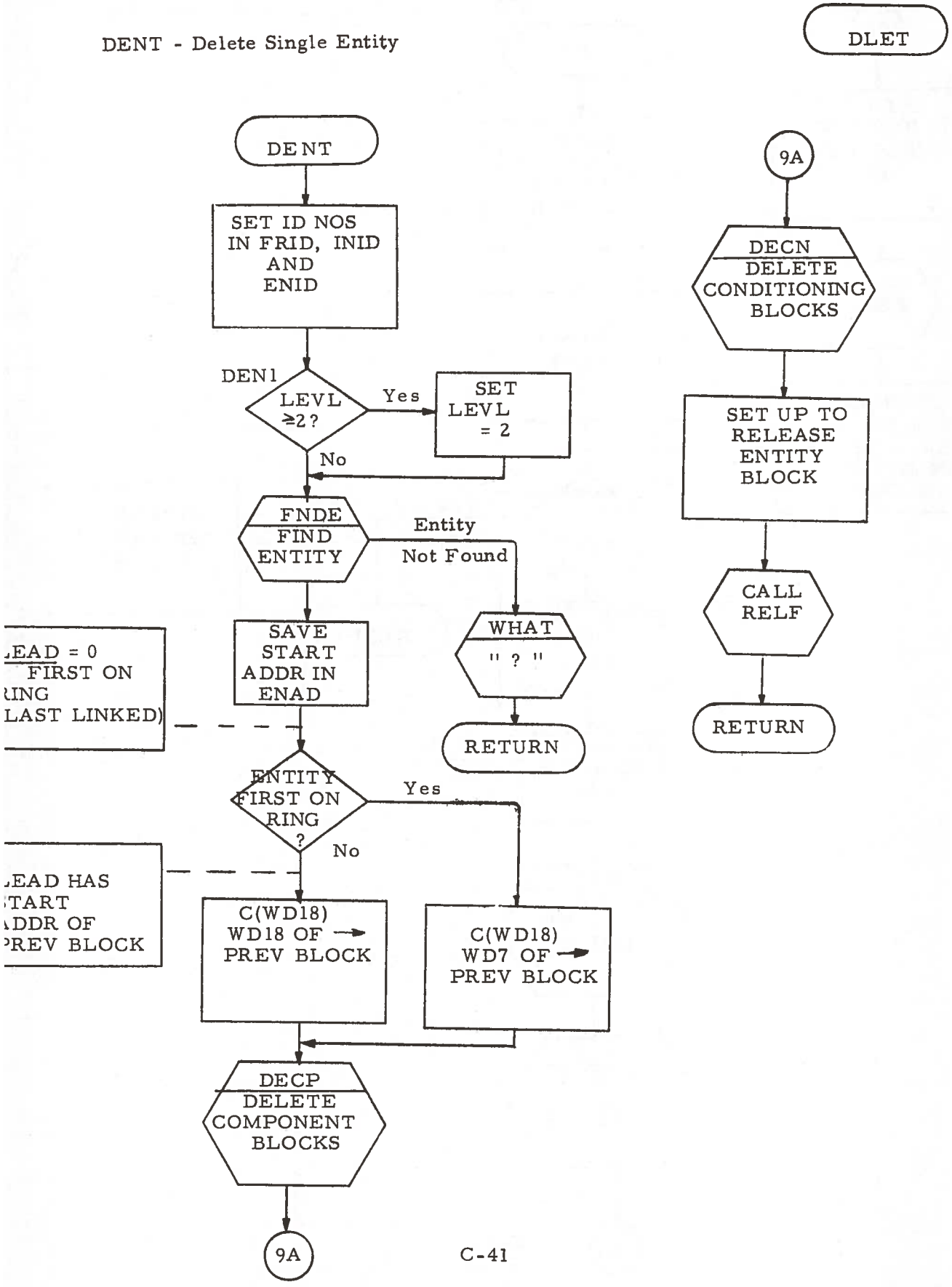


DECP - Delete Component Blocks on Entry AR = Addr of Entity

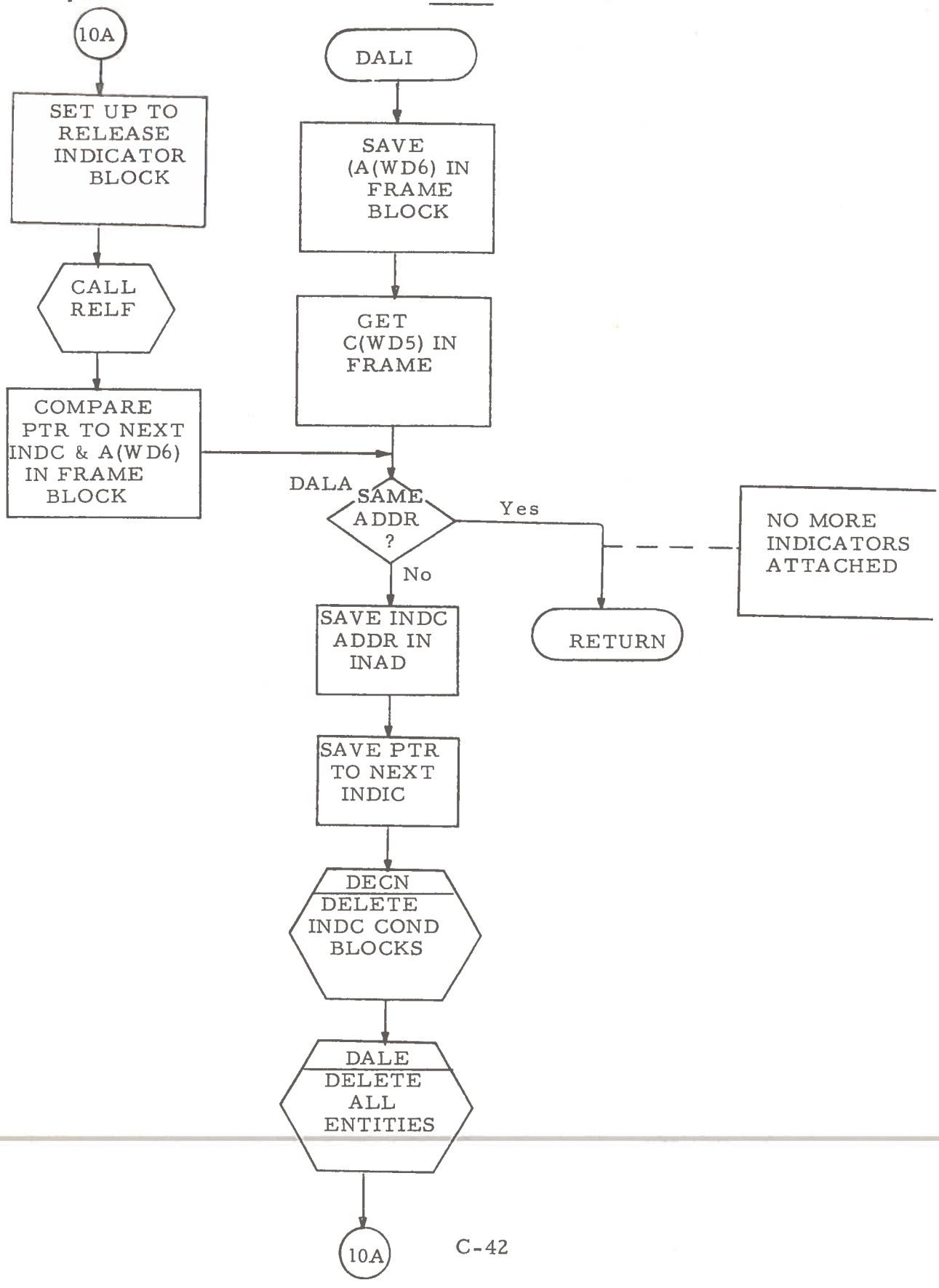
DLET



DENT - Delete Single Entity



DALI - Delete all Indicators in given frame
Input - Start Address of Frame in FRAD



SUBROUTINE: DOS

1. PURPOSE:

Return control to Disc Operating System.

2. CALLING SEQUENCE:

```
                JST*        PTR
PTR             DAC         COML+N
                 .
                 .
                 .
COML+N          XAC         DOS
```

3. INPUT:

None

4. OUTPUT:

None

5. ACTION:

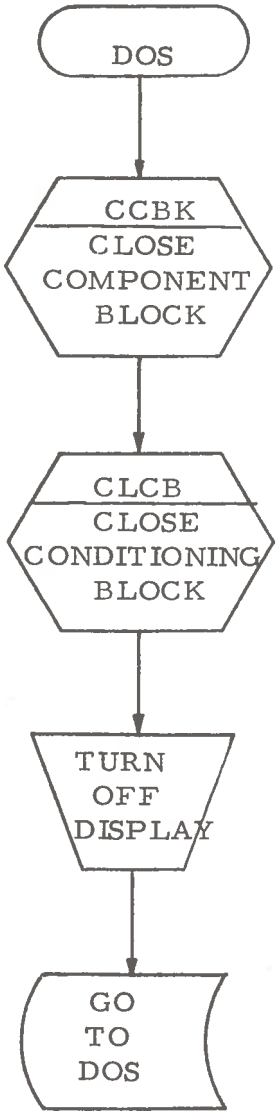
The working level is set to zero, and the component and conditioning blocks are closed. The display is shut off and control passes to the Disc Operating System.

6. EXTERNAL REFERENCES:

CLVL - Current Working Level.

7. CORE USED:

14_8


DOS
CCBK
CLOSE
COMPONENT
BLOCK
CLCB
CLOSE
CONDITIONING
BLOCK
TURN
OFF
DISPLAY
GO
TO
DOS

SUBROUTINE DREG

1. PURPOSE:

Set up Register Definition input for later analysis

2. CALLING SEQUENCE:

```
                JST*    PTR
                 :
PTR             DAC     COML+N
                 :
COML+N          XAC     DREG
```

3. INPUT:

a) Register to be defined in form Ri
where i=9, ..., 99

b) An expression defining the given register

4. OUTPUT:

The Register Definition is inserted into the Register Definition Block in the following format:

1	0	Register Number		
0	7-Bit ASCII CHAR	X	7-Bit ASCII CHAR	
0	⋮	X	⋮	
0	7-Bit ASCII CHAR	X	7-Bit ASCII CHAR or 0	

5. ACTION:

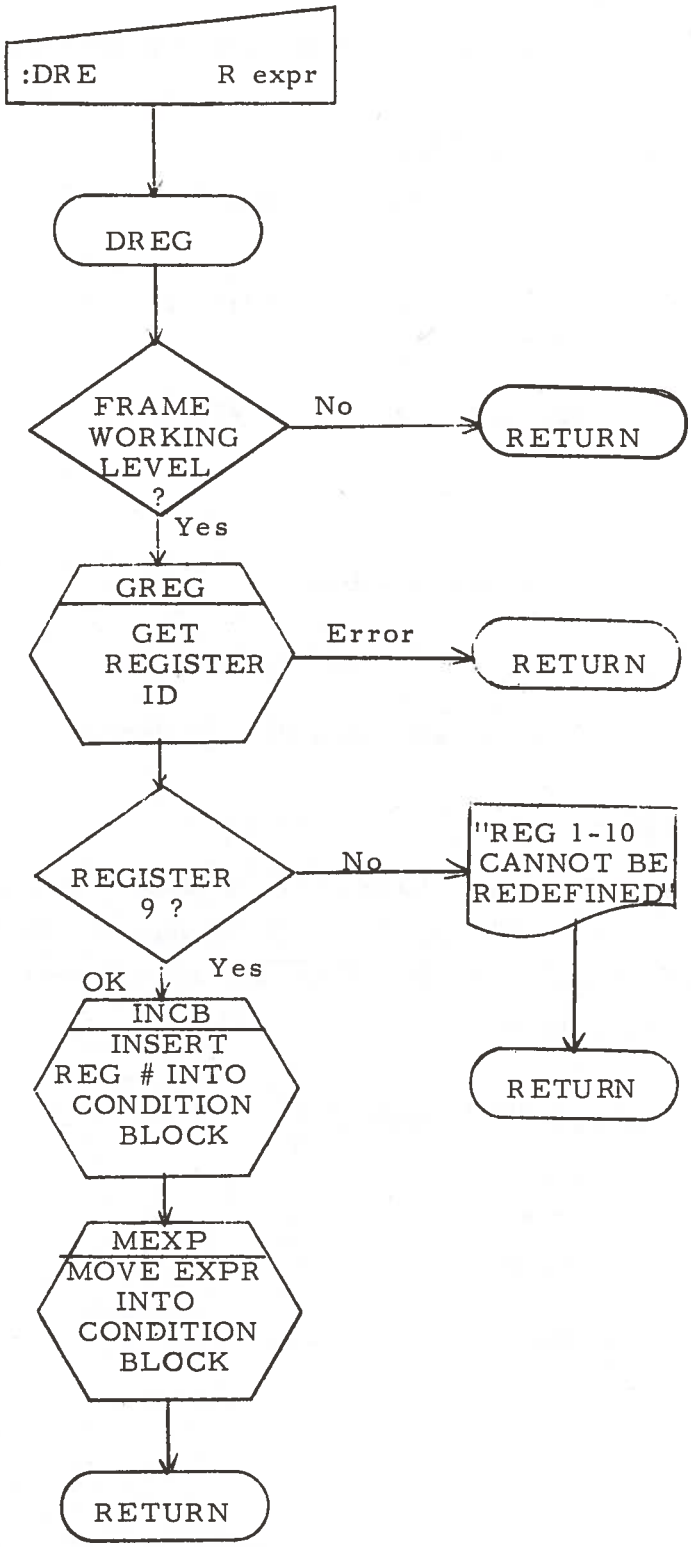
Checks for frame working level. Checks register to be defined. Ignores comma immediately following defined register, and spaces and tabs leading to expression. Moves expression, checking only for carriage return into Register Definition Block.

6. EXTERNAL REFERENCES:

MEXP
CMOD
GTIC
GTNO
WHAT
INCB

7. CORE USED:

42_8


:DRE R expr
DREG
FRAME WORKING LEVEL ?
No
RETURN
Yes
GREG
GET REGISTER ID
Error
RETURN
REGISTER 9 ?
No
"REG 1-10 CANNOT BE REDEFINED"
RETURN
OK
Yes
INCB
INSERT REG # INTO CONDITION BLOCK
MEXP
MOVE EXPR INTO CONDITION BLOCK
RETURN

SUBROUTINE ENTY

1. PURPOSE:

Get or create entity; set working level to entity

2. CALLING SEQUENCE:

```
          JST*    PTR
           :
PTR       DAC     COML+N
           :
COML+N    XAC     ENTY
```

3. INPUT:

ID from input buffer

4. OUTPUT:

Sets current entity ID and address

5. ACTION:

Retrieves ID from input buffer. Retrieves or creates entity. If retrieved, the entity display is intensified. The current entity ID and address are set and control returns to the calling program.

6. EXTERNAL REFERENCES:

CLVL
ENAD
ENID

7. CORE USED:

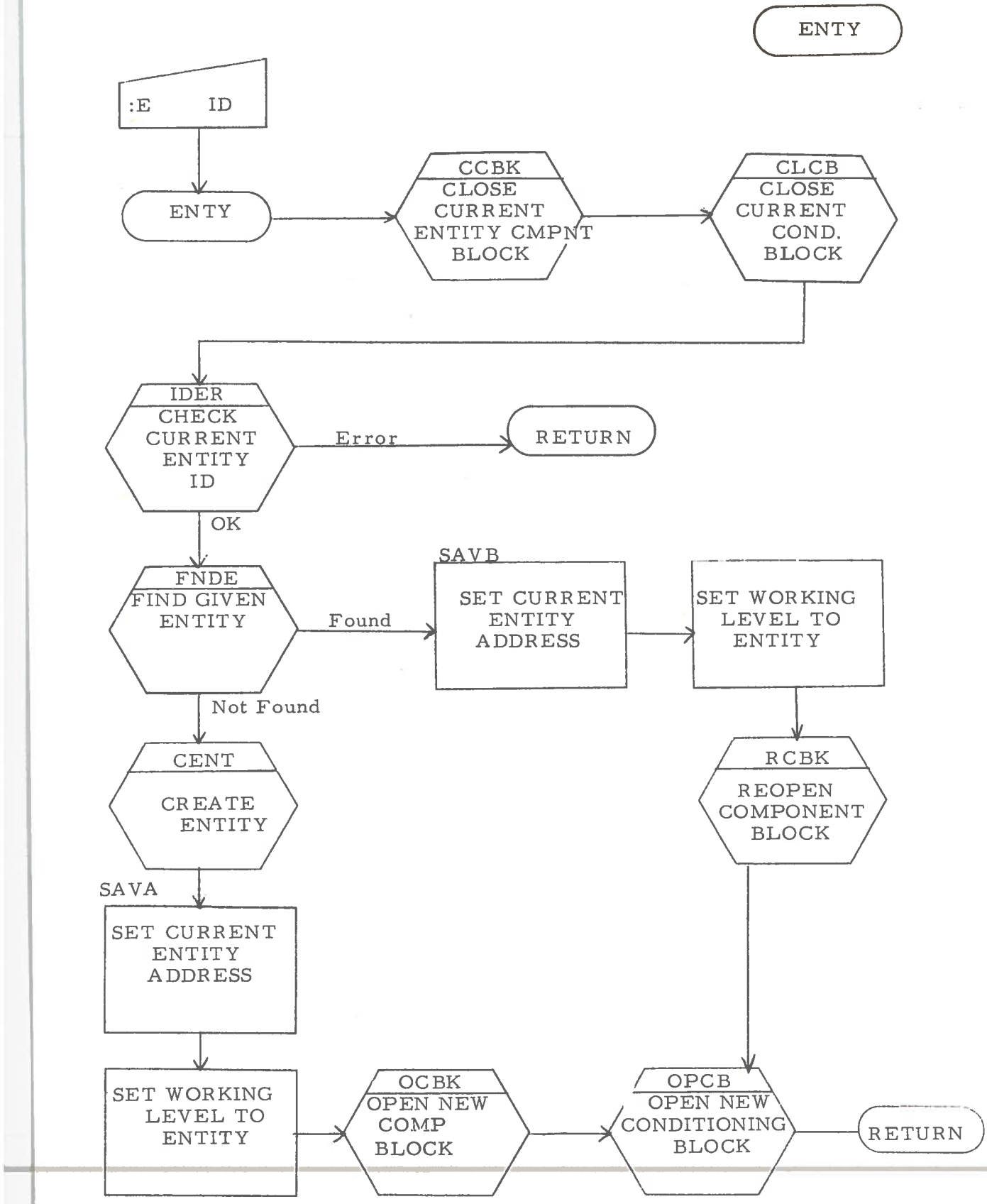
34_8

6. EXTERNAL REFERENCES:

IDER
FNDE
CENT
STEX
INEN
CLVL
ENID
ENAD
CCBK
OCBK
CLCB
OPCB

7. CORE USED:

32_8


ENTY
:E ID
ENTY
CCBK
CLOSE CURRENT ENTITY CMPNT BLOCK
CLCB
CLOSE CURRENT COND. BLOCK
IDER
CHECK CURRENT ENTITY ID
Error
RETURN
OK
FNDE
FIND GIVEN ENTITY
Found
SAVB
SET CURRENT ENTITY ADDRESS
SET WORKING LEVEL TO ENTITY
Not Found
CENT
CREATE ENTITY
RCBK
REOPEN COMPONENT BLOCK
SAVA
SET CURRENT ENTITY ADDRESS
SET WORKING LEVEL TO ENTITY
OCBK
OPEN NEW COMP BLOCK
OPCB
OPEN NEW CONDITIONING BLOCK
RETURN

SUBROUTINE: FRAM

1. PURPOSE:

Find frame or create new frame with given ID

2. CALLING SEQUENCE:

```
            JST*    PTR
             :
PTR         DAC     COML+N
             :
COML+N      XAC     FRAM
```

3. INPUT:

ID number retrieved from input buffer

4. OUTPUT:

Frame address set

5. ACTION:

ID number is retrieved. An attempt is made to locate a frame with the given ID. If successful, the frame address is saved; frame is displayed; and working level set to frame.

If unsuccessful, a new frame is created before returning. If storage is not available for the creation of a new frame, an appropriate message is typed before returning.

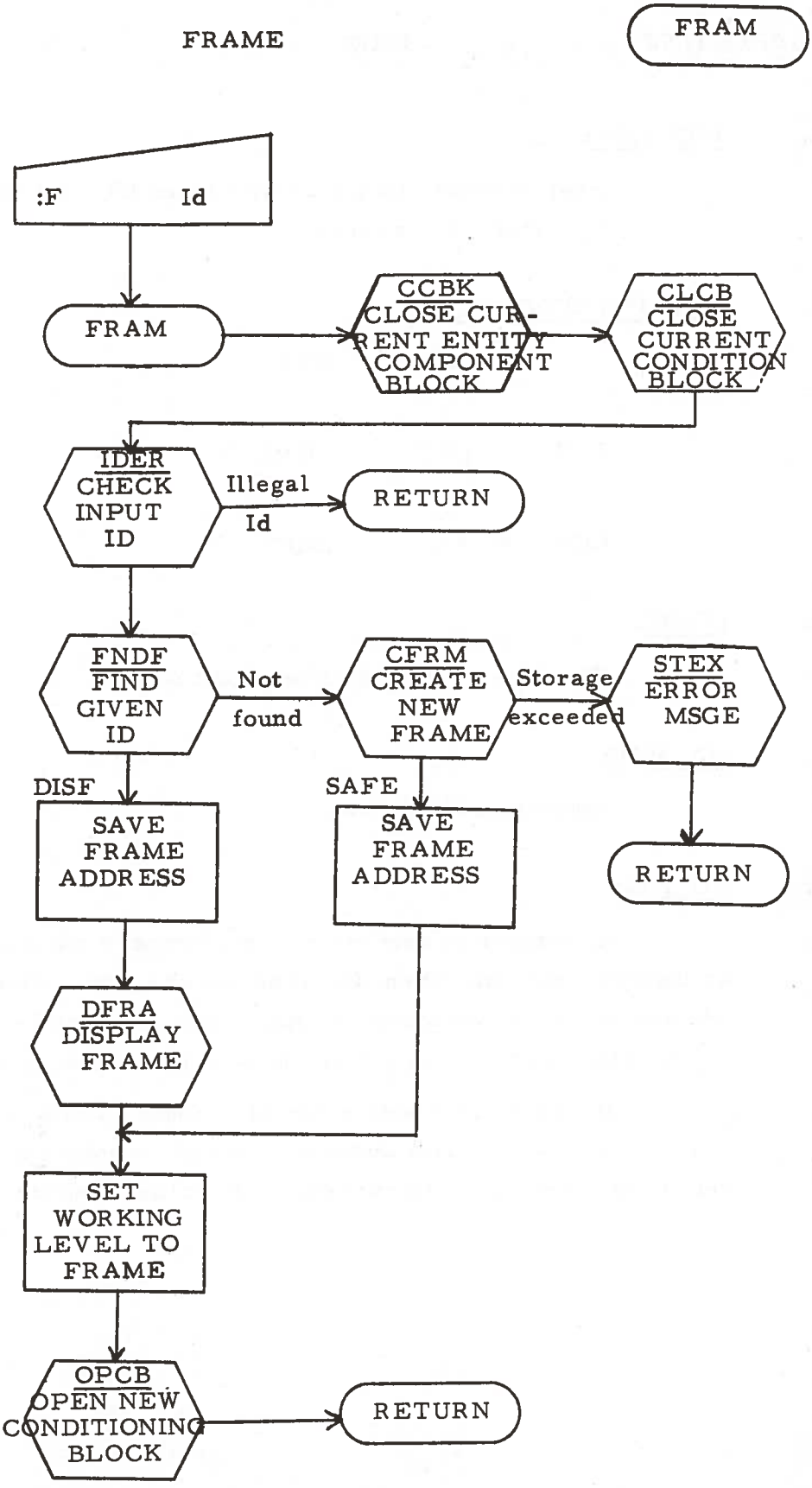
6. EXTERNAL REFERENCES:

IDER
FNDF
CFRM
STEX
DFRA
FRAD
FRID
CCBK
CLVL
OPCB
CLCB

7. CORE USED:

30_8

FRAME


FRAM
:F Id
FRAM
CCBK
CLOSE CURRENT ENTITY COMPONENT BLOCK
CLCB
CLOSE CURRENT CONDITION BLOCK
IDER
CHECK INPUT ID
Illegal Id
RETURN
FNDF
FIND GIVEN ID
Not found
CFRM
CREATE NEW FRAME
Storage exceeded
STEX
ERROR MSGE
DISF
SAFE
SAVE FRAME ADDRESS
SAVE FRAME ADDRESS
RETURN
DFRA
DISPLAY FRAME
SET WORKING LEVEL TO FRAME
OPCB
OPEN NEW CONDITIONING BLOCK
RETURN

SUBROUTINE INDC

1. PURPOSE:

Find or create indicator with given ID. Set to indicator working level.

2. CALLING SEQUENCE:

```
          JST*    PTR
           :
PTR       DAC     COML+N
           :
COML+N    XAC     INDC
```

3. INPUT:

ID number retrieved from input buffer

4. OUTPUT:

Indicator address set

5. ACTION:

ID number is retrieved. An attempt is made to locate an indicator with the given ID. If successful, the indicator address is saved; indicator is intensified; working level is set to indicator, and control returns to calling routine.

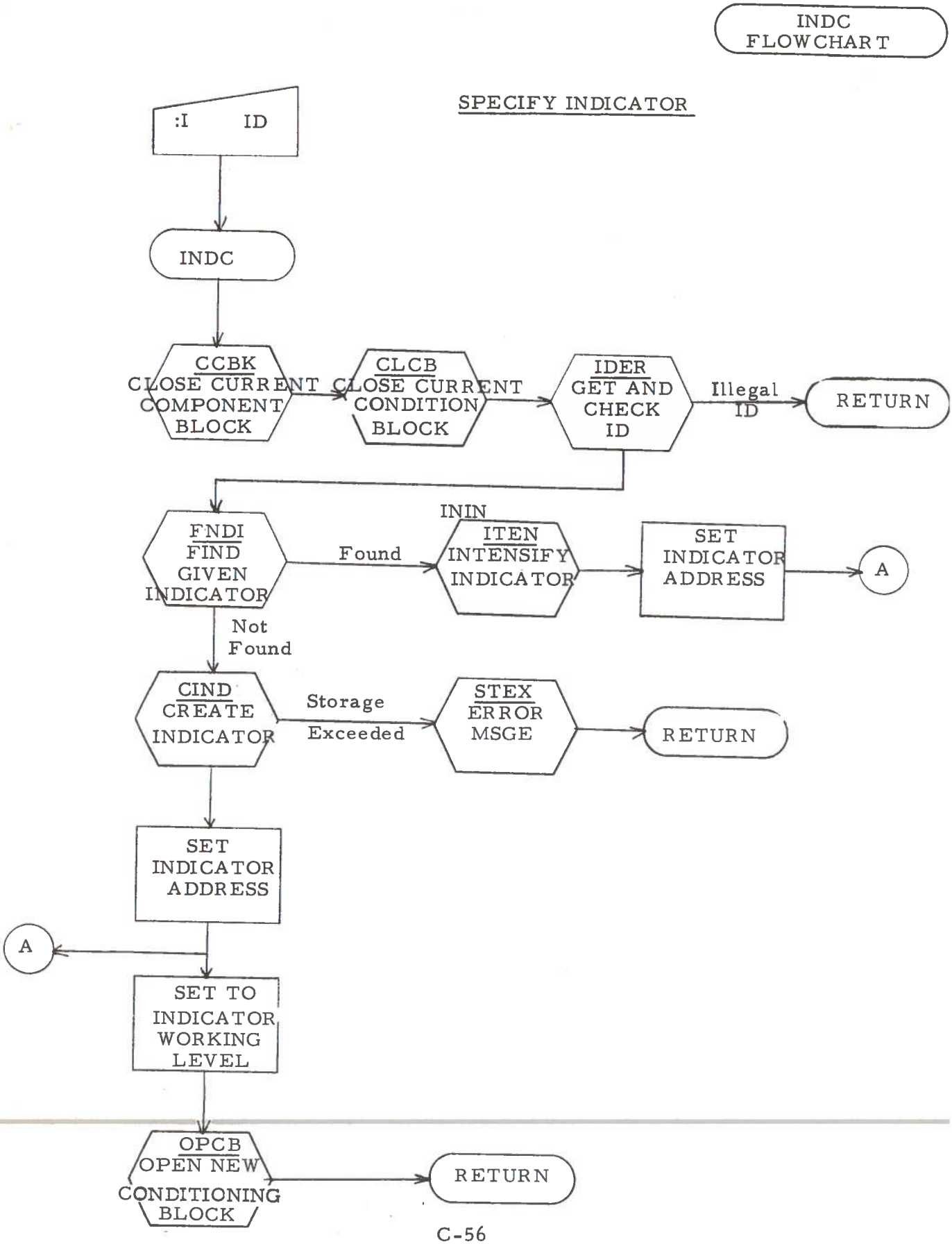
If no indicator with given ID is found, a new indicator is created; the indicator address is saved; working level is set to indicator, and control returns to calling routine.

6. EXTERNAL REFERENCES:

INAD
INID
CLVL

7. CORE USED:

27_8


INDC FLOWCHART
SPECIFY INDICATOR
:I ID
INDC
CCBK CLOSE CURRENT COMPONENT BLOCK
CLCB CLOSE CURRENT CONDITION BLOCK
IDER GET AND CHECK ID
Illegal ID
RETURN
FNDI FIND GIVEN INDICATOR
Found
ININ ITEN INTENSIFY INDICATOR
SET INDICATOR ADDRESS
A
Not Found
CIND CREATE INDICATOR
Storage Exceeded
STEX ERROR MSGE
RETURN
SET INDICATOR ADDRESS
A
SET TO INDICATOR WORKING LEVEL
OPCB OPEN NEW CONDITIONING BLOCK
RETURN

ROTT

1. PURPOSE:

Set up dynamic expressions for entity rotation

2. CALLING SEQUENCE:

```
          JST*    PTR
           :
PTR       DAC     COML+N
           :
COML+N    XAC     ROTT
```

3. INPUT:

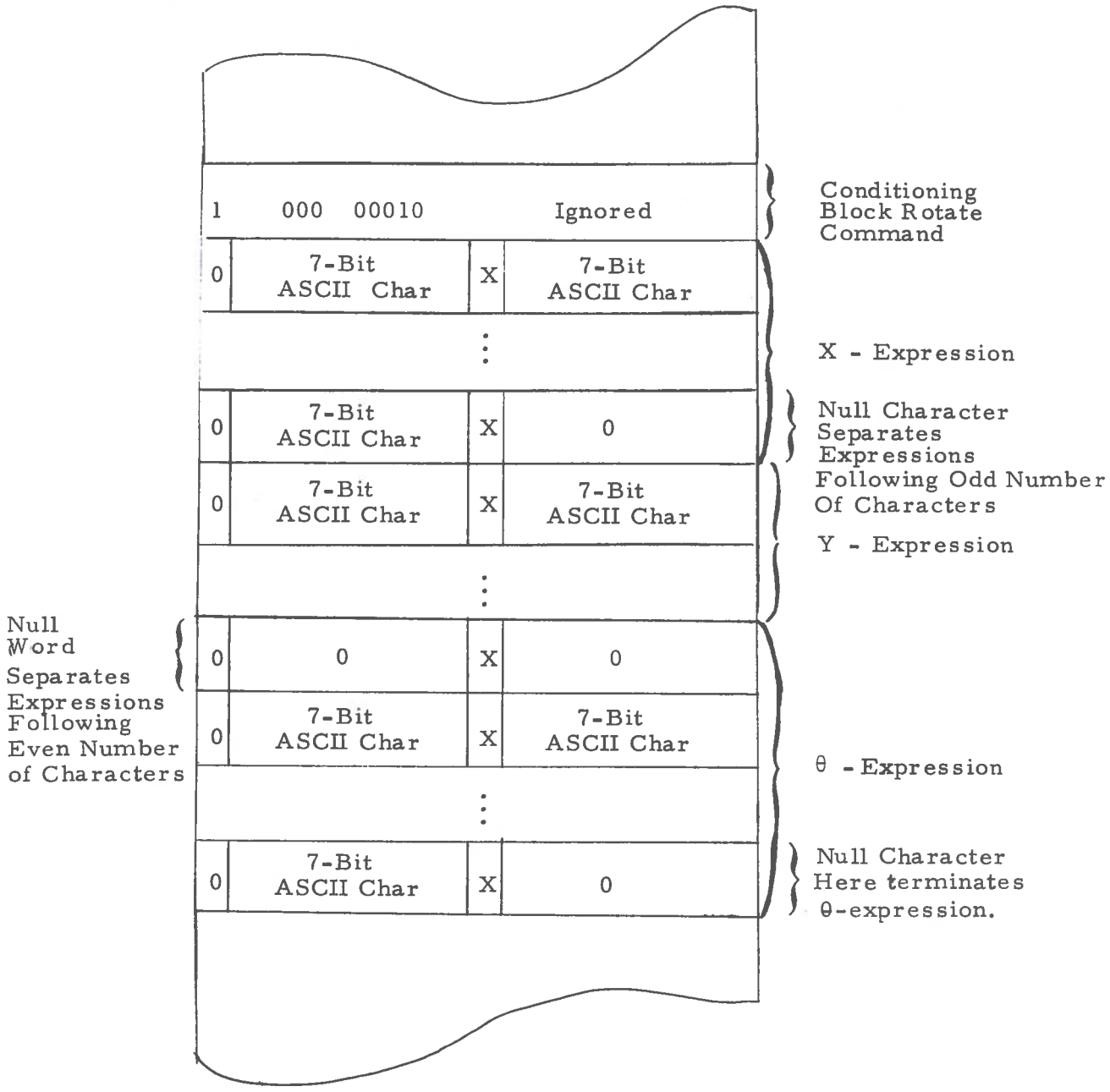
Dynamic expressions for x, y, and Θ from input buffer

4. OUTPUT:

Expression packed two ASCII characters per word in conditioning block for current entity

5. ACTION:

Moves expressions into conditioning block. Inserts null word if no null characters packed by move routine, MEXP. Set up of expressions in conditioning block is Δ x. Then Δ y followed by Θ, as below

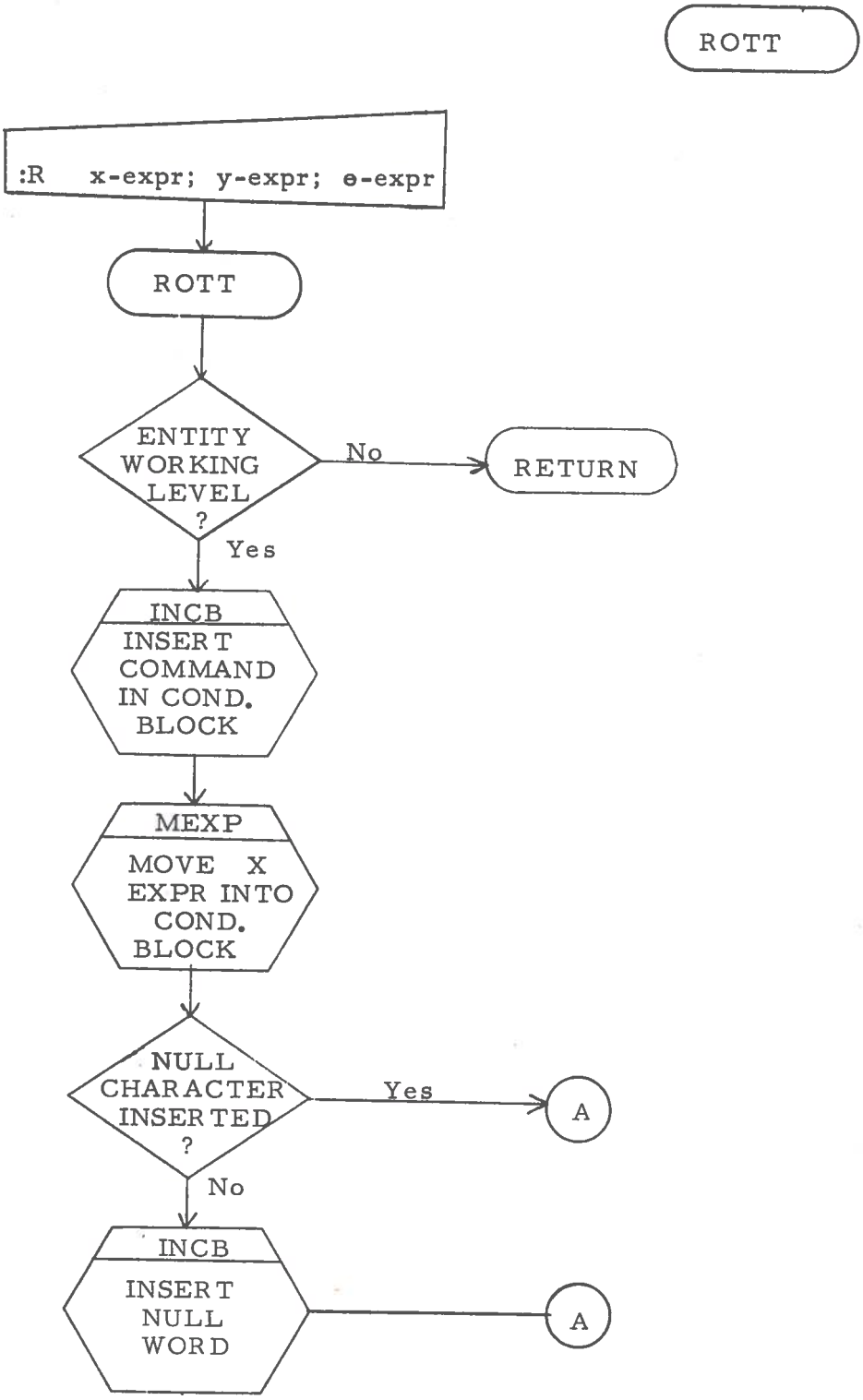

1 000 00010 Ignored
Conditioning Block Rotate Command
0 7-Bit ASCII Char X 7-Bit ASCII Char
X - Expression
0 7-Bit ASCII Char X 0
Null Character Separates Expressions Following Odd Number Of Characters
0 7-Bit ASCII Char X 7-Bit ASCII Char
Y - Expression
Null Word Separates Expressions Following Even Number of Characters
0 0 X 0
0 7-Bit ASCII Char X 7-Bit ASCII Char
θ - Expression
0 7-Bit ASCII Char X 0
Null Character Here terminates θ-expression.

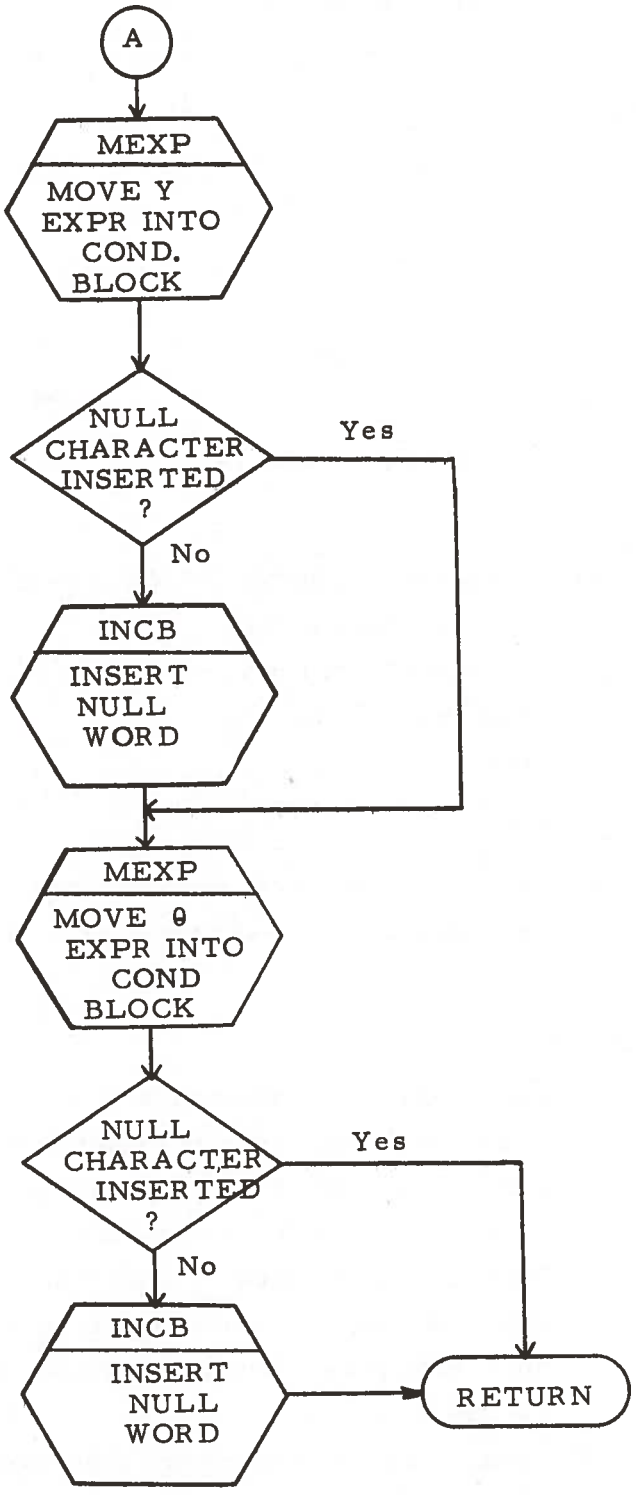
6. EXTERNAL REFERENCES:

CMOD
INCB
MEXP

7. CORE USED:

22_8


ROTT
:R x-expr; y-expr; e-expr
ROTT
ENTITY WORKING LEVEL ?
No
RETURN
Yes
INCB
INSERT COMMAND IN COND. BLOCK
MEXP
MOVE X EXPR INTO COND. BLOCK
NULL CHARACTER INSERTED ?
Yes
A
No
INCB
INSERT NULL WORD
A


A
MEXP
MOVE Y EXPR INTO COND. BLOCK
NULL CHARACTER INSERTED ?
Yes
No
INCB
INSERT NULL WORD
MEXP
MOVE θ EXPR INTO COND BLOCK
NULL CHARACTER INSERTED ?
Yes
No
INCB
INSERT NULL WORD
RETURN

SUBROUTINE: SAVE, RLOD, INIT

1. PURPOSE:

   a) Save data structure on disc or paper tape
   b) Reload data structure from disc or paper tape
   c) Initialize data structure; ie, set up free storage ring

2. CALLING SEQUENCE:

```
                    JST*            PTR
                     .
                     .
                     .
      PTR           DAC             COML+N
   COML+N           XAC             PROG
```

   Where PROG = SAVE, RLOD, INIT

3. INPUT:

   a) File name, output device (disc or paper tape) and phase from the input buffer.
   b) File name, input device (disc or paper tape) and phase from the input buffer.
   c) None

4. OUTPUT:

   a) Data structure saved on disc or paper tape
   b) Data structure reloaded from disc or paper tape
   c) None

5. ACTION:

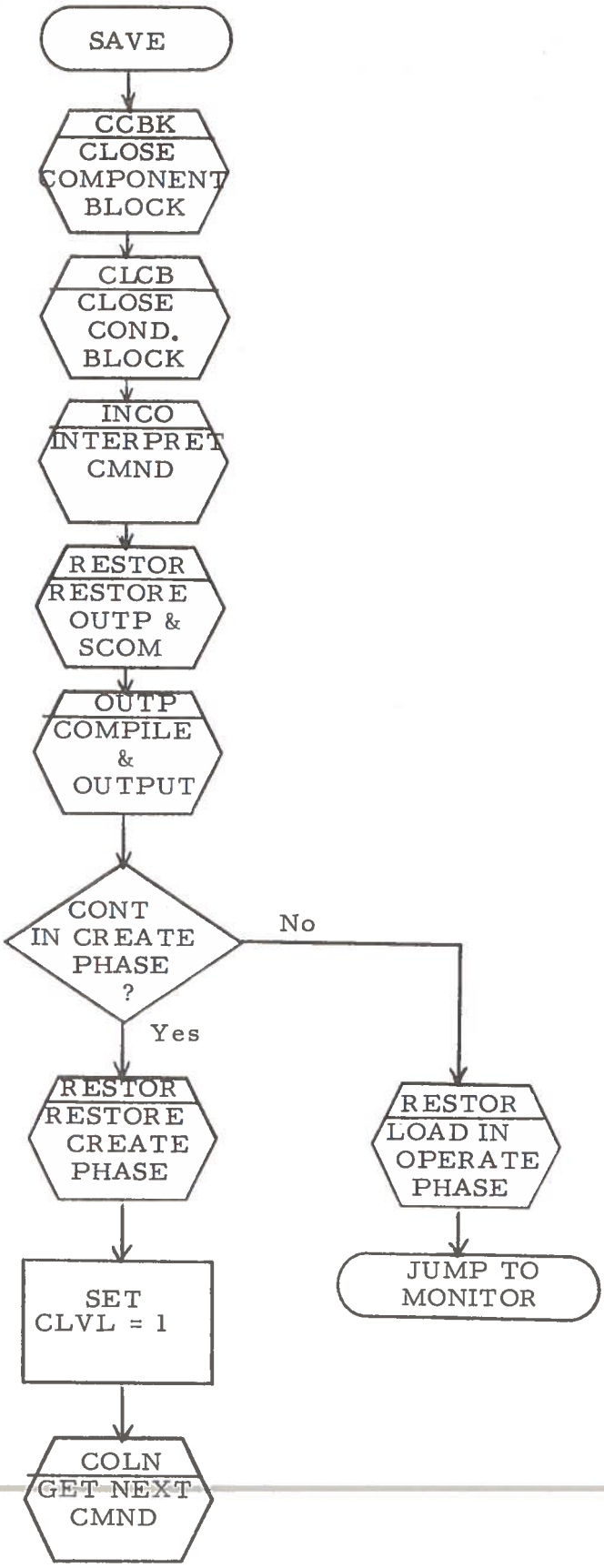
   a) The conditioning commands in the data structure are compiled and the data structure is output to the specified device. Control will pass to either the create phase or operate phase depending upon the "C" or "O" parameter.
   b) Free storage is initialized and the data structure is read in from either the disc or paper tape reader. Control will pass to either the create phase or operate phase, depending upon the "C" or "O" parameter.
   c) Pointers are initialized and free storage is attached to the free storage ring using 100_8 word blocks.

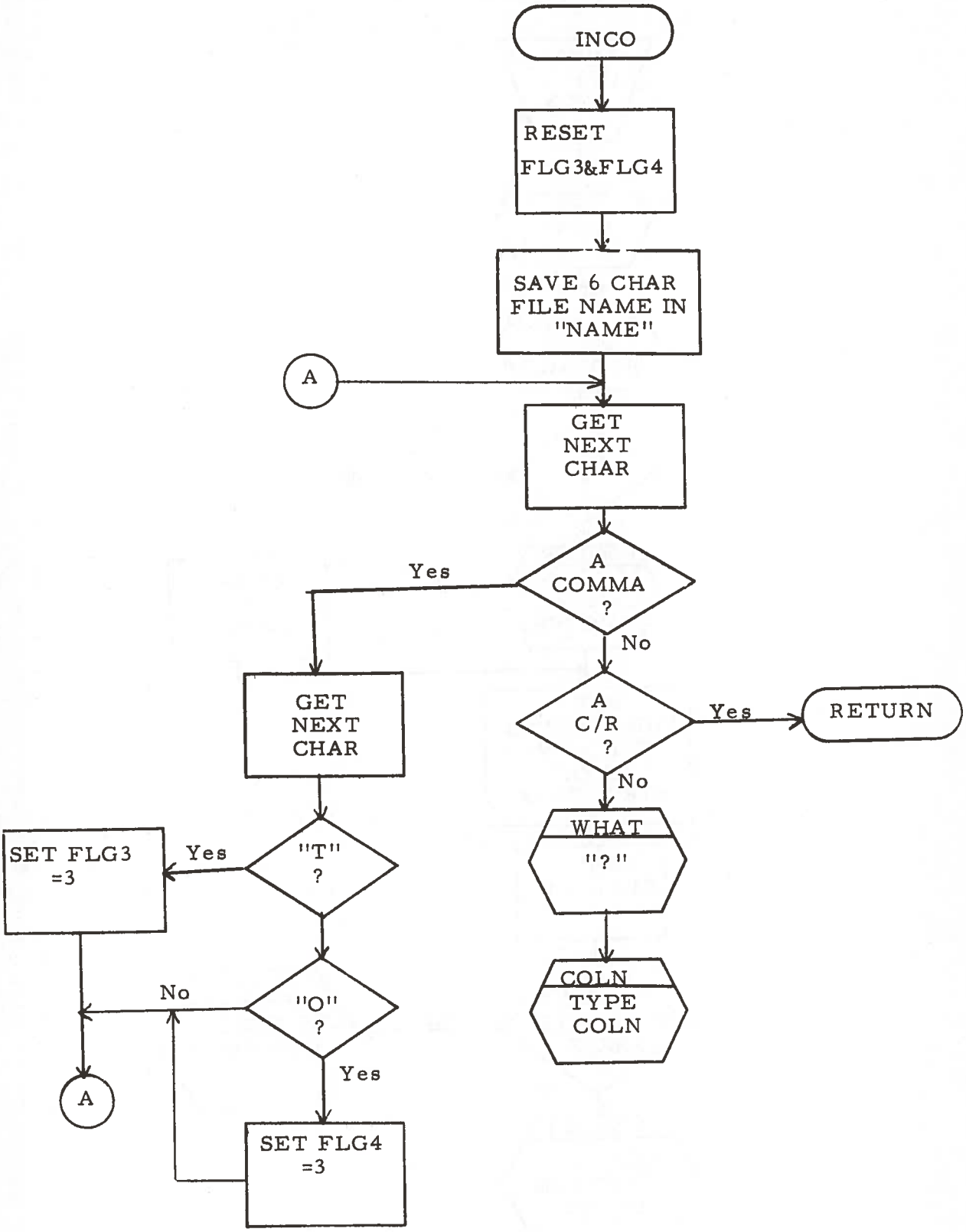
6. EXTERNAL REFERENCES:

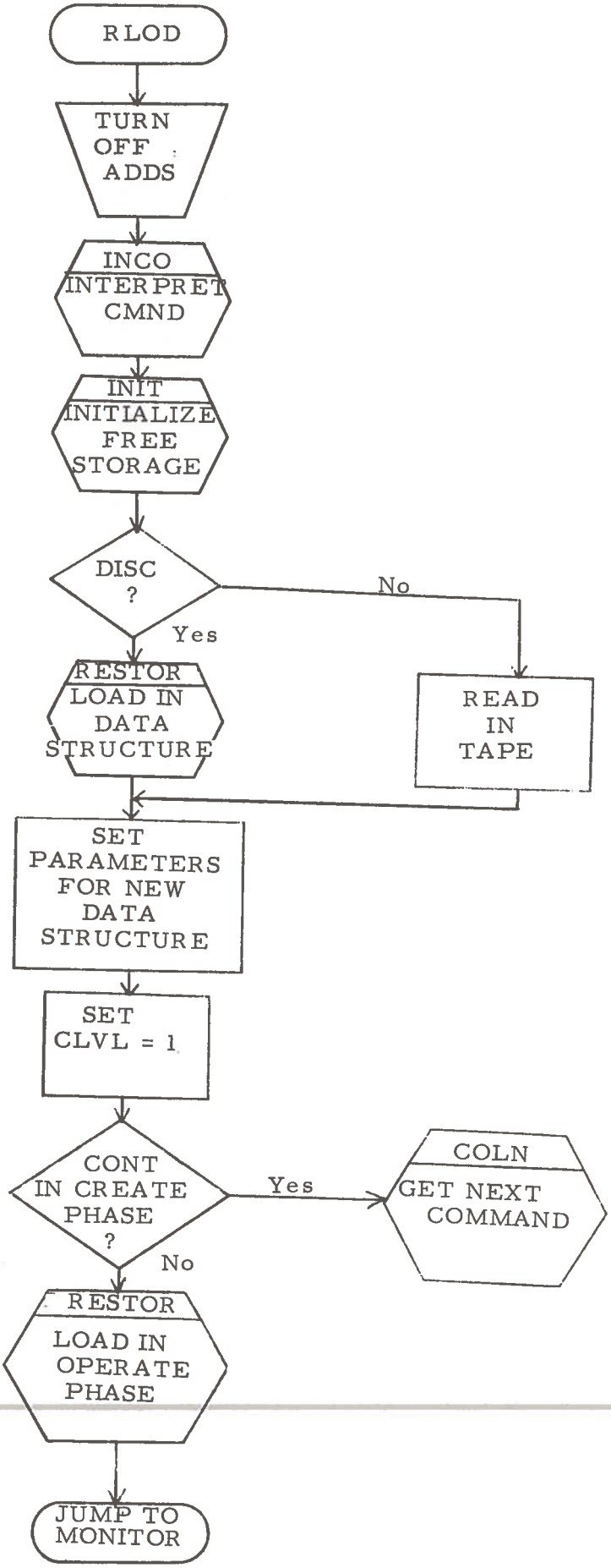
GETF

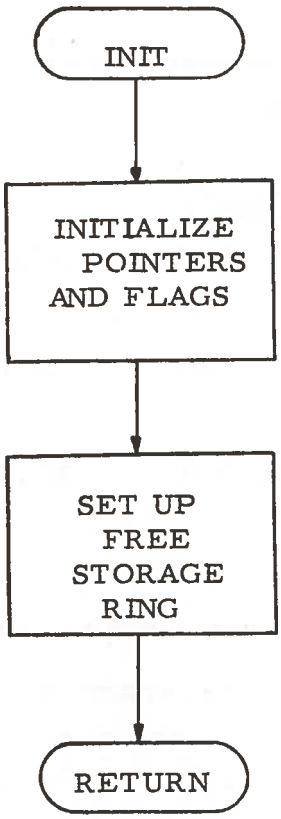
7. CORE USED:

413_8


SAVE
CCBK
CLOSE
COMPONENT
BLOCK
CLCB
CLOSE
COND.
BLOCK
INCO
INTERPRET
CMND
RESTOR
RESTORE
OUTP &
SCOM
OUTP
COMPILE
&
OUTPUT
CONT
IN CREATE
PHASE
?
No
Yes
RESTOR
RESTORE
CREATE
PHASE
RESTOR
LOAD IN
OPERATE
PHASE
SET
CLVL = 1
JUMP TO
MONITOR
COLN
GET NEXT
CMND


INCO
RESET FLG3&FLG4
SAVE 6 CHAR FILE NAME IN "NAME"
A
GET NEXT CHAR
A COMMA ?
Yes
No
GET NEXT CHAR
A C/R ?
Yes
RETURN
No
WHAT "?"
COLN TYPE COLN
"T" ?
Yes
SET FLG3 =3
"O" ?
No
Yes
A
SET FLG4 =3


RLOD
TURN OFF ADDS
INCO
INTERPRET CMND
INIT
INITIALIZE FREE STORAGE
DISC ?
No
Yes
RESTOR
LOAD IN DATA STRUCTURE
READ IN TAPE
SET PARAMETERS FOR NEW DATA STRUCTURE
SET CLVL = 1
CONT IN CREATE PHASE ?
Yes
COLN
GET NEXT COMMAND
No
RESTOR
LOAD IN OPERATE PHASE
JUMP TO MONITOR


INIT
INITIALIZE POINTERS AND FLAGS
SET UP FREE STORAGE RING
RETURN

SUBROUTINE SCAL

1. PURPOSE:

Insert commands to draw numeric labels (scales) for lines

2. CALLING SEQUENCE:

```
          JST*    PTR
           :
PTR       DAC     COML+N
           :
COML+N    XAC     SCAL
```

3. INPUT:

The following from teletype input buffer

a) x_i x-component; Initial point relative
b) y_i y-component; Initial point relative
c) x_f x-component; Final point relative
d) y_f y-component; Final point relative
e) A Initial Value of Scale
f) B Final Value of Scale
g) N Number of labels
h) SIZE Character size

4. OUTPUT:

Display commands are inserted into the current component block to draw and position scale labels.

5 ACTION:

Checks working level and inputs x_i, y_i, x_f, y_f, A, B, N, and SIZE. Returns after calling WHAT if early carriage return encountered. Calculates x, y position increments:

$$\Delta x = \frac{x_f - x_i}{N-1} \quad , \quad \Delta y = \frac{y_f - y_i}{N-1}$$

and value increment:

$$\Delta S = \frac{B-A}{N-1}$$

Inserts size command and sets up spacing according to the following table:

SIZE	SPACING (raster units)
1	15
2	30
3	45
4	60

Calculates position of leftmost character relative to value center point and moves to that point. Set command to enter character mode, inserts characters, and returns to coordinate mode. Either the next scale value is then set up or a blank move to bring beam to (x_f, y_f) relative to original position is set up and control is returned to calling program.

6. EXTERNAL REFERENCES:

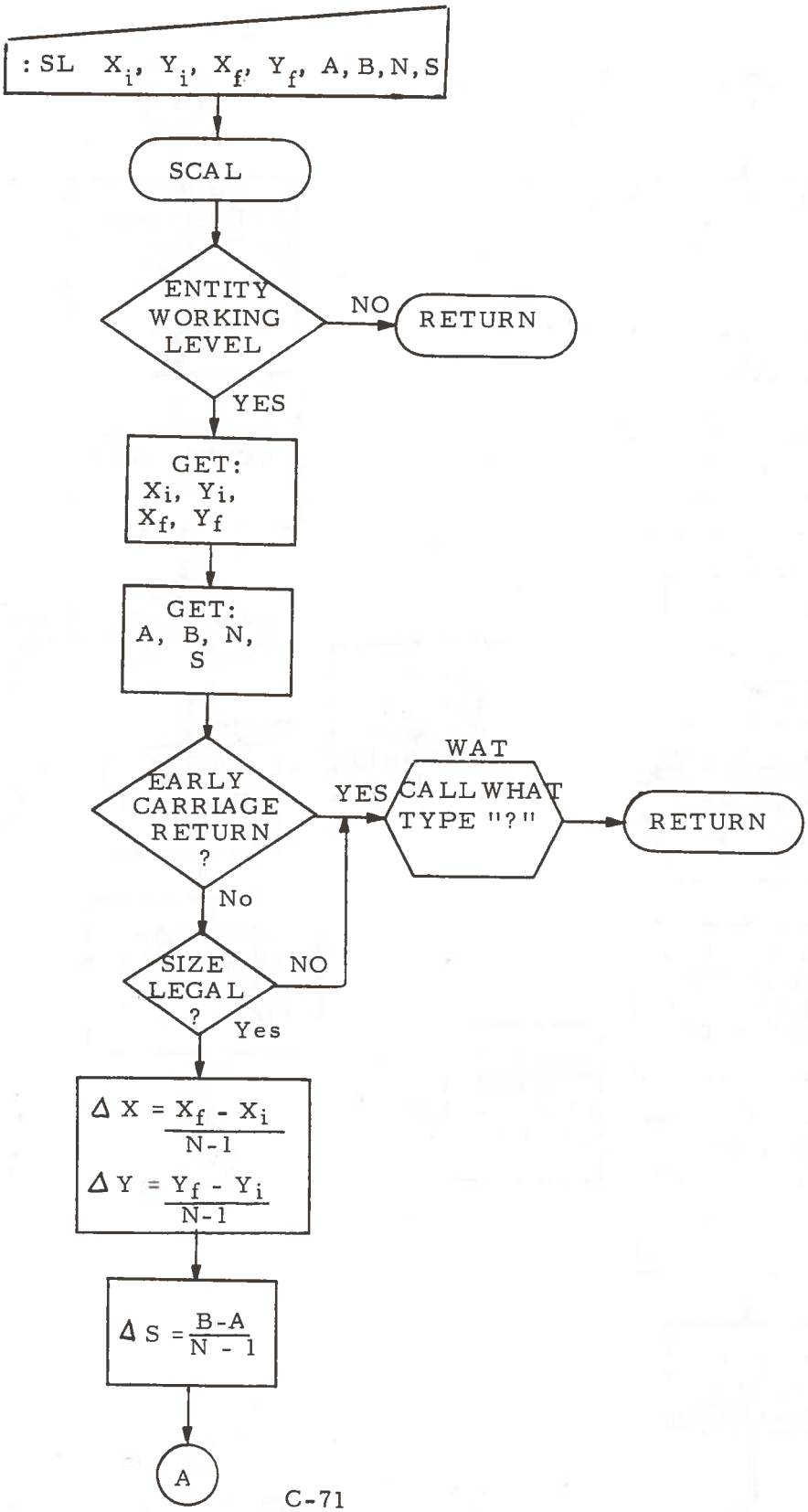
CMOD
GTNO
NONO
WHAT
COP
CIN
CHAD
CLO

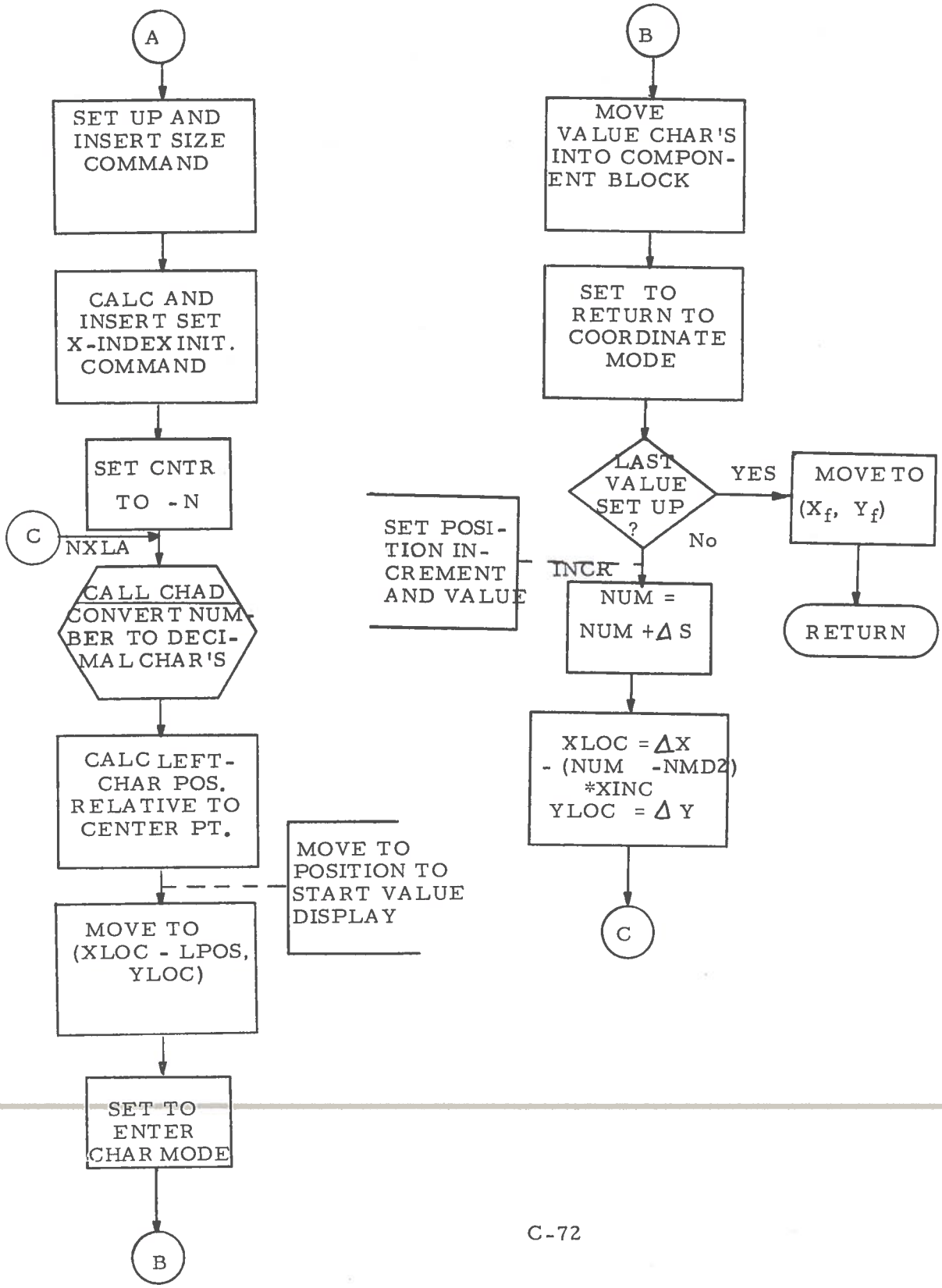
7. CORE USED:

257_8

Scale for Lines

SCAL Flowchart




A
SET UP AND INSERT SIZE COMMAND
CALC AND INSERT SET X-INDEX INIT. COMMAND
SET CNTR TO -N
C
NXLA
CALL CHAD CONVERT NUMBER TO DECIMAL CHAR'S
CALC LEFT-CHAR POS. RELATIVE TO CENTER PT.
MOVE TO POSITION TO START VALUE DISPLAY
MOVE TO (XLOC - LPOS, YLOC)
SET TO ENTER CHAR MODE
B
B
MOVE VALUE CHAR'S INTO COMPONENT BLOCK
SET TO RETURN TO COORDINATE MODE
LAST VALUE SET UP ?
YES
MOVE TO (X_f, Y_f)
RETURN
No
SET POSITION INCREMENT AND VALUE
INCR
NUM = NUM + Δ S
XLOC = ΔX - (NUM -NMD2) *XINC
YLOC = Δ Y
C

SUBROUTINE SCAR

1. PURPOSE:

Set up commands to display scaled values for arcs

2. CALLING SEQUENCE:

```
          JST*     PTR
           .
           .
           .
PTR       DAC      COML+N
           .
           .
           .
COML+N    XAC      SCAR
```

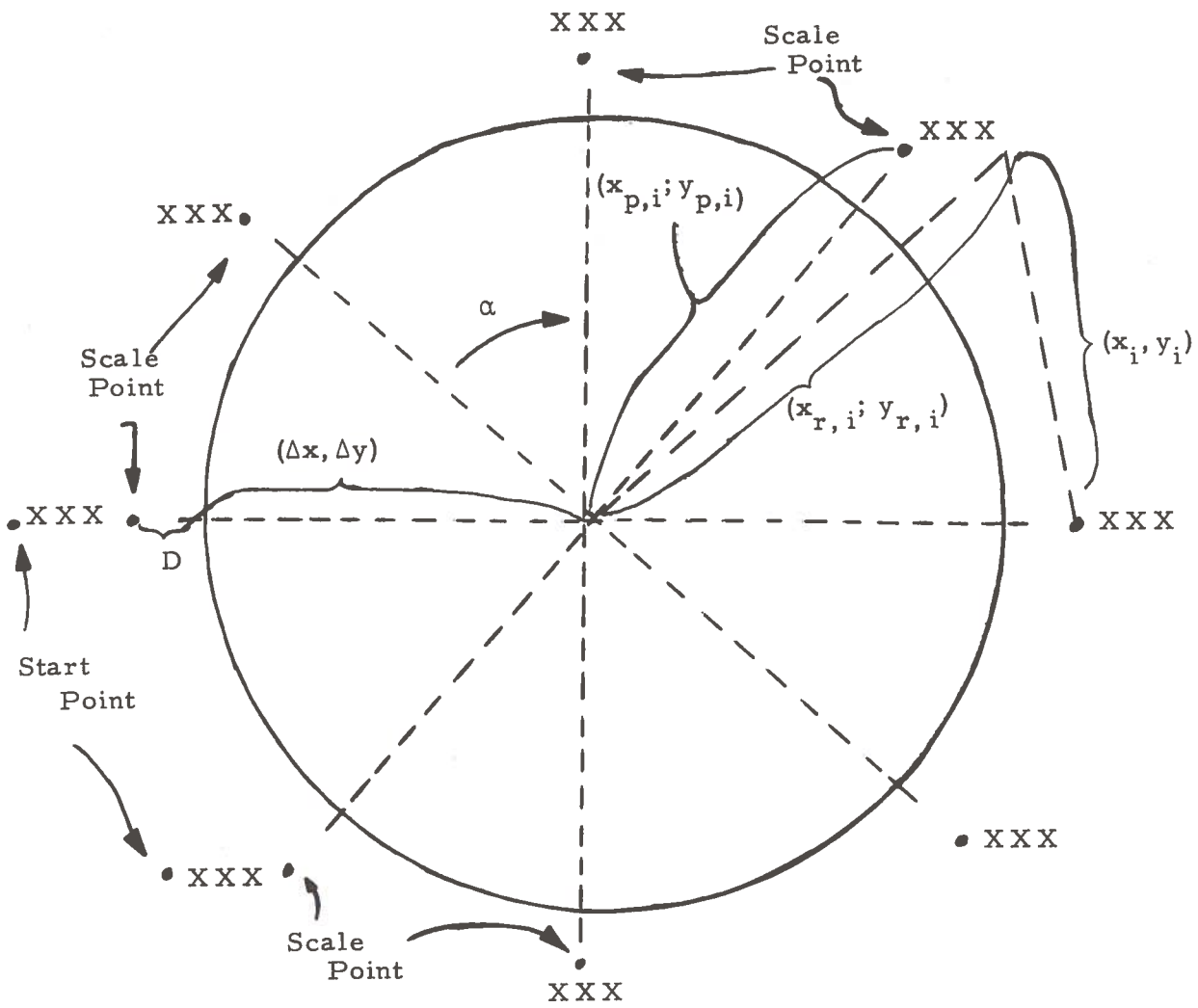
3. INPUT:

From the teletype buffer:

Δx	x-component, start position on arc
Δy	y-component, start position on arc
A	start value
B	end value
N	number of values
SIZE	character size; range = 1, 2, 3, 4
D	distance from arc to scale point along radius
θ	angle of arc; (drawn clockwise for positive θ ; default gives full circle)

4. OUTPUT:

Display commands are inserted into component block.



Positioning of Scale Values on Arc

Figure 1.

5. ACTION:

Scale values are set up for display as follows (see Figure 1):

i) Calculate initial scale point $(X_{p,i}, Y_{p,i})$

ii) Apply offset for size and number of characters and for position along arc, to get start point (X_i, Y_i).

iii) Move to (X_i, Y_i) and display digits

iv) Calculate return vector $(X_{r,i}, Y_{r,i})$

v) Calculate new scale point $(X_{p,i}, Yp,i)$

vi) Go to ii

When all scale values are set for display, a command to move back to the center of the arc using the current return vector is generated.

Terms used for calculations are:

NC = Number of char's for current scale value

CS = Horizontal character spacing in raster units

CSY = Vertical character spacing in raster units

α = Angle between scale points as fraction of circle $*2^{15}$

$$\alpha = \frac{-0*2^{15}}{360\ (N-1),} \quad , \text{ if } \theta \neq 0$$

$$\alpha = \frac{-2^{15}}{N} \quad , \text{ if } \theta \text{ (input)} = 0 \qquad (1)$$

In equation (1), the effect is to set $\theta = 360$ and to eliminate repetition of scaling at the first scale point.

ΔR = length of radius $= \sqrt{\Delta x^2 + \Delta y^2}$

R = distance from center to scale point = $D + \Delta R$

For $i = 1$;

$$x_{p,i} = R \cdot \frac{\Delta x}{\Delta R} \quad ; \quad y_{p,i} = R \cdot \frac{\Delta y}{\Delta R}$$

For $i = 2, \ldots, N$

$$x_{p,i} = x_{p,i-1} \cos\alpha - y_{p,i-1} \sin\alpha$$

$$y_{p,i} = x_{p,i-1} \sin\alpha + y_{p,i-1} \cos\alpha$$

For $i = 1, \ldots, N$

$$x_i = x_{p,i} + (x_{p,i} - R) \left[\frac{NC*CS}{2} - 2*SIZE\right] \frac{1}{R} + x_{r,i-1}$$

$$y_i = y_{p,i} + (y_{p,i} - R) \quad \frac{CSY}{2} \quad \frac{1}{R} + y_{r,i-1}$$

$$x_{r,i} = x_{r,i-1} - (x_i + NC*CS)$$

$$y_{r,i} = y_{r,i-1} - y_i$$

Where $x_{r,o} = y_{r,o} = 0$

Finally, the beam is returned to the center of the arc, $(x_{r,n}, y_{r,n})$

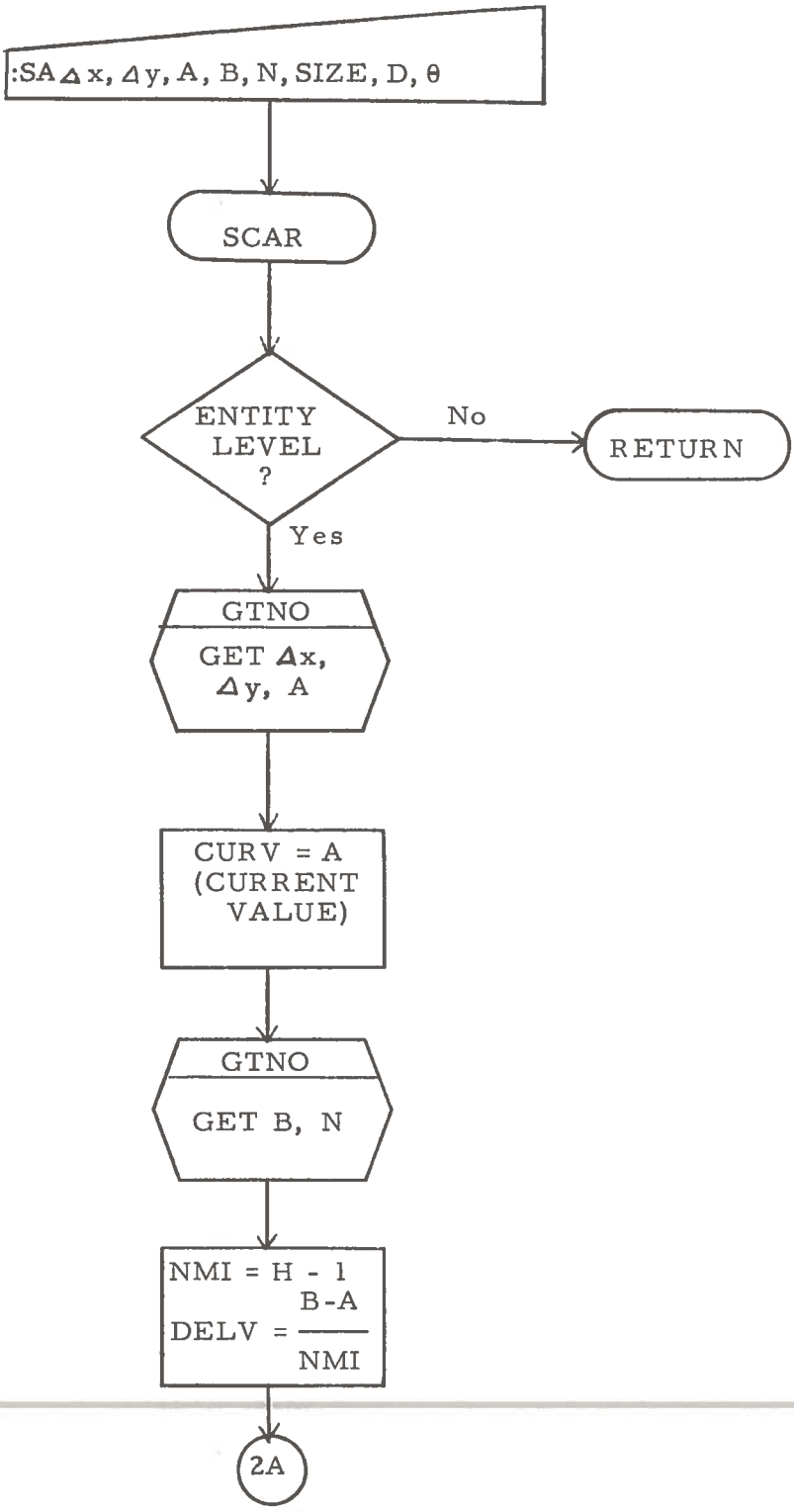
6. EXTERNAL REFERENCES

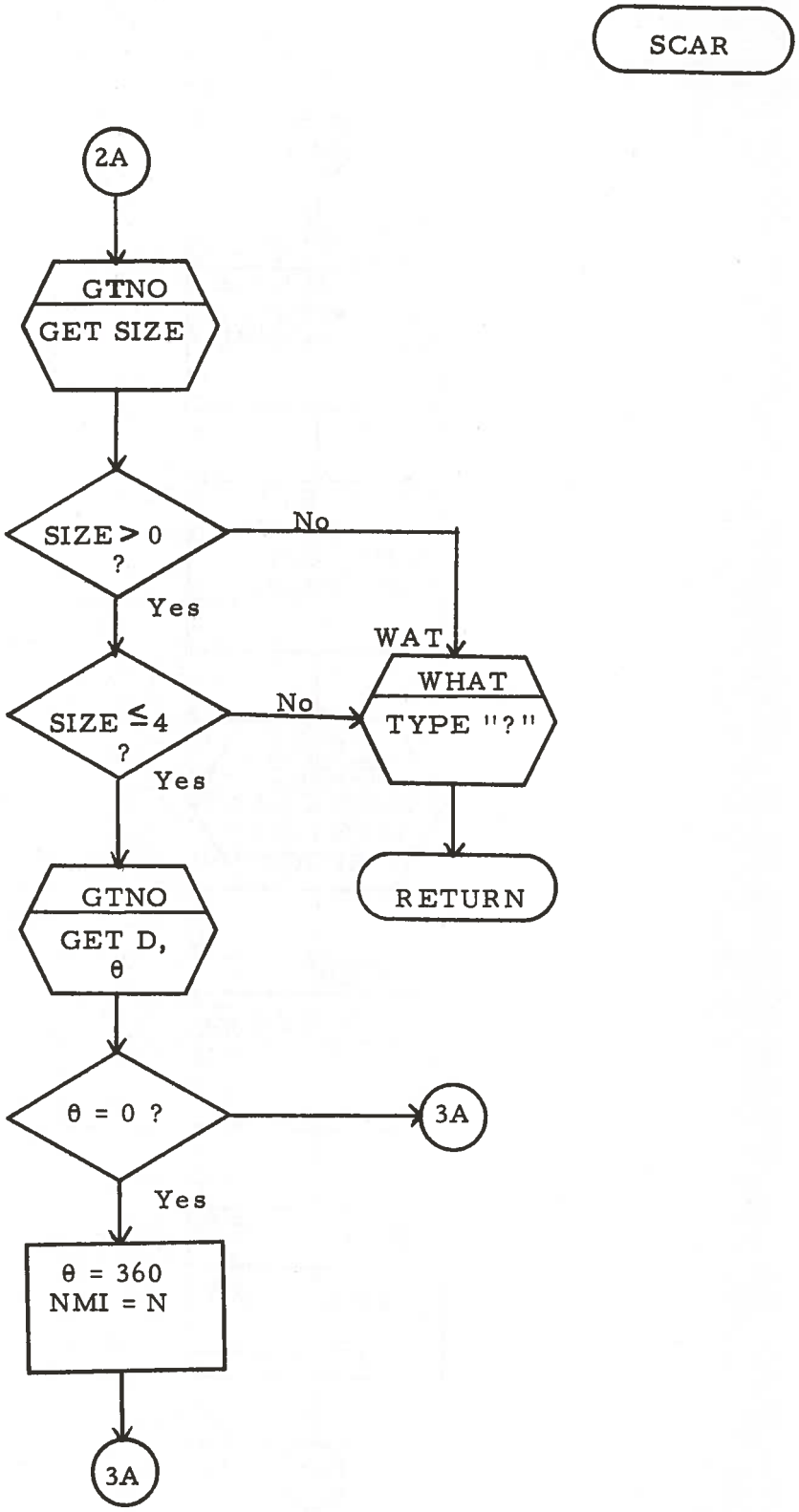
CMOD, GTNO, WHAT, SINX2, COSX2, COP, CIN, SQRJAM, CHAD, CLO

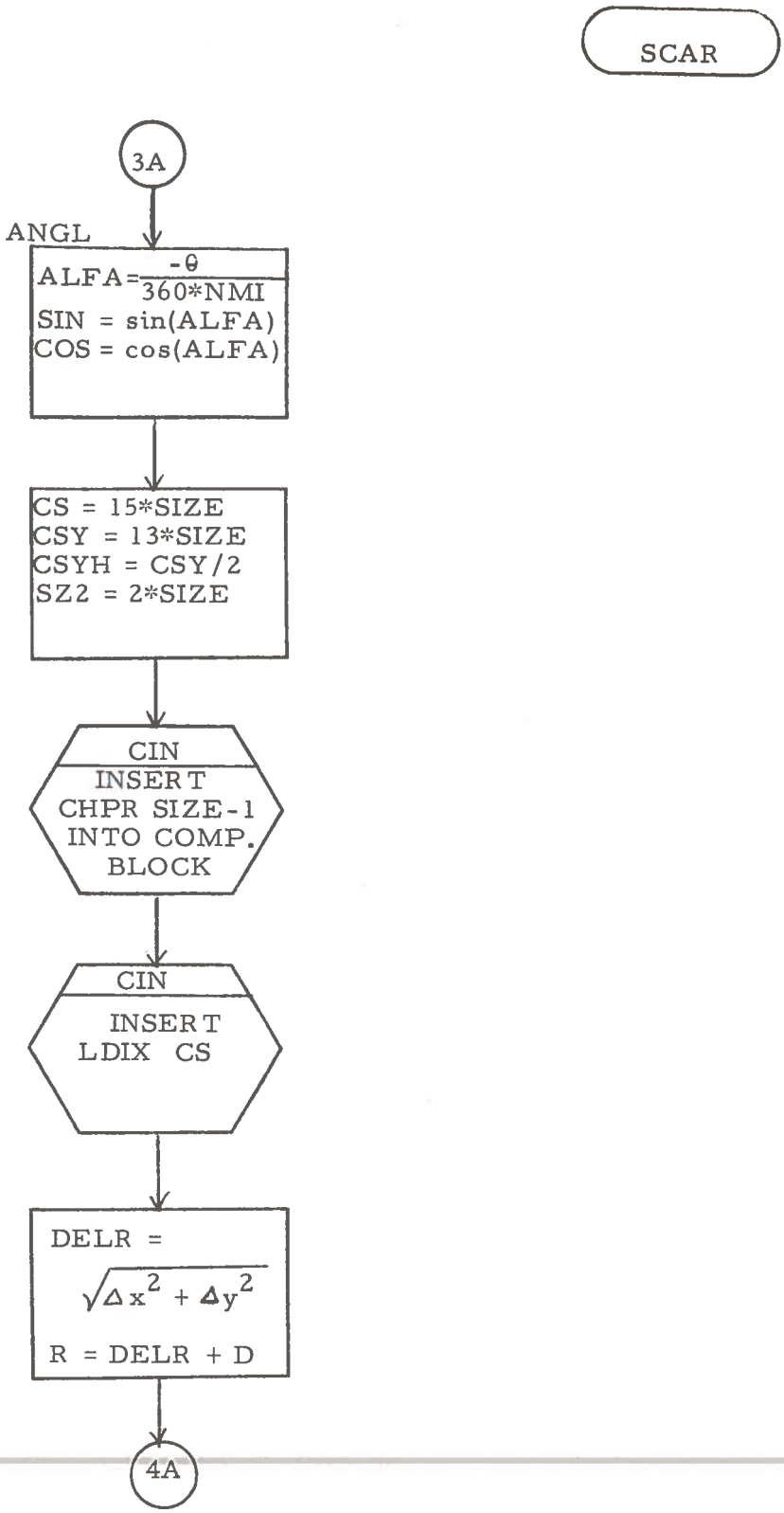
7. CORE USED:

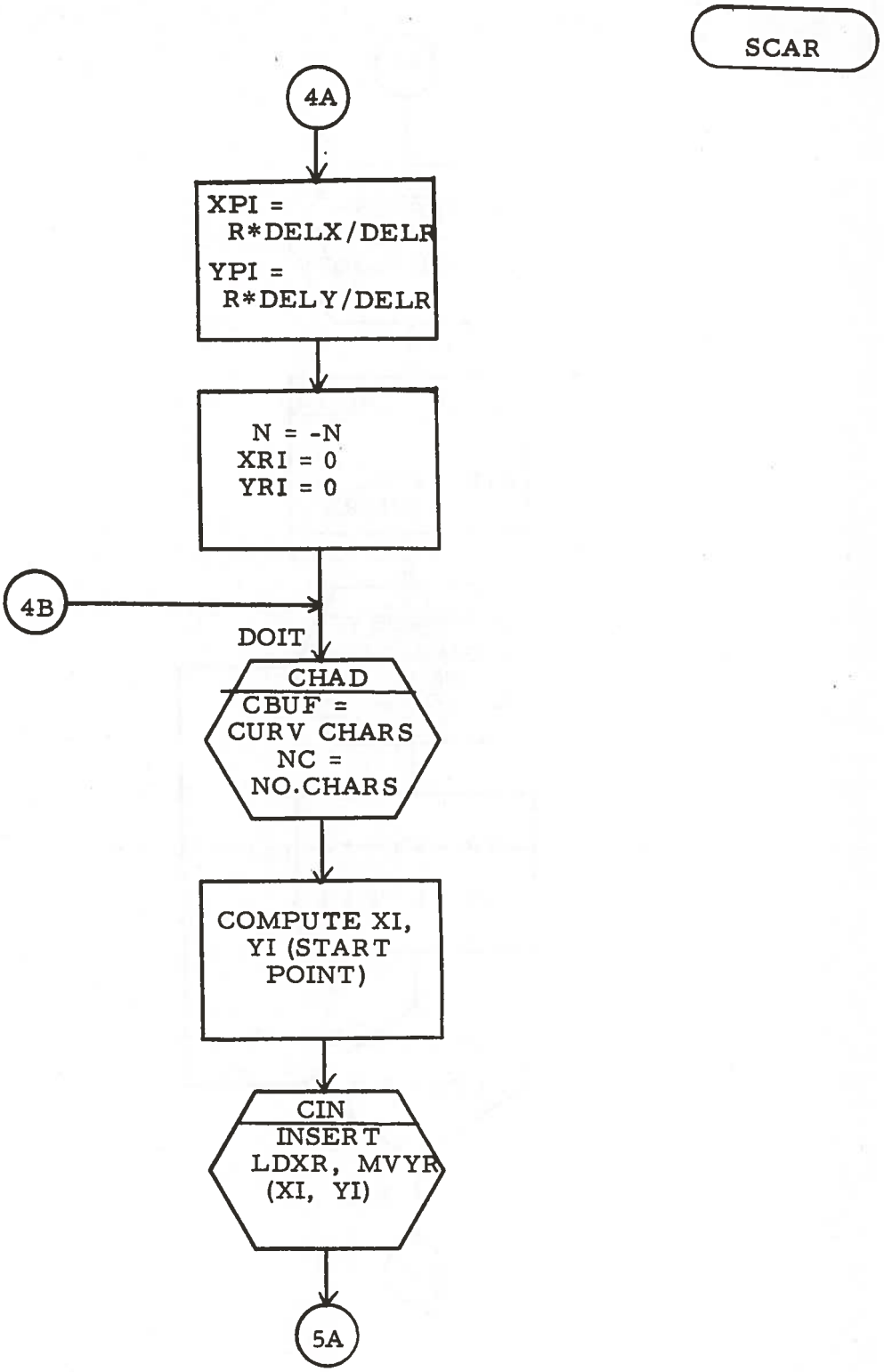
400_8

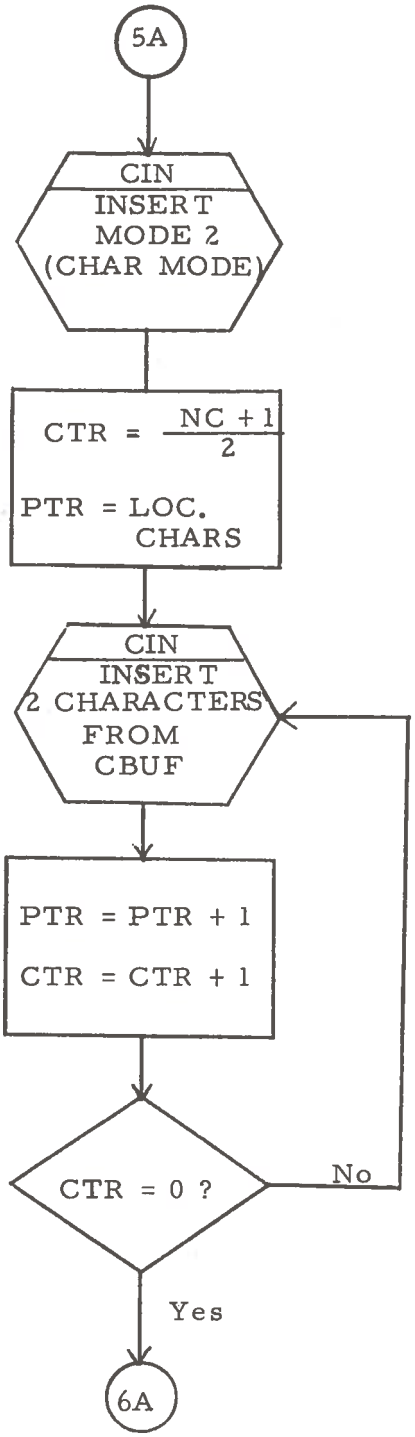
SCAR

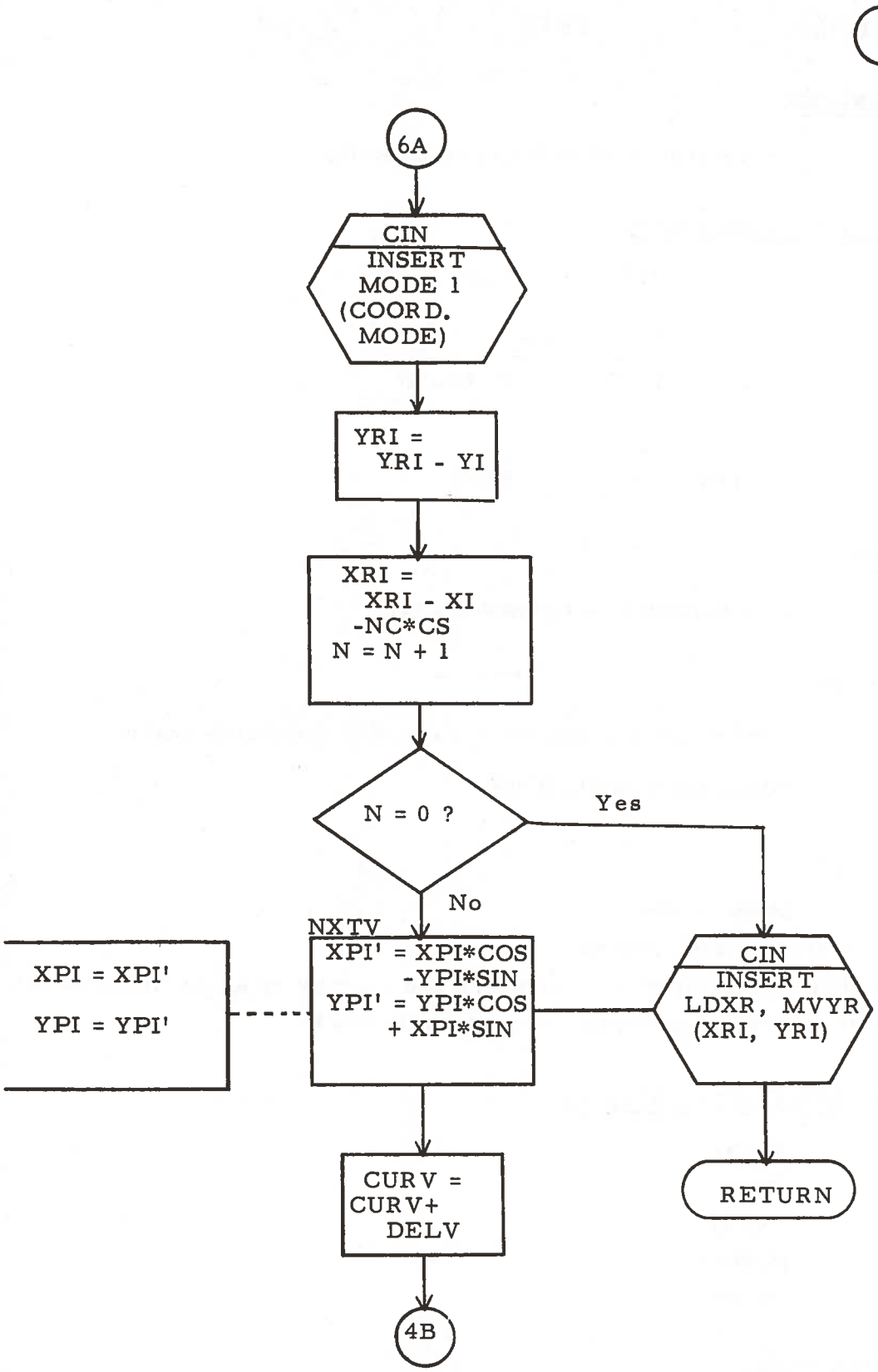

:SA Δx, Δy, A, B, N, SIZE, D, θ
SCAR
ENTITY LEVEL ?
No
RETURN
Yes
GTNO
GET Δx, Δy, A
CURV = A (CURRENT VALUE)
GTNO
GET B, N
NMI = H - 1
DELV = B-A / NMI
2A


SCAR
2A
GTNO
GET SIZE
SIZE > 0 ?
No
Yes
WAT
WHAT
TYPE "?"
SIZE ≤ 4 ?
No
Yes
RETURN
GTNO
GET D, θ
θ = 0 ?
3A
Yes
θ = 360
NMI = N
3A


SCAR
3A
ANGL
ALFA=-θ/360*NMI
SIN = sin(ALFA)
COS = cos(ALFA)
CS = 15*SIZE
CSY = 13*SIZE
CSYH = CSY/2
SZ2 = 2*SIZE
CIN
INSERT CHPR SIZE-1 INTO COMP. BLOCK
CIN
INSERT LDIX CS
DELR = √(Δx² + Δy²)
R = DELR + D
4A


SCAR
4A
XPI =
R*DELX/DELR
YPI =
R*DELY/DELR
N = -N
XRI = 0
YRI = 0
4B
DOIT
CHAD
CBUF =
CURV CHARS
NC =
NO.CHARS
COMPUTE XI,
YI (START
POINT)
CIN
INSERT
LDXR, MVYR
(XI, YI)
5A


5A
CIN
INSERT
MODE 2
(CHAR MODE)
CTR = NC + 1 / 2
PTR = LOC.
CHARS
CIN
INSERT
2 CHARACTERS
FROM
CBUF
PTR = PTR + 1
CTR = CTR + 1
CTR = 0 ?
No
Yes
6A


SCAR
6A
CIN
INSERT
MODE 1
(COORD.
MODE)
YRI =
YRI - YI
XRI =
XRI - XI
-NC*CS
N = N + 1
N = 0 ?
Yes
No
NXTV
XPI = XPI'
YPI = YPI'
XPI' = XPI*COS
-YPI*SIN
YPI' = YPI*COS
+ XPI*SIN
CIN
INSERT
LDXR, MVYR
(XRI, YRI)
CURV =
CURV+
DELV
RETURN
4B

SUBROUTINE SETP

1. PURPOSE:

Set reference point for current entity

2. CALLING SEQUENCE:

```
                JST*    LOC
                 :
LOC             DAC     COML+N
                 :
COM+N           XAC     SETP
```

3. INPUT:

x, y positions from input buffer

4. OUTPUT:

Load x absolute and move y absolute commands inserted into current entity block.

5. ACTION:

Mode is checked.
If not entity mode, return.
If entity mode, input x, y positions and insert display command to set entity reference point into current entity block.

6. EXTERNAL REFERENCES:

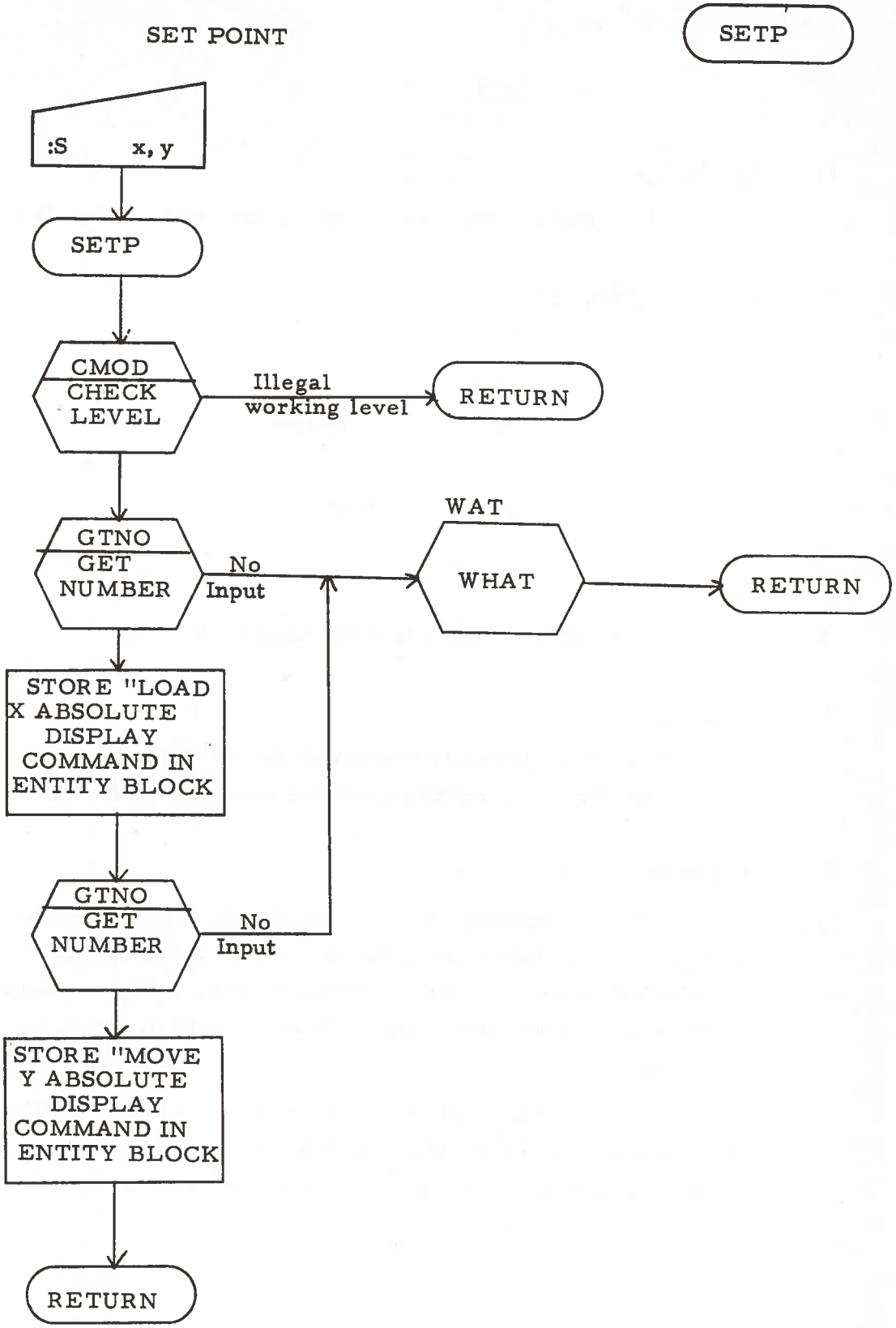
GTNO -
ENAD -
CMOD -
NONO -
WHAT

7. CORE USED:

51_8

SET POINT

SETP



SITI

1. PURPOSE:

Set initial texture and intensity for entity or indicator

2. CALLING SEQUENCE:

```
            JST*      PTR
             :
PTR         DAC       COML+N
             :
COML+N      XAC       SITI
```

3. INPUT:

Texture and Intensity from input buffer

4. OUTPUT:

Texture, Intensity command inserted into given entity block, or all entity blocks for given indicator.

5. ACTION:

Checks working level. Gets texture input. If no texture input, check for default and intensity input. If texture input is present, then gets intensity. If intensity input is first, then attempts to get texture input again. If still no input, check for texture default.

After texture and intensity are set, they are combined into a display command, and inserted into an entity block, or if in indicator mode, inserted into each entity block on the indicated ring.

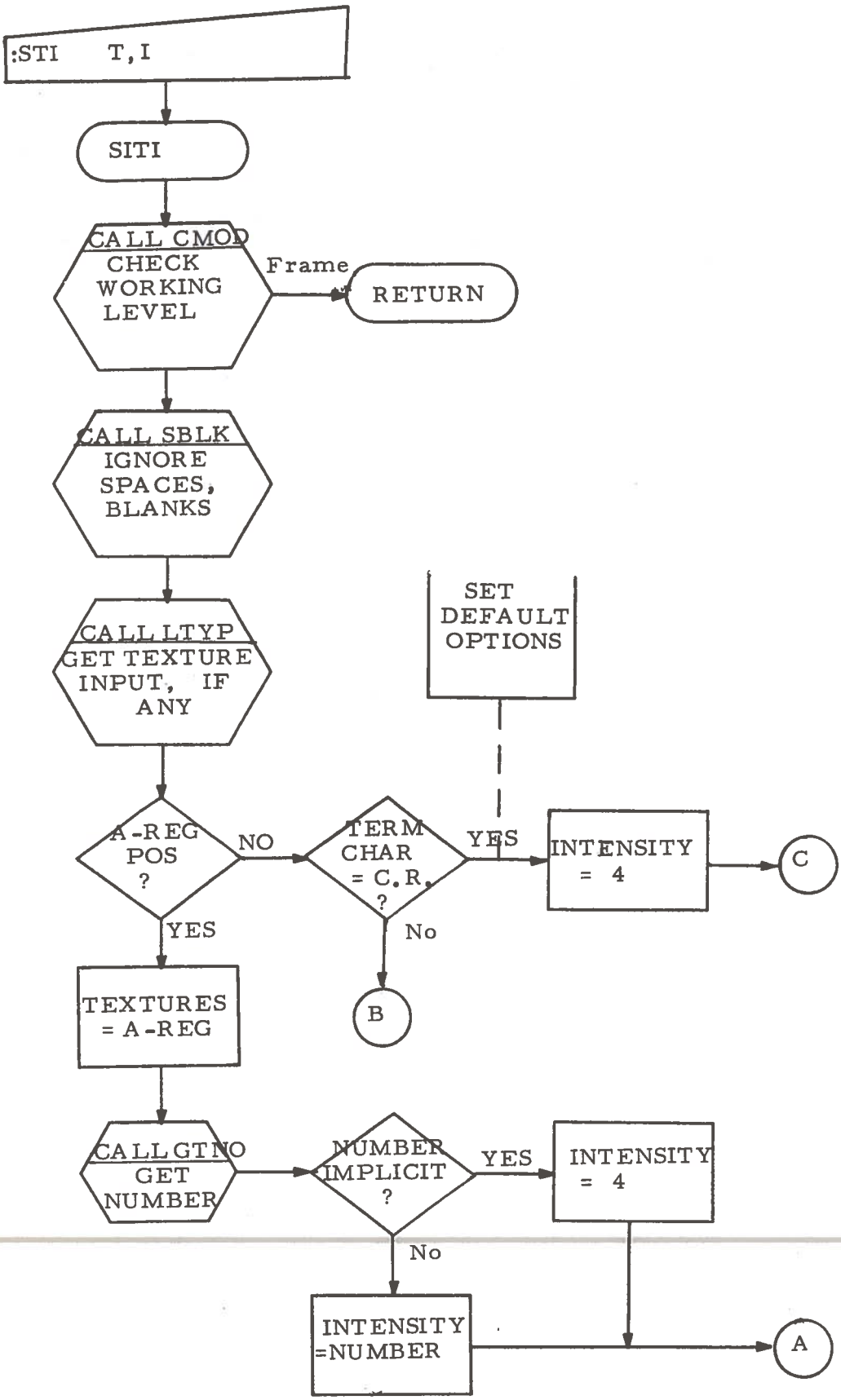
6. EXTERNAL REFERENCES:

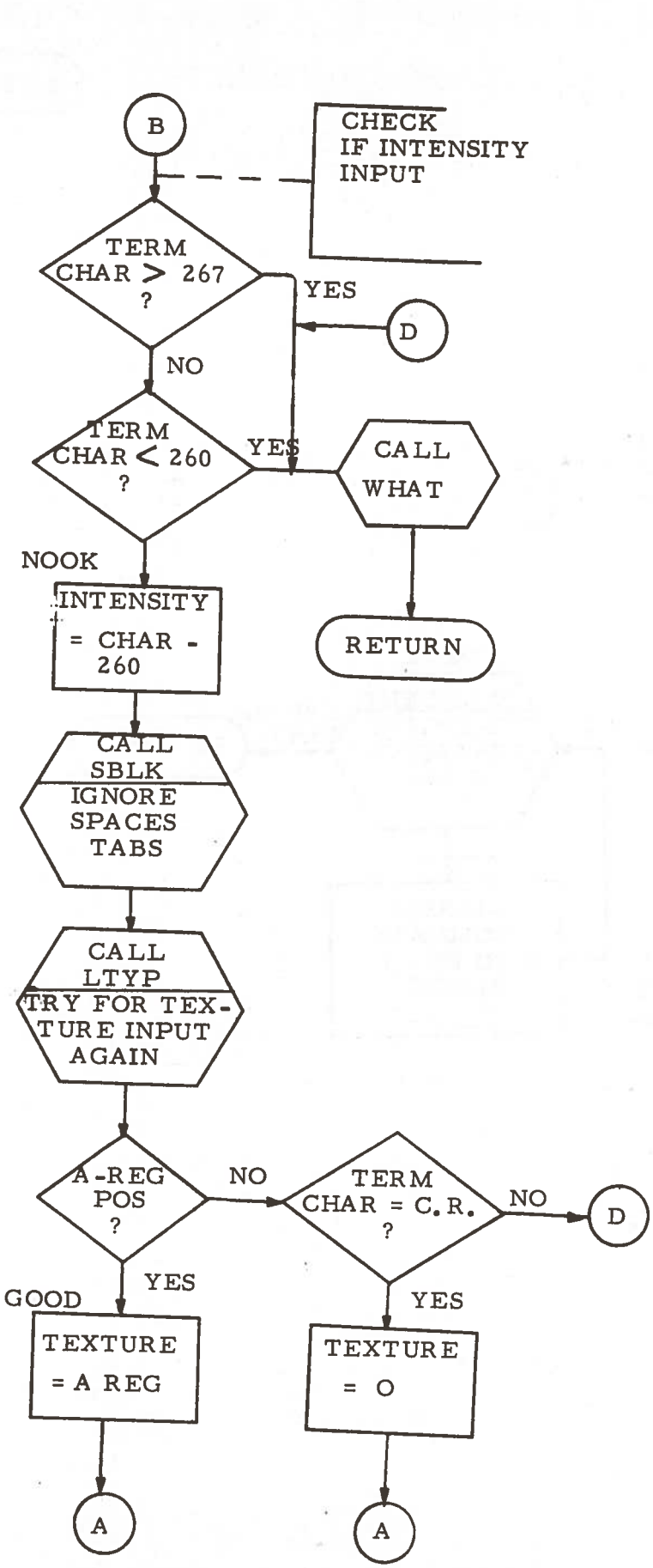
CMOD
SBLK
LTYP
GTNO
NONO
WHAT
CLVL
GENT
ENAD

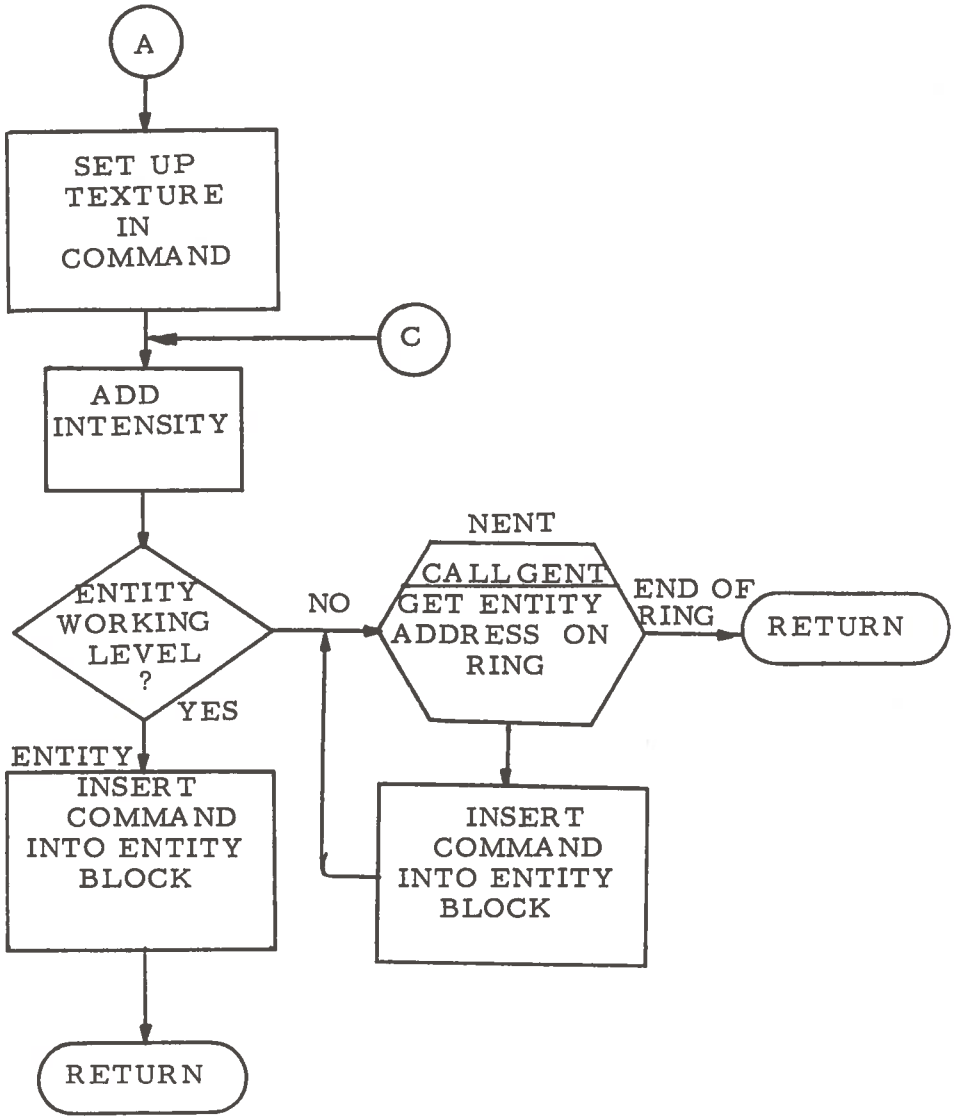
7. CORE USED:

143_8

Set Initial Texture, Intensity




SITI
B
CHECK IF INTENSITY INPUT
TERM CHAR > 267 ?
YES
D
NO
TERM CHAR < 260 ?
YES
CALL WHAT
RETURN
NOOK
INTENSITY = CHAR - 260
CALL SBLK
IGNORE SPACES TABS
CALL LTYP
TRY FOR TEXTURE INPUT AGAIN
A-REG POS ?
NO
TERM CHAR = C.R. ?
NO
D
YES
GOOD
TEXTURE = A REG
YES
TEXTURE = O
A
A


A
SET UP TEXTURE IN COMMAND
C
ADD INTENSITY
ENTITY WORKING LEVEL ?
NO
YES
NENT
CALLGENT
GET ENTITY ADDRESS ON RING
END OF RING
RETURN
ENTITY
INSERT COMMAND INTO ENTITY BLOCK
INSERT COMMAND INTO ENTITY BLOCK
RETURN

SKIP

1. PURPOSE:

Process skip instructions

2. CALLING SEQUENCE:

```
          JST*     PTR
           :
PTR       DAC      COML+N
           :
COML+N    XAC      SUBR
```

where:

SUBR = SKIP, EQAL, GRET, GREQ, LESS, LEQU

3. INPUT:

Register numbers and skip numbers from input buffer

4. OUTPUT:

Skip instruction in conditioning block

5. ACTION:

a) SKIP - checks working level
Gets skip number, checks if greater than 0. Combined with SKIP code and inserted into conditioning block.

b) EQAL - Skip if A = B

c) GRET - Skip if A > B

d) GREQ - Skip if A $\geq$ B

e) LESS - Skip if A < B

f) LEQU - Skip if A $\leq$ B

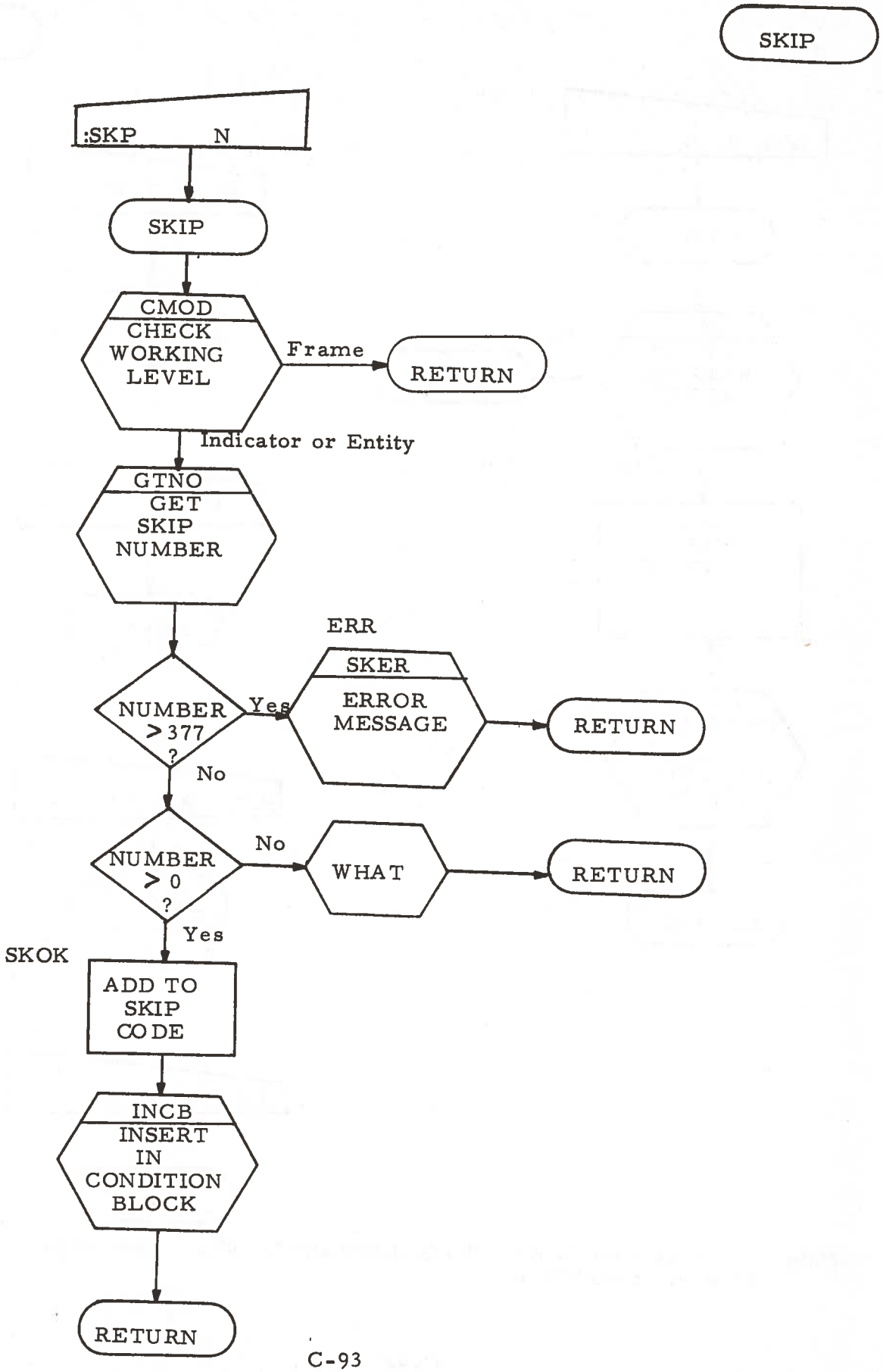
b-f) Check working level, load appropriate skip code, CALL GABN.

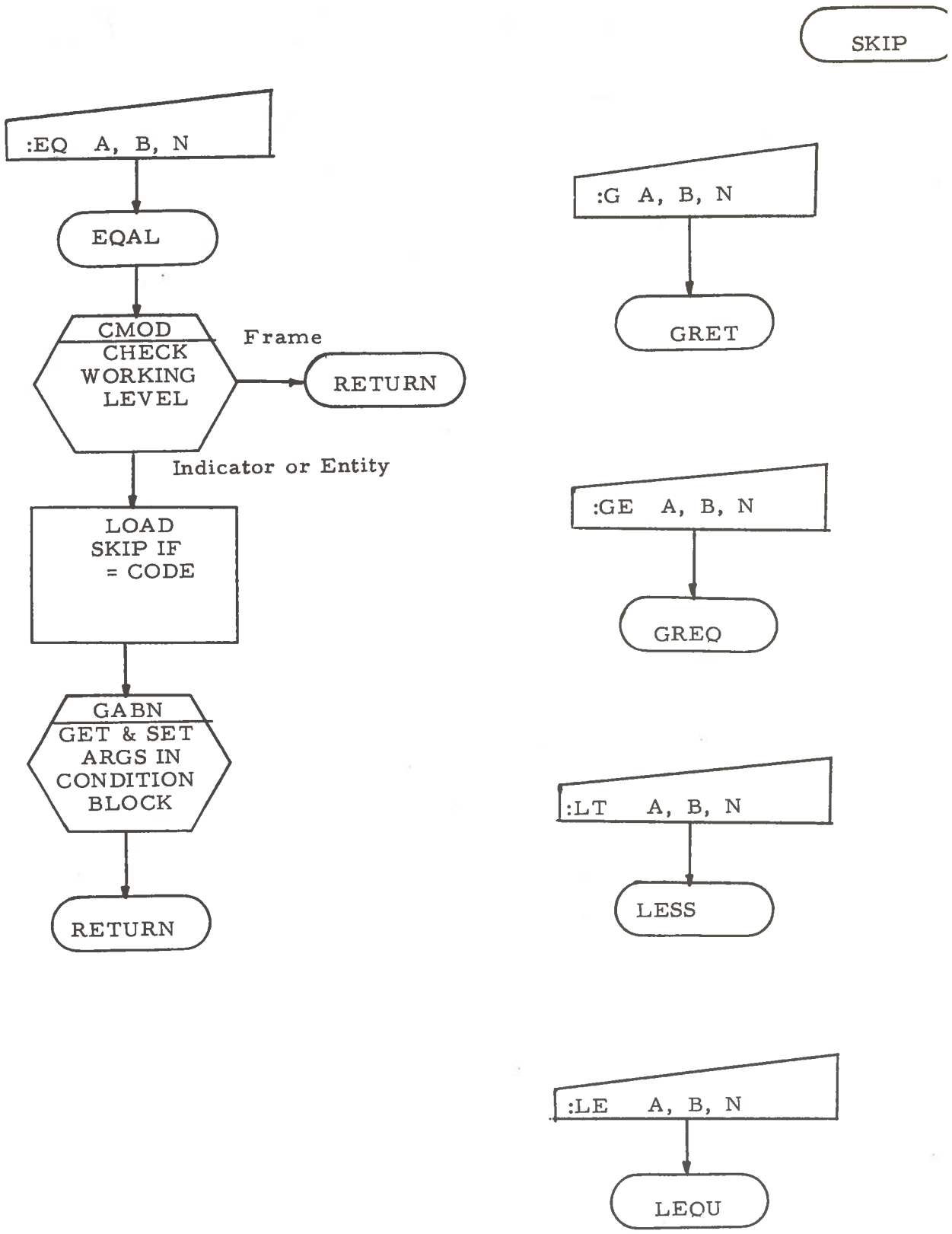
6. EXTERNAL REFERENCES:

CMOD
GTNO
WHAT
INCB
GABN
SKER

7. CORE USED:

76_8


SKIP
:SKP N
SKIP
CMOD
CHECK WORKING LEVEL
Frame
RETURN
Indicator or Entity
GTNO
GET SKIP NUMBER
ERR
SKER
ERROR MESSAGE
RETURN
NUMBER > 377 ?
Yes
No
NUMBER > 0 ?
No
WHAT
RETURN
Yes
SKOK
ADD TO SKIP CODE
INCB
INSERT IN CONDITION BLOCK
RETURN



Note: Flowcharts for GRET, GREQ, LESS and LEQU are similar to the Flowcharts for EQAL.

SUBROUTINE TICK

1. PURPOSE:

Insert component display commands to draw tick marks as indicated by input.

2. CALLING SEQUENCE:

```
          JST*    PTR
           :
PTR       DAC     COML+N
           :
COML+N    XAC     TICK
```

3. INPUT:

Arguments for tick mark specifications from input buffer

4. OUTPUT:

Display Commands to draw tick marks inserted in component buffer

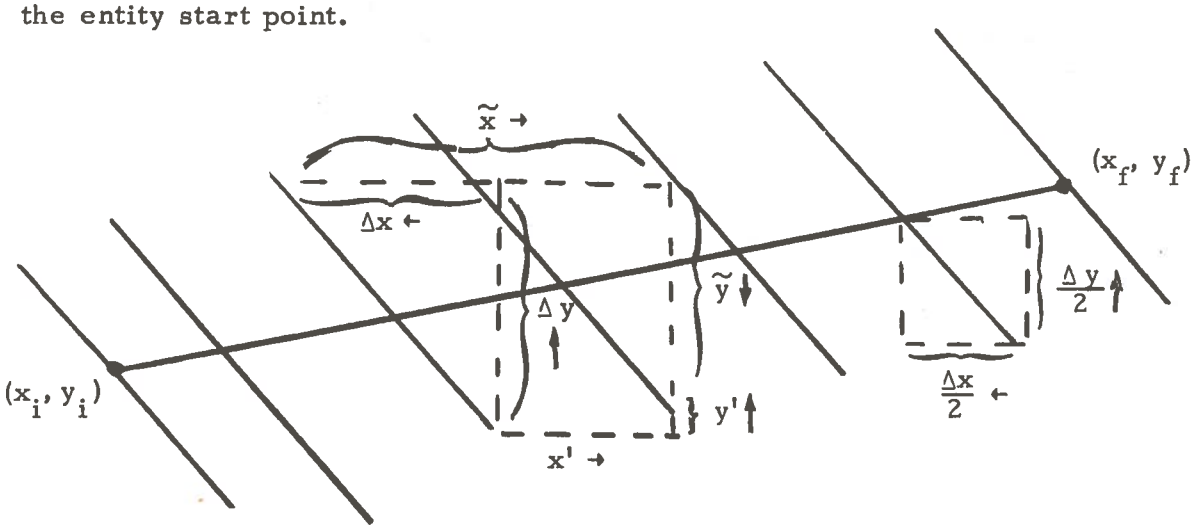
5. ACTION:

Retrieves number of ticks, N, initial point (x_i, y_i) and final point $(x_f y_f)$. If early termination of line, error return is made.

Δx, Δy are retrieved. If neither are present, tick marks are automatically of length 12 rasters and are perpendicular to the line connecting (x_i, y_i) and (x_f, y_f). If only Δx is present, it is assumed to be the magnitude of the tick mark length, and again tick marks are perpendicular to the line connecting (x_i, y_i) and (x_f, y_f). If both Δx, Δy are present, then they are assumed to be both magnitude and direction of the x, y components of the tick, respectively.

The parameters described in the following figure are calculated and the appropriate display commands are inserted into the component block. These commands leave the final position of the beam

at (x_f, y_f). NOTE: (x_i, y_i) and (x_f, y_f) are assumed to be relative to the entity start point.



(Arrows indicate direction of quantity.)

Where:

N given

$$\Delta x = \frac{\text{magn.} * (y_i - y_f)}{\sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}} \quad \text{or given}$$

$$\Delta y = \frac{\text{magn.} * (x_f - x_i)}{\sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}} \quad \text{or given}$$

$$x' = \frac{x_f - x_i}{N-1}$$

$$y' = \frac{y_f - y_i}{N-1}$$

$$\tilde{x} = x' - \Delta x$$

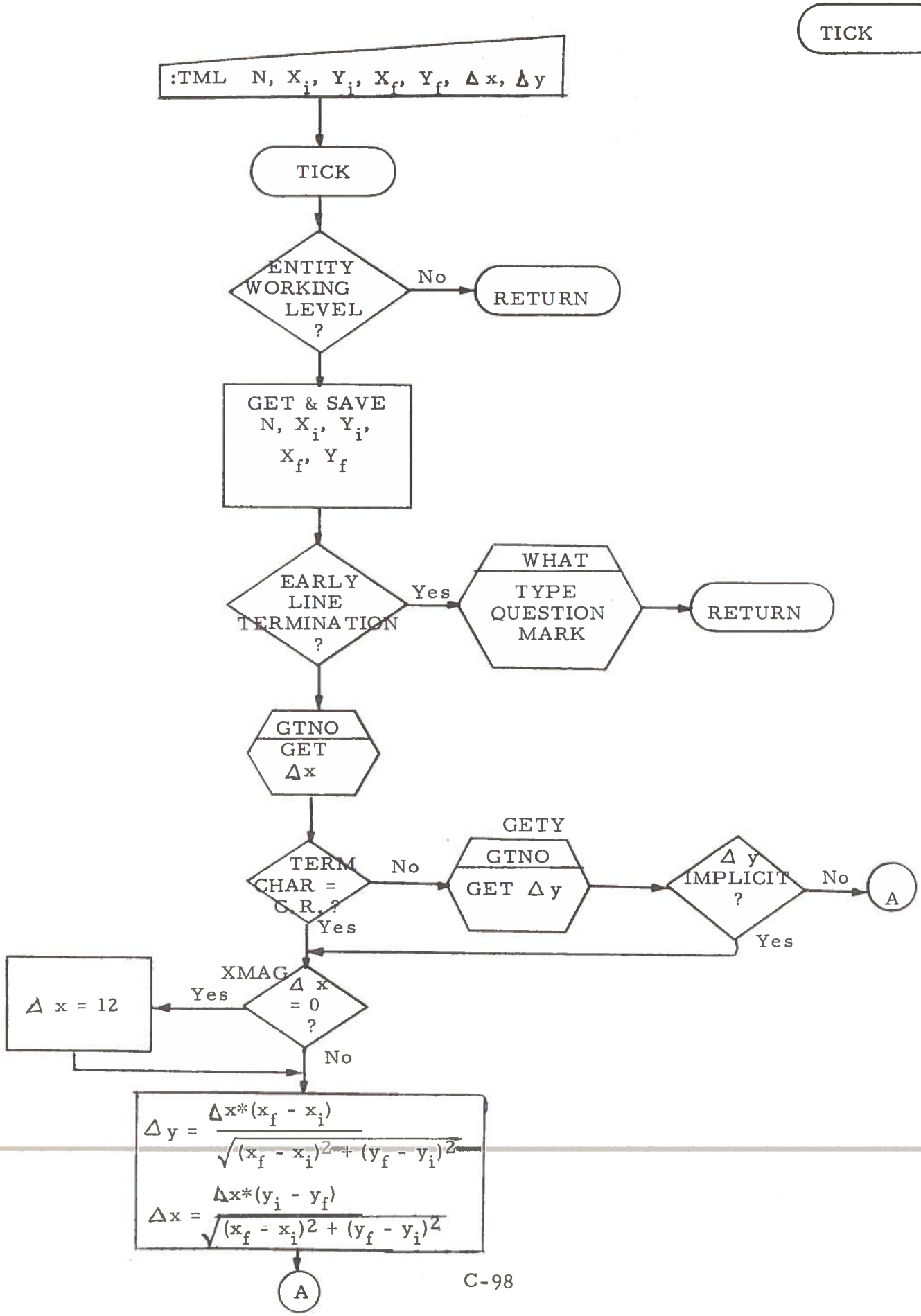
$$\tilde{y} = y' - \Delta y$$

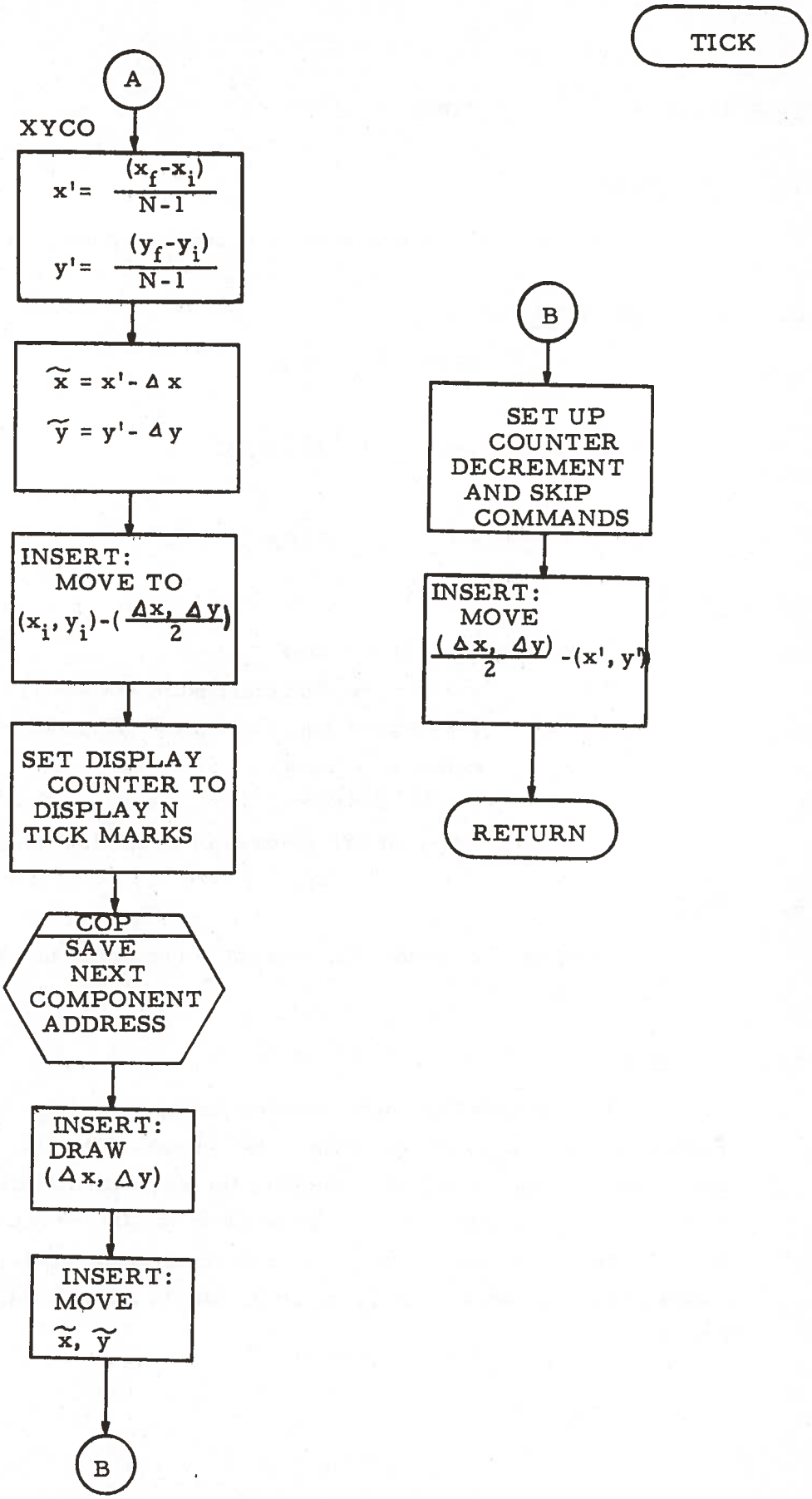
6. EXTERNAL REFERENCES:

CMOD
GTNO
NONO
SQRJAM
COP
CIN
CLO
CPAD

7. CORE USED:

254_8


:TML N, X_i, Y_i, X_f, Y_f, Δx, Δy
TICK
ENTITY WORKING LEVEL ?
No
RETURN
GET & SAVE N, X_i, Y_i, X_f, Y_f
EARLY LINE TERMINATION ?
Yes
WHAT TYPE QUESTION MARK
RETURN
GTNO GET Δx
TERM CHAR = C.R. ?
No
GETY GTNO GET Δy
Δy IMPLICIT ?
No
A
Yes
Yes
XMAG Δx = 0 ?
Yes
Δx = 12
No
$\Delta y = \frac{\Delta x*(x_f - x_i)}{\sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}}$
$\Delta x = \frac{\Delta x*(y_i - y_f)}{\sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}}$
A


TICK
A
XYCO
$x' = \frac{(x_f - x_i)}{N-1}$
$y' = \frac{(y_f - y_i)}{N-1}$
$\tilde{x} = x' - \Delta x$
$\tilde{y} = y' - \Delta y$
INSERT:
MOVE TO
$(x_i, y_i) - (\frac{\Delta x, \Delta y}{2})$
SET DISPLAY COUNTER TO DISPLAY N TICK MARKS
COP
SAVE NEXT COMPONENT ADDRESS
INSERT:
DRAW
$(\Delta x, \Delta y)$
INSERT:
MOVE
$\tilde{x}, \tilde{y}$
B
B
SET UP COUNTER DECREMENT AND SKIP COMMANDS
INSERT:
MOVE
$\frac{(\Delta x, \Delta y)}{2} - (x', y')$
RETURN

SUBROUTINE TIKA

1. PURPOSE:

Set up display commands to draw tick marks for arcs

2. CALLING SEQUENCE:

```
        JST*    PTR
        :
PTR     PAC     COML+N
        :
COML+N  XAC     TIKA
```

3. INPUT:

From teletype input buffer:

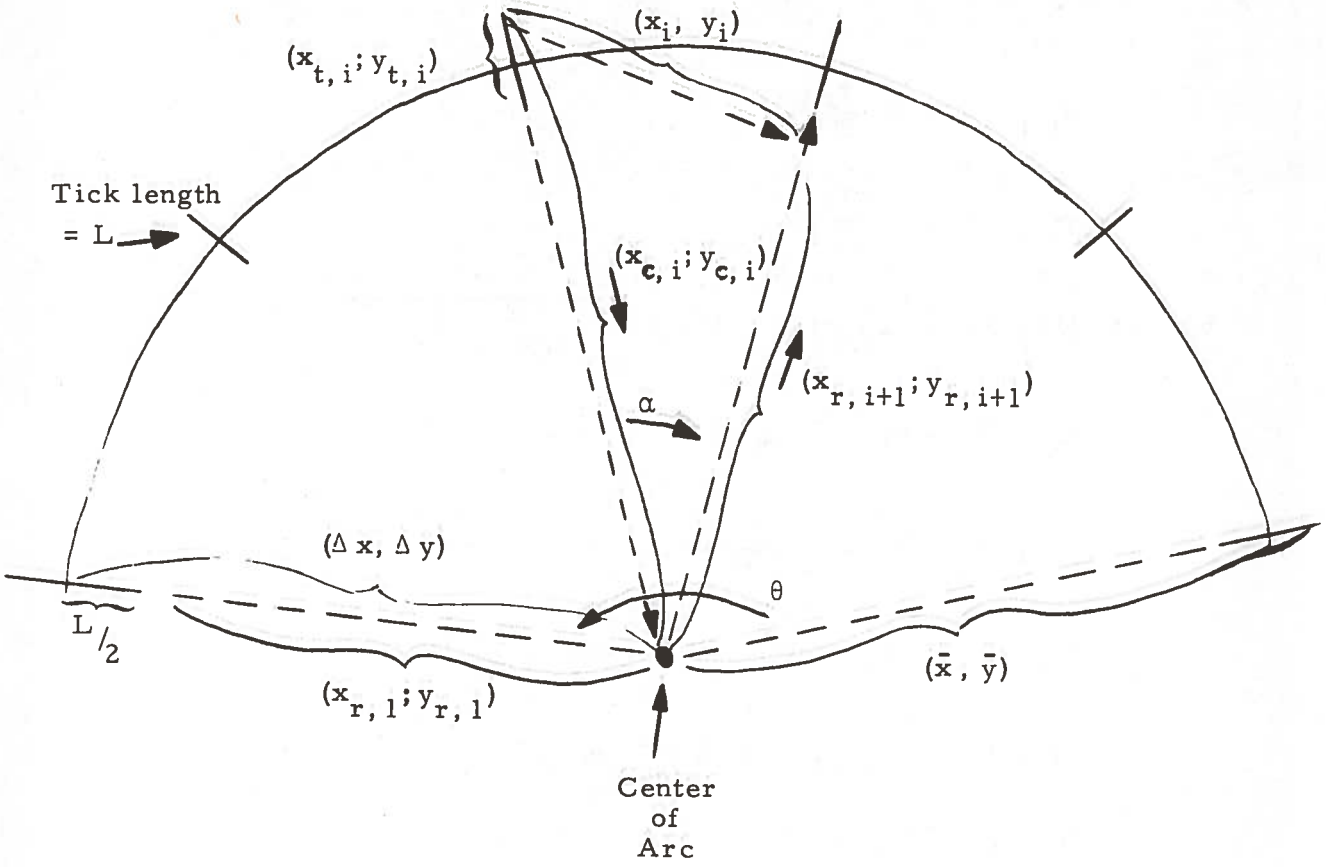
Δx	x-component of start point (relative)
Δy	y-component of start point (relative)
N	number of ticks
L	length of ticks
θ	Angle of arc (between 1st and last ticks)

4. OUTPUT:

Display commands inserted into component block for current entity

5. ACTION:

Determines the angle between ticks, α; and the length of a radial vector, (x_r, y_r), drawing of tick mark. The tick mark is drawn perpendicular to and straddling the arc with radius $|(\Delta x, \Delta y)|$. The vector (x_i, y_i) used to move to position for the next tick mark is the resultant of a vector, (x_c, y_c), returning to the center of the arc followed by the radial vector, (x_r, y_r), for the next tick (see figure below).



Construction of Tick Marks

Figure 1.

The following formula are used:

$$\alpha = \frac{-\theta}{360\,(n-1)} \quad ; \quad \begin{array}{l} \alpha \text{ is in fractions of a circle} \\ \theta \text{ is in degrees} \end{array}$$

$$x_{r,1} = \Delta x - \frac{L}{2} * \frac{\Delta x}{D_0}$$

$$y_{r,1} = \Delta y - \frac{L}{2} * \frac{\Delta y}{D_0}$$

where: $D_0 = |(\Delta x, \Delta y)| = \sqrt{\Delta x^2 + \Delta y^2}$

$$x_{t,i} = L * \frac{x_{r,i}}{DI}$$

$$y_{t,i} = L * \frac{y_{r,i}}{DI}$$

where: $$DI = |(x_{r,1}; y_{r,1})| = \sqrt{x_{r,1}^2 + y_{r,1}^2}$$

$$x_{c,i} = -(x_{r,i} + x_{t,i})$$

$$y_{c,i} = -(y_{r,i} + y_{t,i})$$

$$x_{r,i+1} = x_{r,i} \cos \alpha - y_{r,i} \sin \alpha$$

$$y_{r,i+1} = x_{r,i} \sin \alpha + y_{r,i} \cos \alpha$$

$$x_i = x_{e,i} + x_{r,i+1}$$

$$x_i = y_{c,i} + y_{r,i+1}$$

The return vector $(\bar{x}, \bar{y})$ to return to the center of the arc is defined by:

$$\bar{x} = -(x_{r,1} + \sum_{i=1}^{n} x_{t,i} + \sum_{i=1}^{n} x_i)$$

$$\bar{y} = -(y_{r,1} + \sum_{i=1}^{n} y_{t,i} + \sum_{i=1}^{n-1} y_i)$$

6. EXTERNAL REFERENCES:

CMOD

GTNO
COP

CIN

CLO

6. EXTERNAL REFERENCES (cont.):

SINX2

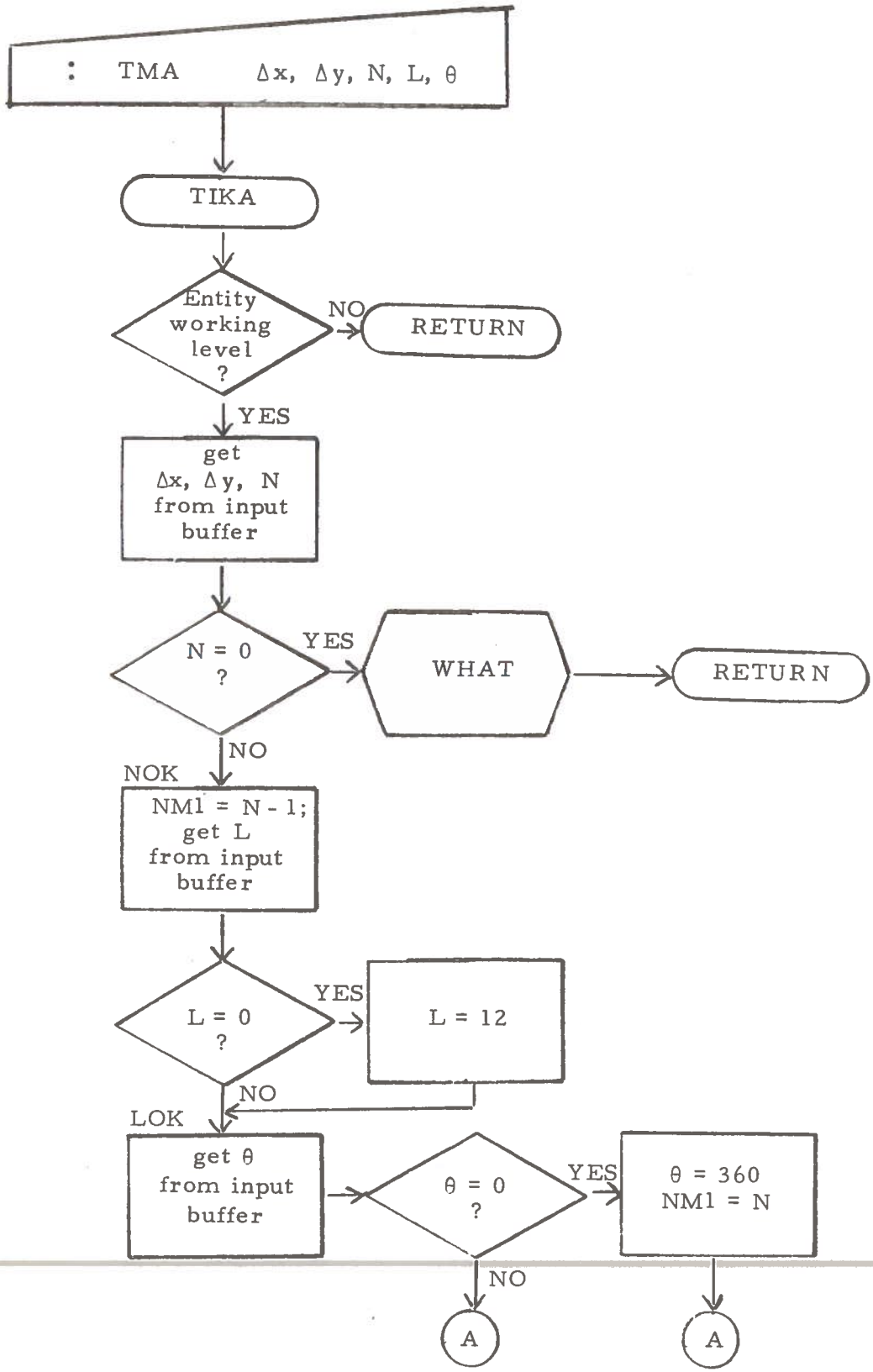
COSX2

SQRJAM

7. CORE USED:

356_8

Tick Marks for Arcs



SUBROUTINE TRLT

1. PURPOSE:

Translate indicators or set up dynamic expression for entity translation.

2. CALLING SEQUENCE:

```
          JST*     PTR
           :
PTR       DAC      COML+N
           :
COML+N    XAC      TRLT
```

3. INPUT:

$\Delta x, \Delta y$ from input buffers.
If in indicator mode, $\Delta x, \Delta y$ assumed to be numbers.
If in entity mode, $\Delta x, \Delta y$ assumed to be dynamic expressions.

4. OUTPUT:

For indicators, all entities in entity ring are translated $\Delta x, \Delta y$ relative to current set points by resetting the set points.

For entities, a dynamic expression is set in the current entities conditioning block.

5. ACTION:

In indicator mode, retrieves $\Delta x, \Delta y$. Cycle thru the entity ring, resetting the entity set points.

In entity mode, moves dynamic expressions into conditioning block. At end of moving an expression, check to insure that a null character has been moved into conditioning block. The null character/word is used as a delimiter for the expressions. The code set in the conditioning block is in the following format:

1	000 000 1		ignored	
0	7-bit ASCII char.	X	7-bit ASCII char.	
0	⋮	X	⋮	
0	7-bit ASCII char.		7-bit ASCII char.	

Δx and/or Δy expressions

If no Δx-expression, first word is the null word. A null word or char. will separate the Δx- and Δy-expressions. If no Δy-expression, a null word will be packed if Δx-expression has an odd number of characters. The Δx-expression will then be followed by a null character and a null word, then the next conditioning command.

E. g.,

0	7-bit ASCII char.	X	7-bit ASCII char.	← Δx-expr. with even number of char.
0	0 0 0 0 0 0 0	X	0 0 0 0 0 0 0	← null to terminate Δx-expr.
0	0 0 0 0 0 0 0	X	0 0 0 0 0 0 0	← null for Δy-expr., here none
1	command			

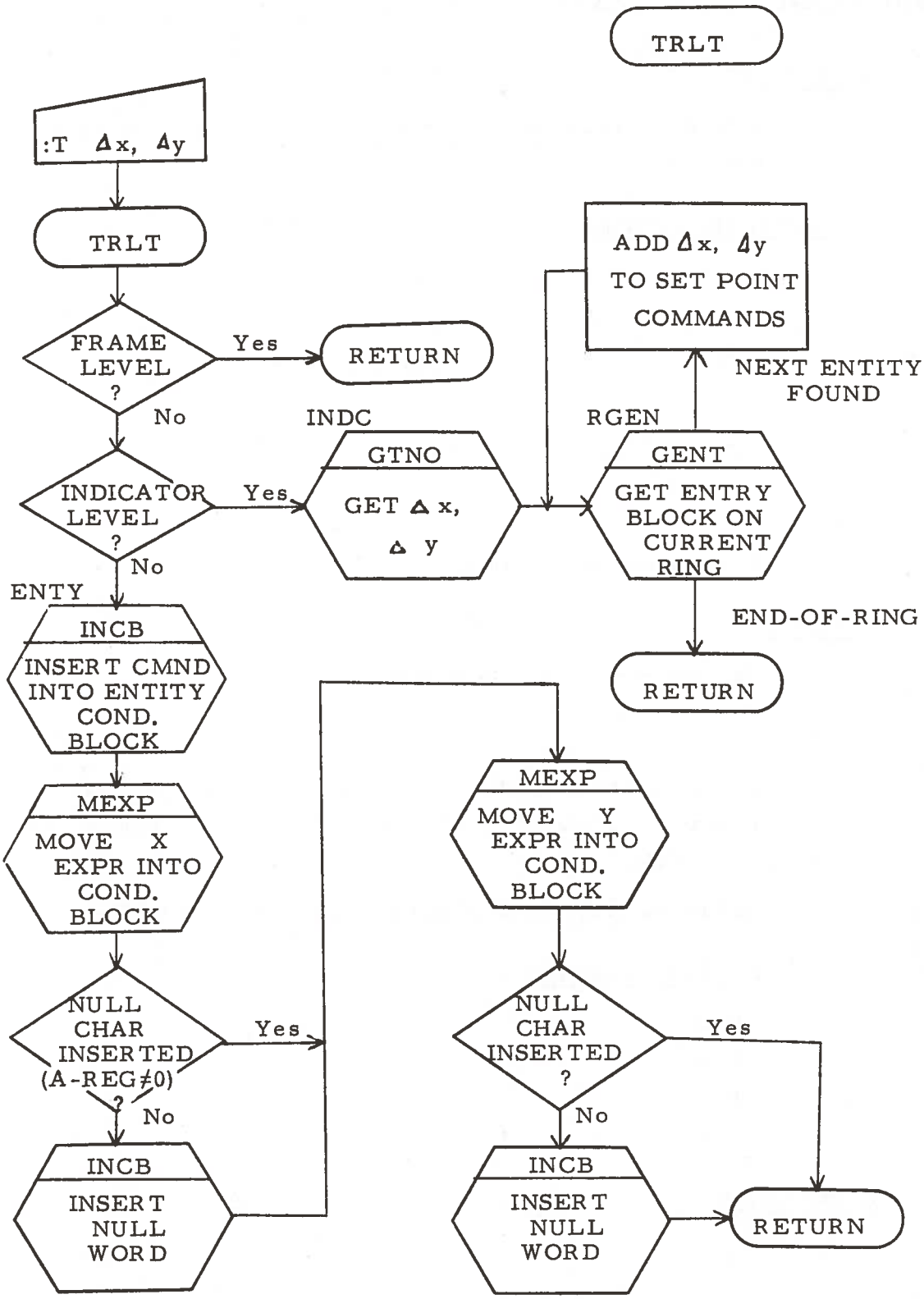
0	7-bit ASCII character	X	0 0 0 0 0 0 0	← Δx-expr. with odd number of char.
0	0 0 0 0 0 0 0	X	0 0 0 0 0 0 0	← Null for Δy-expr., none
1	command	X		

6. EXTERNAL REFERENCES:

CLVL

7. CORE USED:

72_8


TRLT
:T Δx, Δy
TRLT
FRAME LEVEL ?
Yes
RETURN
No
INDC
GTNO
GET Δ x, Δ y
INDICATOR LEVEL ?
Yes
No
ENTY
INCB
INSERT CMND INTO ENTITY COND. BLOCK
MEXP
MOVE X EXPR INTO COND. BLOCK
NULL CHAR INSERTED (A-REG≠0) ?
Yes
No
INCB
INSERT NULL WORD
ADD Δx, Δy TO SET POINT COMMANDS
NEXT ENTITY FOUND
RGEN
GENT
GET ENTRY BLOCK ON CURRENT RING
END-OF-RING
RETURN
MEXP
MOVE Y EXPR INTO COND. BLOCK
NULL CHAR INSERTED ?
Yes
No
INCB
INSERT NULL WORD
RETURN

SUBROUTINE TXTR

1. PURPOSE:

Interpret argument for line type (texture) specification and insert into conditioning block.

2. CALLING SEQUENCE:

```
        JST*    LOC
         .
         .
         .
LOC     DAC     COML+N
         .
         .
         .
COML+N  XAC     TXTR
```

3. INPUT:

Char. from input buffer.

4. OUTPUT:

Texture command in conditioning block.

5. ACTION:

Checks working level. Get code for line type specified. If illegal character indicated, set default code if illegal char. = C. R., otherwise call WHAT.

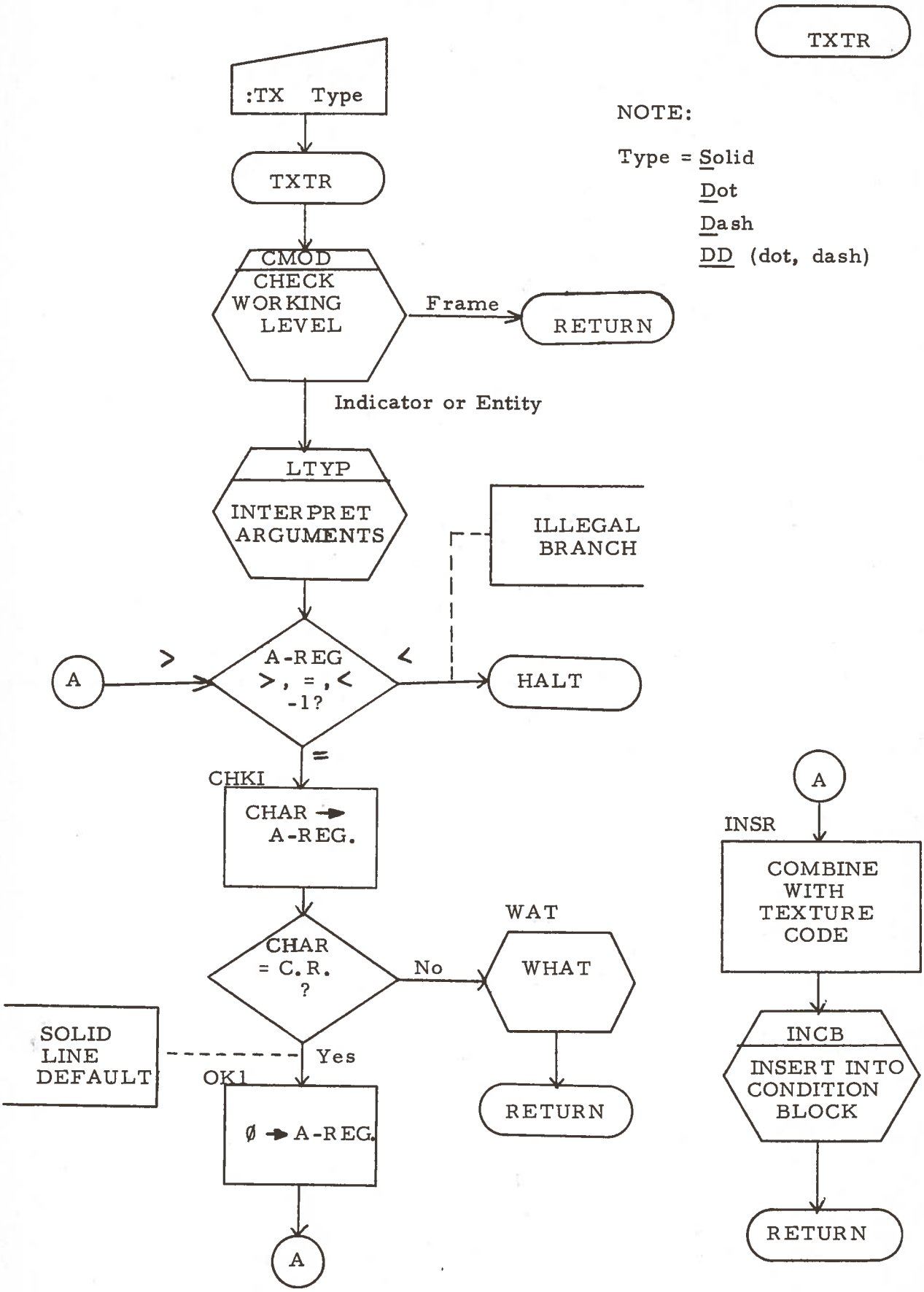
Add to texture command and set in conditioning block.

6. EXTERNAL REFERENCES:

CMOD
LTYP
WHAT
INCB

7. CORE USED:

27_8


TXTR
:TX Type
TXTR
CMOD
CHECK WORKING LEVEL
Frame
RETURN
Indicator or Entity
LTYP
INTERPRET ARGUMENTS
ILLEGAL BRANCH
A
>
A-REG
>, =, <
-1?
<
HALT
=
CHKI
CHAR → A-REG.
CHAR = C.R. ?
No
WAT
WHAT
RETURN
SOLID LINE DEFAULT
Yes
OK1
∅ → A-REG.
A
NOTE:
Type = Solid
Dot
Dash
DD (dot, dash)
A
INSR
COMBINE WITH TEXTURE CODE
INCB
INSERT INTO CONDITION BLOCK
RETURN

GREG

1. PURPOSE:

   Retrieve and check register number from input buffer.

2. CALLING SEQUENCE:

```
CALL     GREG
ERROR    RETURN
NORMAL RETURN
```

3. INPUT:

   Character string in input buffer

4. OUTPUT:

   Returns register number in A-Reg.

5. ACTION:

   Sets to skip blanks and tabs, retrieves one char. If char = R, continue; otherwise return.

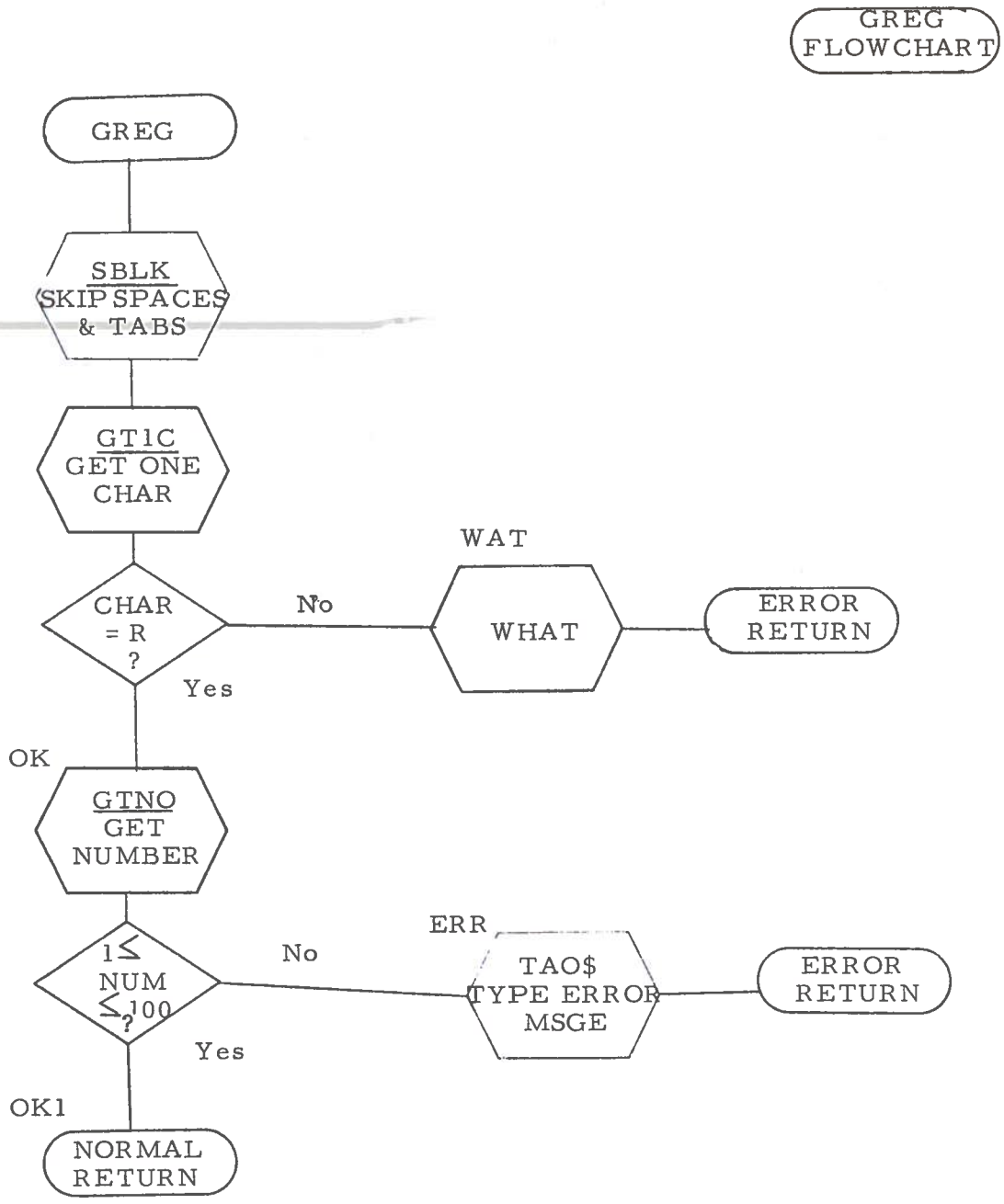
   Get number from input buffer. Check reg. number for range 0-99. Return reg. number in A-Reg.

6. EXTERNAL REFERENCES:

   None

7. CORE USED:

   46_8


GREG FLOWCHART
GREG
SBLK SKIP SPACES & TABS
GT1C GET ONE CHAR
CHAR = R ?
No
WAT
WHAT
ERROR RETURN
Yes
OK
GTNO GET NUMBER
1≤ NUM ≤100 ?
No
ERR
TAO$ TYPE ERROR MSGE
ERROR RETURN
Yes
OK1
NORMAL RETURN

SUBROUTINE: GTNO, NONO

a) Get Number

b) Returns 1 if last number returned was explicit; otherwise 0

1. PURPOSE:

a) Retrieves characters from buffer specifying a decimal number, converts, and outputs equivalent octal number in two's complement notation. Non-numeric characters, spaces, and tabs preceding the number in the input buffer are ignored on option. If no number is found, a zero is returned. Zero also returned on overflow.

b) NONO = Ø if no number found, otherwise 1.
/A-Reg 0 to ignore spaces, tabs, non-numeric char.
/A-Reg = 0 to ignore space, tabs only
/A-Reg 0 do not ignore any char.

2. CALLING SEQUENCE:

a) CALL GTNO

b) XAC NONO

3. INPUT:

Characters from input buffer

4. OUTPUT:

a. 1) Octal number in two's complement notation in A-register

a. 2) Terminating ASCII character in B-register

b) If last number returned by GTNO was a zero because no number found, NONO returns zero.

5. ACTION:

NUM, which is to contain the octal number, is zeroed, and flags are initialized. A character is retrieved from the buffer. If it is numeric, NUM is reset to NUM * 12_8 + (octal value of input numeric), and another character is retrieved.

If the character is not numeric, a check is made to determine if it is a terminating character, i.e., if any previous character was numeric. If so, a jump is made to complement the input number if necessary and return with the terminating character in the B-register.

If the character is not a terminating character and is a negative sign, a flag is set to indicate complementation is needed before returning to the calling routine, and the next character is retrieved.

If the character is a carriage return, the A-register is zeroed, and return is made to the calling routine with the carriage return ASCII code in the B-register.

Otherwise the character is ignored and the next character is received.

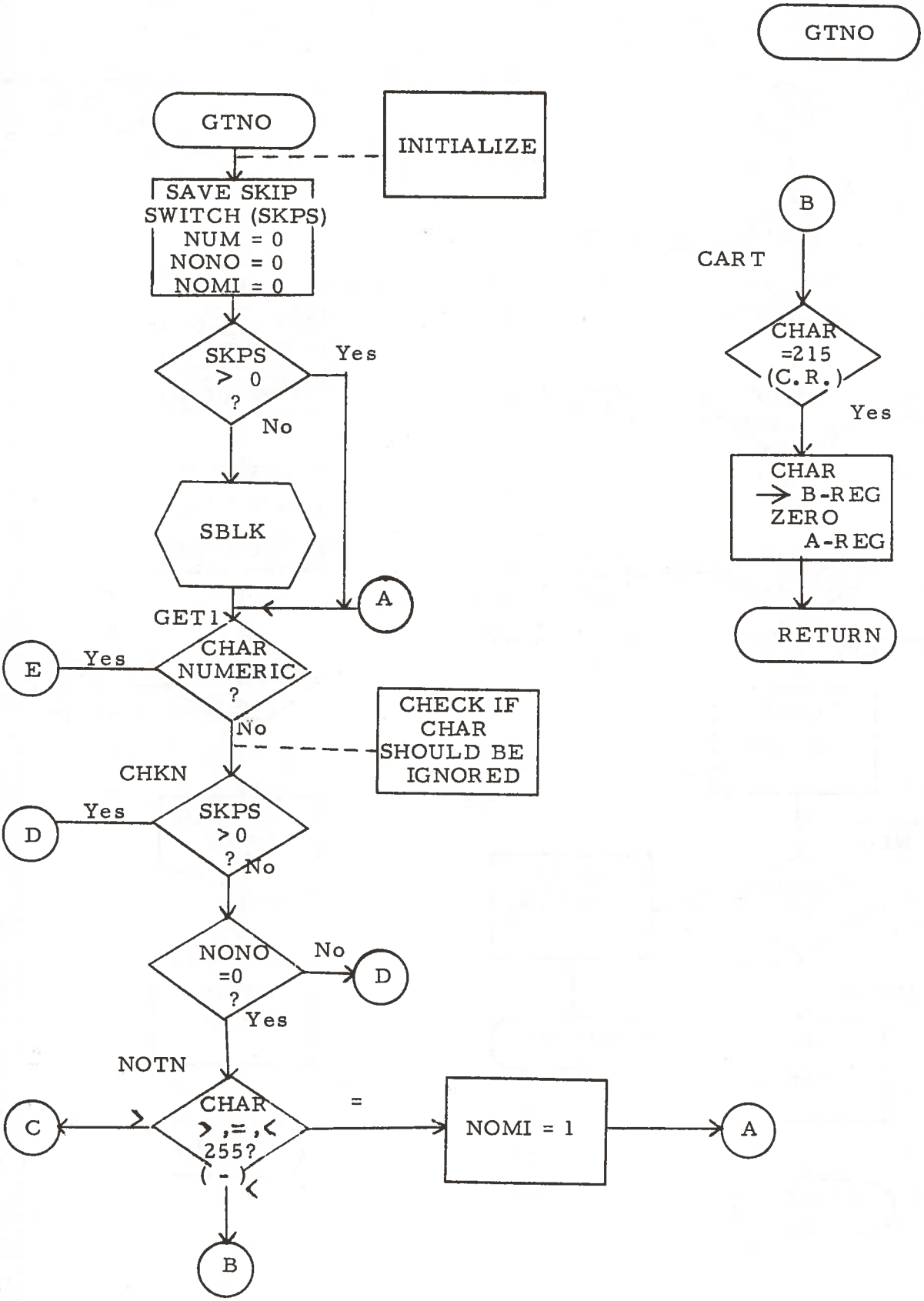
If the input number overflows, an error message is typed, and control is transferred to the calling routine, with the A-register zero.

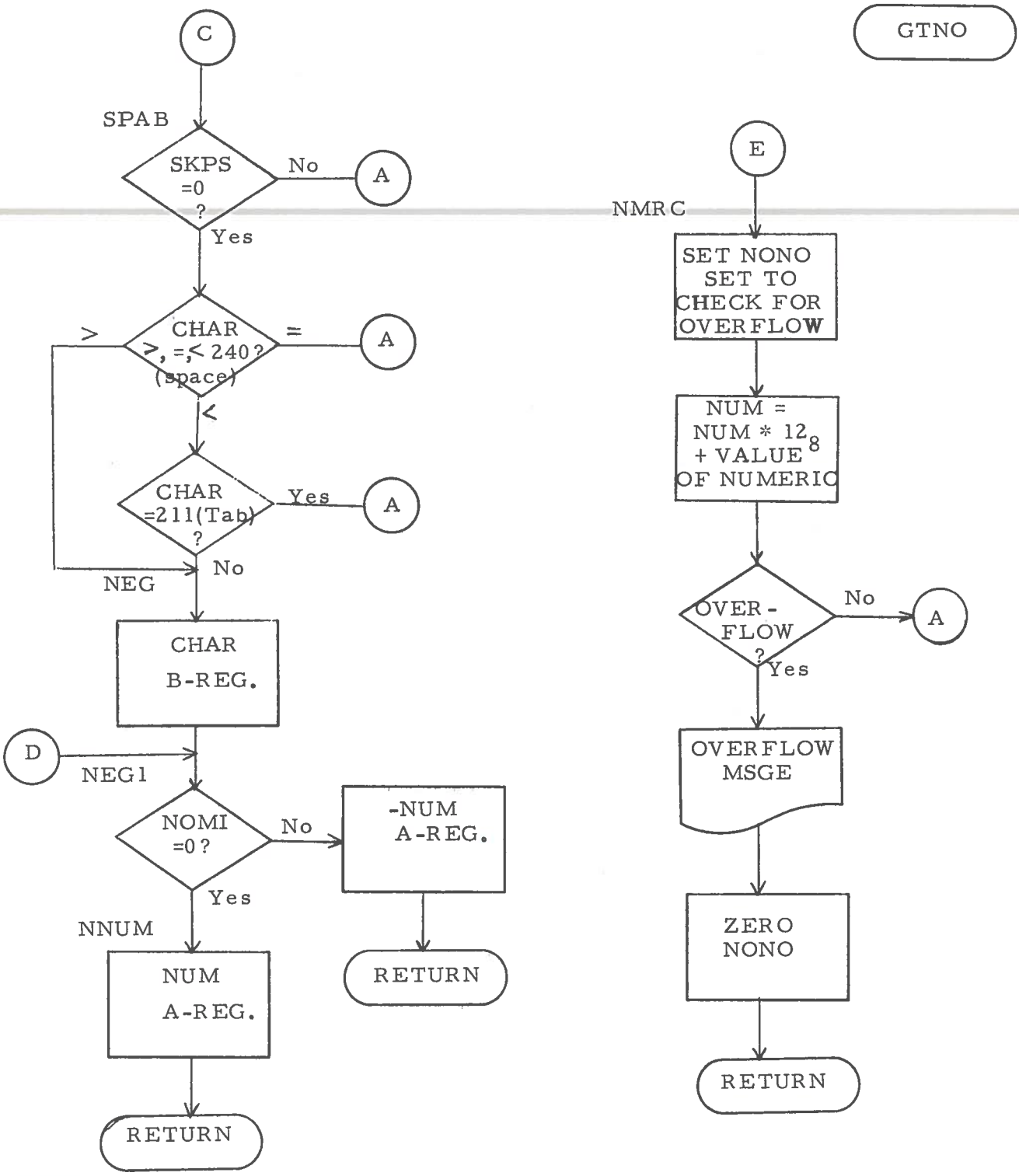
6. EXTERNAL REFERENCES:

SBLK
GT1C
TAO$

7. CORE USED:

161_8


GTNO
GTNO
INITIALIZE
SAVE SKIP SWITCH (SKPS)
NUM = 0
NONO = 0
NOMI = 0
SKPS > 0 ?
Yes
No
SBLK
A
GET1
E
Yes
CHAR NUMERIC ?
No
CHECK IF CHAR SHOULD BE IGNORED
CHKN
D
Yes
SKPS > 0 ?
No
NONO =0 ?
No
D
Yes
NOTN
C
CHAR > ,= ,< 255? (-)
=
NOMI = 1
A
B
B
CART
CHAR =215 (C.R.)
Yes
CHAR → B-REG ZERO A-REG
RETURN


GTNO
C
SPAB
SKPS =0 ?
No
A
Yes
CHAR >, =, < 240? (space)
>
=
A
<
CHAR =211(Tab) ?
Yes
A
No
NEG
CHAR B-REG.
D
NEG1
NOMI =0?
No
-NUM A-REG.
Yes
NNUM
NUM A-REG.
RETURN
RETURN
E
NMRC
SET NONO SET TO CHECK FOR OVERFLOW
NUM = NUM * 12 8 + VALUE OF NUMERIC
OVER - FLOW ?
No
A
Yes
OVERFLOW MSGE
ZERO NONO
RETURN

SUBROUTINE: IDER

Check Identification Number

1. PURPOSE:

Get identification number and check range.

2. CALLING SEQUENCE:

CALL IDER

3. INPUT:

ID number extracted from input buffer

4. OUTPUT:

a) If $1 \leq ID \leq 64$, return ID number in A-register. otherwise output error message.

5. ACTION:

A number is extracted from the input buffer. If it is a number from 1 to 64, it is returned to the calling program in the A-register.

If the number is not in the above range, an error message is typed and the error return is taken.

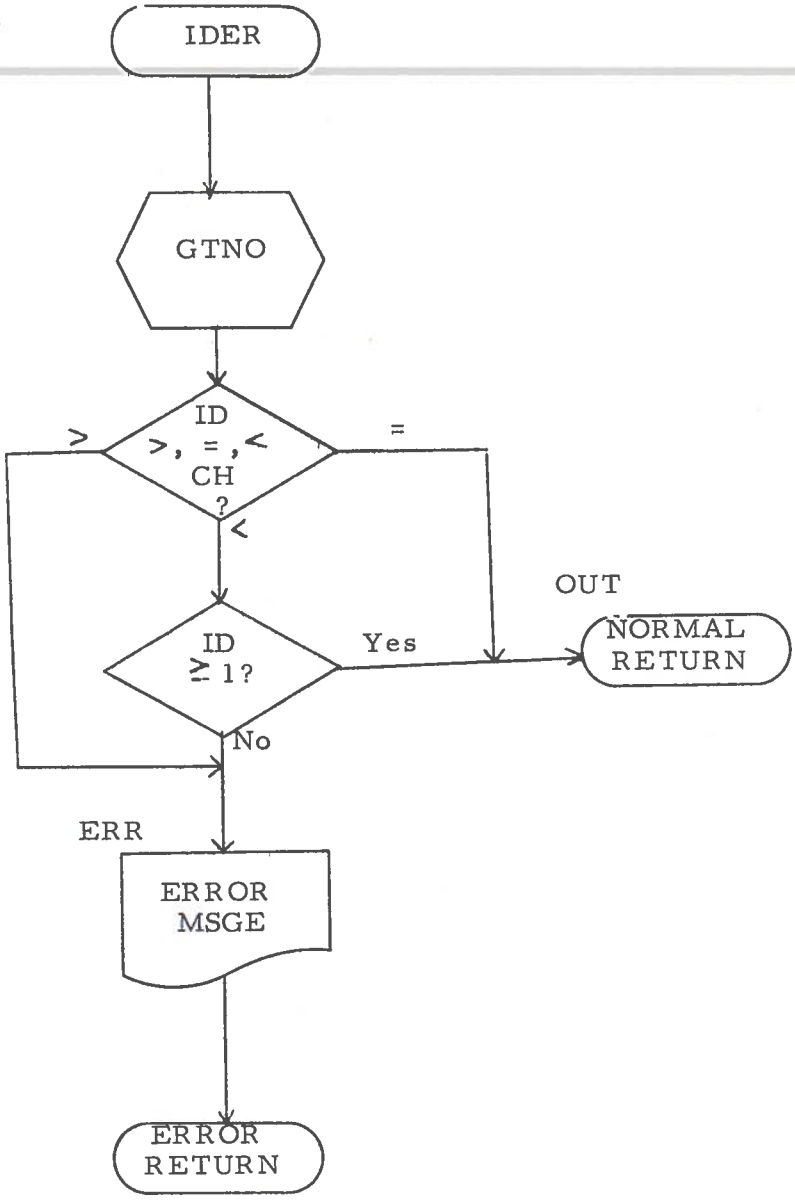
6. EXTERNAL REFERENCE:

GTNO
TAO$

7. CORE USED:

34_8

IDER


IDER
GTNO
ID
>, =, <
CH
?
>
=
<
ID
≥ 1?
Yes
No
OUT
NORMAL
RETURN
ERR
ERROR
MSGE
ERROR
RETURN

INPUT BUFFER SERVICE ROUTINES

1. PURPOSE:

To retrieve characters from input buffer

2. ENTRY POINTS AND CALLING SEQUENCE:

a) CHBF	70 character buffer; no calling sequence
b) CALL CHI	Initializes flags and pointers to retrieve initial character from buffer ignoring spaces and tabs
c) CALL SBLK	Resets flag to ignore spaces and tabs
d) CALL GTCH	Retrieve two characters; return with first in A-register, second in B-register
e) CALL GT1C	Retrieve one character; return with character in A-register
f) CALL SAVP	Saves flags and ptrs to return to current retrieval location
g) CALL RESP	Restores flags and ptrs to previously saved location

3. INPUT:

Characters in CHBF

4. OUTPUT:

None for a, b, c, f, g. Two characters from buffer for d. One character from buffer for e.

5. ACTION:

a) No action

b) Pointer is set to beginning of input buffer (CHBF). Half word indicator (LEFT) is zeroed to indicate character is to be retrieved from left half of buffer word. Resets switch (BSSW) to ignore spaces and tabs. Returns to calling program.

c) BSSW reset to ignore spaces and tabs. Returns to calling program.

d) Calls GT1C to get one character. Stores in B-register. Calls GT1C to get second character. Exchanges A-and B-registers. Returns to calling program.

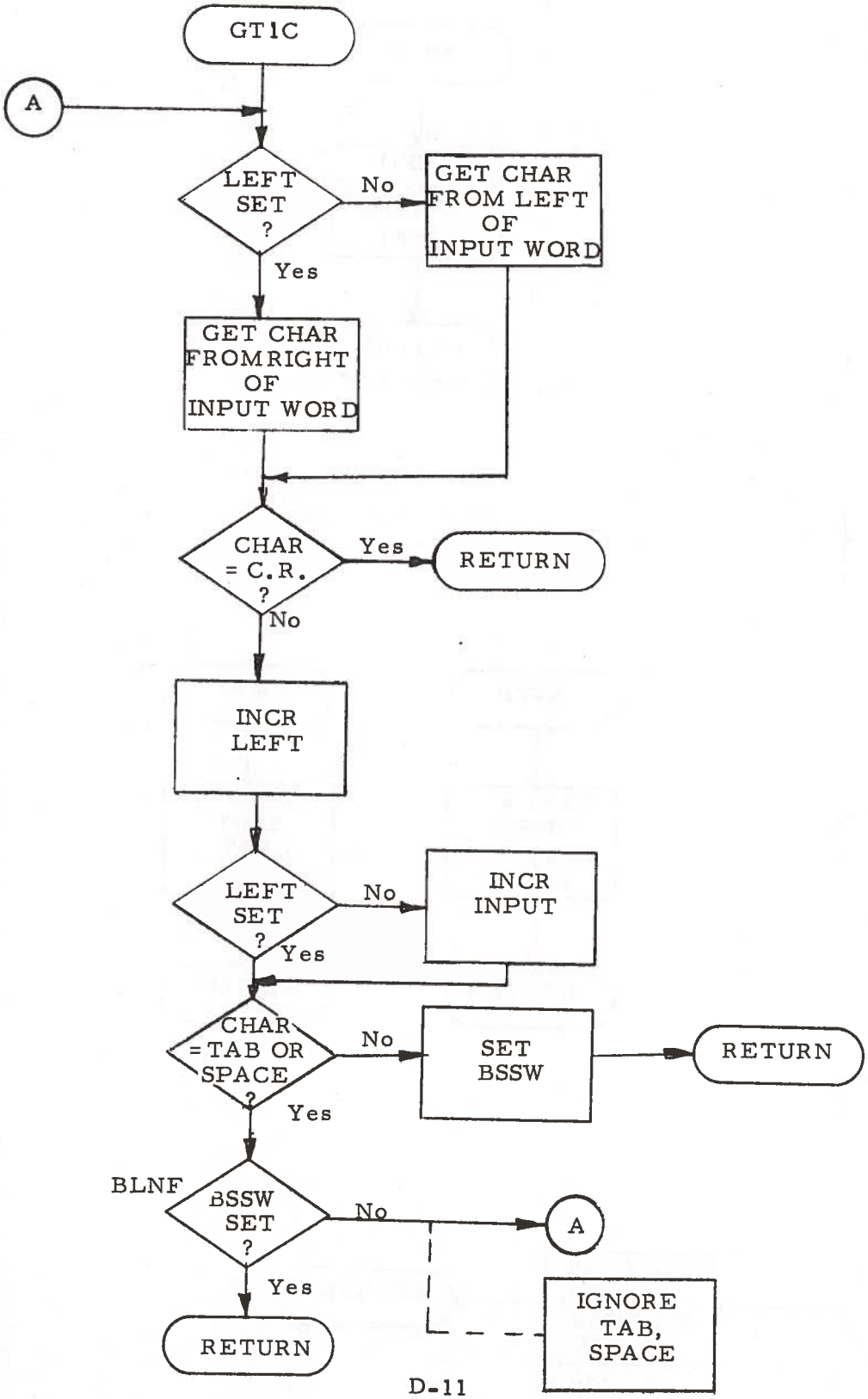
e) Checks LEFT to determine which half of buffer word to take character from. If character is a carriage return, pointers are not incremented and control is returned to the calling program. Otherwise LEFT is switched to get next character from opposite half of word. The buffer pointer is incremented if LEFT now indicates the left half of the word. If the character is a space or a tab, BSSW is checked to determine if the space or tab should be ignored and another character retrieved. When an acceptable character is found, it is returned to the calling program through the A-register.

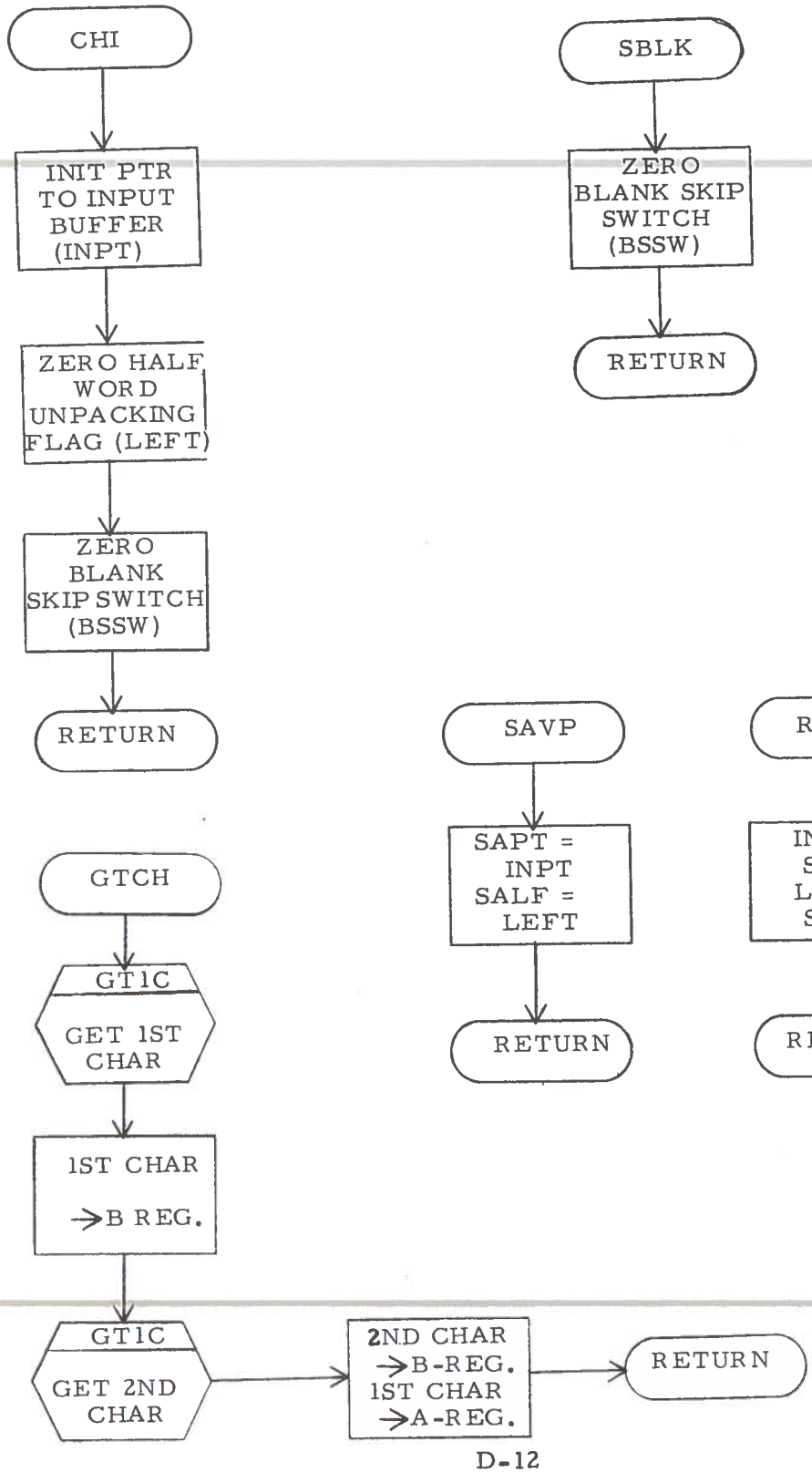
f) Stores ptrs and flags used for retrieval of chars. in temporary locations.

g) Restores ptrs and flags previously saved.

7. CORE USED:

151_8


GT1C
A
LEFT SET ?
No
GET CHAR FROM LEFT OF INPUT WORD
Yes
GET CHAR FROMRIGHT OF INPUT WORD
CHAR = C.R. ?
Yes
RETURN
No
INCR LEFT
LEFT SET ?
No
INCR INPUT
Yes
CHAR = TAB OR SPACE ?
No
SET BSSW
RETURN
Yes
BLNF
BSSW SET ?
No
A
Yes
RETURN
IGNORE TAB, SPACE


CHI
INIT PTR TO INPUT BUFFER (INPT)
ZERO HALF WORD UNPACKING FLAG (LEFT)
ZERO BLANK SKIP SWITCH (BSSW)
RETURN
SBLK
ZERO BLANK SKIP SWITCH (BSSW)
RETURN
SAVP
SAPT = INPT
SALF = LEFT
RETURN
RESP
INPT = SAPT
LEFT = SALF
RETURN
GTCH
GT1C
GET 1ST CHAR
1ST CHAR →B REG.
GT1C
GET 2ND CHAR
2ND CHAR →B-REG.
1ST CHAR →A-REG.
RETURN

LTYP

1. PURPOSE:

Interprets arguments specifying line type (texture).

2. CALLING SEQUENCE:

CALL LTYP

3. INPUT:

Characters from input buffer

4. OUTPUT:

A-Reg	=	-1;	illegal first char.; illegal char new in B-Reg.
	=	0;	code for solid line
	=	1;	code for dotted line
	=	2;	code for dashed line
	=	3;	code for dot-dash line

5. ACTION:

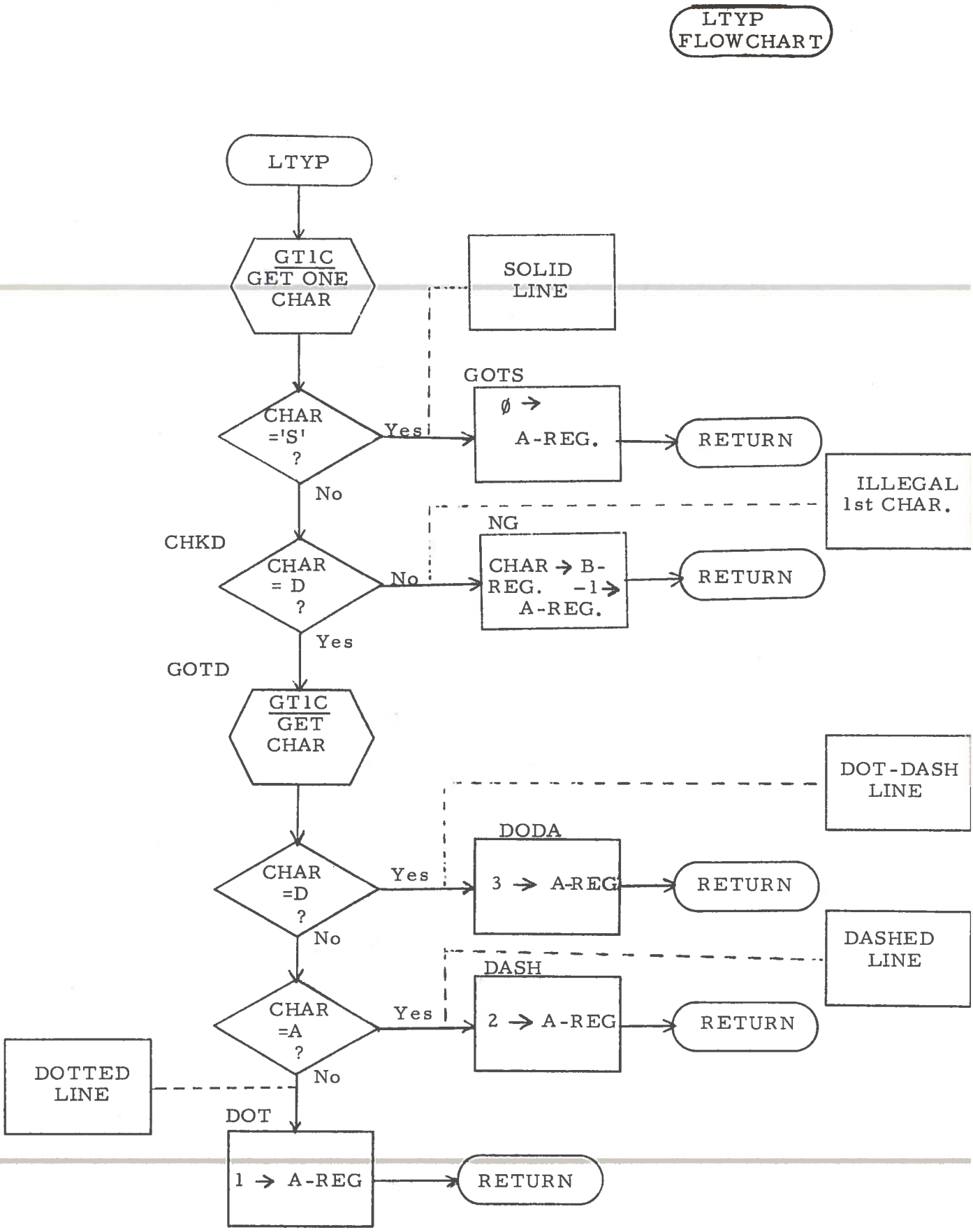
Gets char. and checks for S or D. If D, checks for A, or D for second char. Returns appropriate type or error code.

6. EXTERNAL REFERENCES:

GT1C

7. CORE USED:

41_8


LTYP FLOWCHART
LTYP
GT1C GET ONE CHAR
SOLID LINE
GOTS
CHAR ='S' ?
Yes
Ø → A-REG.
RETURN
No
ILLEGAL 1st CHAR.
NG
CHKD
CHAR = D ?
No
CHAR → B-REG. -1 → A-REG.
RETURN
Yes
GOTD
GT1C GET CHAR
DOT-DASH LINE
DODA
CHAR =D ?
Yes
3 → A-REG
RETURN
No
DASHED LINE
DASH
CHAR =A ?
Yes
2 → A-REG
RETURN
DOTTED LINE
No
DOT
1 → A-REG
RETURN

SUBROUTINE OUTP

1. PURPOSE:

Stores output file on disc or paper tape after compiling any expression found in register definition and conditioning blocks.

2. CALLING SEQUENCE:

```
JST  OUTP
DAC  FRHD
DAC  FPNT
DAC  FLG3
XAC  GETF
DAC  RELF
DAC  FREE
DAC  LRG
DAC  LST
(NAME)
(NAME + 1)
(NAME + 2)
```

3. INPUT:

See calling sequence.

4. OUTPUT:

The data structure is saved on disc or paper tape after the dynamic expressions are compiled and the conditioning blocks are compressed.

5. ACTION:

The conditioning blocks in the data structure are searched for any dynamic expressions which have not been previously compiled. If not, the mini-compiler (SCOM) is called and the start address of the object code is stored in the conditioning block. After conditioning block compression, unused storage is returned to the free storage ring.

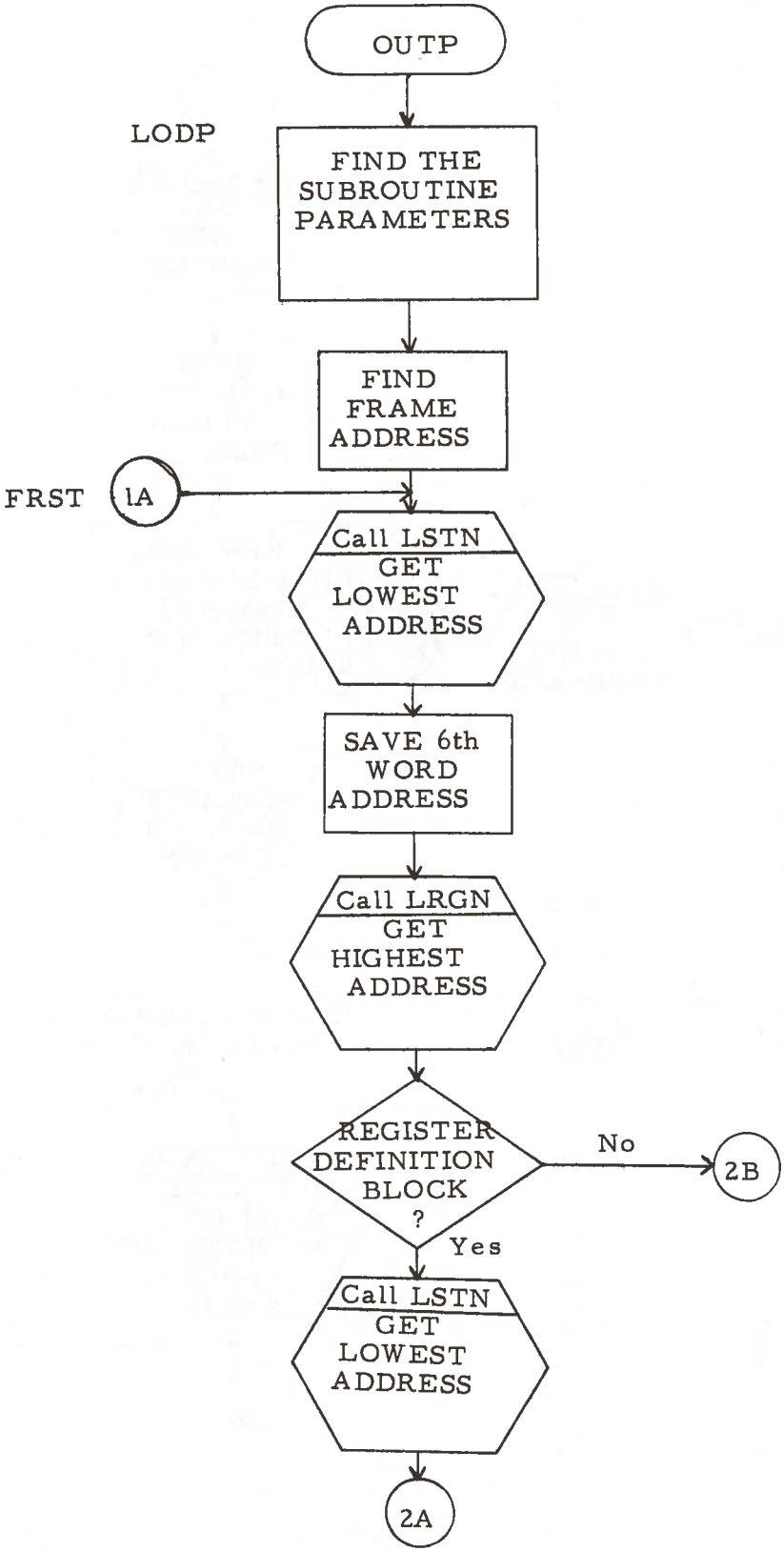
Finally, the data structure is output on either disc or paper tape.

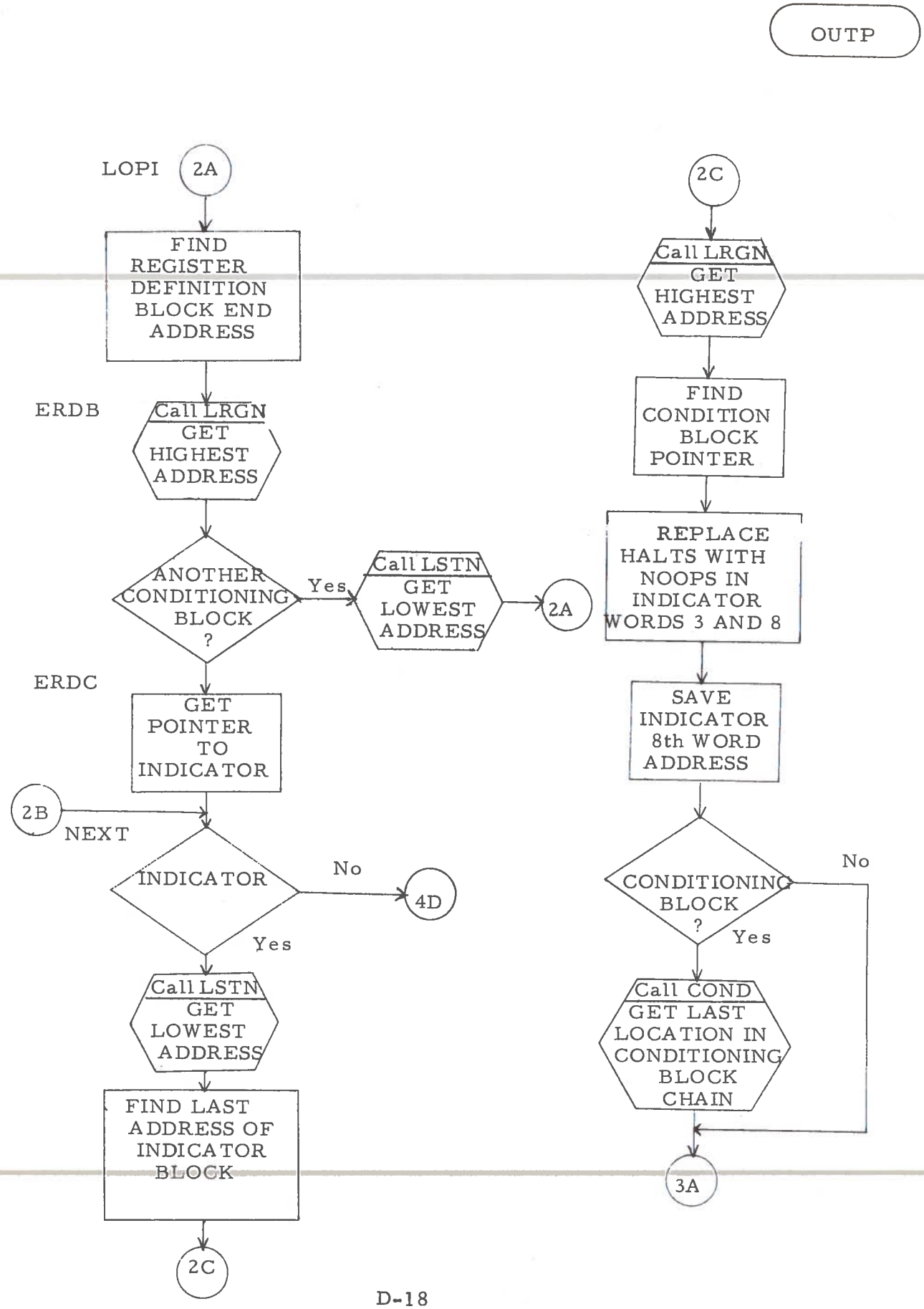
6. EXTERNAL REFERENCES:

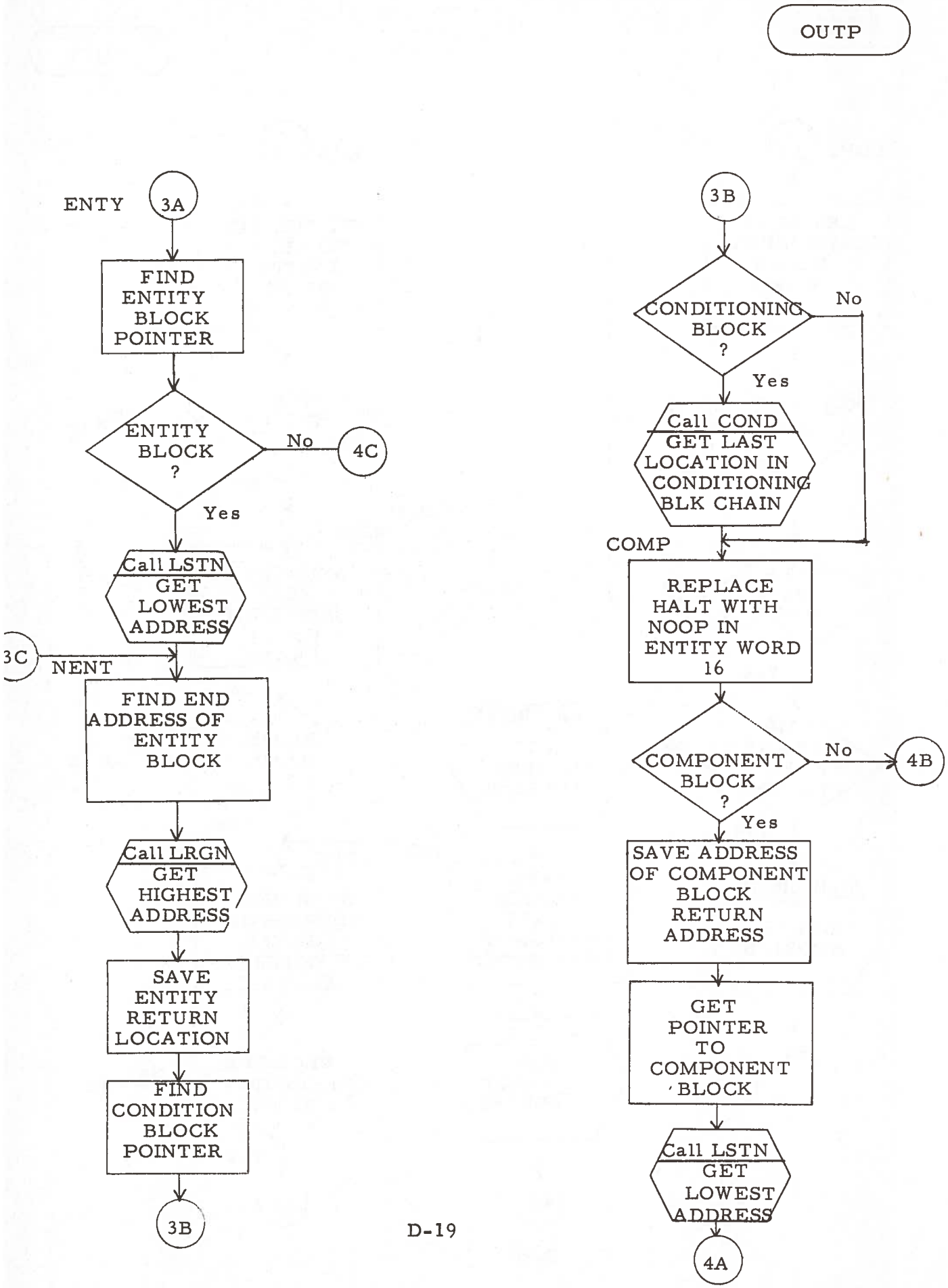
External subroutines and addresses are referenced by the calling sequence.

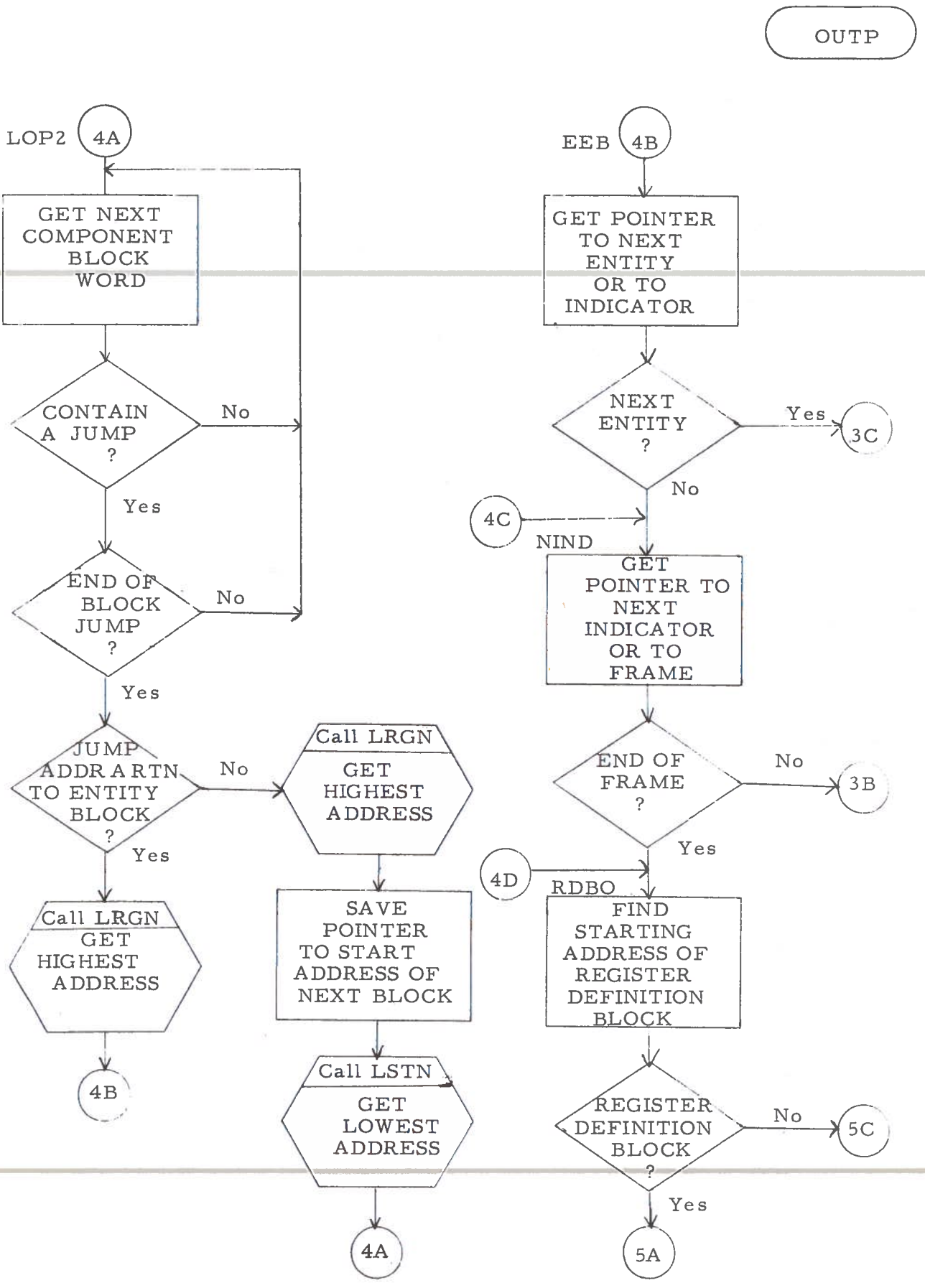
7. CORE USED:

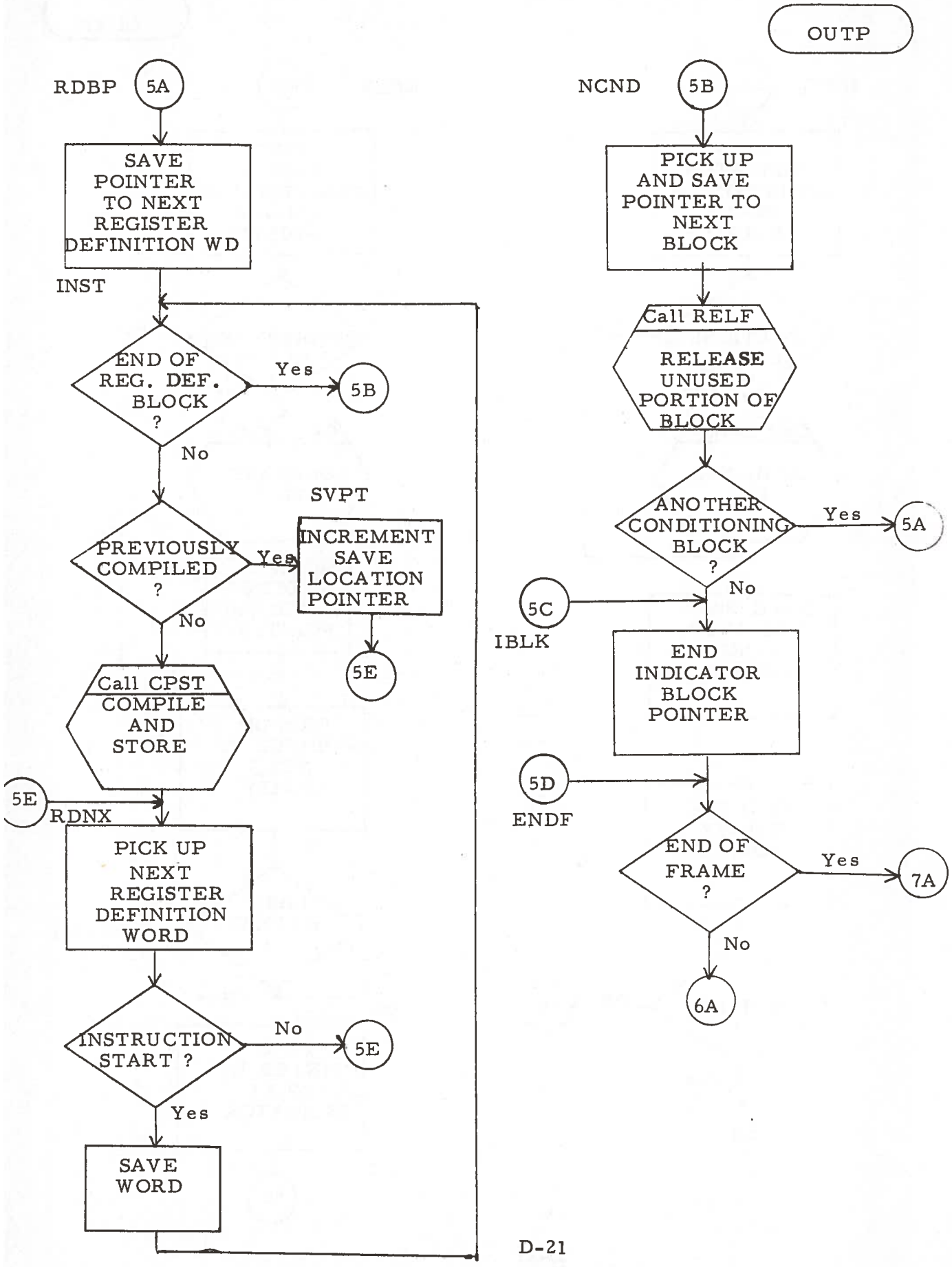
1103_8

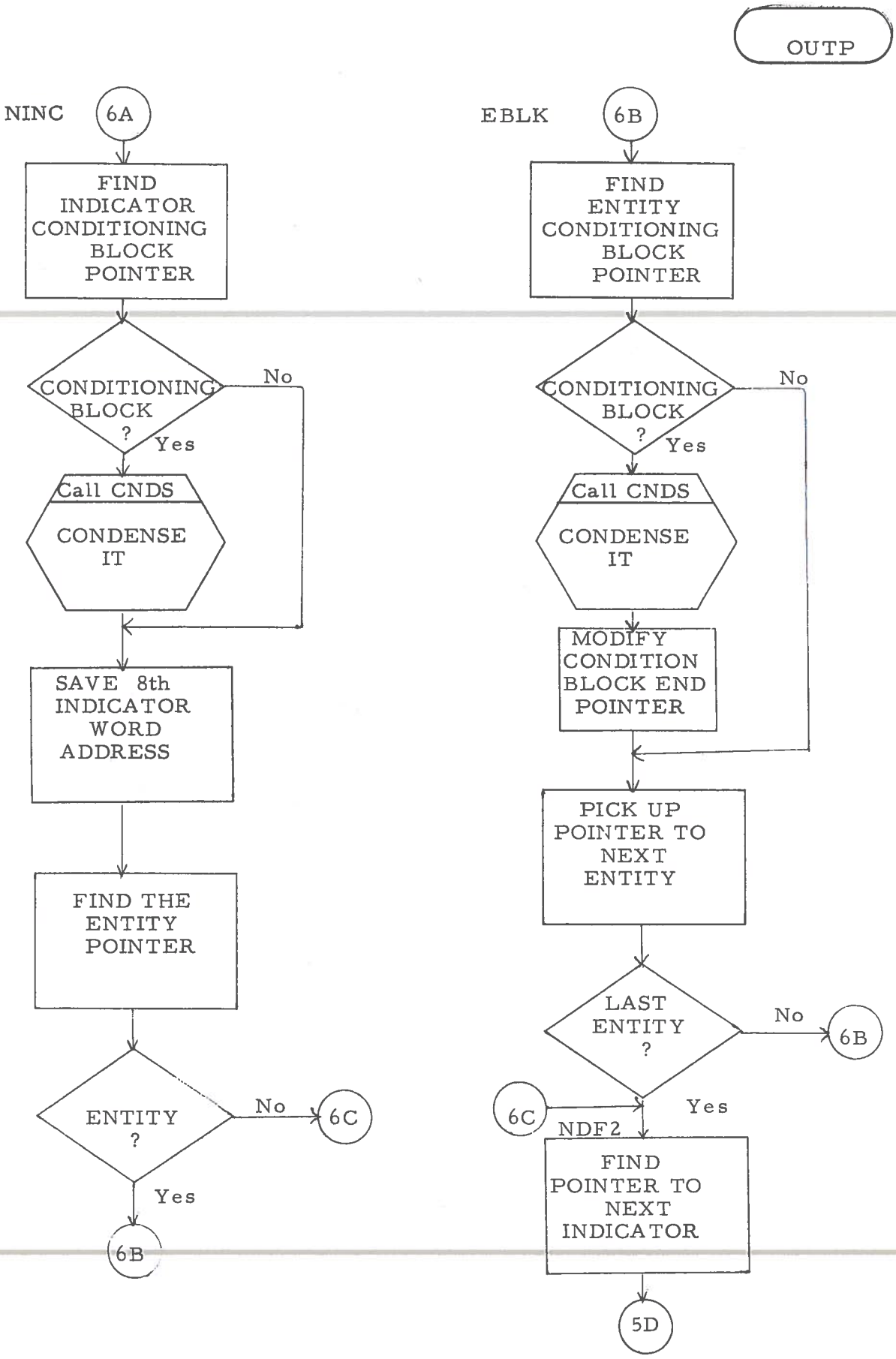

OUTP
LODP
FIND THE SUBROUTINE PARAMETERS
FIND FRAME ADDRESS
FRST
1A
Call LSTN
GET LOWEST ADDRESS
SAVE 6th WORD ADDRESS
Call LRGN
GET HIGHEST ADDRESS
REGISTER DEFINITION BLOCK ?
No
2B
Yes
Call LSTN
GET LOWEST ADDRESS
2A

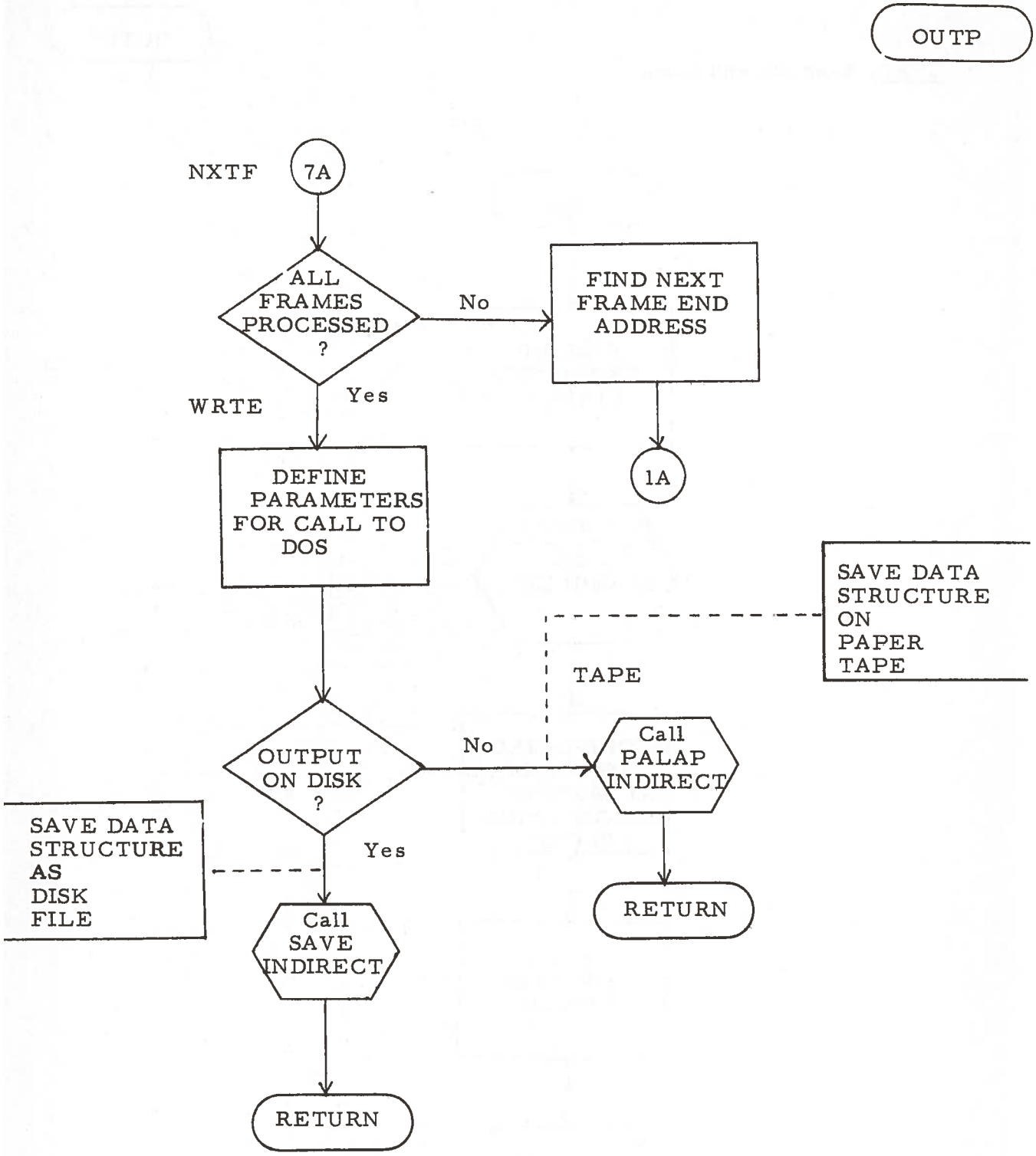

OUTP
LOPI
2A
FIND REGISTER DEFINITION BLOCK END ADDRESS
ERDB
Call LRGN
GET HIGHEST ADDRESS
ANOTHER CONDITIONING BLOCK ?
Yes
Call LSTN
GET LOWEST ADDRESS
2A
ERDC
GET POINTER TO INDICATOR
2B
NEXT
INDICATOR
No
4D
Yes
Call LSTN
GET LOWEST ADDRESS
FIND LAST ADDRESS OF INDICATOR BLOCK
2C
2C
Call LRGN
GET HIGHEST ADDRESS
FIND CONDITION BLOCK POINTER
REPLACE HALTS WITH NOOPS IN INDICATOR WORDS 3 AND 8
SAVE INDICATOR 8th WORD ADDRESS
CONDITIONING BLOCK ?
No
Yes
Call COND
GET LAST LOCATION IN CONDITIONING BLOCK CHAIN
3A


OUTP
ENTY
3A
FIND ENTITY BLOCK POINTER
ENTITY BLOCK ?
No
4C
Yes
Call LSTN
GET LOWEST ADDRESS
3C
NENT
FIND END ADDRESS OF ENTITY BLOCK
Call LRGN
GET HIGHEST ADDRESS
SAVE ENTITY RETURN LOCATION
FIND CONDITION BLOCK POINTER
3B
3B
CONDITIONING BLOCK ?
No
Yes
Call COND
GET LAST LOCATION IN CONDITIONING BLK CHAIN
COMP
REPLACE HALT WITH NOOP IN ENTITY WORD 16
COMPONENT BLOCK ?
No
4B
Yes
SAVE ADDRESS OF COMPONENT BLOCK RETURN ADDRESS
GET POINTER TO COMPONENT BLOCK
Call LSTN
GET LOWEST ADDRESS
4A


OUTP
LOP2
4A
GET NEXT COMPONENT BLOCK WORD
CONTAIN A JUMP ?
No
Yes
END OF BLOCK JUMP ?
No
Yes
JUMP ADDR A RTN TO ENTITY BLOCK ?
No
Yes
Call LRGN
GET HIGHEST ADDRESS
4B
Call LRGN
GET HIGHEST ADDRESS
SAVE POINTER TO START ADDRESS OF NEXT BLOCK
Call LSTN
GET LOWEST ADDRESS
4A
EEB
4B
GET POINTER TO NEXT ENTITY OR TO INDICATOR
NEXT ENTITY ?
Yes
3C
No
4C
NIND
GET POINTER TO NEXT INDICATOR OR TO FRAME
END OF FRAME ?
No
3B
Yes
4D
RDBO
FIND STARTING ADDRESS OF REGISTER DEFINITION BLOCK
REGISTER DEFINITION BLOCK ?
No
5C
Yes
5A

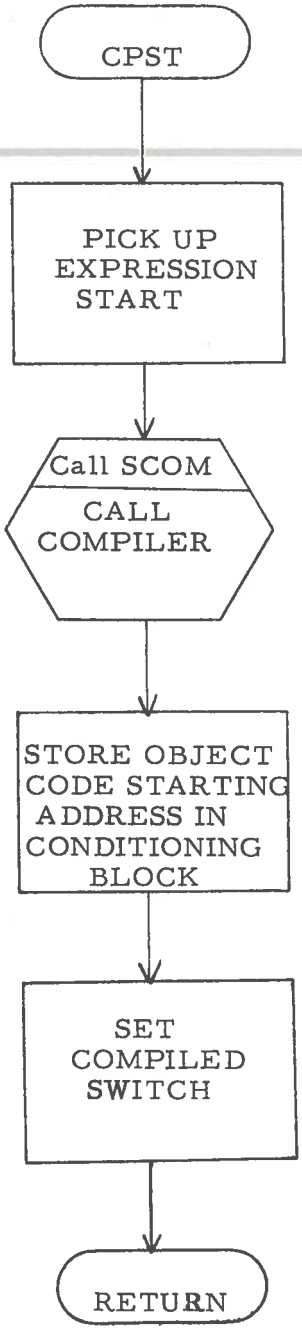

OUTP
RDBP
5A
SAVE POINTER TO NEXT REGISTER DEFINITION WD
INST
END OF REG. DEF. BLOCK ?
Yes
5B
No
SVPT
PREVIOUSLY COMPILED ?
Yes
INCREMENT SAVE LOCATION POINTER
5E
No
Call CPST
COMPILE AND STORE
5E
RDNX
PICK UP NEXT REGISTER DEFINITION WORD
INSTRUCTION START ?
No
5E
Yes
SAVE WORD
NCND
5B
PICK UP AND SAVE POINTER TO NEXT BLOCK
Call RELF
RELEASE UNUSED PORTION OF BLOCK
ANOTHER CONDITIONING BLOCK ?
Yes
5A
No
5C
IBLK
END INDICATOR BLOCK POINTER
5D
ENDF
END OF FRAME ?
Yes
7A
No
6A


OUTP
NINC
6A
FIND INDICATOR CONDITIONING BLOCK POINTER
CONDITIONING BLOCK ?
No
Yes
Call CNDS
CONDENSE IT
SAVE 8th INDICATOR WORD ADDRESS
FIND THE ENTITY POINTER
ENTITY ?
No
6C
Yes
6B
EBLK
6B
FIND ENTITY CONDITIONING BLOCK POINTER
CONDITIONING BLOCK ?
No
Yes
Call CNDS
CONDENSE IT
MODIFY CONDITION BLOCK END POINTER
PICK UP POINTER TO NEXT ENTITY
LAST ENTITY ?
No
6B
6C
Yes
NDF2
FIND POINTER TO NEXT INDICATOR
5D


OUTP
NXTF
7A
ALL FRAMES PROCESSED ?
No
FIND NEXT FRAME END ADDRESS
WRTE
Yes
1A
DEFINE PARAMETERS FOR CALL TO DOS
SAVE DATA STRUCTURE ON PAPER TAPE
TAPE
OUTPUT ON DISK ?
No
Call PALAP INDIRECT
SAVE DATA STRUCTURE AS DISK FILE
Yes
RETURN
Call SAVE INDIRECT
RETURN

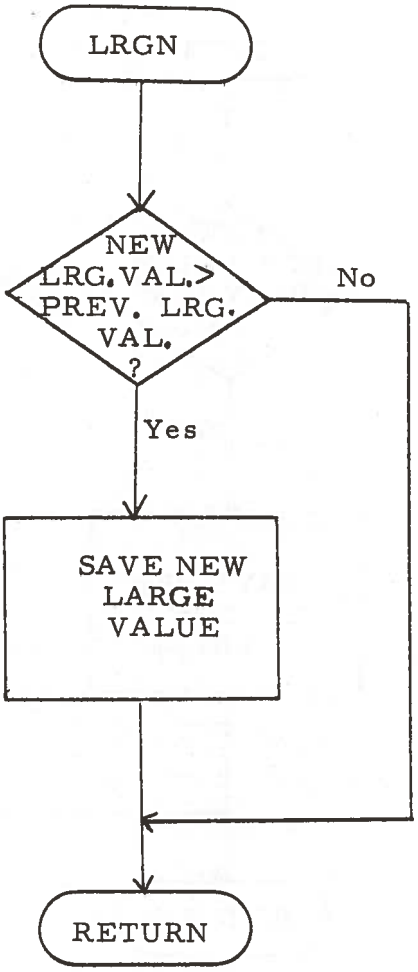
OUTP

CPST: Compile and Store



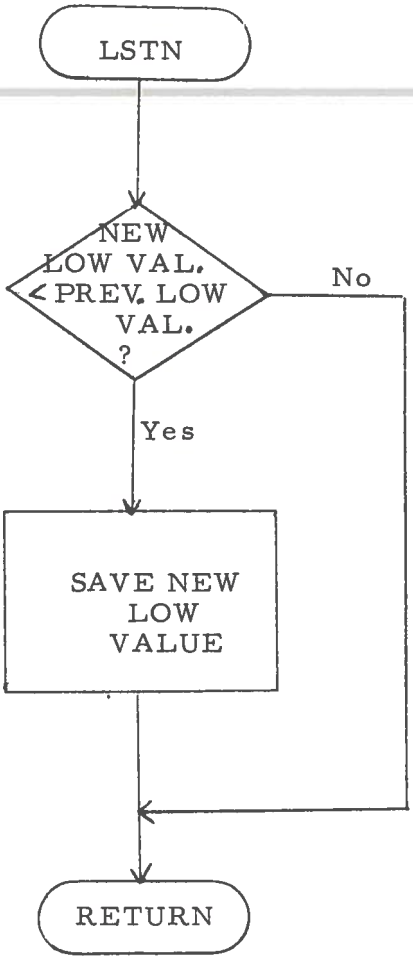
OUTP

LRGN: Find Highest Value



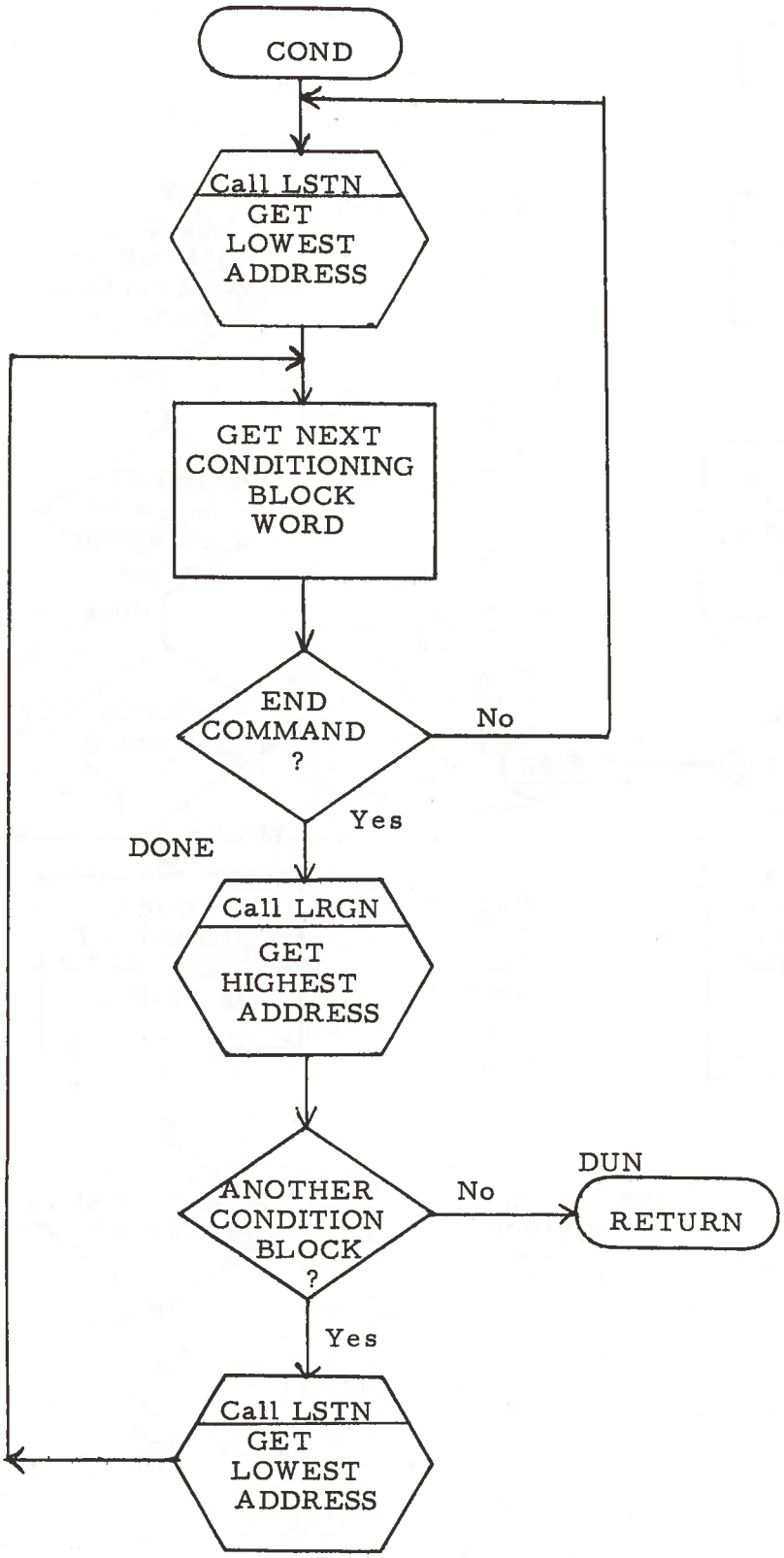
OUTP

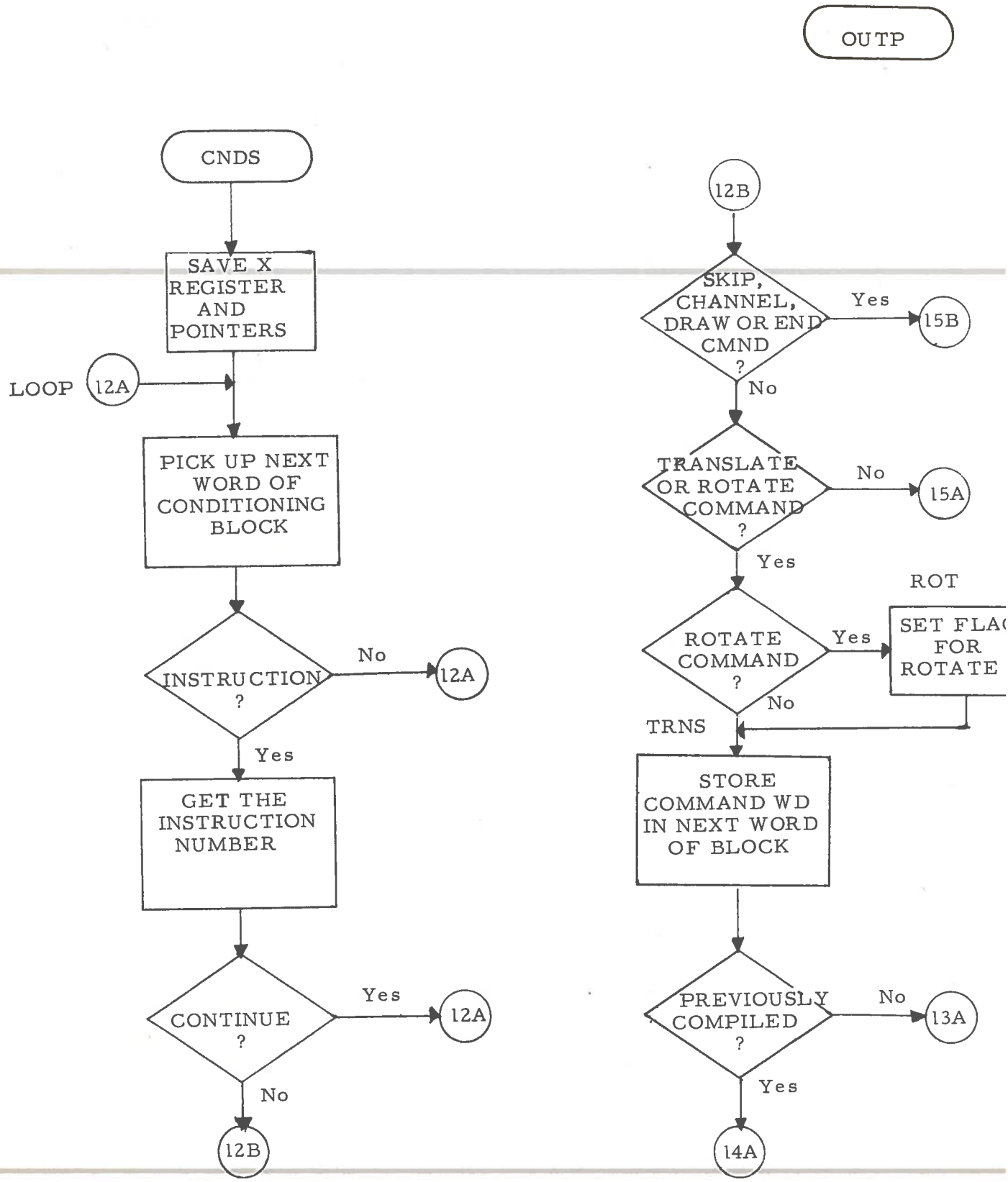
LSTN: Find Lowest Value

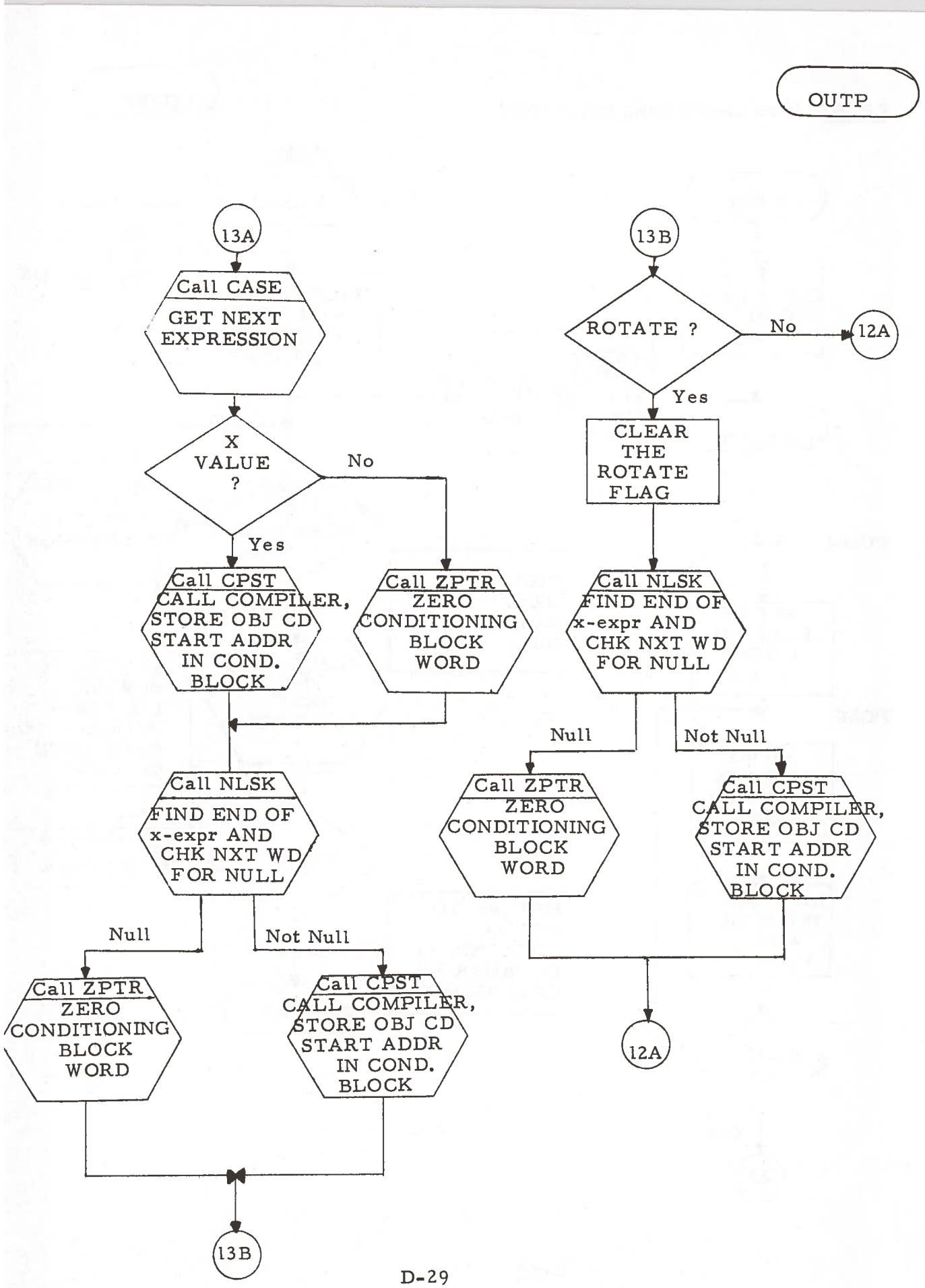


OUTP

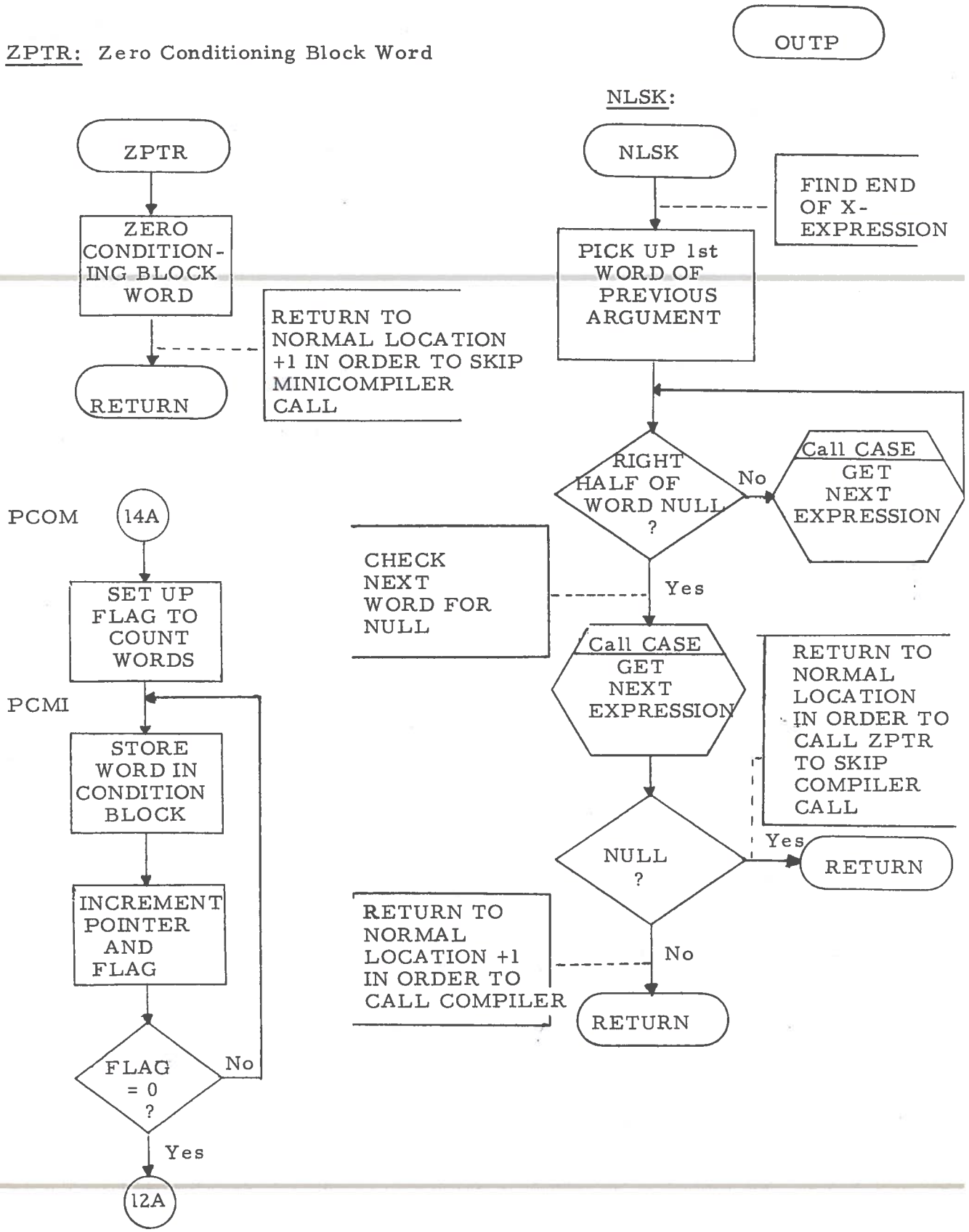
COND: Get Last Location in Conditioning Block Chain

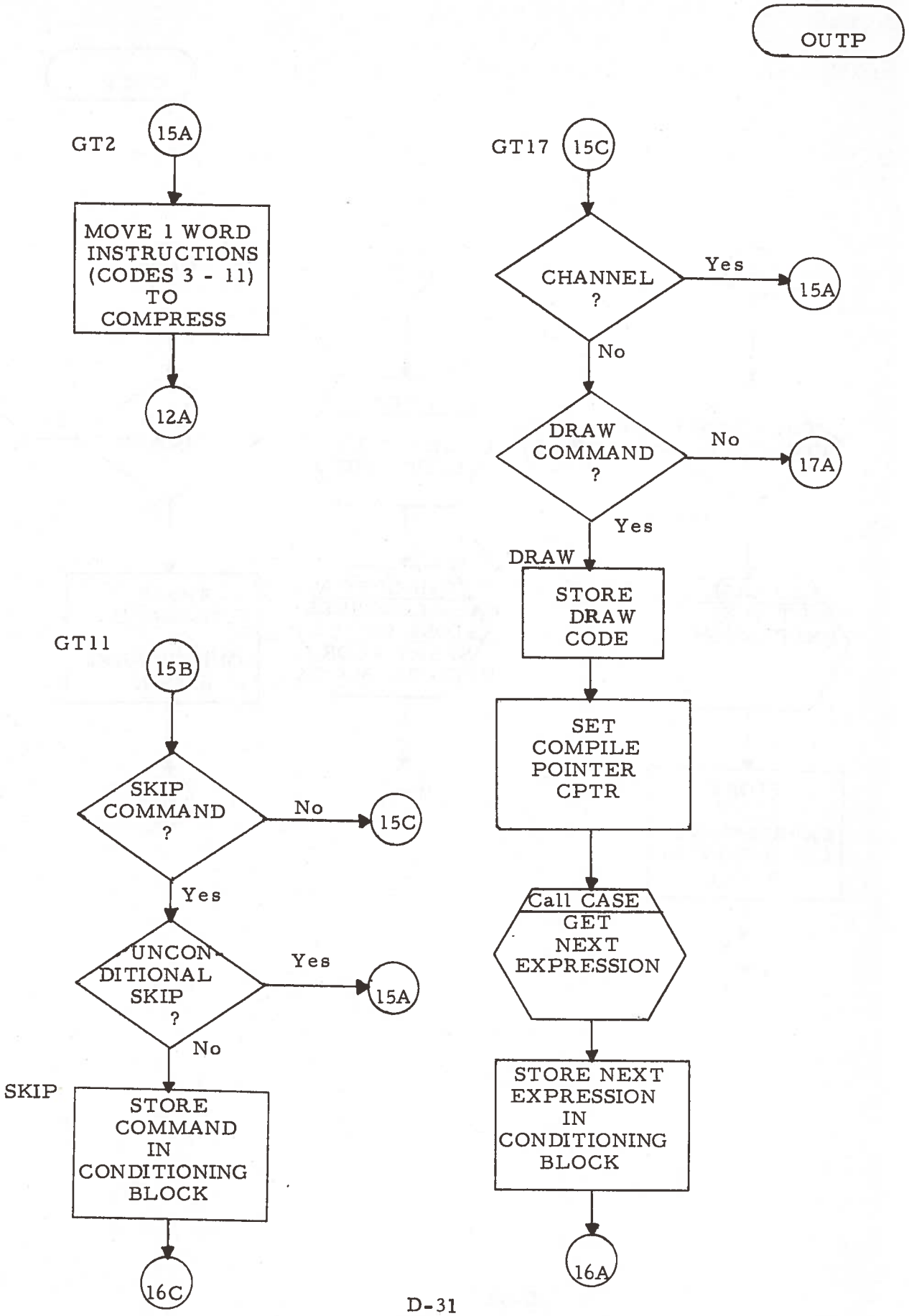


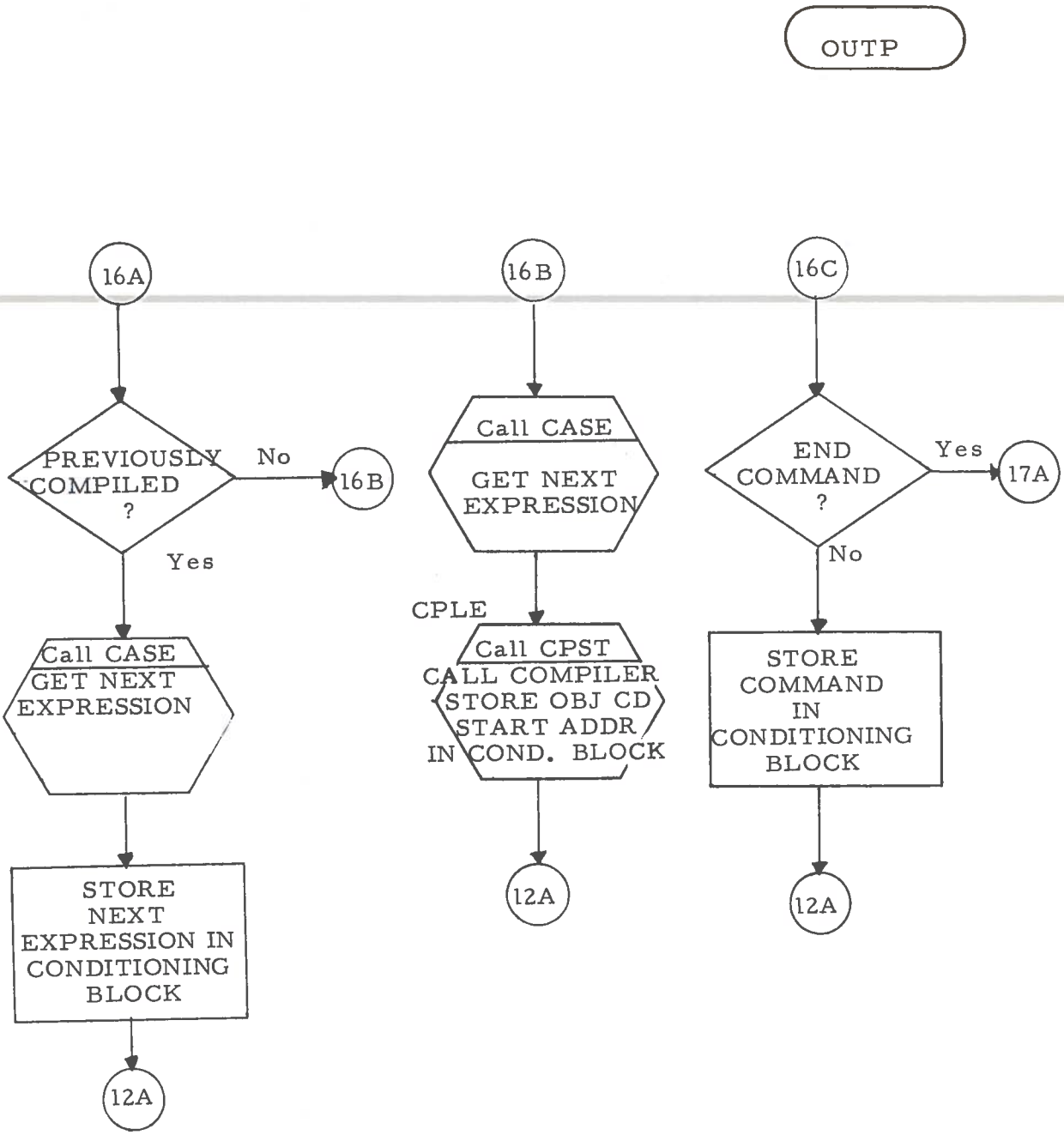

OUTP
CNDS
SAVE X REGISTER AND POINTERS
LOOP
12A
PICK UP NEXT WORD OF CONDITIONING BLOCK
INSTRUCTION ?
No
12A
Yes
GET THE INSTRUCTION NUMBER
CONTINUE ?
Yes
12A
No
12B
12B
SKIP, CHANNEL, DRAW OR END CMND ?
Yes
15B
No
TRANSLATE OR ROTATE COMMAND ?
No
15A
Yes
ROT
ROTATE COMMAND ?
Yes
SET FLA FOR ROTATE
No
TRNS
STORE COMMAND WD IN NEXT WORD OF BLOCK
PREVIOUSLY COMPILED ?
No
13A
Yes
14A

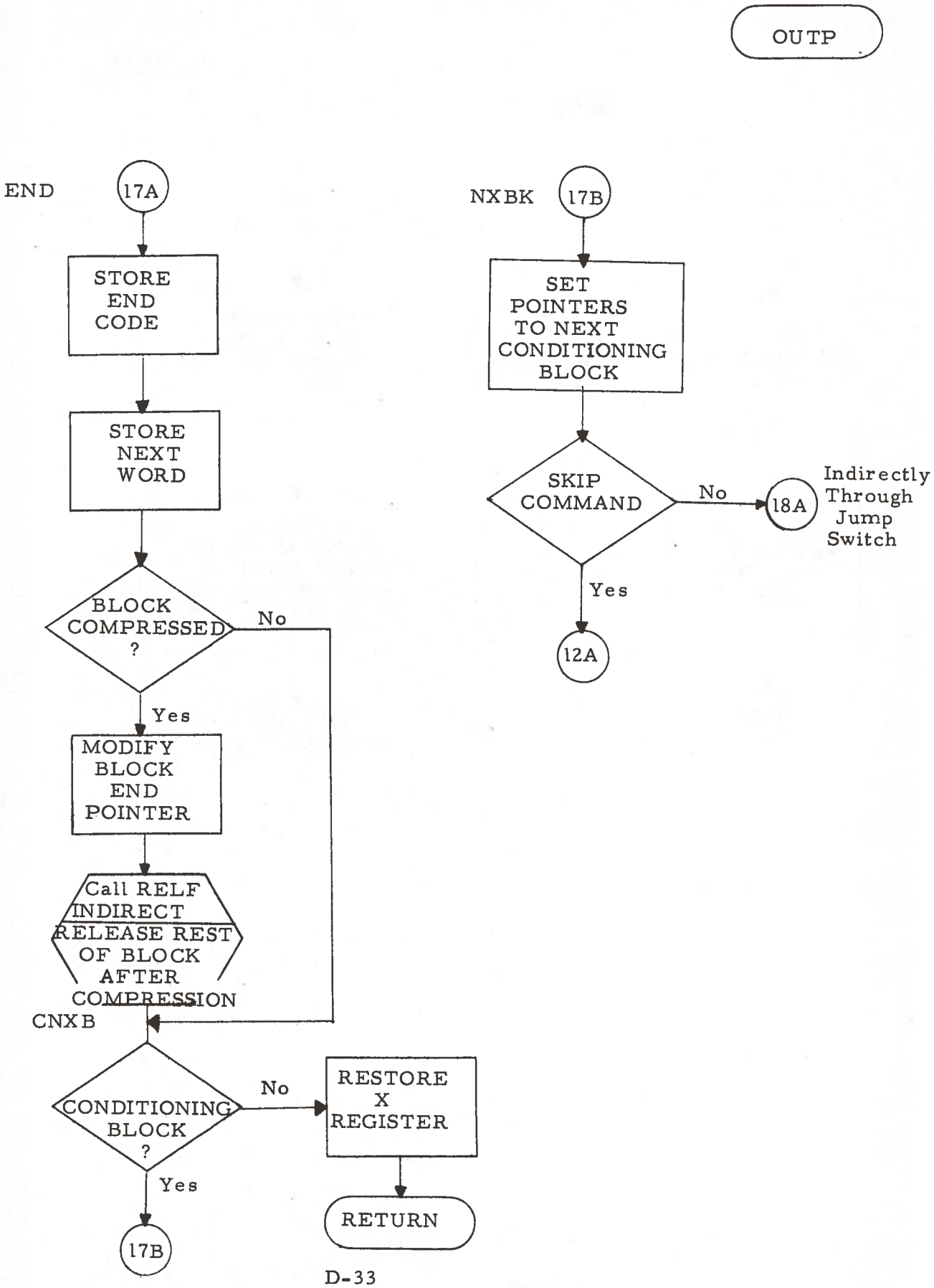

OUTP
13A
Call CASE
GET NEXT EXPRESSION
X VALUE ?
No
Yes
Call CPST
CALL COMPILER, STORE OBJ CD START ADDR IN COND. BLOCK
Call ZPTR
ZERO CONDITIONING BLOCK WORD
Call NLSK
FIND END OF x-expr AND CHK NXT WD FOR NULL
Null
Not Null
Call ZPTR
ZERO CONDITIONING BLOCK WORD
Call CPST
CALL COMPILER, STORE OBJ CD START ADDR IN COND. BLOCK
13B
ROTATE ?
No
12A
Yes
CLEAR THE ROTATE FLAG
Call NLSK
FIND END OF x-expr AND CHK NXT WD FOR NULL
Null
Not Null
Call ZPTR
ZERO CONDITIONING BLOCK WORD
Call CPST
CALL COMPILER, STORE OBJ CD START ADDR IN COND. BLOCK
12A
13B

ZPTR: Zero Conditioning Block Word

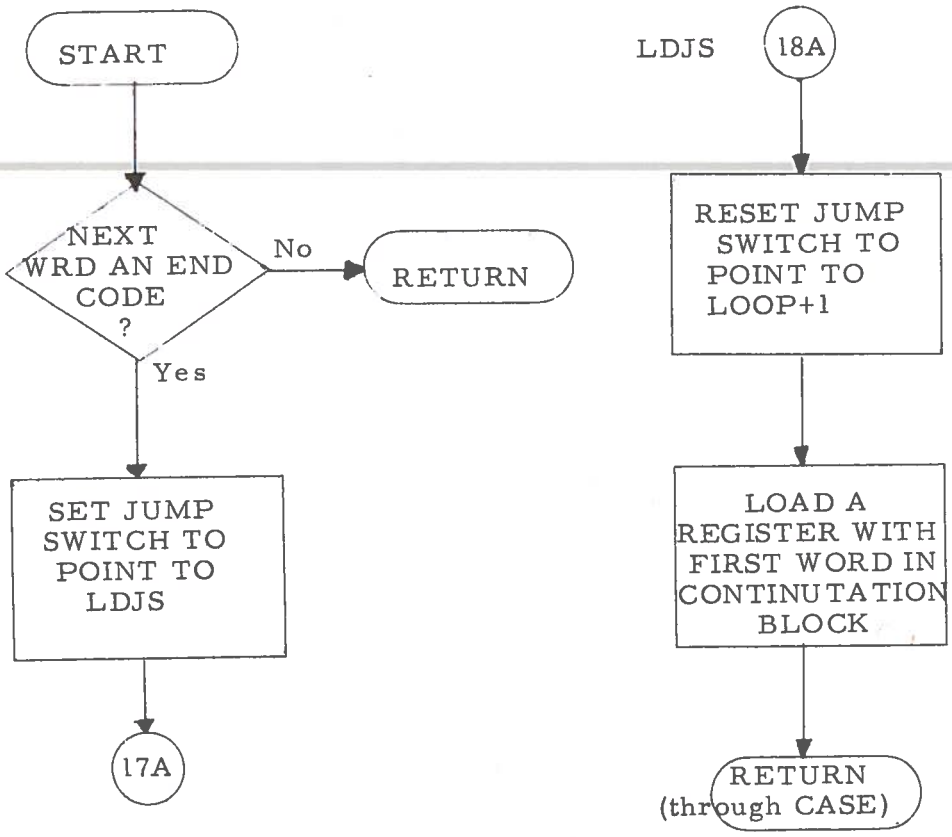



OUTP
GT2
15A
MOVE 1 WORD INSTRUCTIONS (CODES 3 - 11) TO COMPRESS
12A
GT17
15C
CHANNEL ?
Yes
15A
No
DRAW COMMAND ?
No
17A
Yes
DRAW
STORE DRAW CODE
SET COMPILE POINTER CPTR
Call CASE
GET NEXT EXPRESSION
STORE NEXT EXPRESSION IN CONDITIONING BLOCK
16A
GT11
15B
SKIP COMMAND ?
No
15C
Yes
UNCON-DITIONAL SKIP ?
Yes
15A
No
SKIP
STORE COMMAND IN CONDITIONING BLOCK
16C


OUTP
16A
PREVIOUSLY COMPILED ?
No
16B
Yes
Call CASE
GET NEXT EXPRESSION
STORE NEXT EXPRESSION IN CONDITIONING BLOCK
12A
16B
Call CASE
GET NEXT EXPRESSION
CPLE
Call CPST
CALL COMPILER
STORE OBJ CD START ADDR IN COND. BLOCK
12A
16C
END COMMAND ?
Yes
17A
No
STORE COMMAND IN CONDITIONING BLOCK
12A


OUTP
END
17A
STORE END CODE
STORE NEXT WORD
BLOCK COMPRESSED ?
No
Yes
MODIFY BLOCK END POINTER
Call RELF INDIRECT RELEASE REST OF BLOCK AFTER COMPRESSION
CNXB
CONDITIONING BLOCK ?
No
RESTORE X REGISTER
RETURN
Yes
17B
NXBK
17B
SET POINTERS TO NEXT CONDITIONING BLOCK
SKIP COMMAND
No
18A
Indirectly Through Jump Switch
Yes
12A

CASE


START
NEXT WRD AN END CODE ?
No
RETURN
Yes
SET JUMP SWITCH TO POINT TO LDJS
17A
LDJS
18A
RESET JUMP SWITCH TO POINT TO LOOP+1
LOAD A REGISTER WITH FIRST WORD IN CONTINUTATION BLOCK
RETURN (through CASE)

SUBROUTINE SCOM

1. PURPOSE:

Compile Fortran type arithmetic statements using integer arithmetic and register numbers as variables.

2. CALLING SEQUENCE:

```
LDA     PTR
CALL    SCOM
```

Where PTR is start address of dynamic expression.

3. INPUT:

Start address of dynamic expression from calling sequence.

4. OUTPUT:

Start address of compiled object code in A register.

5. ACTION:

SCOM compiles a dynamic expression found in the register definition or conditioning blocks. The free storage necessary for the object code is obtained from the free storage ring. The hierarchy of arithmetic operation is:

	Subroutines	Class 0
**	Exponentiation	Class 1
/	Division	Class 2
*	Multiplication	
+	Addition	Class 3
-	Subtraction	

6. EXTERNAL REFERENCES:

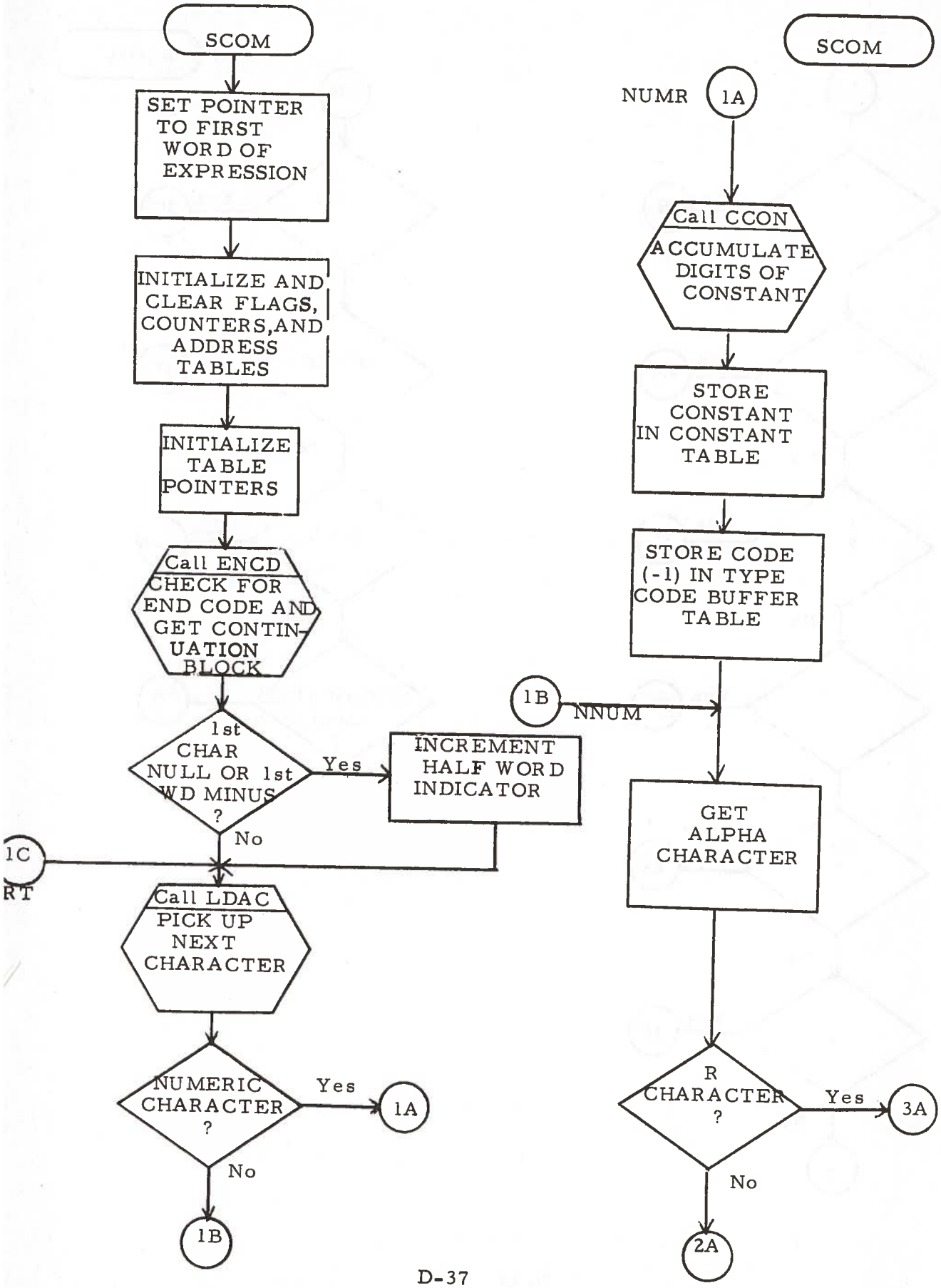
GETF	Get block from free storage
IABS	Computes absolute value of integer argument
ISQRT	Computes square root of integer argument
IMAX	Maximum value of specified variables
IMIN	Minimum value of specified variables
ISIGN	Transfers sign from argument j to absolute value of argument i. (ISIGN (i, j)).

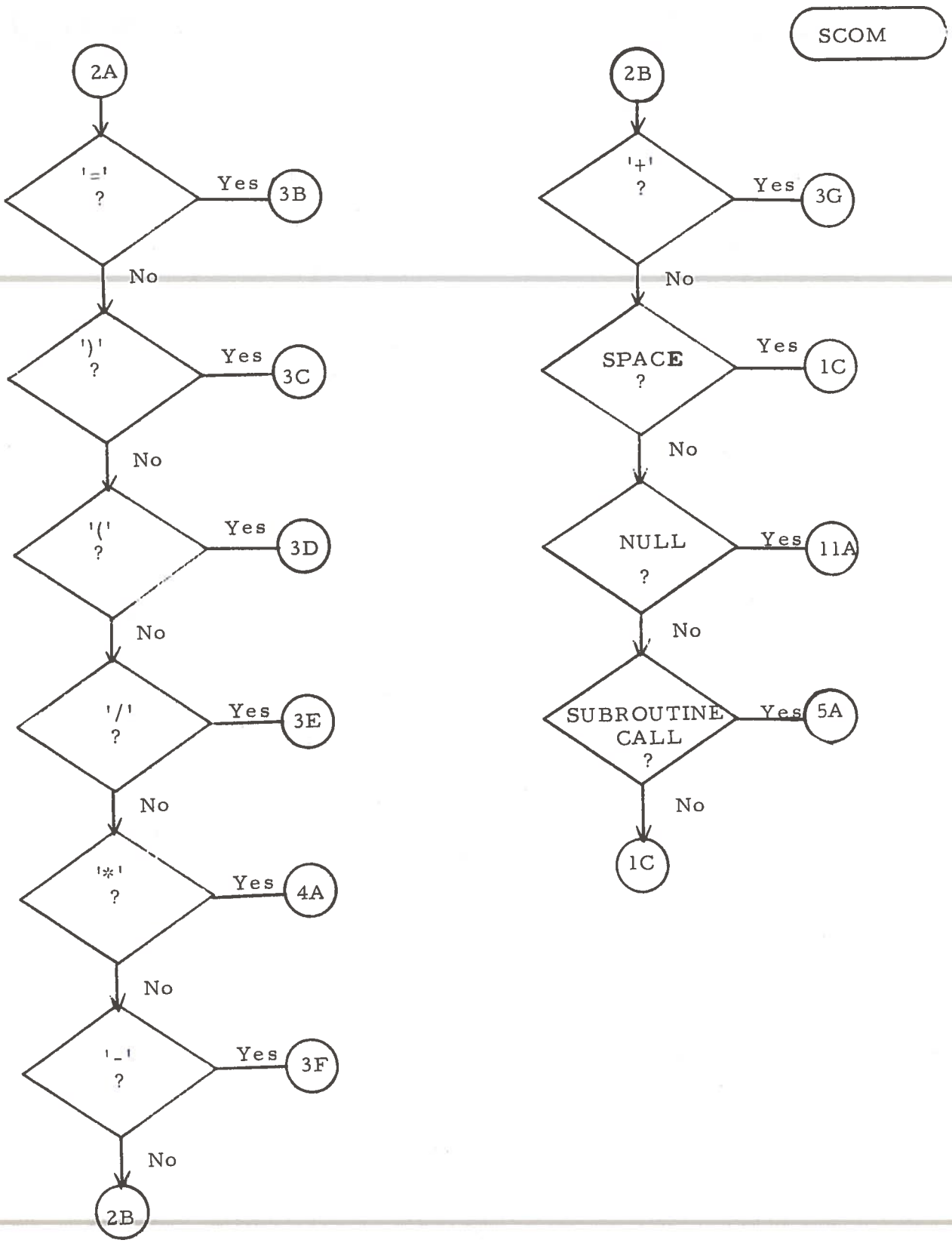
IDIM

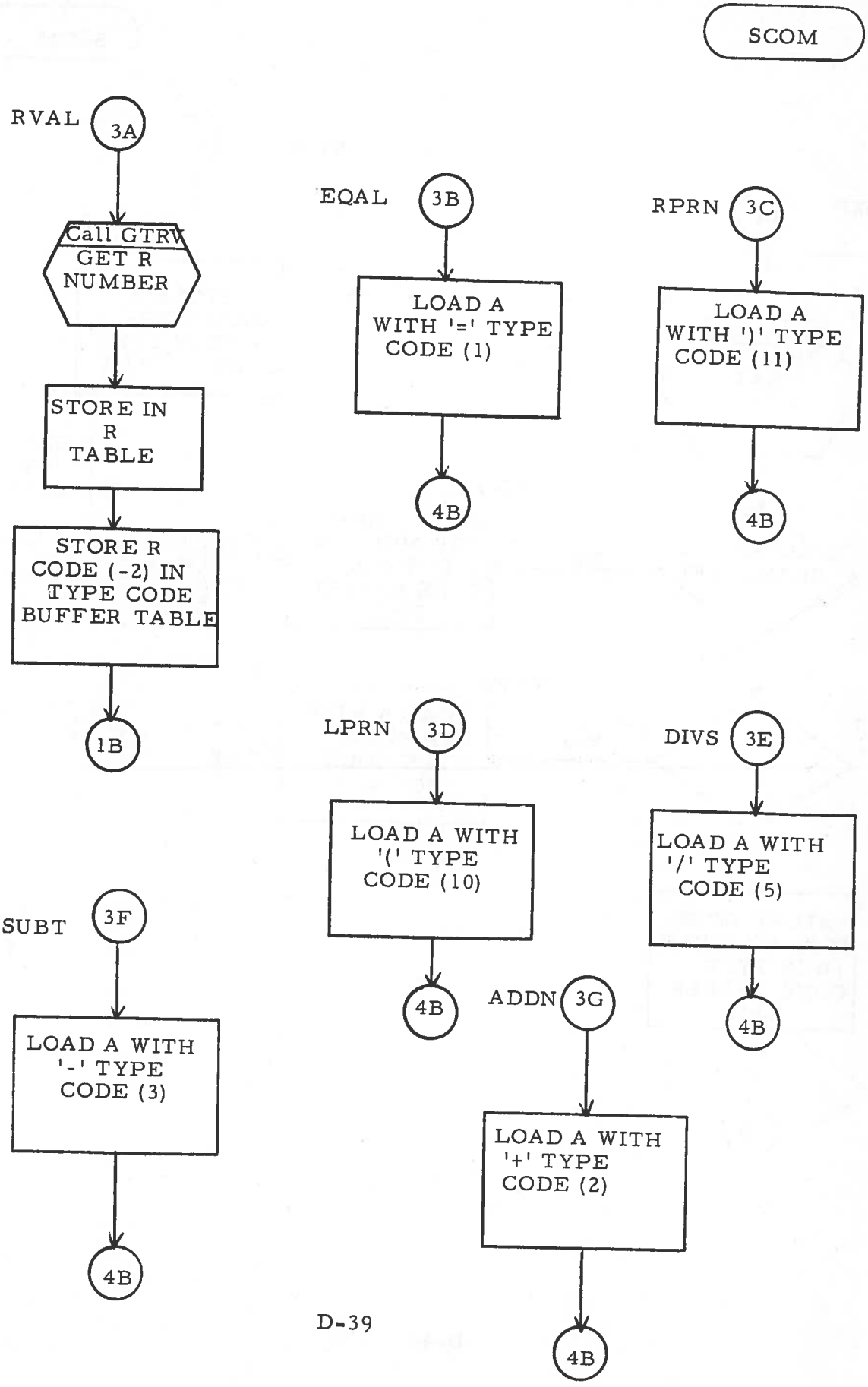
IMOD

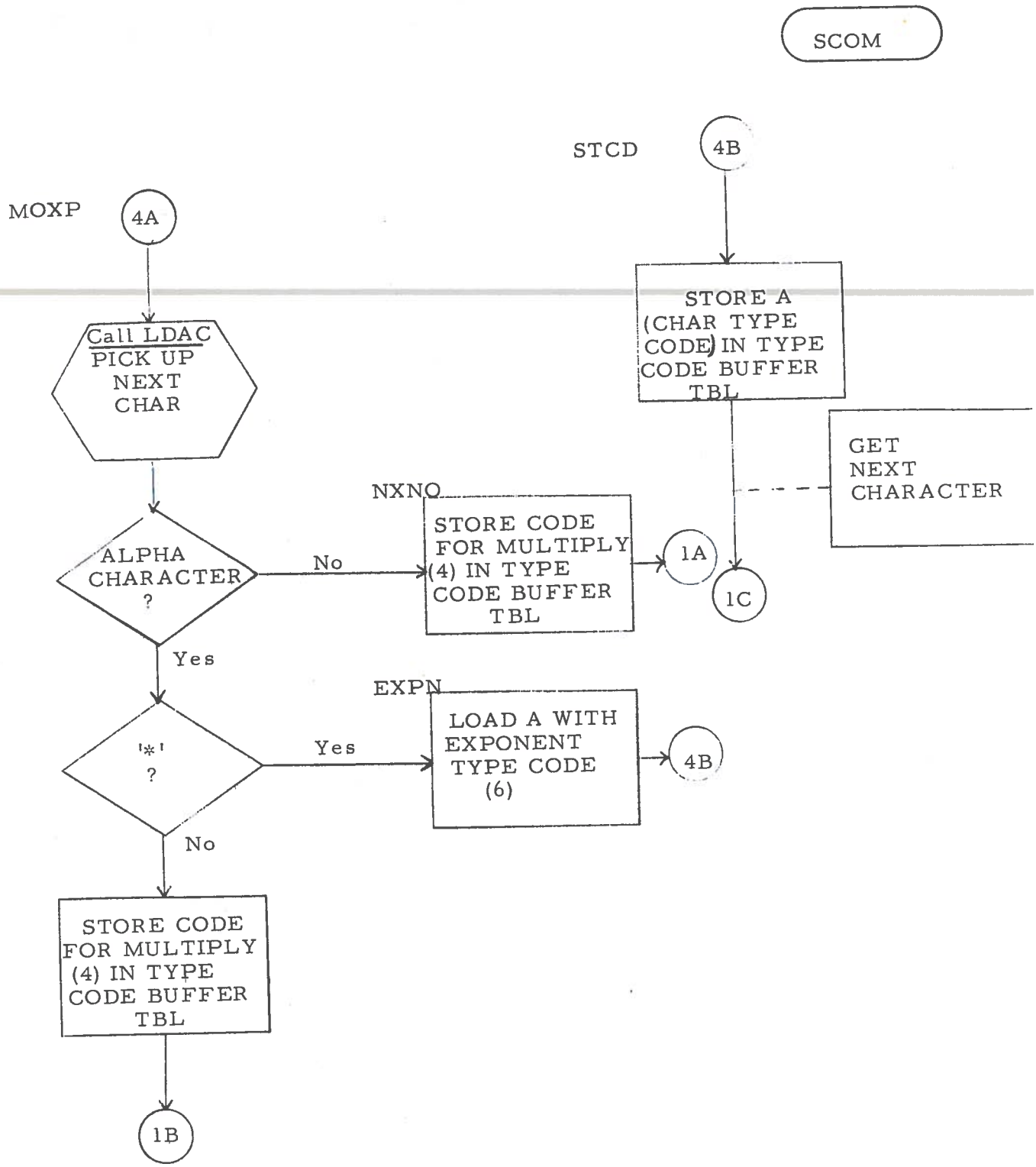
7. CORE USED:

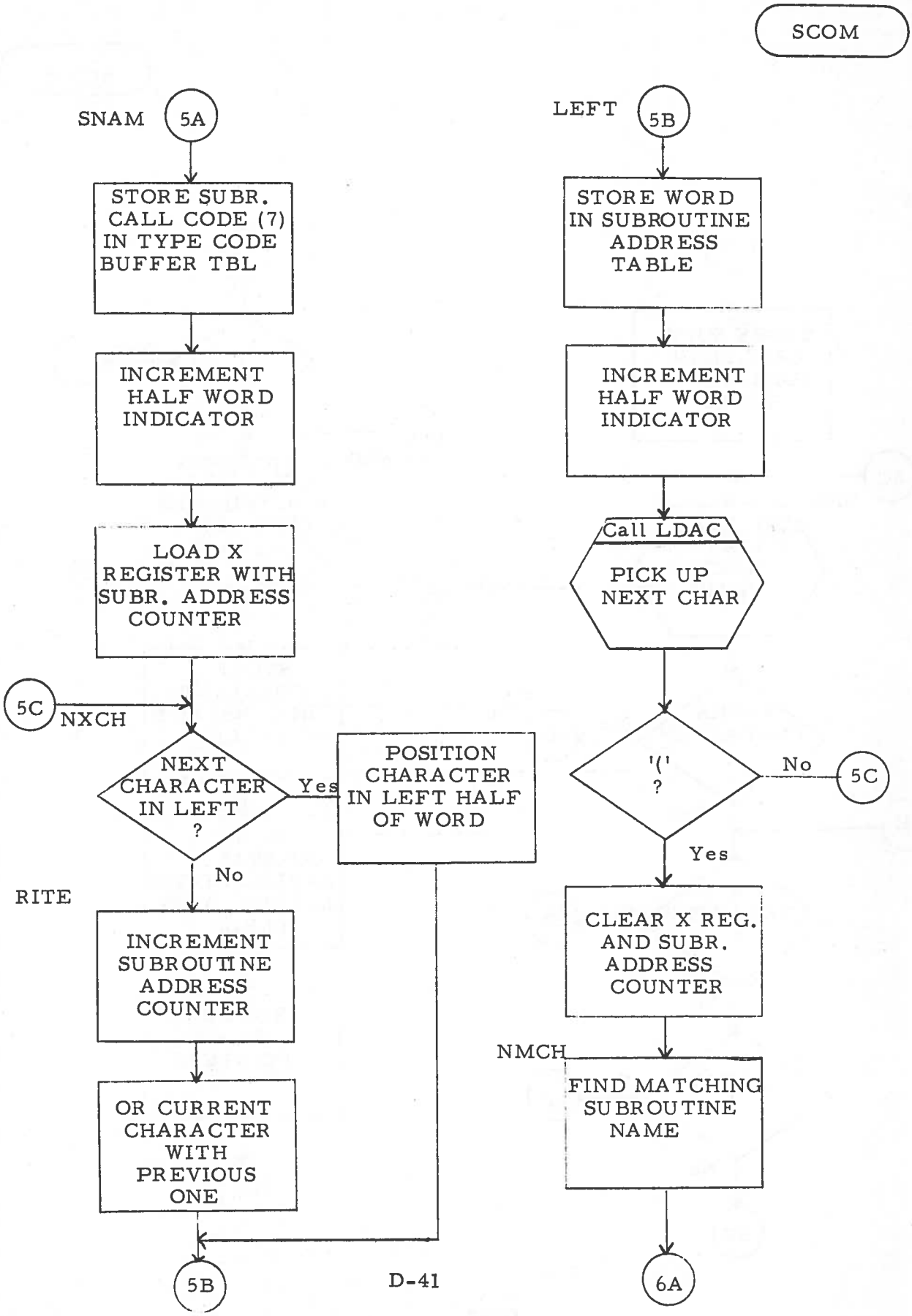
2032_8

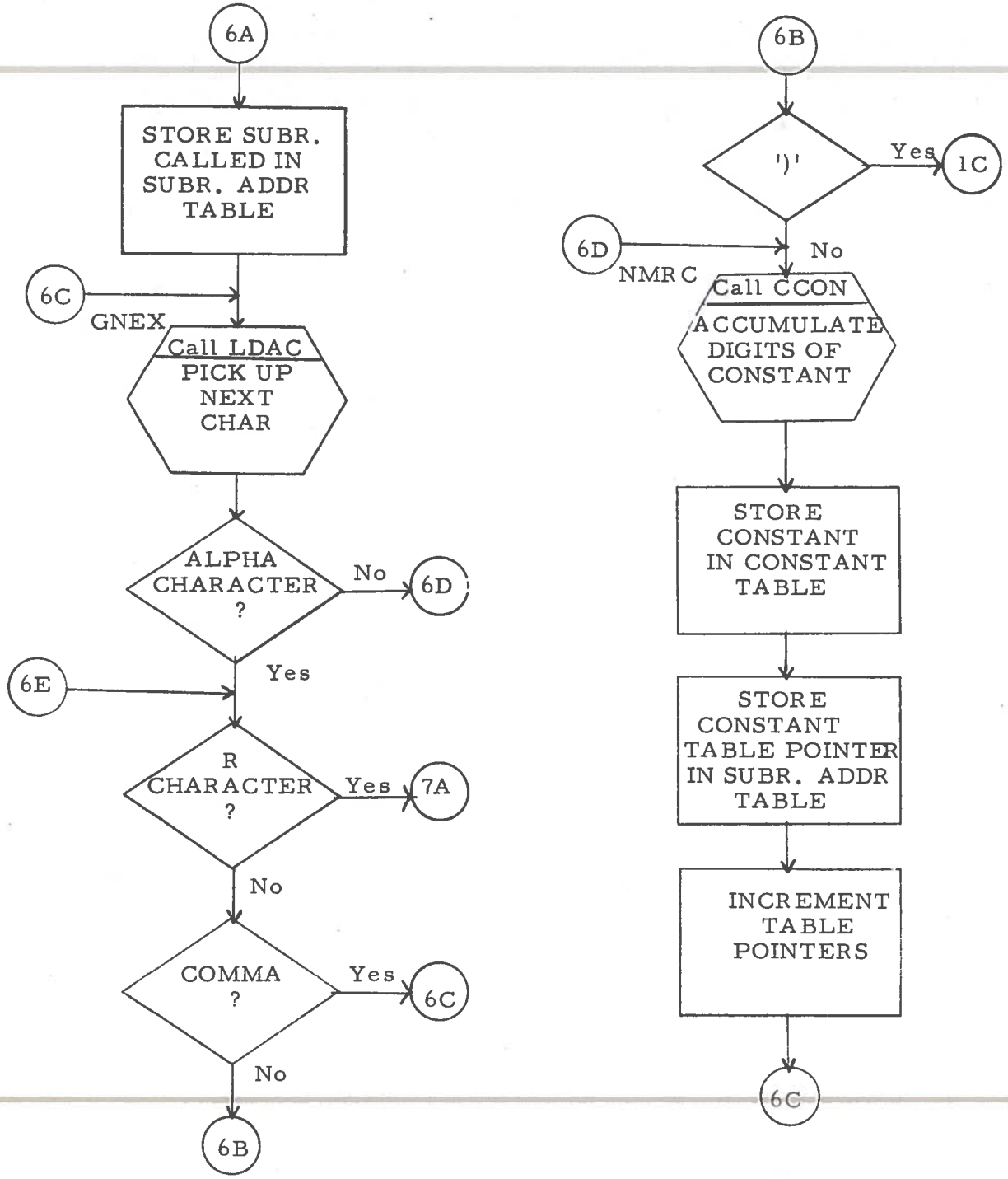

SCOM
SET POINTER TO FIRST WORD OF EXPRESSION
INITIALIZE AND CLEAR FLAGS, COUNTERS, AND ADDRESS TABLES
INITIALIZE TABLE POINTERS
Call ENCD
CHECK FOR END CODE AND GET CONTIN-UATION BLOCK
1st CHAR NULL OR 1st WD MINUS ?
Yes
INCREMENT HALF WORD INDICATOR
No
1C
RT
Call LDAC
PICK UP NEXT CHARACTER
NUMERIC CHARACTER ?
Yes
1A
No
1B
SCOM
NUMR
1A
Call CCON
ACCUMULATE DIGITS OF CONSTANT
STORE CONSTANT IN CONSTANT TABLE
STORE CODE (-1) IN TYPE CODE BUFFER TABLE
1B
NNUM
GET ALPHA CHARACTER
R CHARACTER ?
Yes
3A
No
2A


SCOM
2A
'='
?
Yes
3B
No
')'
?
Yes
3C
No
'('
?
Yes
3D
No
'/'
?
Yes
3E
No
'*'
?
Yes
4A
No
'-'
?
Yes
3F
No
2B
2B
'+'
?
Yes
3G
No
SPACE
?
Yes
1C
No
NULL
?
Yes
11A
No
SUBROUTINE
CALL
?
Yes
5A
No
1C

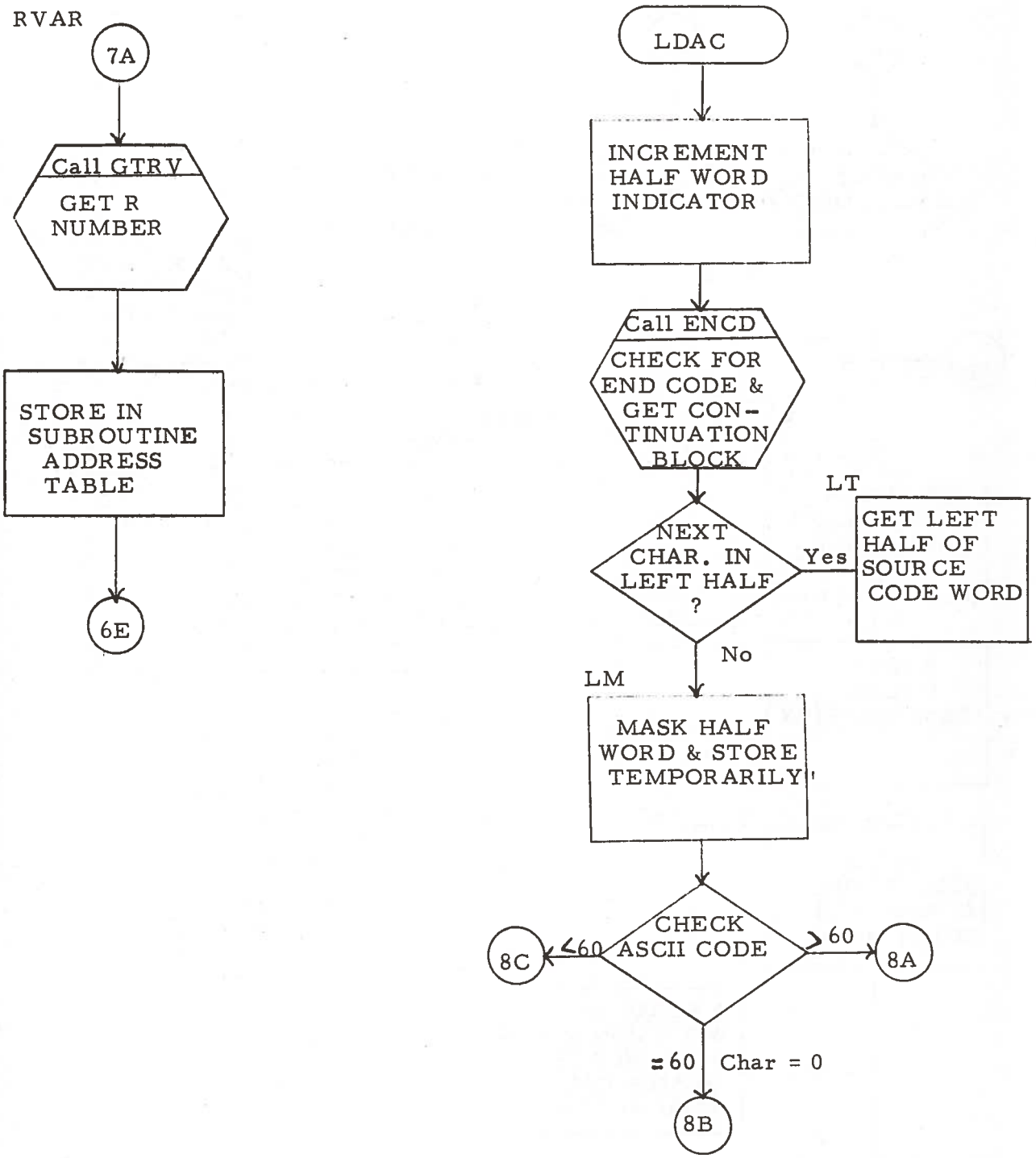

SCOM
RVAL
3A
Call GTRV
GET R
NUMBER
STORE IN
R
TABLE
STORE R
CODE (-2) IN
TYPE CODE
BUFFER TABLE
1B
EQAL
3B
LOAD A
WITH '=' TYPE
CODE (1)
4B
RPRN
3C
LOAD A
WITH ')' TYPE
CODE (11)
4B
LPRN
3D
LOAD A WITH
'(' TYPE
CODE (10)
4B
DIVS
3E
LOAD A WITH
'/' TYPE
CODE (5)
4B
SUBT
3F
LOAD A WITH
'-' TYPE
CODE (3)
4B
ADDN
3G
LOAD A WITH
'+' TYPE
CODE (2)
4B

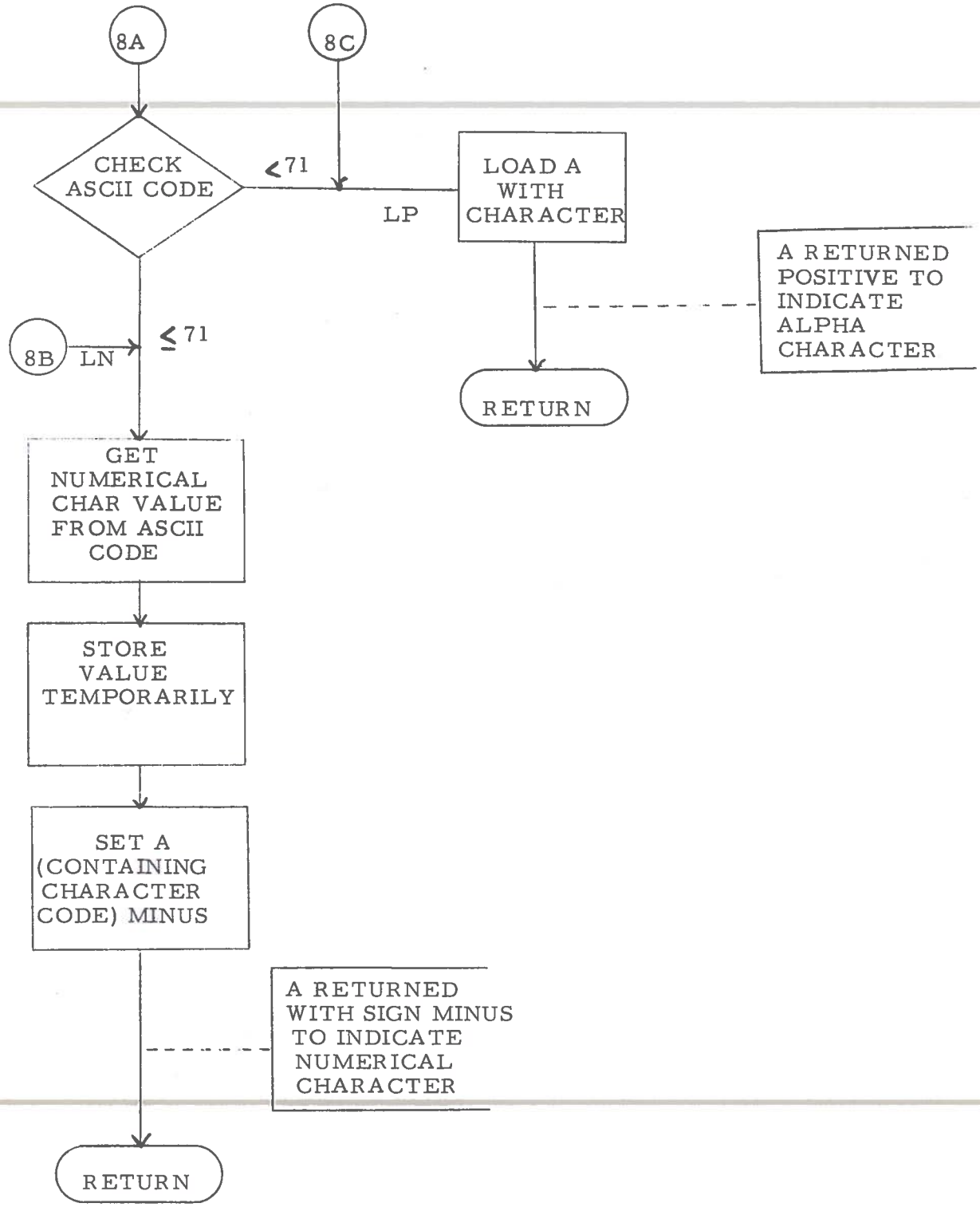

SCOM
STCD
4B
MOXP
4A
STORE A (CHAR TYPE CODE) IN TYPE CODE BUFFER TBL
Call LDAC
PICK UP NEXT CHAR
GET NEXT CHARACTER
NXNO
ALPHA CHARACTER ?
No
STORE CODE FOR MULTIPLY (4) IN TYPE CODE BUFFER TBL
1A
1C
Yes
EXPN
'*' ?
Yes
LOAD A WITH EXPONENT TYPE CODE (6)
4B
No
STORE CODE FOR MULTIPLY (4) IN TYPE CODE BUFFER TBL
1B


SCOM
SNAM
5A
STORE SUBR. CALL CODE (7) IN TYPE CODE BUFFER TBL
INCREMENT HALF WORD INDICATOR
LOAD X REGISTER WITH SUBR. ADDRESS COUNTER
5C
NXCH
NEXT CHARACTER IN LEFT ?
Yes
POSITION CHARACTER IN LEFT HALF OF WORD
No
RITE
INCREMENT SUBROUTINE ADDRESS COUNTER
OR CURRENT CHARACTER WITH PREVIOUS ONE
5B
LEFT
5B
STORE WORD IN SUBROUTINE ADDRESS TABLE
INCREMENT HALF WORD INDICATOR
Call LDAC
PICK UP NEXT CHAR
'(' ?
No
5C
Yes
CLEAR X REG. AND SUBR. ADDRESS COUNTER
NMCH
FIND MATCHING SUBROUTINE NAME
6A


6A
STORE SUBR. CALLED IN SUBR. ADDR TABLE
6C
GNEX
Call LDAC
PICK UP NEXT CHAR
ALPHA CHARACTER ?
No
6D
Yes
6E
R CHARACTER ?
Yes
7A
No
COMMA ?
Yes
6C
No
6B
6B
')'
Yes
1C
6D
NMRC
No
Call CCON
ACCUMULATE DIGITS OF CONSTANT
STORE CONSTANT IN CONSTANT TABLE
STORE CONSTANT TABLE POINTER IN SUBR. ADDR TABLE
INCREMENT TABLE POINTERS
6C

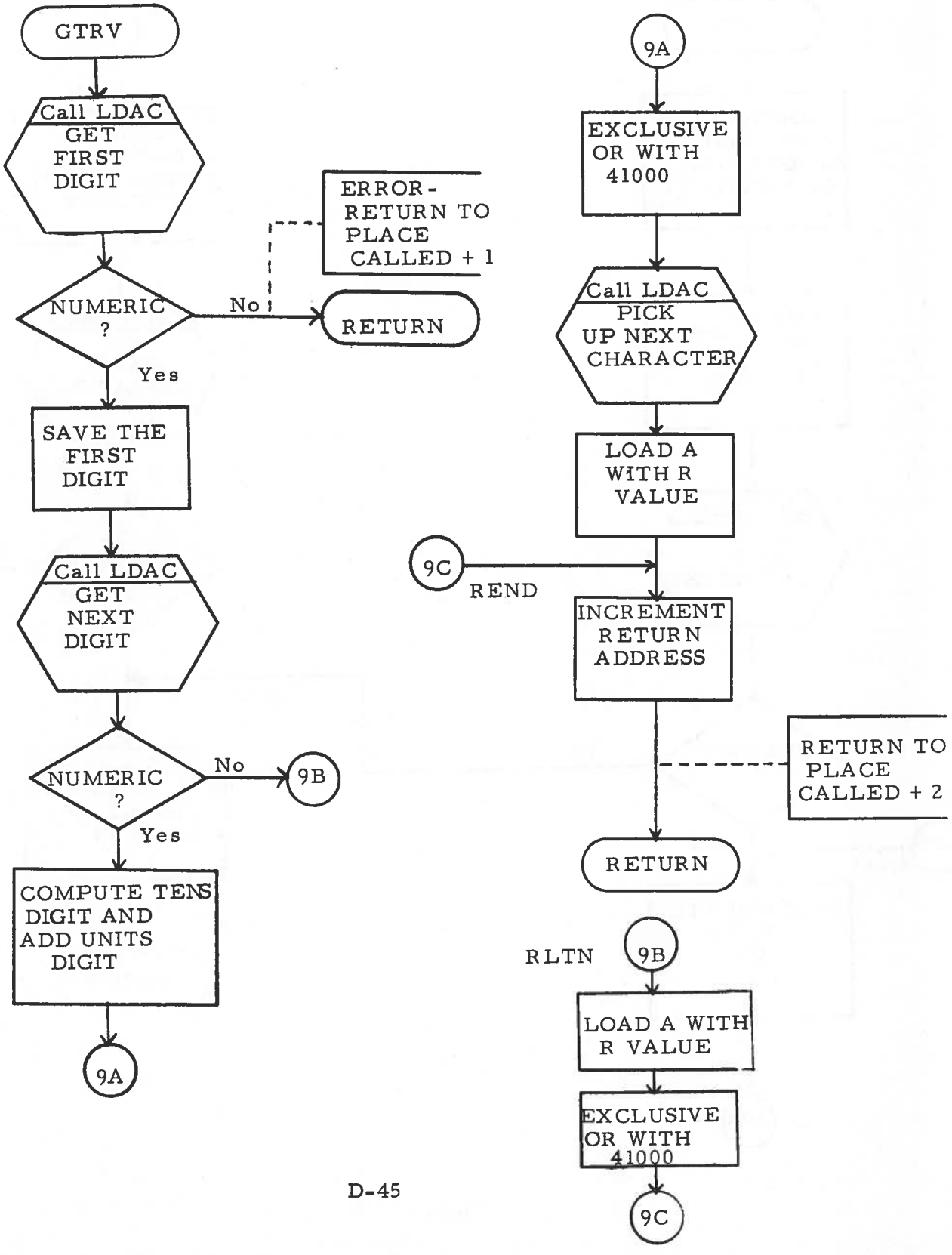
LDAC: Get Next Character




8A
8C
CHECK ASCII CODE
<71
LP
LOAD A WITH CHARACTER
A RETURNED POSITIVE TO INDICATE ALPHA CHARACTER
RETURN
8B
LN
≤71
GET NUMERICAL CHAR VALUE FROM ASCII CODE
STORE VALUE TEMPORARILY
SET A (CONTAINING CHARACTER CODE) MINUS
A RETURNED WITH SIGN MINUS TO INDICATE NUMERICAL CHARACTER
RETURN

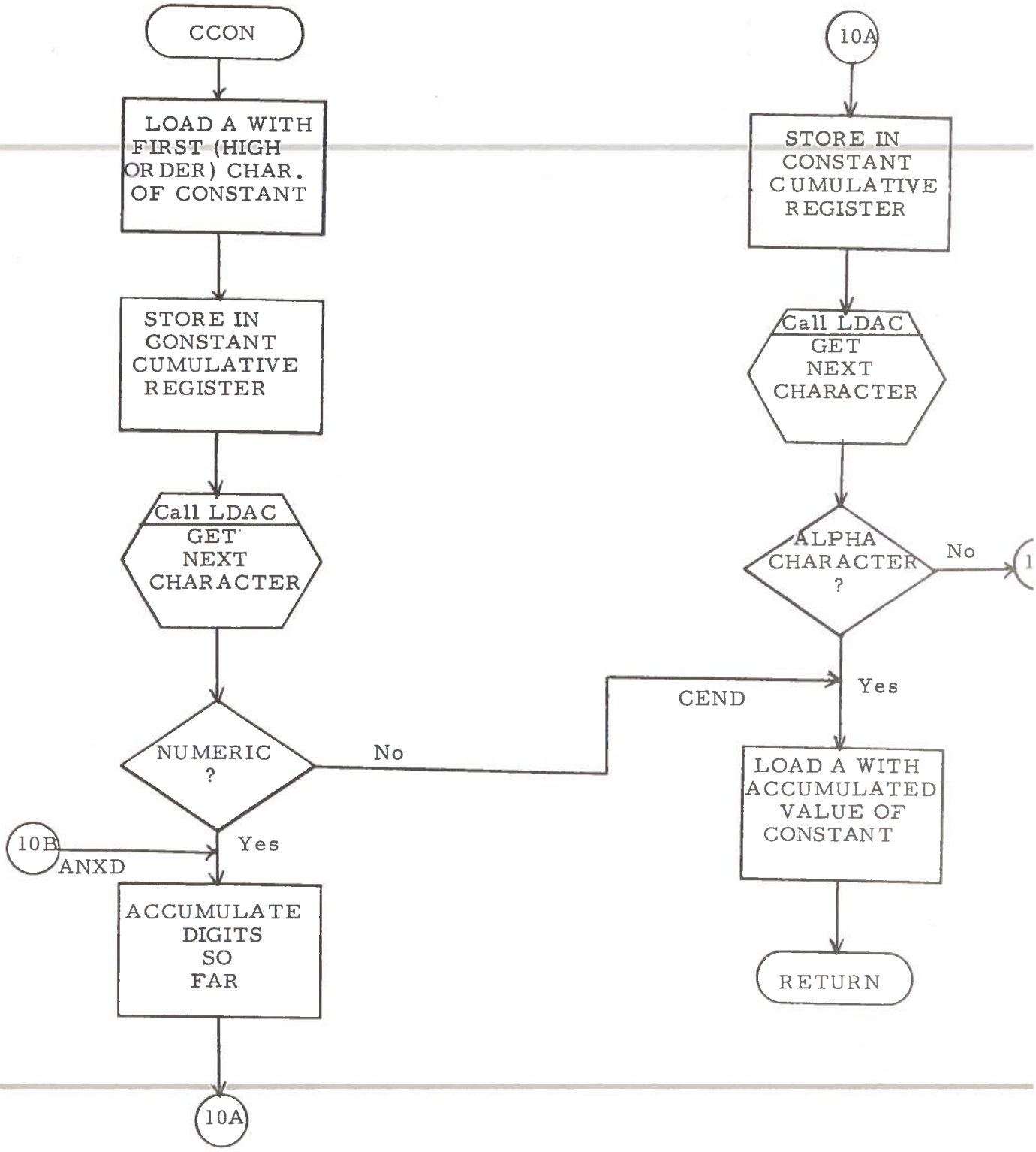
GTRV: Get R Number

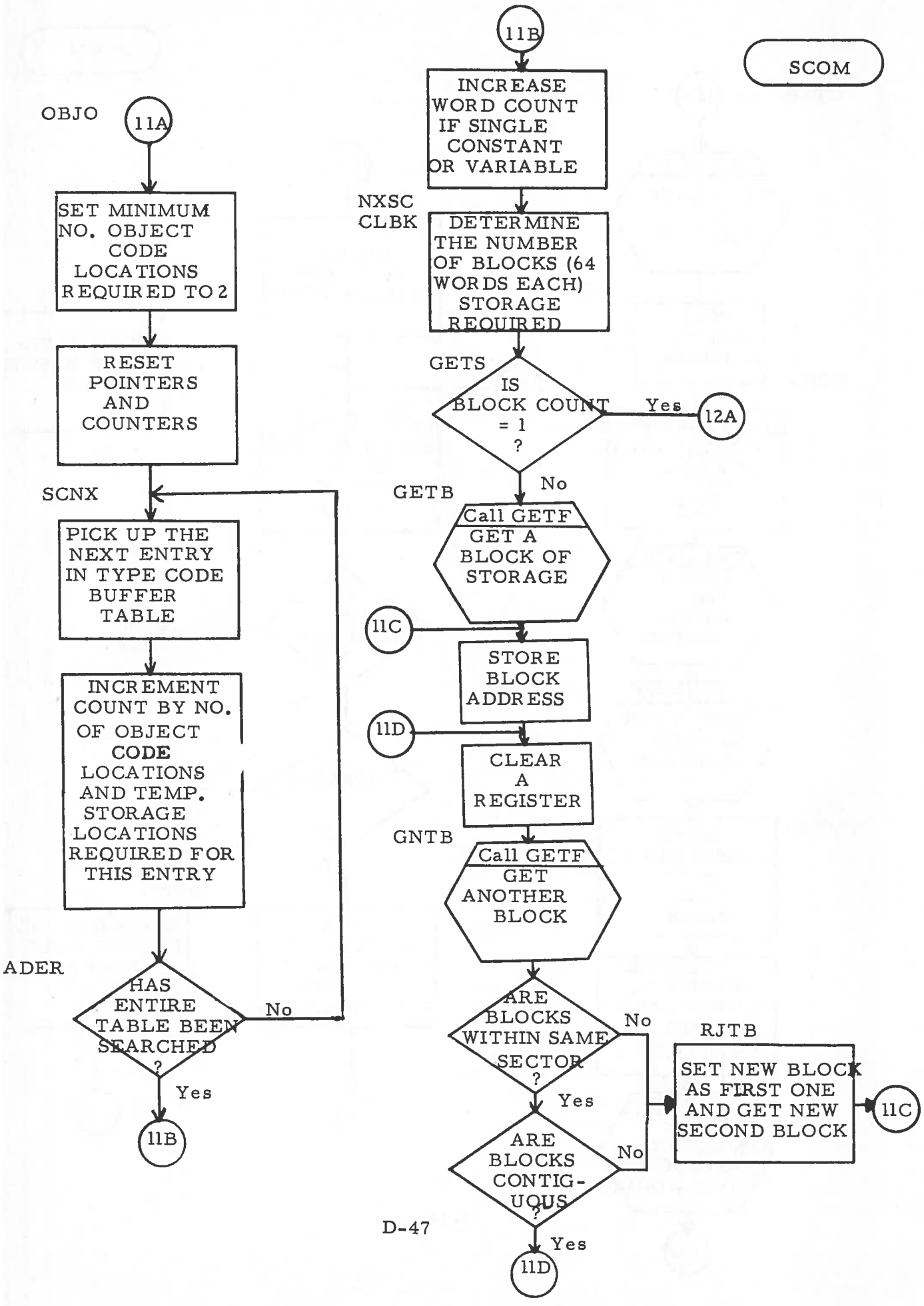
SCOM

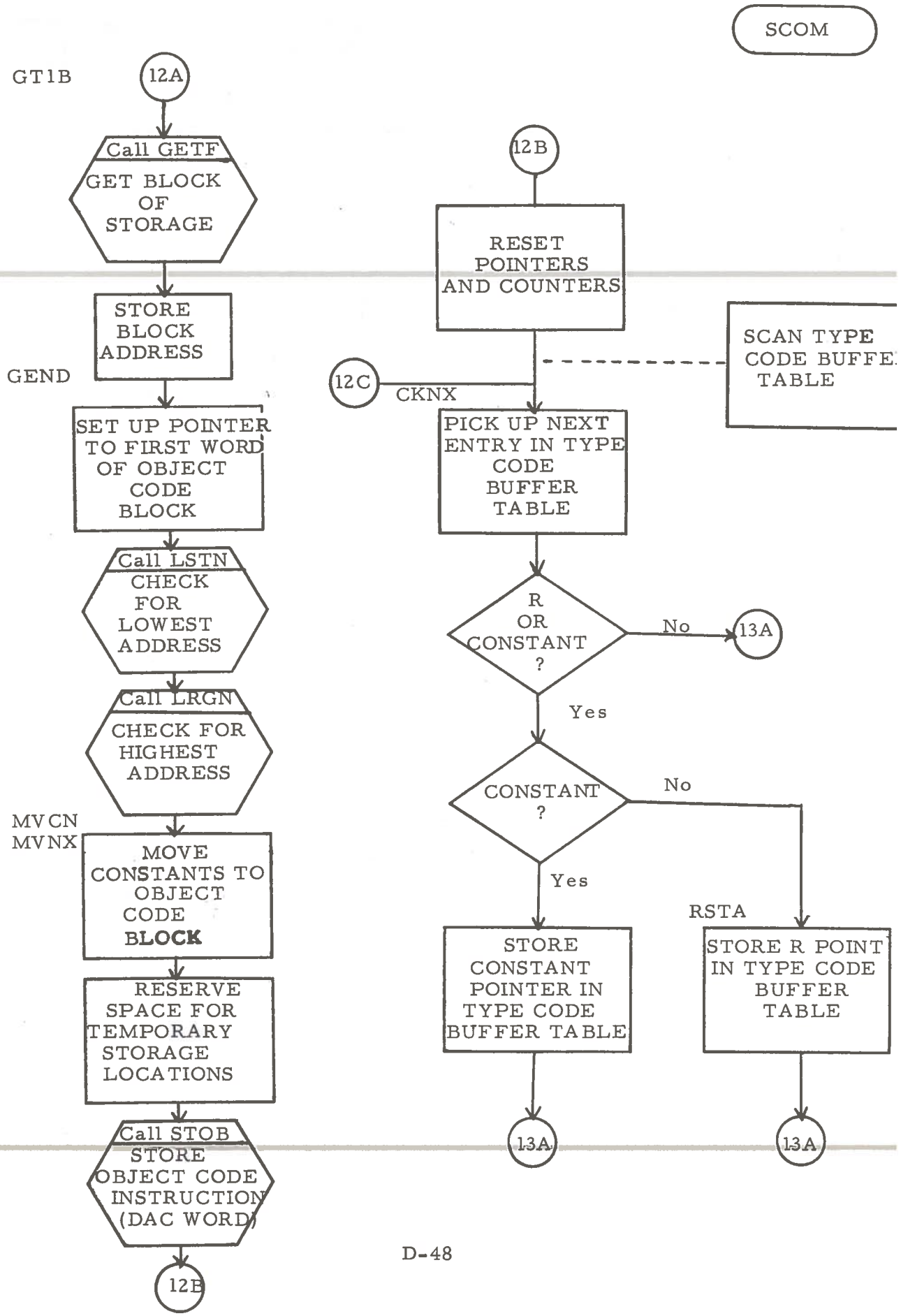


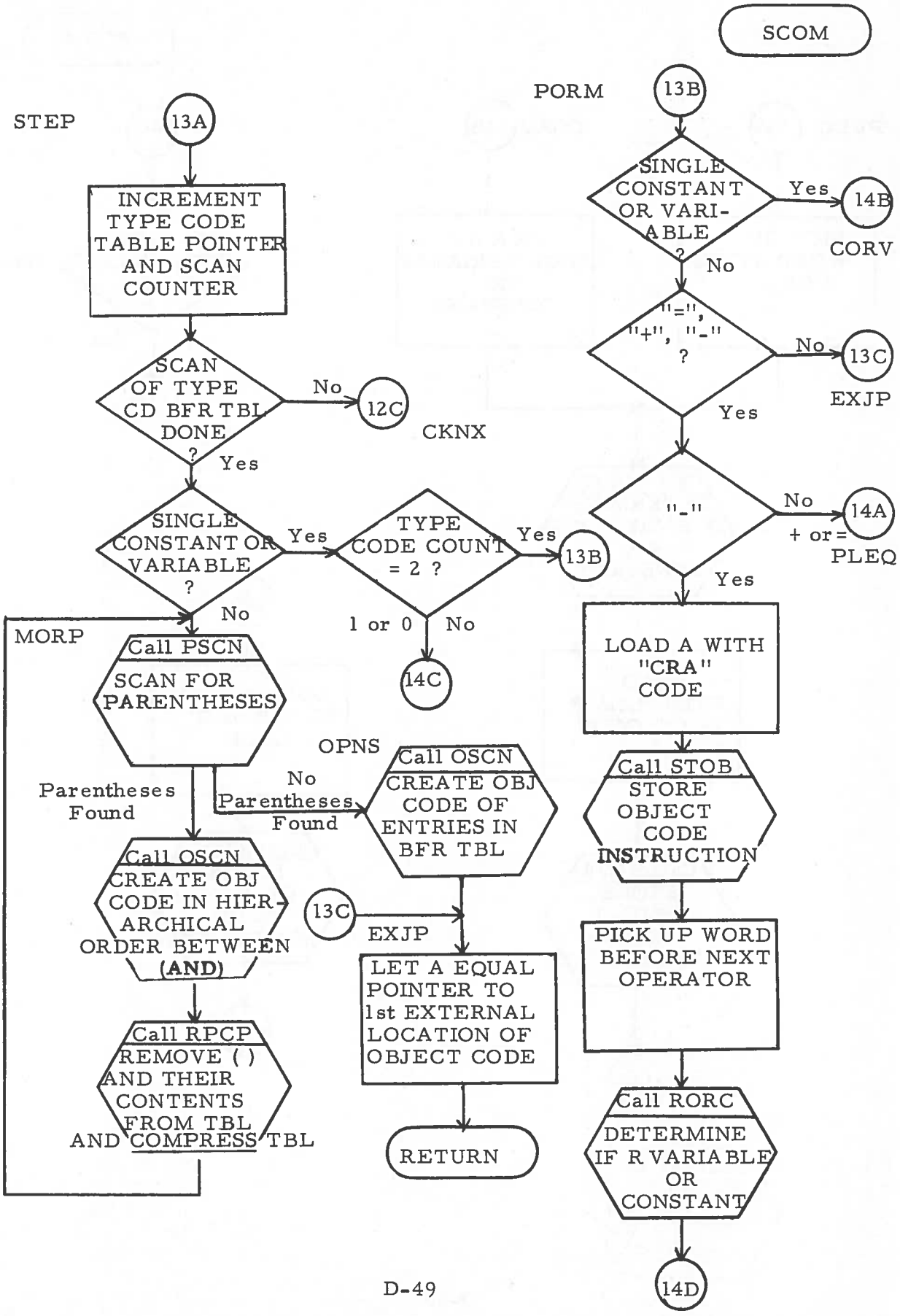
SCOM

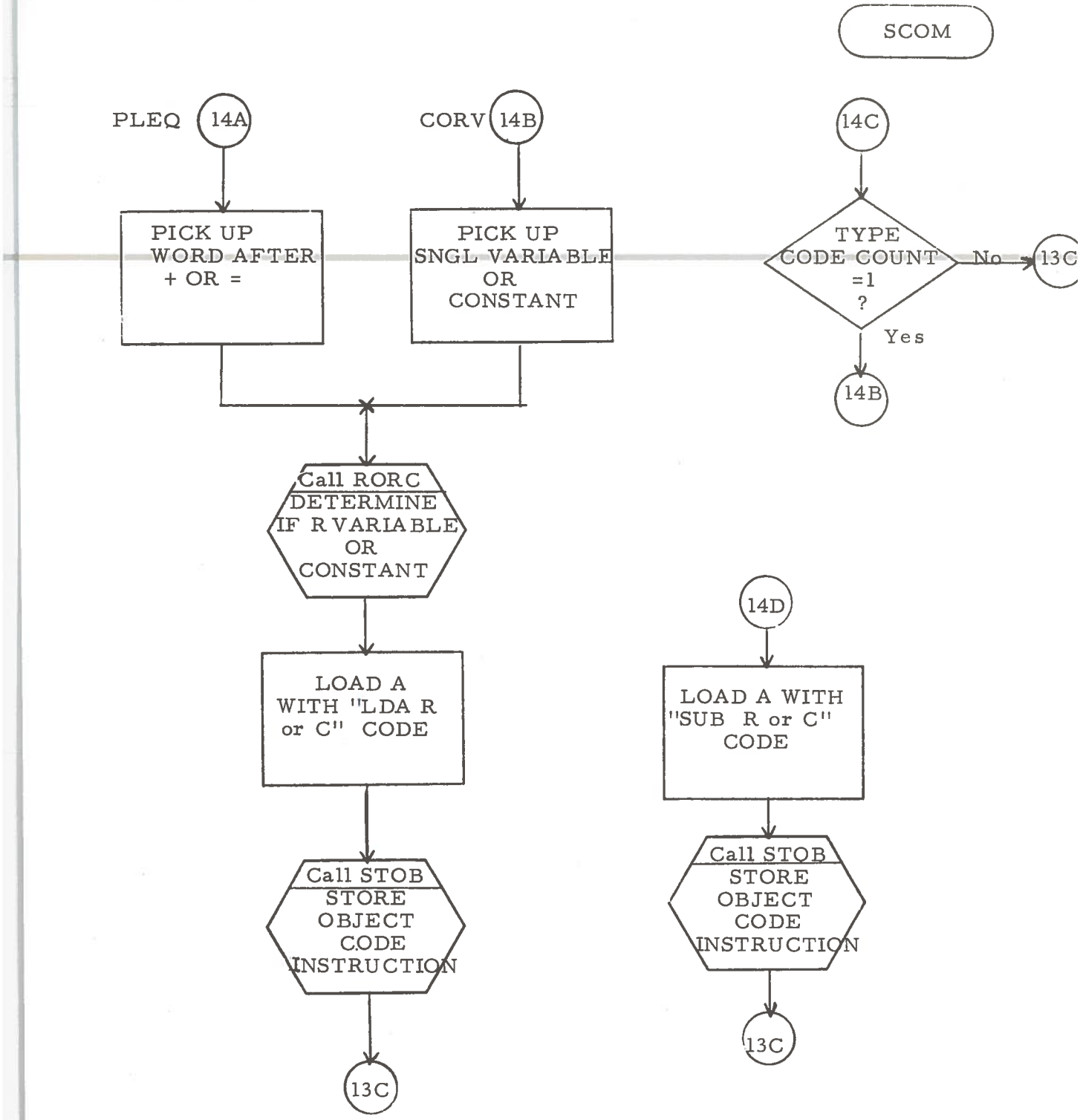
CCON: Accumulate Digits of Constant



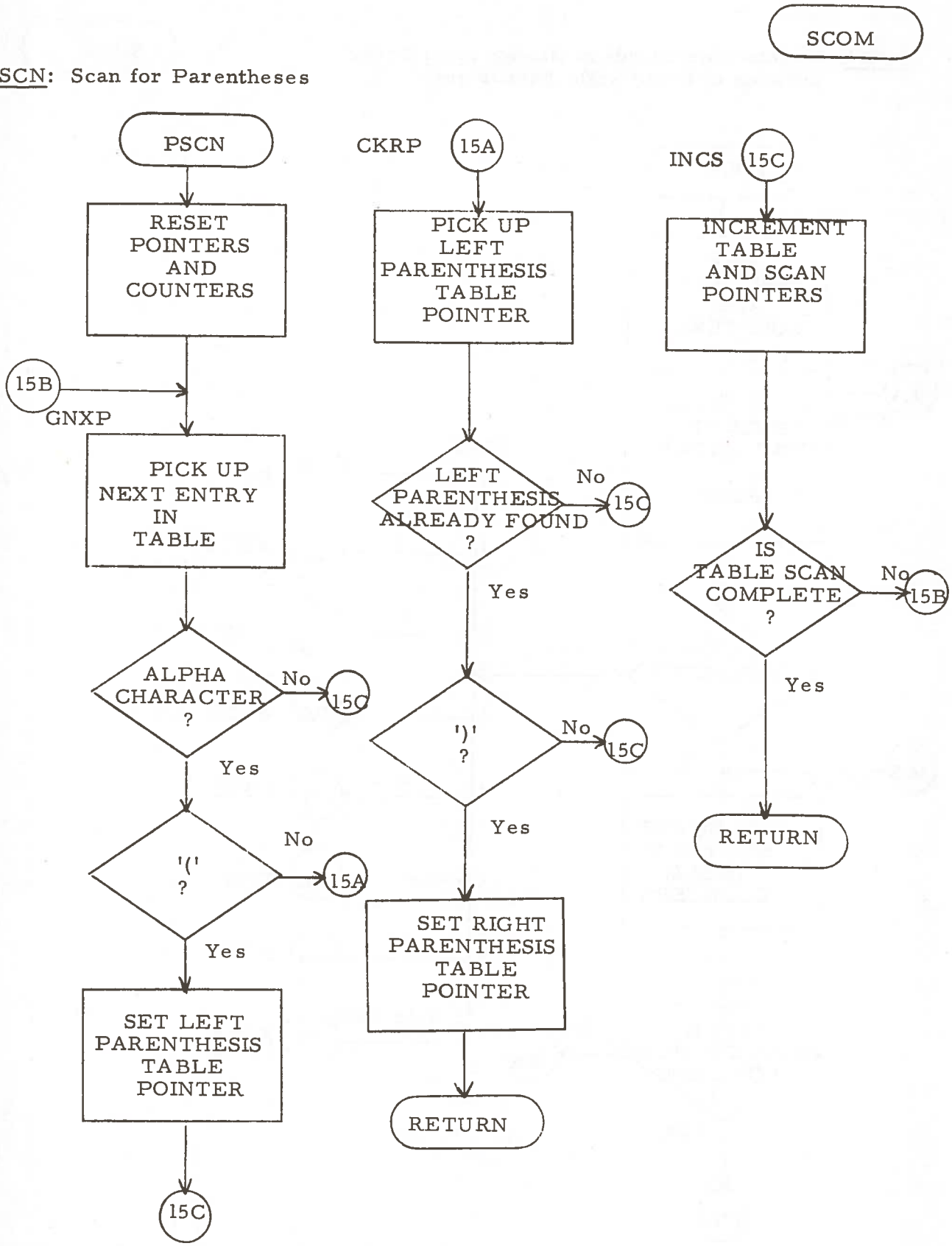

SCOM
OBJO
11A
SET MINIMUM NO. OBJECT CODE LOCATIONS REQUIRED TO 2
RESET POINTERS AND COUNTERS
SCNX
PICK UP THE NEXT ENTRY IN TYPE CODE BUFFER TABLE
INCREMENT COUNT BY NO. OF OBJECT CODE LOCATIONS AND TEMP. STORAGE LOCATIONS REQUIRED FOR THIS ENTRY
ADER
HAS ENTIRE TABLE BEEN SEARCHED ?
No
Yes
11B
11B
INCREASE WORD COUNT IF SINGLE CONSTANT OR VARIABLE
NXSC
CLBK
DETERMINE THE NUMBER OF BLOCKS (64 WORDS EACH) STORAGE REQUIRED
GETS
IS BLOCK COUNT = 1 ?
Yes
12A
No
GETB
Call GETF
GET A BLOCK OF STORAGE
11C
11C
STORE BLOCK ADDRESS
11D
11D
CLEAR A REGISTER
GNTB
Call GETF
GET ANOTHER BLOCK
ARE BLOCKS WITHIN SAME SECTOR ?
No
RJTB
SET NEW BLOCK AS FIRST ONE AND GET NEW SECOND BLOCK
11C
Yes
ARE BLOCKS CONTIG-UOUS ?
No
Yes
11D


GT1B
12A
Call GETF
GET BLOCK OF STORAGE
STORE BLOCK ADDRESS
GEND
SET UP POINTER TO FIRST WORD OF OBJECT CODE BLOCK
Call LSTN
CHECK FOR LOWEST ADDRESS
Call LRGN
CHECK FOR HIGHEST ADDRESS
MVCN
MVNX
MOVE CONSTANTS TO OBJECT CODE BLOCK
RESERVE SPACE FOR TEMPORARY STORAGE LOCATIONS
Call STOB
STORE OBJECT CODE INSTRUCTION (DAC WORD)
12B
12B
RESET POINTERS AND COUNTERS
SCAN TYPE CODE BUFFE TABLE
12C
CKNX
PICK UP NEXT ENTRY IN TYPE CODE BUFFER TABLE
R OR CONSTANT ?
No
13A
Yes
CONSTANT ?
No
Yes
RSTA
STORE CONSTANT POINTER IN TYPE CODE BUFFER TABLE
STORE R POINT IN TYPE CODE BUFFER TABLE
13A
13A

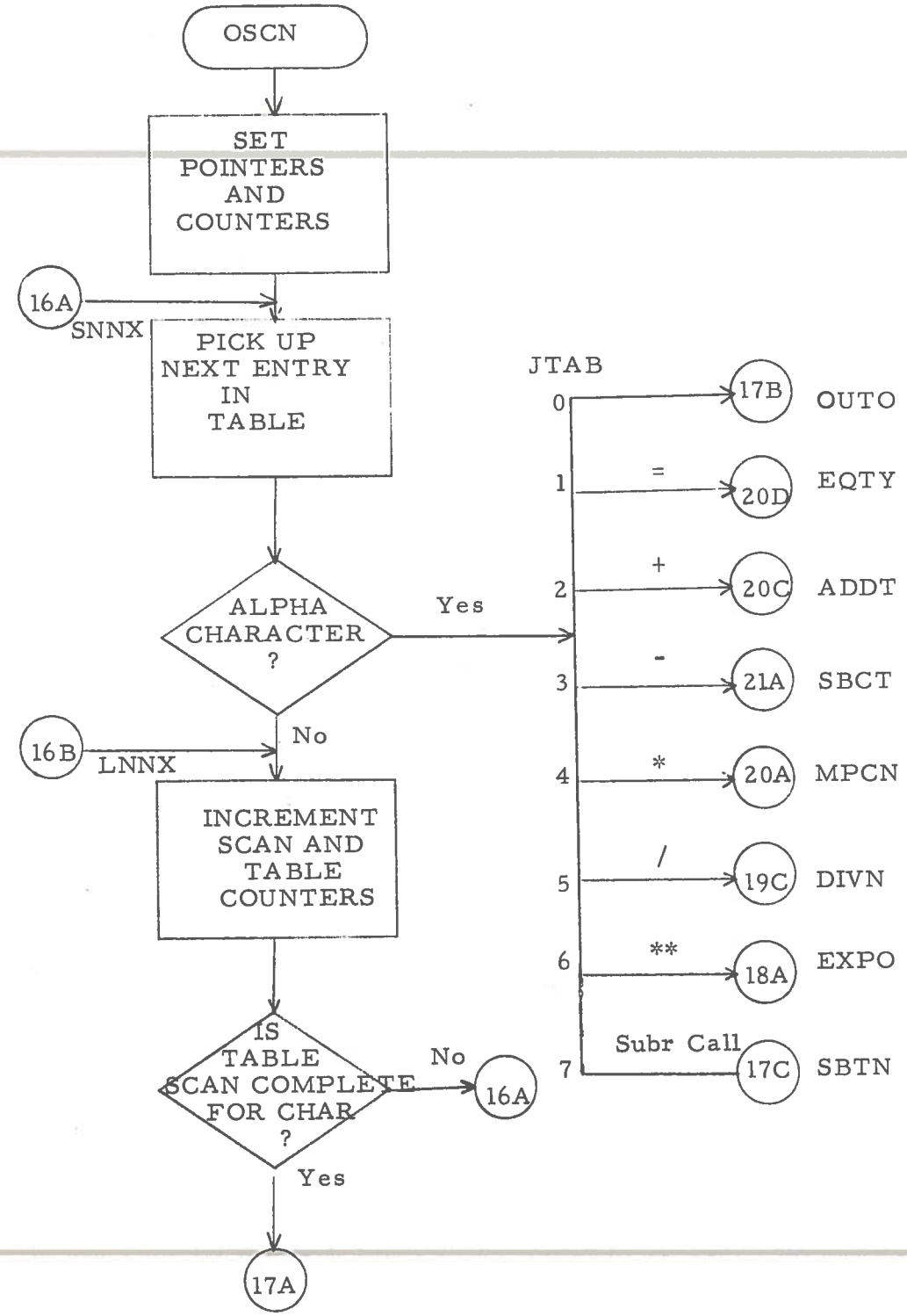

SCOM
PORM
13B
SINGLE CONSTANT OR VARIABLE ?
Yes
14B
CORV
No
"=", "+", "-" ?
No
13C
EXJP
Yes
"-"
No
14A
+ or =
PLEQ
Yes
LOAD A WITH "CRA" CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
PICK UP WORD BEFORE NEXT OPERATOR
Call RORC
DETERMINE IF R VARIABLE OR CONSTANT
14D
STEP
13A
INCREMENT TYPE CODE TABLE POINTER AND SCAN COUNTER
SCAN OF TYPE CD BFR TBL DONE ?
No
12C
CKNX
Yes
SINGLE CONSTANT OR VARIABLE ?
Yes
TYPE CODE COUNT = 2 ?
Yes
13B
1 or 0
No
14C
No
MORP
Call PSCN
SCAN FOR PARENTHESES
Parentheses Found
OPNS
No Parentheses Found
Call OSCN
CREATE OBJ CODE OF ENTRIES IN BFR TBL
Call OSCN
CREATE OBJ CODE IN HIERARCHICAL ORDER BETWEEN (AND)
13C
EXJP
LET A EQUAL POINTER TO 1st EXTERNAL LOCATION OF OBJECT CODE
Call RPCP
REMOVE ( ) AND THEIR CONTENTS FROM TBL AND COMPRESS TBL
RETURN

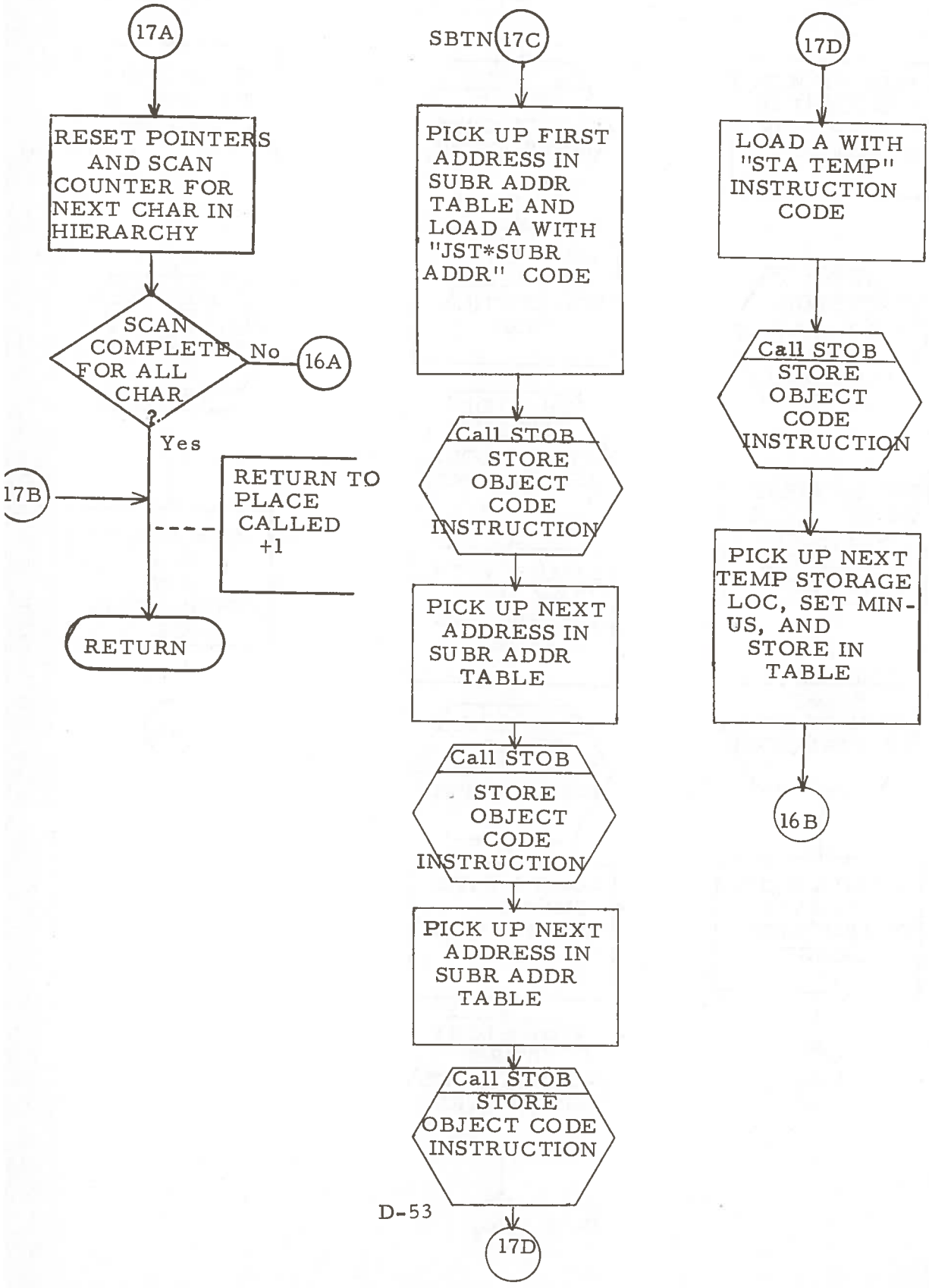

SCOM
PLEQ
14A
CORV
14B
14C
PICK UP WORD AFTER + OR =
PICK UP SNGL VARIABLE OR CONSTANT
TYPE CODE COUNT =1 ?
No
13C
Yes
14B
Call RORC
DETERMINE IF R VARIABLE OR CONSTANT
LOAD A WITH "LDA R or C" CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
13C
14D
LOAD A WITH "SUB R or C" CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
13C

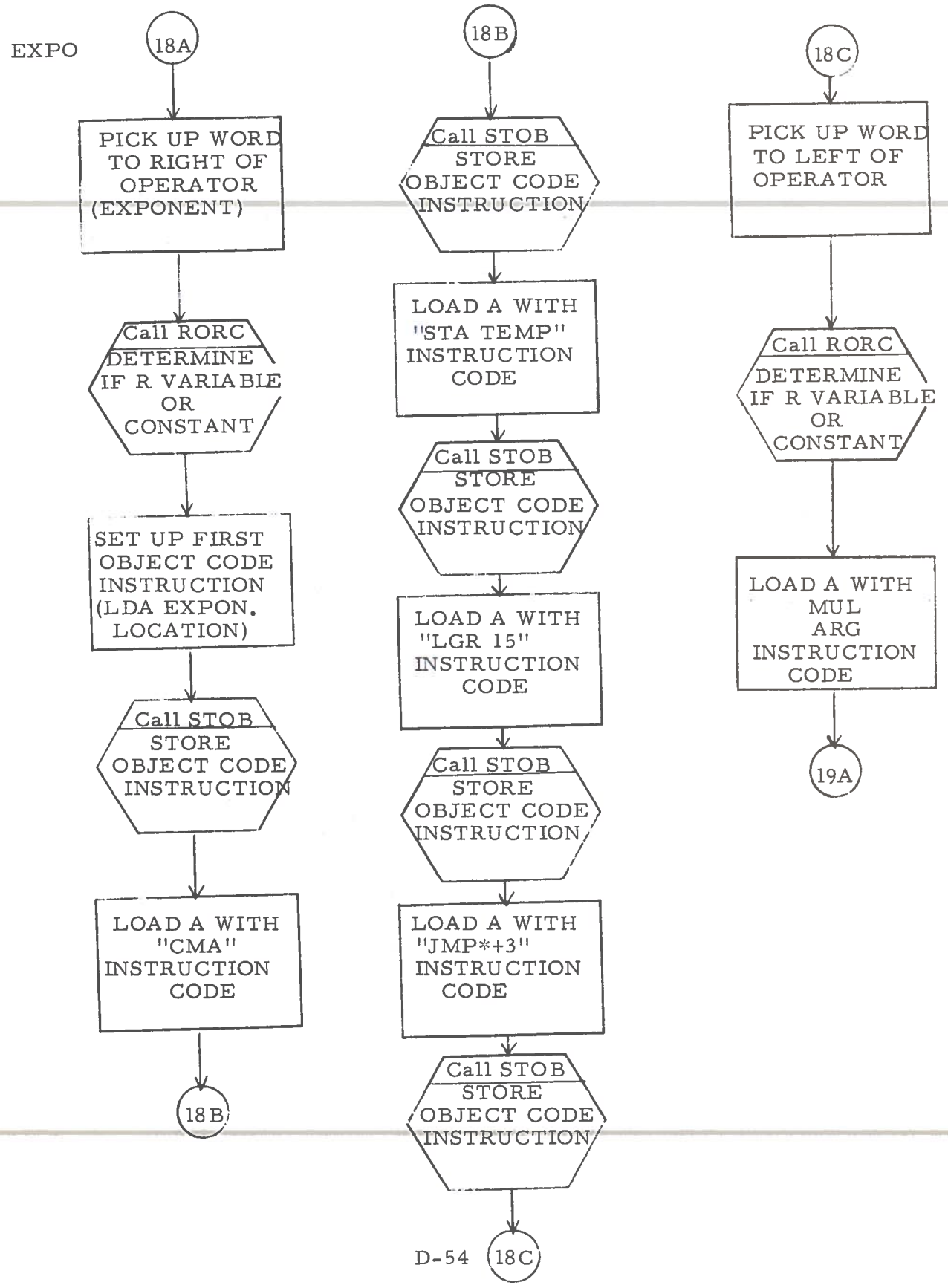
PSCN: Scan for Parentheses

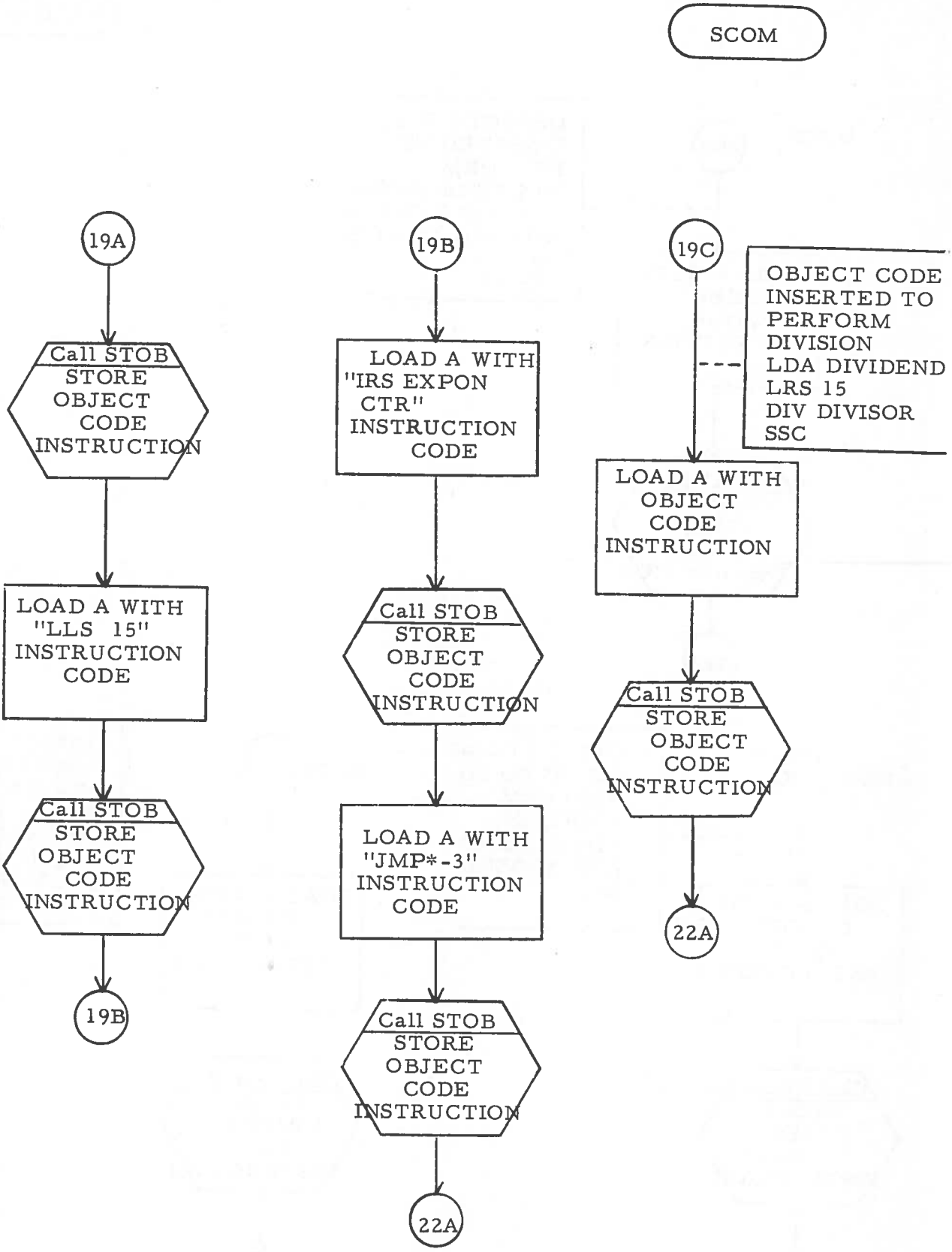


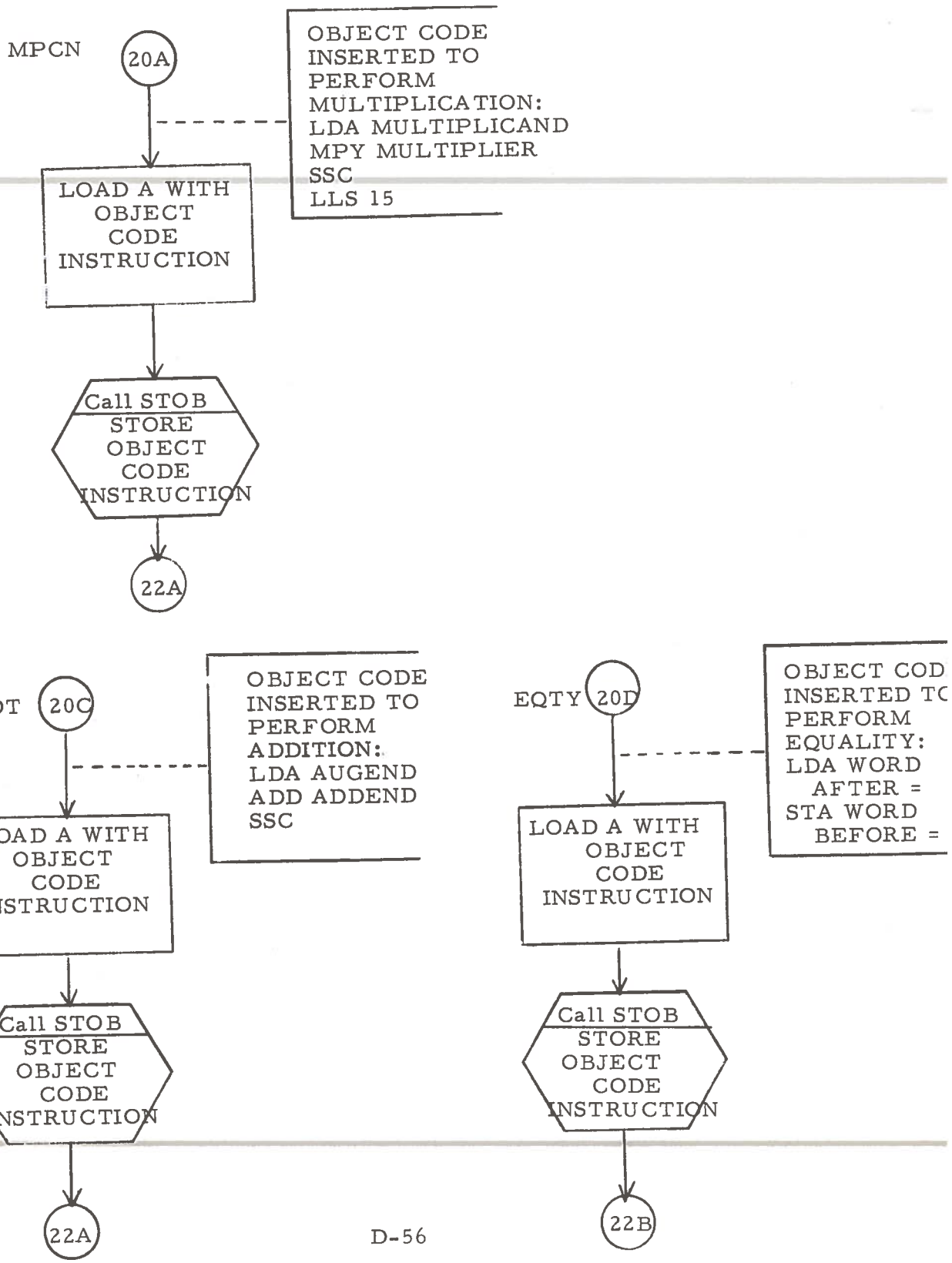
OSCN: Create Object Code in Hierarchical Order between Left and Right Parentheses

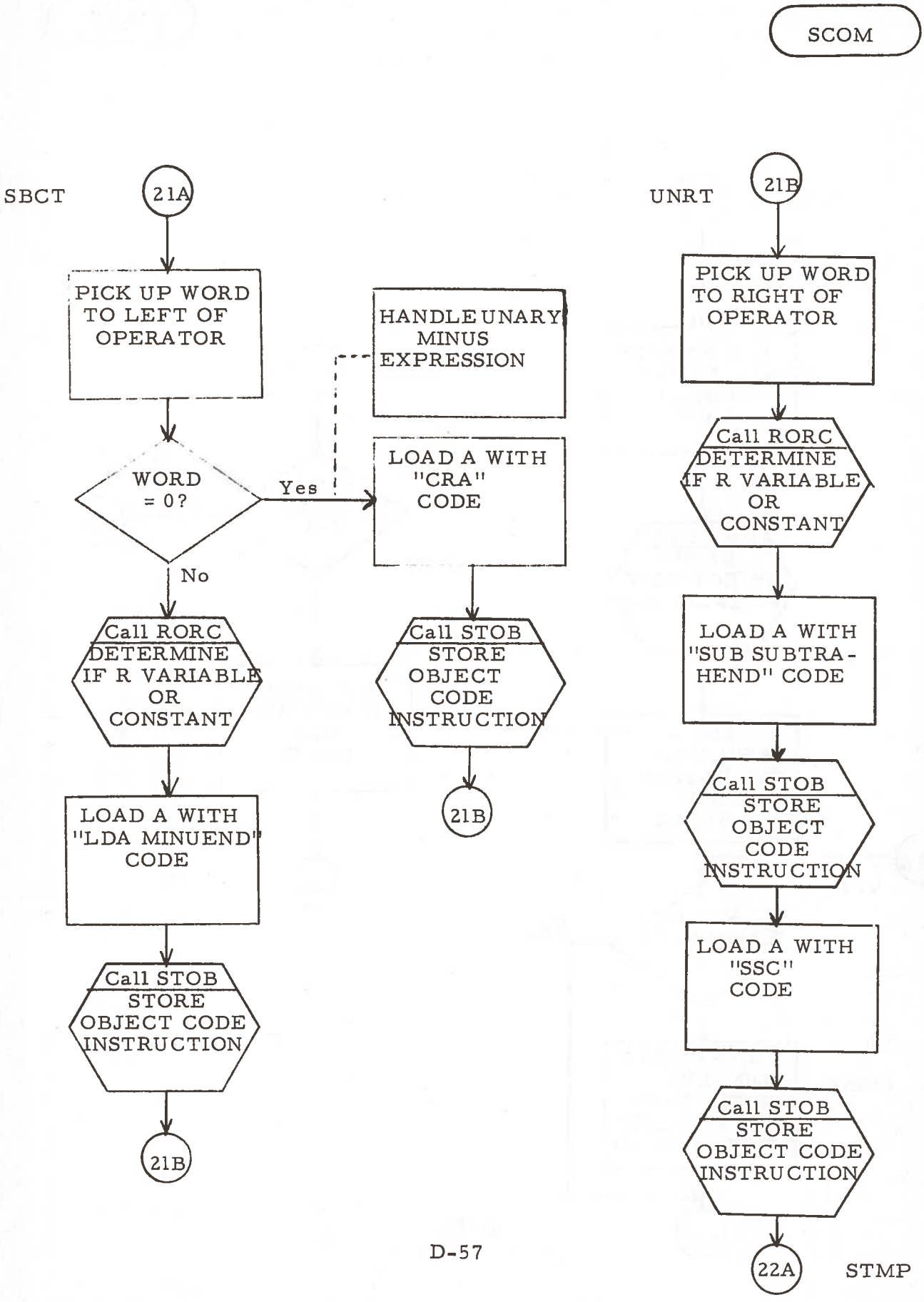


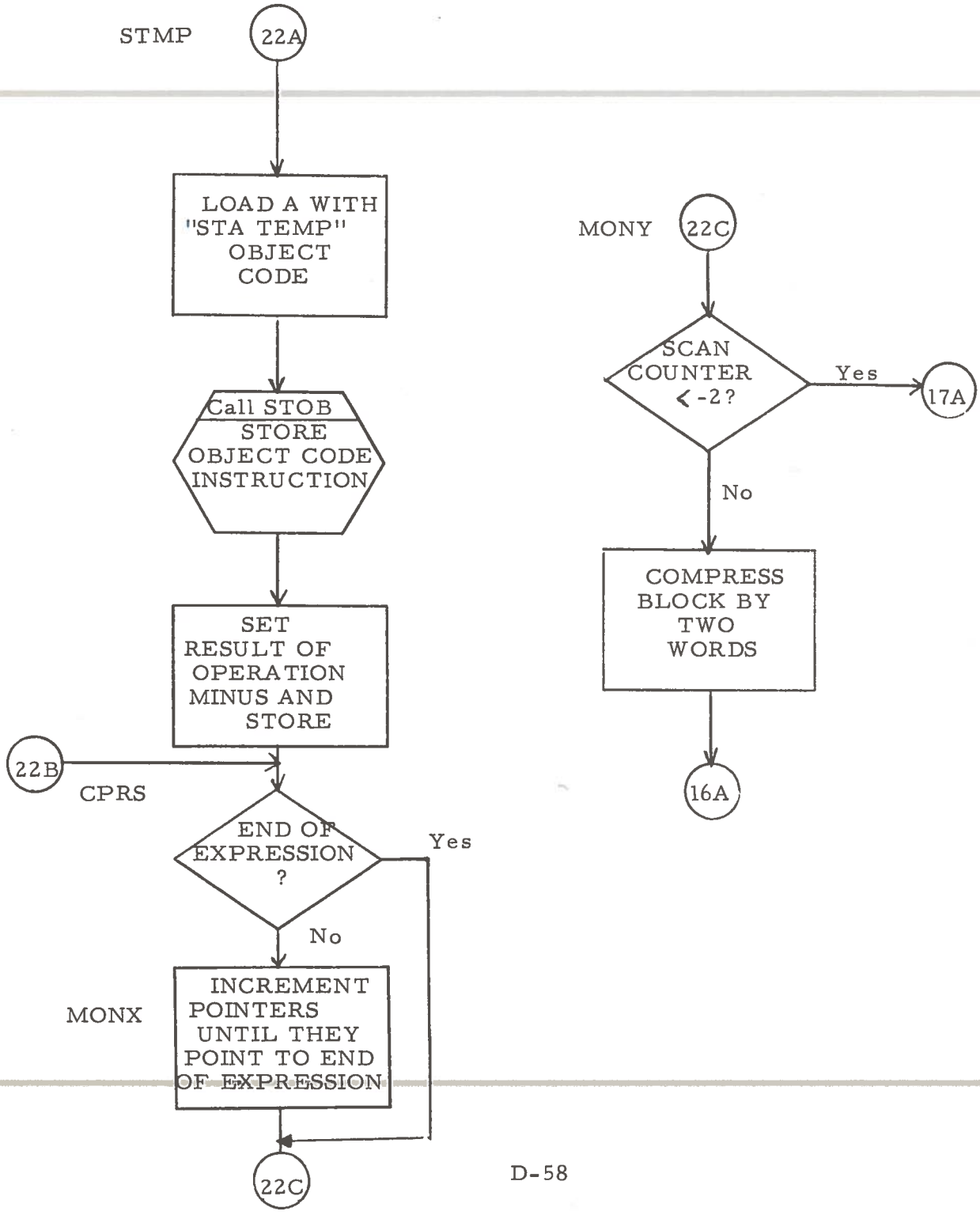

17A
RESET POINTERS AND SCAN COUNTER FOR NEXT CHAR IN HIERARCHY
SCAN COMPLETE FOR ALL CHAR ?
No
16A
Yes
17B
RETURN TO PLACE CALLED +1
RETURN
SBTN
17C
PICK UP FIRST ADDRESS IN SUBR ADDR TABLE AND LOAD A WITH "JST*SUBR ADDR" CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
PICK UP NEXT ADDRESS IN SUBR ADDR TABLE
Call STOB
STORE OBJECT CODE INSTRUCTION
PICK UP NEXT ADDRESS IN SUBR ADDR TABLE
Call STOB
STORE OBJECT CODE INSTRUCTION
17D
17D
LOAD A WITH "STA TEMP" INSTRUCTION CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
PICK UP NEXT TEMP STORAGE LOC, SET MINUS, AND STORE IN TABLE
16B

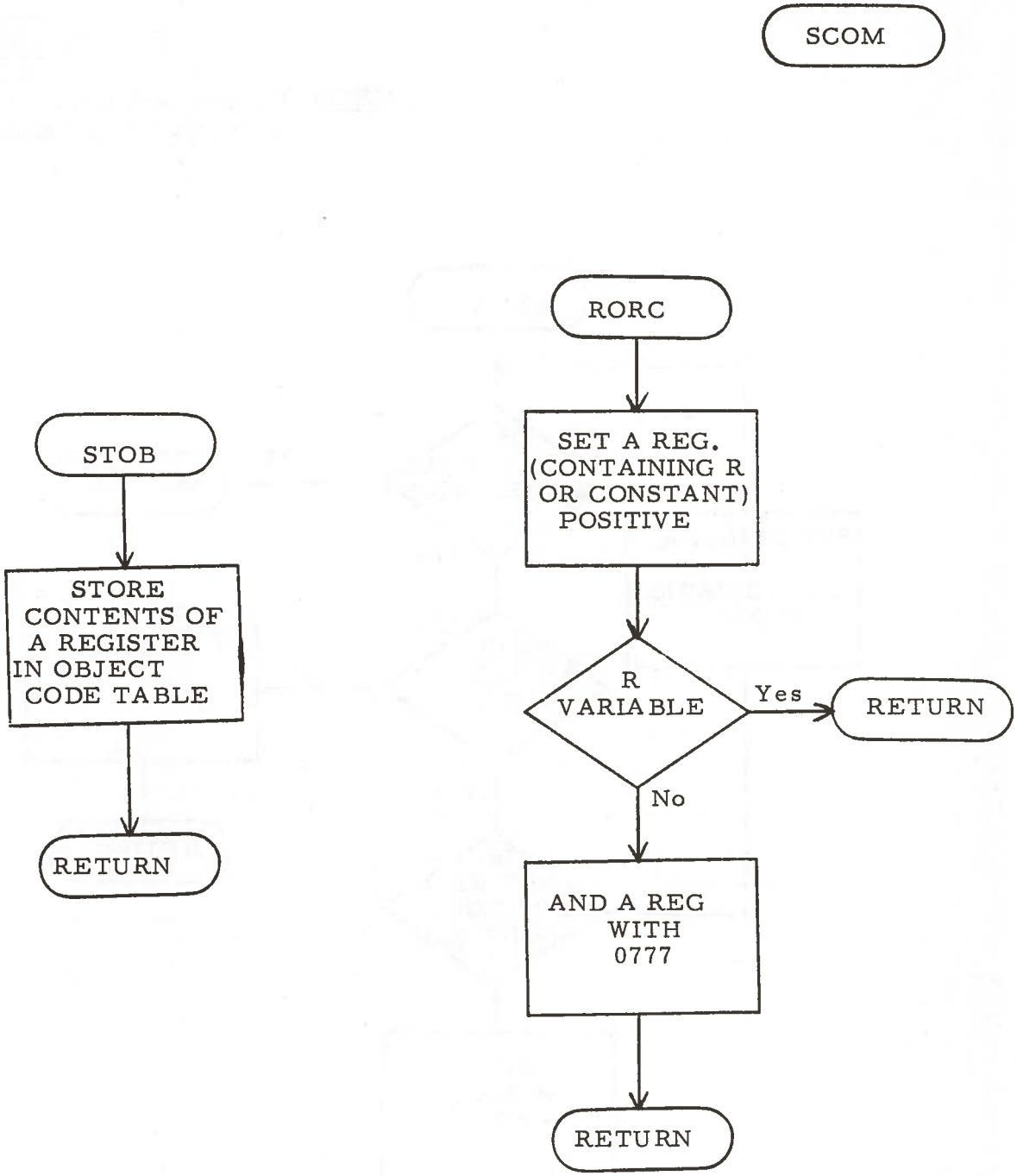

EXPO
18A
PICK UP WORD TO RIGHT OF OPERATOR (EXPONENT)
Call RORC
DETERMINE IF R VARIABLE OR CONSTANT
SET UP FIRST OBJECT CODE INSTRUCTION (LDA EXPON. LOCATION)
Call STOB
STORE OBJECT CODE INSTRUCTION
LOAD A WITH "CMA" INSTRUCTION CODE
18B
18B
Call STOB
STORE OBJECT CODE INSTRUCTION
LOAD A WITH "STA TEMP" INSTRUCTION CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
LOAD A WITH "LGR 15" INSTRUCTION CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
LOAD A WITH "JMP*+3" INSTRUCTION CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
18C
18C
PICK UP WORD TO LEFT OF OPERATOR
Call RORC
DETERMINE IF R VARIABLE OR CONSTANT
LOAD A WITH MUL ARG INSTRUCTION CODE
19A


SCOM
19A
Call STOB
STORE
OBJECT
CODE
INSTRUCTION
LOAD A WITH
"LLS 15"
INSTRUCTION
CODE
Call STOB
STORE
OBJECT
CODE
INSTRUCTION
19B
19B
LOAD A WITH
"IRS EXPON
CTR"
INSTRUCTION
CODE
Call STOB
STORE
OBJECT
CODE
INSTRUCTION
LOAD A WITH
"JMP*-3"
INSTRUCTION
CODE
Call STOB
STORE
OBJECT
CODE
INSTRUCTION
22A
19C
OBJECT CODE
INSERTED TO
PERFORM
DIVISION
LDA DIVIDEND
LRS 15
DIV DIVISOR
SSC
LOAD A WITH
OBJECT
CODE
INSTRUCTION
Call STOB
STORE
OBJECT
CODE
INSTRUCTION
22A


MPCN
20A
OBJECT CODE INSERTED TO PERFORM MULTIPLICATION:
LDA MULTIPLICAND
MPY MULTIPLIER
SSC
LLS 15
LOAD A WITH OBJECT CODE INSTRUCTION
Call STOB
STORE OBJECT CODE INSTRUCTION
22A
ADDT
20C
OBJECT CODE INSERTED TO PERFORM ADDITION:
LDA AUGEND
ADD ADDEND
SSC
LOAD A WITH OBJECT CODE INSTRUCTION
Call STOB
STORE OBJECT CODE INSTRUCTION
22A
EQTY
20D
OBJECT COD
INSERTED TO
PERFORM
EQUALITY:
LDA WORD
AFTER =
STA WORD
BEFORE =
LOAD A WITH OBJECT CODE INSTRUCTION
Call STOB
STORE OBJECT CODE INSTRUCTION
22B

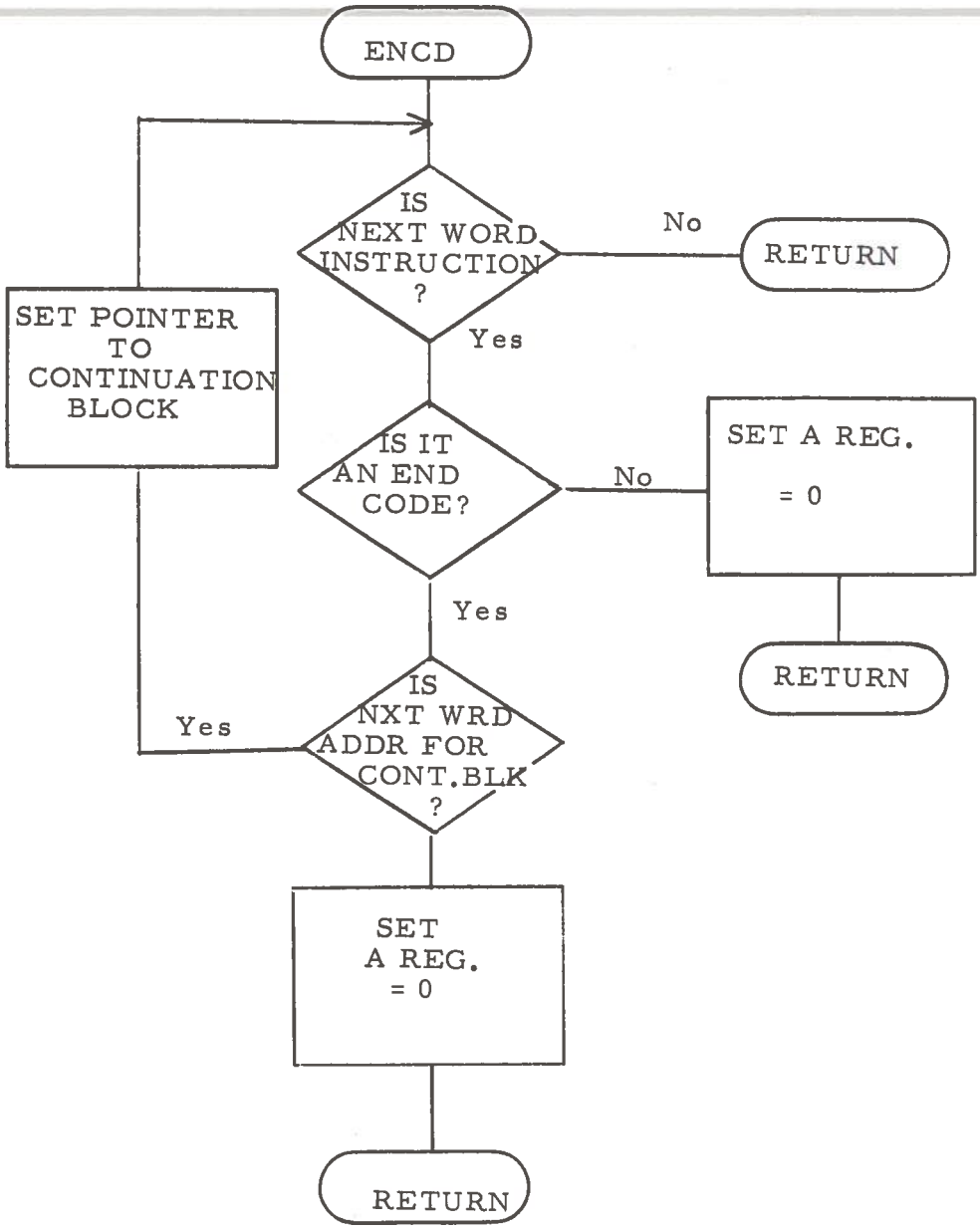

SCOM
SBCT
21A
PICK UP WORD TO LEFT OF OPERATOR
WORD = 0?
Yes
No
HANDLE UNARY MINUS EXPRESSION
LOAD A WITH "CRA" CODE
Call STOB STORE OBJECT CODE INSTRUCTION
21B
Call RORC DETERMINE IF R VARIABLE OR CONSTANT
LOAD A WITH "LDA MINUEND" CODE
Call STOB STORE OBJECT CODE INSTRUCTION
21B
UNRT
21B
PICK UP WORD TO RIGHT OF OPERATOR
Call RORC DETERMINE IF R VARIABLE OR CONSTANT
LOAD A WITH "SUB SUBTRA-HEND" CODE
Call STOB STORE OBJECT CODE INSTRUCTION
LOAD A WITH "SSC" CODE
Call STOB STORE OBJECT CODE INSTRUCTION
22A
STMP


STMP
22A
LOAD A WITH "STA TEMP" OBJECT CODE
Call STOB
STORE OBJECT CODE INSTRUCTION
SET RESULT OF OPERATION MINUS AND STORE
22B
CPRS
END OF EXPRESSION ?
Yes
No
MONX
INCREMENT POINTERS UNTIL THEY POINT TO END OF EXPRESSION
22C
MONY
22C
SCAN COUNTER < -2?
Yes
17A
No
COMPRESS BLOCK BY TWO WORDS
16A


SCOM
RORC
SET A REG.
(CONTAINING R
OR CONSTANT)
POSITIVE
R
VARIABLE
Yes
RETURN
No
AND A REG
WITH
0777
RETURN
STOB
STORE
CONTENTS OF
A REGISTER
IN OBJECT
CODE TABLE
RETURN

SCOM

ENCD: Checks each word for End Code and gets continuation block



SUBROUTINE: STEX

Storage Exceeded

1. PURPOSE:

Type storage exceeded message

2. CALLING SEQUENCE:

CALL STEX

3. INPUT:

None

4. OUTPUT:

Storage exceeded message on teletype

5. ACTION:

Types message. Returns to calling program.

6. EXTERNAL REFERENCES:

TAO$

7. CORE USED:

24_8


STEX
STORAGE
EXCEEDED
MESSAGE
RETURN

SUBROUTINE: TAI1

Teletype ASCII Input - 1 Character

1. PURPOSE:

Inputs 1 character from teletype

2. CALLING SEQUENCE:

CALL TAI1

3. INPUT:

One character from teletype (no characters or control input rejected)

4. OUTPUT:

Input character right-adjusted in A-register.

5. ACTION:

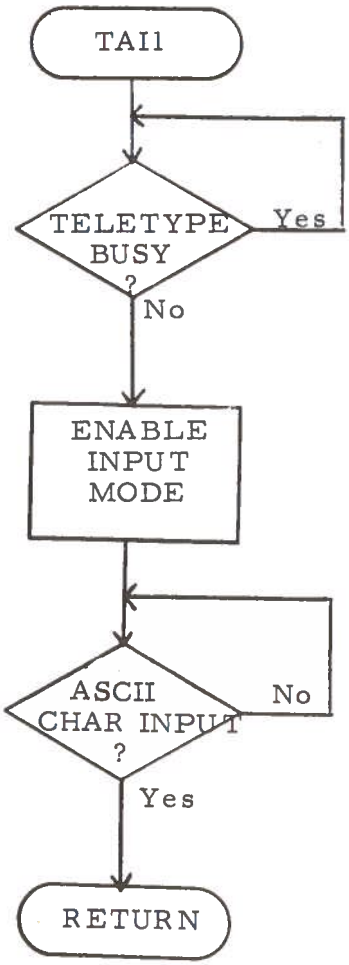
Enables teletype for input. Waits for character. Returns after character input.

6. EXTERNAL REFERENCES:

None

7. CORE USED:

7


TAI1
TELETYPE BUSY ?
Yes
No
ENABLE INPUT MODE
ASCII CHAR INPUT ?
No
Yes
RETURN

SUBROUTINE: TAI$

1. PURPOSE:

Monitors teletype input of up to 70 characters into given buffer

2. CALLING SEQUENCE:

```
CALL    TAI$
DAC     BUFL    BUFL = location of 35 word buffer
```

3. INPUT:

Up to 70 characters terminating with but including carriage return from teletype

4. OUTPUT:

a) characters in given buffer
b) number of characters input in A-register

5. ACTION:

After initialization, characters are accepted from the keyboard one at a time. The "rubout" character will cause the previously input character to be deleted from the buffer, the pointers to back up, and the typing of a reverse arrow. The program is then ready to accept another character. Input of a control X will cause a reverse arrow followed by an "X" to be typed. Control will then be returned to the calling program with a zero in the A-register to indicate the entire input buffer is invalid.

After 64 characters have been put in the buffer a bell is sounded to indicate a near full buffer. If an attempt is made to put more than 70 characters in the buffer, an error message is is printed and control is returned to the calling program with a zero in the A-register.

NOTE: The "rubout" character deletes characters from the buffer,

and decrements char count).

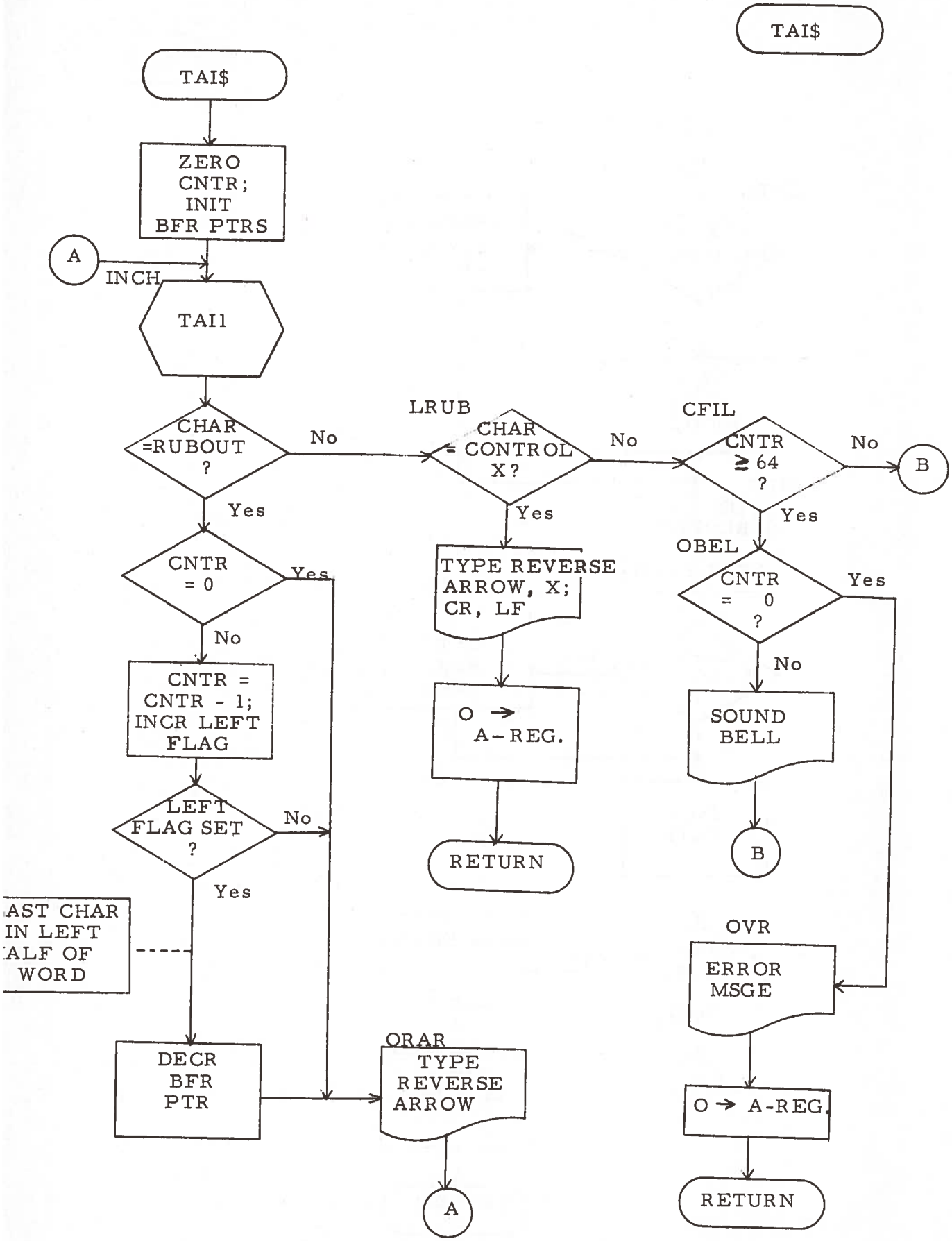
A carriage return will be entered in the buffer and will cause control to be returned to the calling program with the number of input characters in the A-register. Input characters will be packed two to a word in the given buffer. An odd character will be left-adjusted.

6. EXTERNAL REFERENCES:

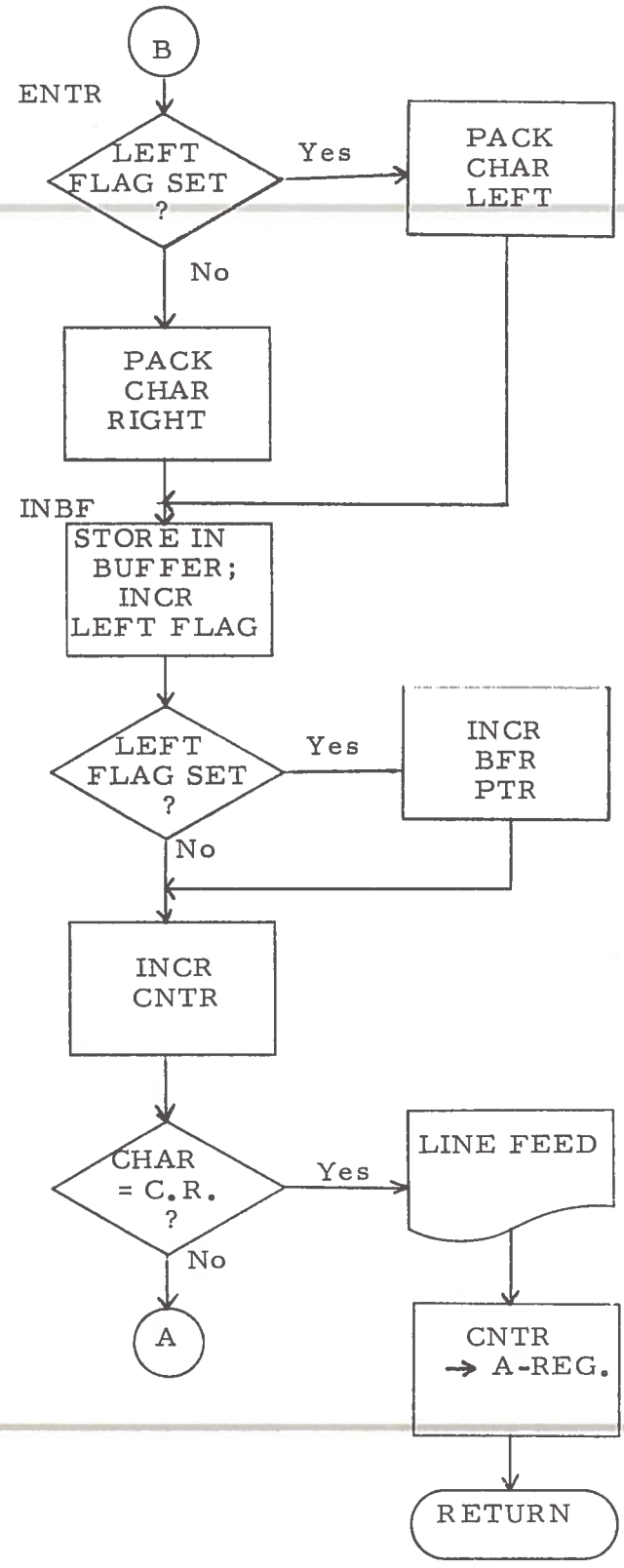
TAI1
TAO1
TAO$

7. CORE USED:

167_8


TAI$
TAI$
ZERO CNTR; INIT BFR PTRS
A
INCH
TAI1
CHAR =RUBOUT ?
No
LRUB
CHAR = CONTROL X?
No
CFIL
CNTR ≥ 64 ?
No
B
Yes
Yes
Yes
CNTR = 0
Yes
No
CNTR = CNTR - 1; INCR LEFT FLAG
LEFT FLAG SET ?
No
Yes
AST CHAR IN LEFT ALF OF WORD
DECR BFR PTR
ORAR
TYPE REVERSE ARROW
A
TYPE REVERSE ARROW, X; CR, LF
O → A-REG.
RETURN
OBEL
CNTR = 0 ?
Yes
No
SOUND BELL
B
OVR
ERROR MSGE
O → A-REG.
RETURN

TAI$


B
ENTR
LEFT FLAG SET ?
Yes
PACK CHAR LEFT
No
PACK CHAR RIGHT
INBF
STORE IN BUFFER; INCR LEFT FLAG
LEFT FLAG SET ?
Yes
INCR BFR PTR
No
INCR CNTR
CHAR = C.R. ?
Yes
LINE FEED
No
A
CNTR → A-REG.
RETURN

SUBROUTINE: TAO1

Teletype ASCII output - 1 character

1. PURPOSE:

Outputs given ASCII character on teletype

2. CALLING SEQUENCE:

```
LDA     CHAR
CALL    TAO1
```

3. INPUT:

One ASCII character right-adjusted in A-register.

4. OUTPUT:

Character is typed on teletype

5. ACTION:

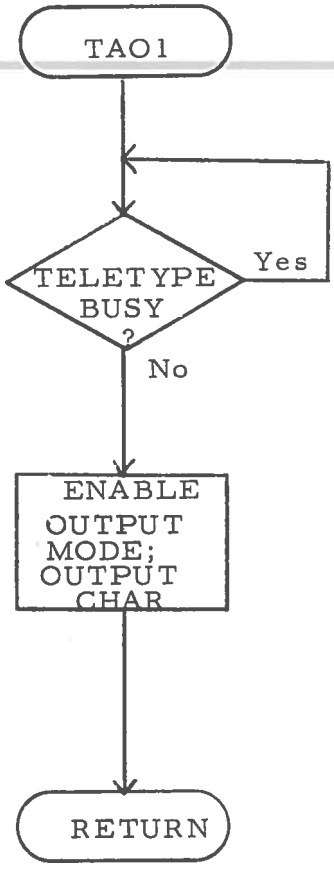
Teletype is enabled for output. Character is output. Return is made to calling routine.

6. EXTERNAL REFERENCES:

None

7. CORE USED:

7


TAO1
TELETYPE BUSY ?
Yes
No
ENABLE OUTPUT MODE; OUTPUT CHAR
RETURN

SUBROUTINE: TAO$

Teletype ASCII output - given buffer

1. PURPOSE:

Outputs characters stored in given buffer

2. CALLING SEQUENCE:

```
LDA     N
TCA             / (optional, if N neg, CR & LF typed)
CALL    TAO$
DAC     BUF
```

3. INPUT:

N	Number of characters to be output
BUF	Location of first word of buffer containing characters for output.

NOTE:

If the A-register is negative on entry to TAO$, it is assumed to be the two's complement of the number of characters, N, and a carriage return and line feed are generated after the buffer is outputted. If the A-register is positive on entry, it is two's complemented by TAO$, and set to count the number of characters, and a carriage return is not generated after the buffer is outputted.

4. OUTPUT:

The characters in the buffer are output on the teletype, optionally followed by a carriage return and line feed.

5. ACTION:

A flag is set according to condition of A-register on entry to indicate whether a carriage return should follow the output. The

character count is set and the pointer and half word indicator are initialized.

A character from the buffer is outputted and the counter and half word indicator are incremented. If the half word indicator now designates a lef-adjusted character, the output pointer is also incremented.

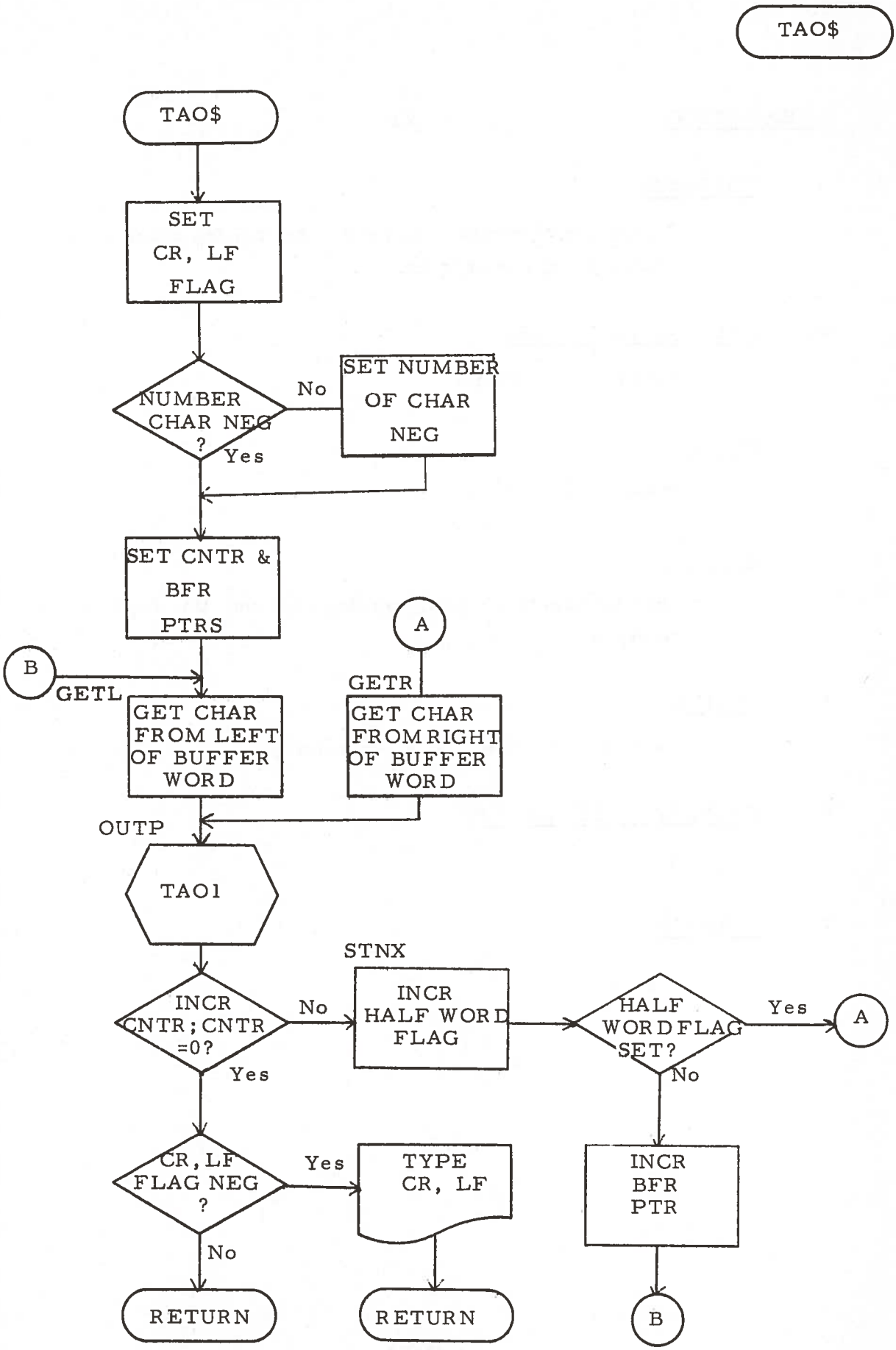
When all characters have been typed, the flag is checked to determine if a carriage return should be output. Then control is returned to the calling program.

6. EXTERNAL REFERENCES:

TAO1

7. CORE USED:

46_8


TAO$
SET CR, LF FLAG
NUMBER CHAR NEG ?
No
SET NUMBER OF CHAR NEG
Yes
SET CNTR & BFR PTRS
A
B
GETL
GETR
GET CHAR FROM LEFT OF BUFFER WORD
GET CHAR FROM RIGHT OF BUFFER WORD
OUTP
TAO1
STNX
INCR CNTR; CNTR =0?
No
INCR HALF WORD FLAG
HALF WORD FLAG SET?
Yes
A
Yes
No
CR, LF FLAG NEG ?
Yes
TYPE CR, LF
INCR BFR PTR
No
RETURN
RETURN
B

SUBROUTINE: WHAT

1. PURPOSE:

   Types question mark indicating that calling routine was unable to interpret input.

2. CALLING SEQUENCE:

   CALL WHAT

3. INPUT:

   None

4. OUTPUT:

   Outputs question mark, carriage return, line feed on teletype

5. ACTION:

   Outputs ?, C.R., L.F. on teletype.

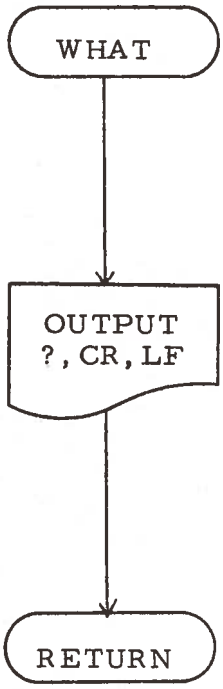
6. EXTERNAL REFERENCES:

   TAO$

7. CORE USED:

   10_8


WHAT


WHAT
OUTPUT
?, CR, LF
RETURN

MONITOR

1. PURPOSE

Handles all initialization, reading of parameters, inputting and displaying the data structure from disc, servicing both photo pen and real time clock interrupts and providing linkage to all routines necessary for the required simulation.

2. CALLING SEQUENCE

NONE (Main program)

3. INPUT

Initial variables from the ASR-35:

a) Data Structure File Name

b) Real Time Clock Interrupt Cycle Time

d) Poll Display Generator

e) Poll PDP-10

f) Poll Pilot Control Station

4. OUTPUT

a) Display of data structure called in from disc

b) Type out requests on the ASR-35

c) Error Messages on the ASR-35

5. ACTION:

After initialization, the monitor requests user input information from the teletype concerning disc file name, update time interval and devices to be polled (See User Guide for type in examples).

The data structure is read in from disc via the Disc Operating System (DOS) and the first frame on the frame ring is displayed.

The monitor then calls the simulate subroutines in a following sequence;

a) Device Polling Subroutines if polling is requested

b) Math Model - update registers RO-R99

c) Perspective - not yet implemented

d) Update - update data structure

e) Background Program

If any of the above subroutines are not present at run time the call will be a dummy subroutine. The background program may retain control on return to the monitor waiting for a real time clock interrupt.

The clock interrupt starts the update cycle. If it occurs before the completion of a full update cycle, an appropriate error message is typed indicating where the time out occurred.

The monitor processes photo pen interrupts by calling the user photo pen interrupt routine, then returning to the interrupted routine for completion of the update cycle.

6. EXTERNAL REFERENCES

PP - photo pen

PSD - pen switch depressed

AN - ASCII control keyboard

FND - function key depress

PSR - pen switch release

FNR - function key release

GC - graphic copy device

X - extra devices

PD10

PPI - photo pen interrupt

RESTORE - restore file (DOS)

TNOUA - type out character array (DOS)

TIIN - read character from teletype (DOS)

MODL- math model

PERS-perspective

UPDT-update

BKGRND-background program

IREG - start address of 100 word register (R0-R99)

PP, PSD, AN, FND, PSR, FNR, GC, and X are called when polling the display generator. The function register contains a one in the bit position related to the external reference by the same title (Refer to the ADDS-900 programmers reference manual). The calling sequence to these routines is:

```
JST*     JMPT + N
return
  :
  :
```

JMPT+N XAC prog name (PP, PSD, AN, FND, etc.)

The calling sequence for polling the PDP-10 is:

```
                JST*     JMPT +8
                return

JMPT+8          XAC      PD10
```

The calling sequence for a photo pen interrupt is:

```
                    JST*    JMPT+9
                    return
                      .
                      .
                      .
                      .
JMPT+9              XAC     PPI
```

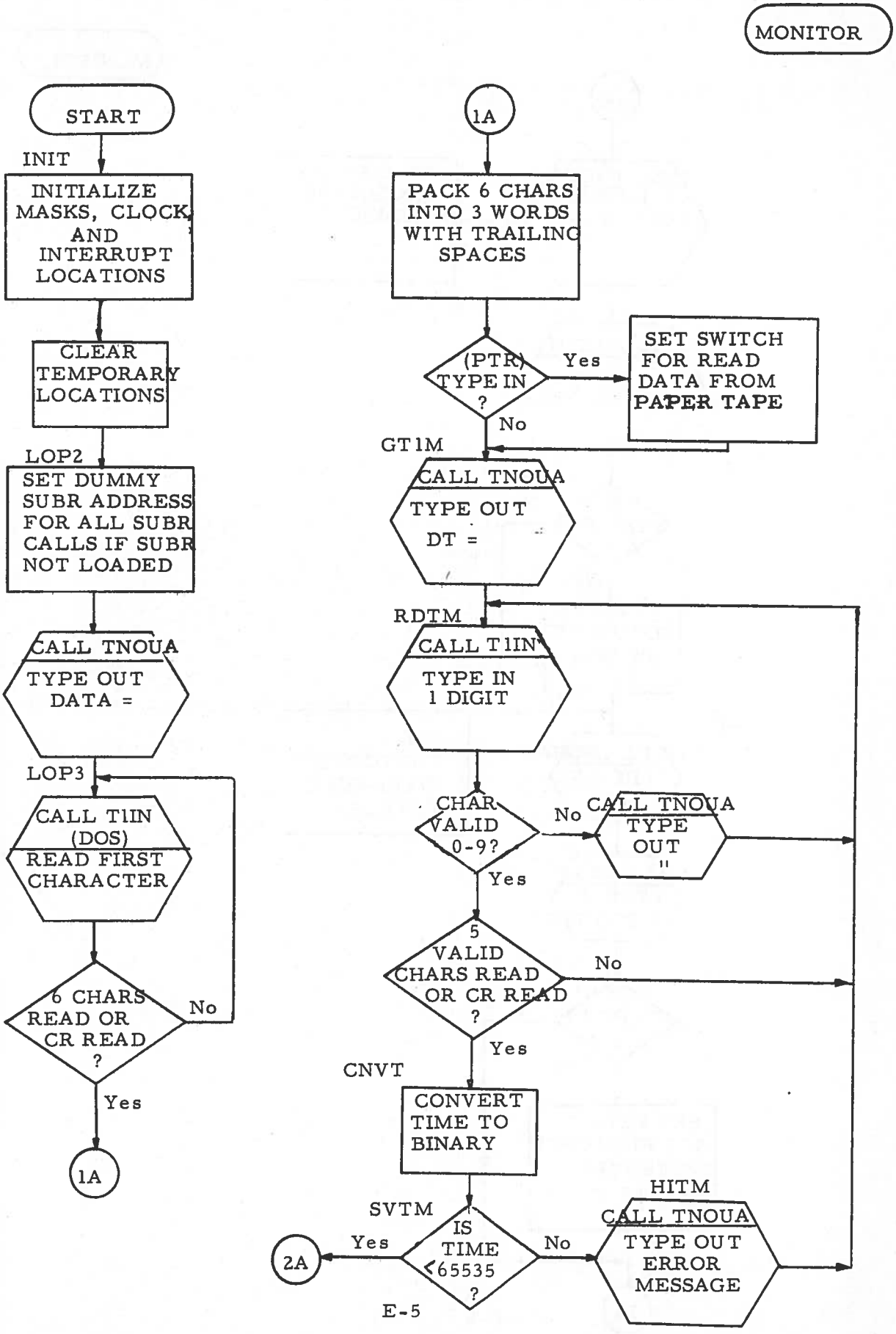
The calling sequence to the math model is

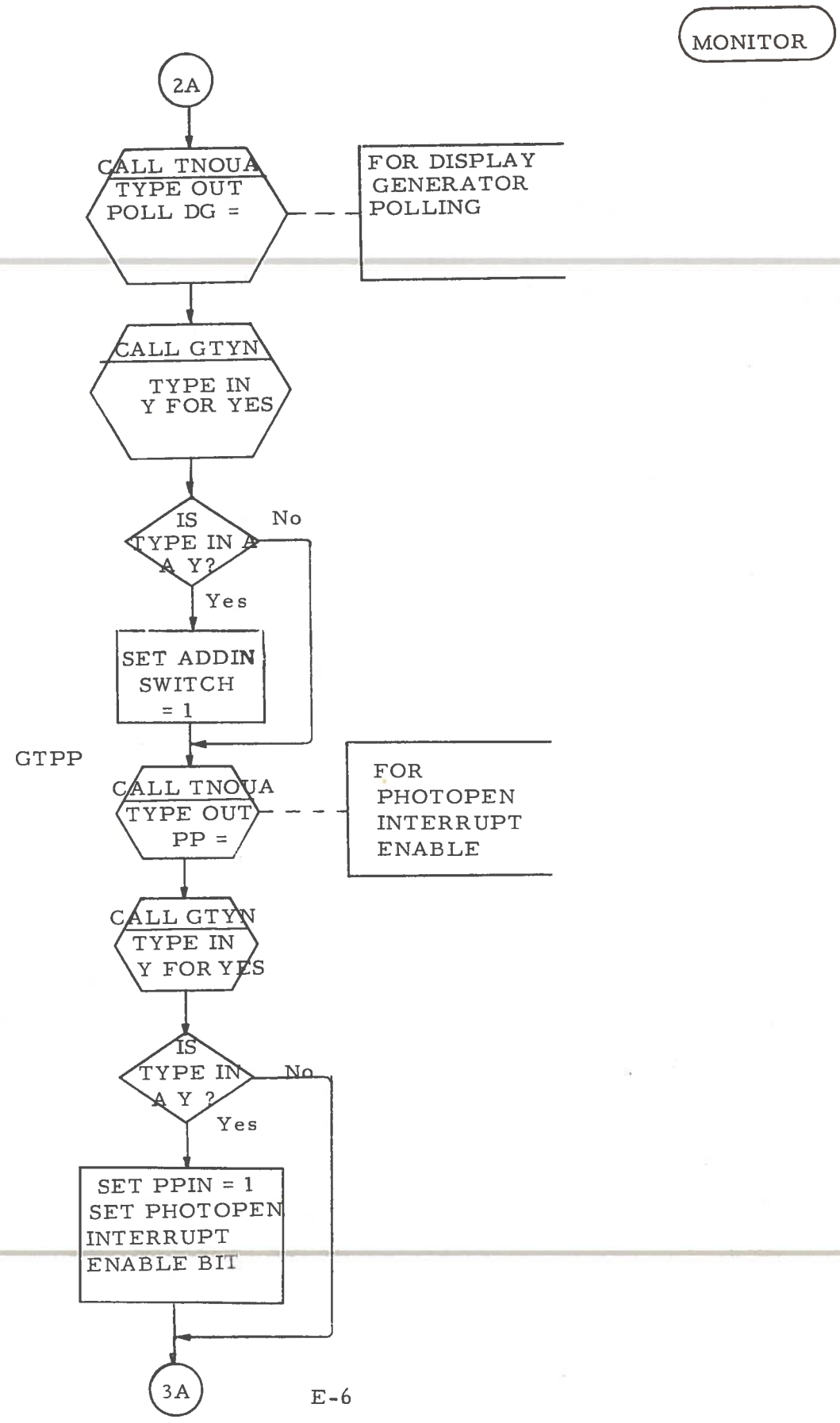
```
                    JST*    JMPT+13
                    DAC
                    return
                      .
                      .
                      .
                      .
JMPT+13             XAC     MODL
```

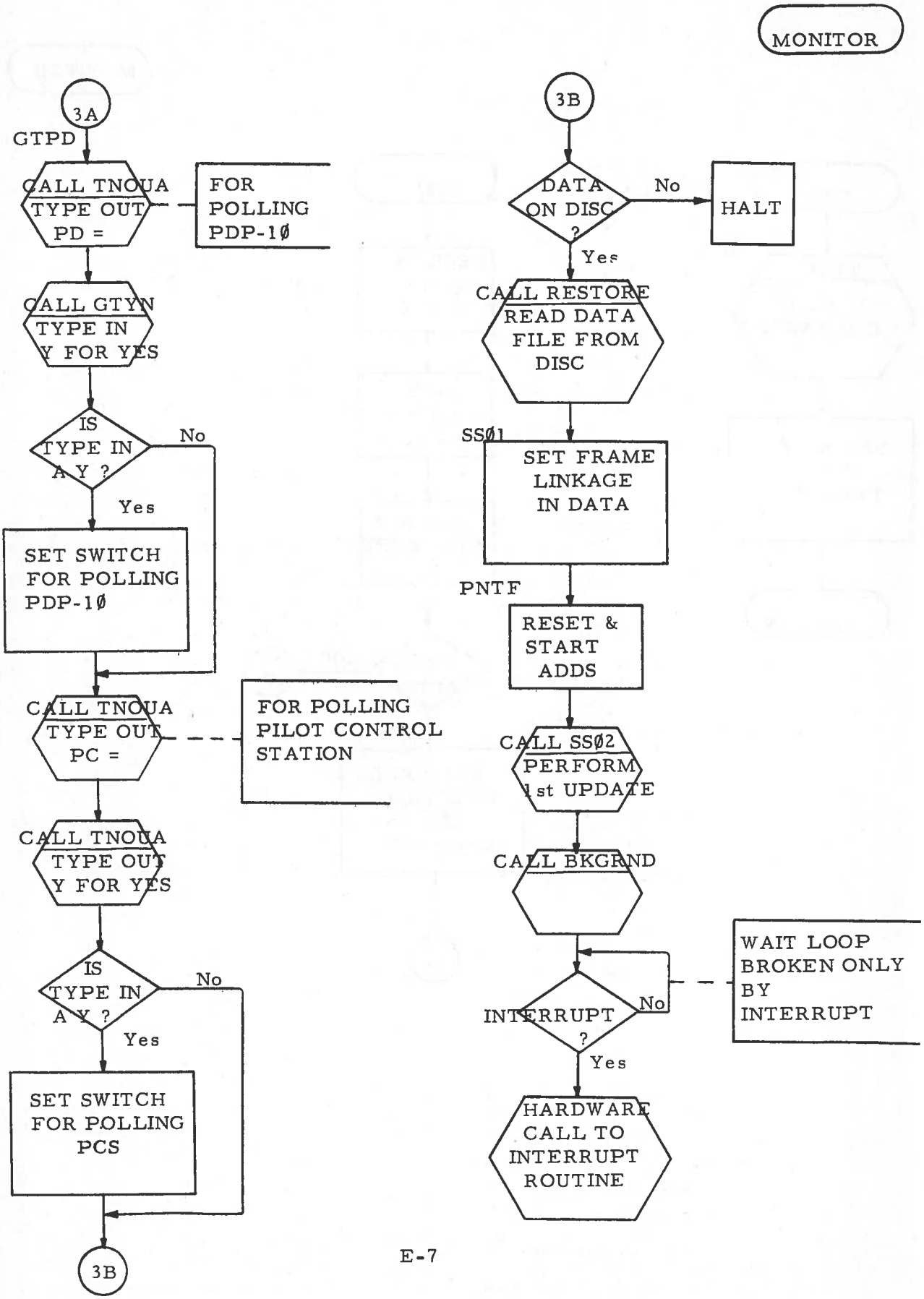
Where:	MPAR+0	Real Time Interrupt internal 0 for 1st entry
	MPAR+1	Not used
	MPAR+2	Frame ID
	MPAR+3	XJOY
	MPAR+4	YJOY
	MPAR+5	THRT
	MPAR+6	LRUD
	MPAR+7	RRUD
	MPAR+8	Not used
	MPAR+9	Not used

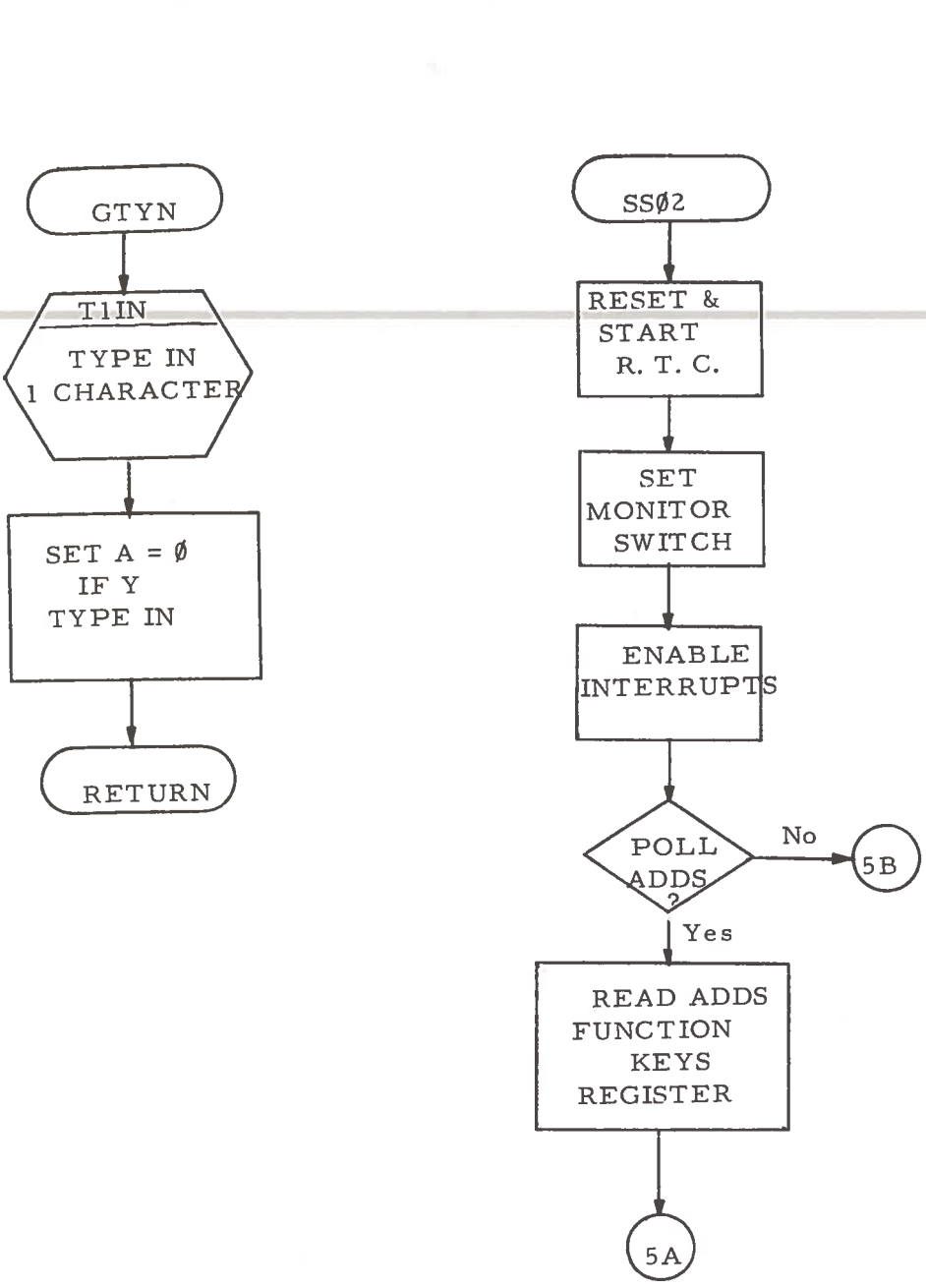
7. CORE USED

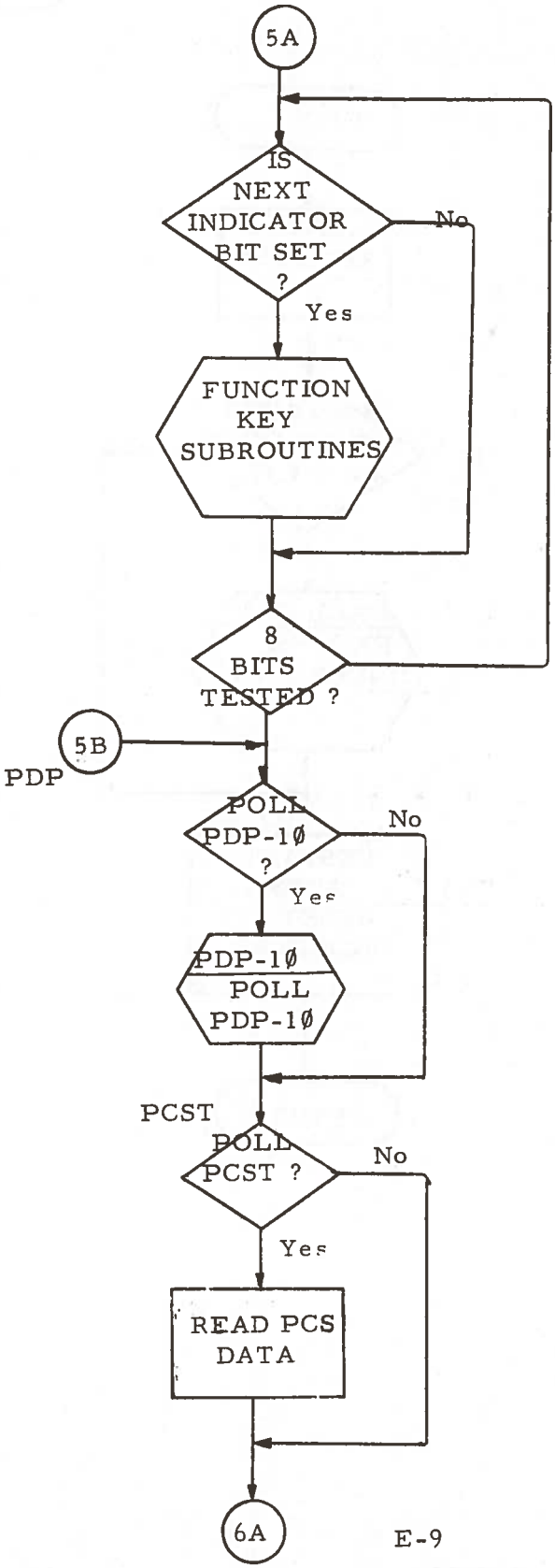
1115_8

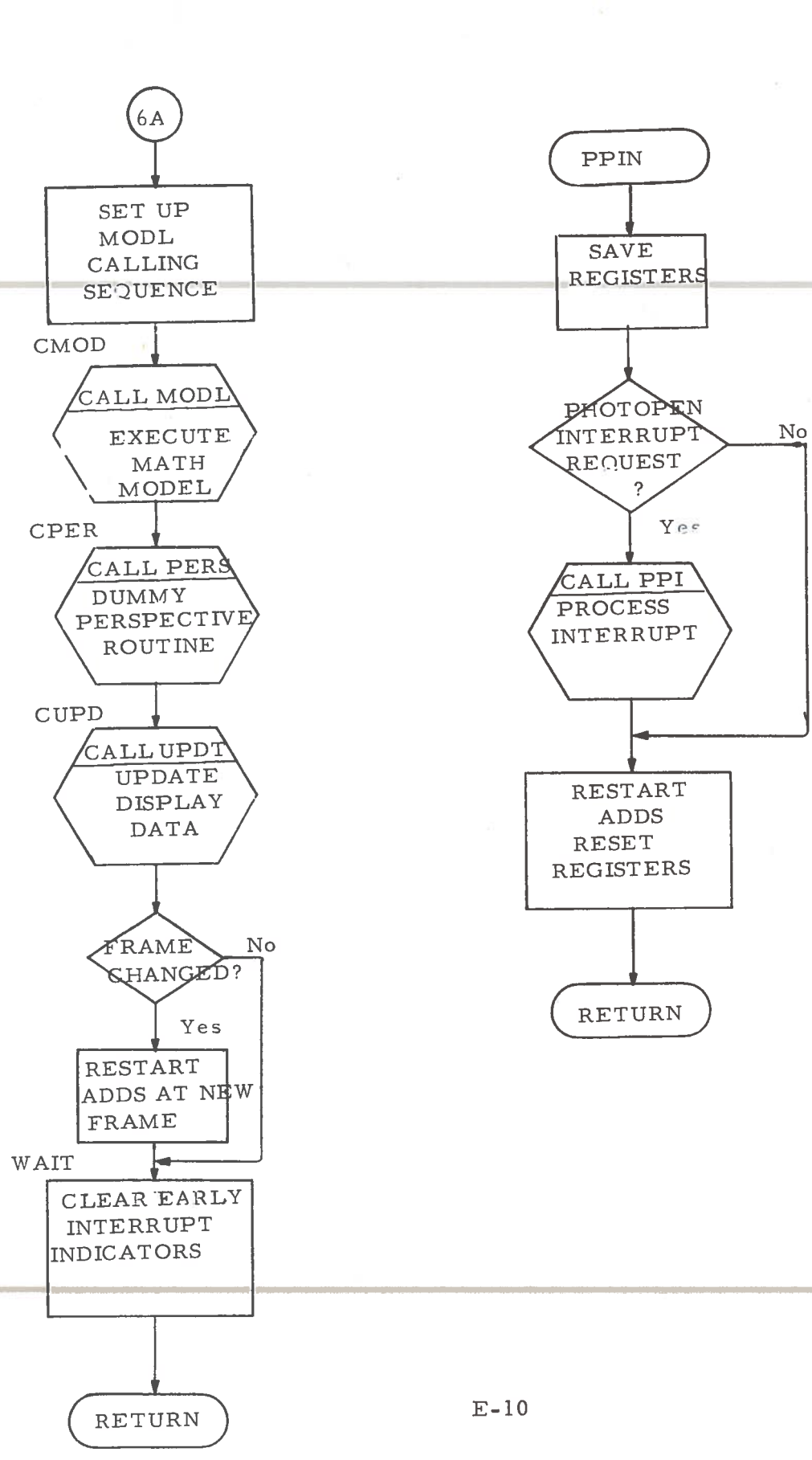

MONITOR
START
INIT
INITIALIZE MASKS, CLOCK, AND INTERRUPT LOCATIONS
CLEAR TEMPORARY LOCATIONS
LOP2
SET DUMMY SUBR ADDRESS FOR ALL SUBR CALLS IF SUBR NOT LOADED
CALL TNOUA
TYPE OUT DATA =
LOP3
CALL T1IN (DOS)
READ FIRST CHARACTER
6 CHARS READ OR CR READ ?
No
Yes
1A
1A
PACK 6 CHARS INTO 3 WORDS WITH TRAILING SPACES
(PTR) TYPE IN ?
Yes
SET SWITCH FOR READ DATA FROM PAPER TAPE
No
GT1M
CALL TNOUA
TYPE OUT DT =
RDTM
CALL T1IN
TYPE IN 1 DIGIT
CHAR VALID 0-9?
No
CALL TNOUA
TYPE OUT "
Yes
5 VALID CHARS READ OR CR READ ?
No
Yes
CNVT
CONVERT TIME TO BINARY
SVTM
IS TIME <65535 ?
Yes
2A
No
HITM
CALL TNOUA
TYPE OUT ERROR MESSAGE
E-5

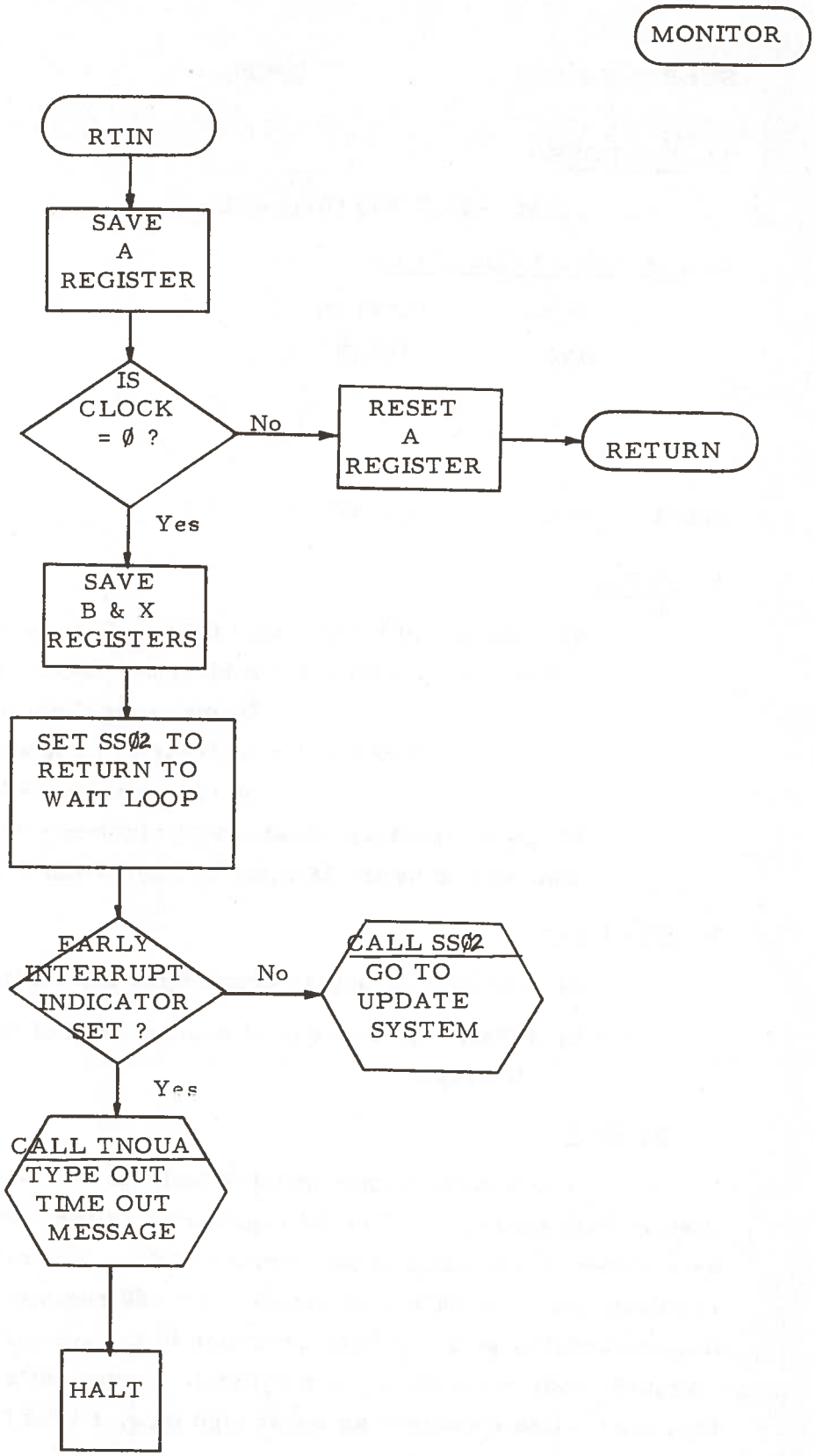

MONITOR
2A
CALL TNOUA
TYPE OUT
POLL DG =
FOR DISPLAY
GENERATOR
POLLING
CALL GTYN
TYPE IN
Y FOR YES
IS
TYPE IN A
A Y?
No
Yes
SET ADDIN
SWITCH
= 1
GTPP
CALL TNOUA
TYPE OUT
PP =
FOR
PHOTOPEN
INTERRUPT
ENABLE
CALL GTYN
TYPE IN
Y FOR YES
IS
TYPE IN
A Y ?
No
Yes
SET PPIN = 1
SET PHOTOPEN
INTERRUPT
ENABLE BIT
3A


MONITOR
3A
GTPD
CALL TNOUA
TYPE OUT
PD =
FOR
POLLING
PDP-1Ø
CALL GTYN
TYPE IN
Y FOR YES
IS
TYPE IN
A Y ?
No
Yes
SET SWITCH
FOR POLLING
PDP-1Ø
CALL TNOUA
TYPE OUT
PC =
FOR POLLING
PILOT CONTROL
STATION
CALL TNOUA
TYPE OUT
Y FOR YES
IS
TYPE IN
A Y ?
No
Yes
SET SWITCH
FOR POLLING
PCS
3B
3B
DATA
ON DISC
?
No
HALT
Yes
CALL RESTORE
READ DATA
FILE FROM
DISC
SSØ1
SET FRAME
LINKAGE
IN DATA
PNTF
RESET &
START
ADDS
CALL SSØ2
PERFORM
1st UPDATE
CALL BKGRND
WAIT LOOP
BROKEN ONLY
BY
INTERRUPT
INTERRUPT
?
No
Yes
HARDWARE
CALL TO
INTERRUPT
ROUTINE


MONITOR
GTYN
T1IN
TYPE IN
1 CHARACTER
SET A = Ø
IF Y
TYPE IN
RETURN
SSØ2
RESET &
START
R. T. C.
SET
MONITOR
SWITCH
ENABLE
INTERRUPTS
POLL
ADDS
?
No
5B
Yes
READ ADDS
FUNCTION
KEYS
REGISTER
5A


5A
IS NEXT INDICATOR BIT SET ?
No
Yes
FUNCTION KEY SUBROUTINES
8 BITS TESTED ?
5B
PDP
POLL PDP-1Ø ?
No
Yes
PDP-1Ø POLL PDP-1Ø
PCST
POLL PCST ?
No
Yes
READ PCS DATA
6A


MONITOR
6A
SET UP MODL CALLING SEQUENCE
CMOD
CALL MODL
EXECUTE MATH MODEL
CPER
CALL PERS
DUMMY PERSPECTIVE ROUTINE
CUPD
CALL UPDT
UPDATE DISPLAY DATA
FRAME CHANGED?
No
Yes
RESTART ADDS AT NEW FRAME
WAIT
CLEAR EARLY INTERRUPT INDICATORS
RETURN
PPIN
SAVE REGISTERS
PHOTOPEN INTERRUPT REQUEST ?
No
Yes
CALL PPI
PROCESS INTERRUPT
RESTART ADDS RESET REGISTERS
RETURN


MONITOR
RTIN
SAVE A REGISTER
IS CLOCK = Ø ?
No
RESET A REGISTER
RETURN
Yes
SAVE B & X REGISTERS
SET SSØ2 TO RETURN TO WAIT LOOP
EARLY INTERRUPT INDICATOR SET ?
No
CALL SSØ2
GO TO UPDATE SYSTEM
Yes
CALL TNOUA
TYPE OUT TIME OUT MESSAGE
HALT

SUBROUTINE UPDT

1. PURPOSE:

Update ADDS/900 Display List

2. CALLING SEQUENCE:

```
        JST*    JMPT+N
        DAC     UPAR
        .
        .
        .
JMPT+N  XAC     UPDT
```

3. INPUT:

a) Address of UPAR from Calling Sequence

Where: UPAR + Ø = Address of second word of frame being displayed

UPAR + 1 = Address of first word of block of 1ØØ registers RØ thru R99.

b) Data structure created and saved in the Create Phase and read in by the Monitor in Simulation Phase.

4. OUTPUT:

a) The data structure updated for the frame to be displayed.

b) UPAR + Ø, address of second word of frame to be displayed.

5. ACTION:

The update routine updates both the 1ØØ registers and the display data structure. The 9Ø registers R1Ø thru R99 are updated as a result of execution of the appropriate mini-compiled code resulting from the DRE statements. All 1ØØ registers RØ thru R99 may be updated as a result of execution of the appropriate mini-compiled code resulting from a dynamic component's expression if that expression contained an equal sign (e.g. L1ØØ# RØ = R90 + 5.)

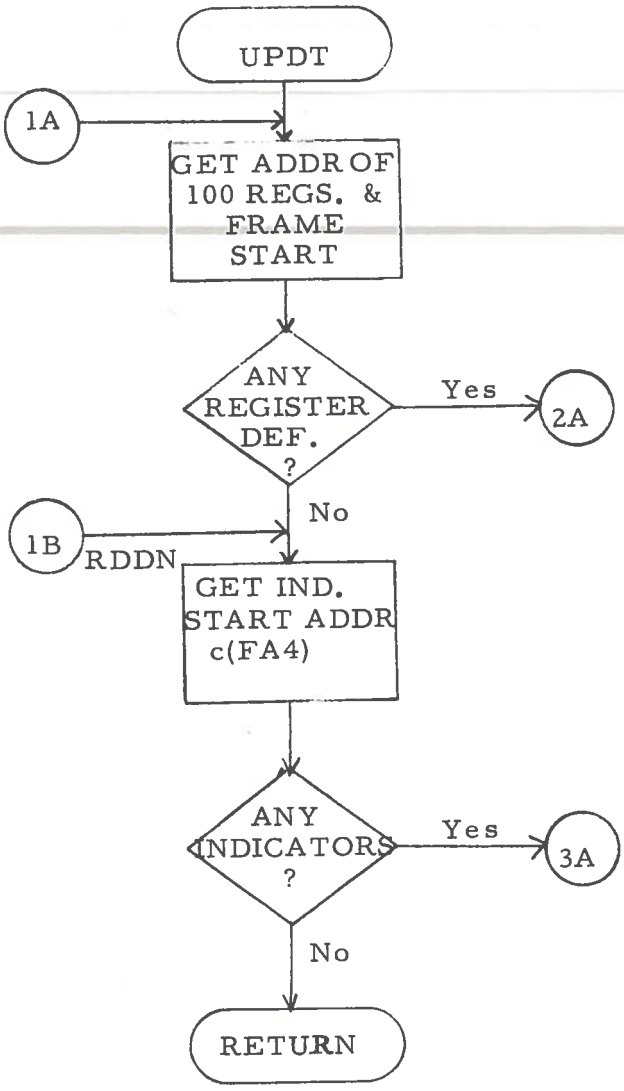
The display data structure is updated as a result of the execution of the conditioning statements at both the indicator and entity levels, and by execution of the appropriate mini-compiled code resulting from dynamic component's expression at the entity level.

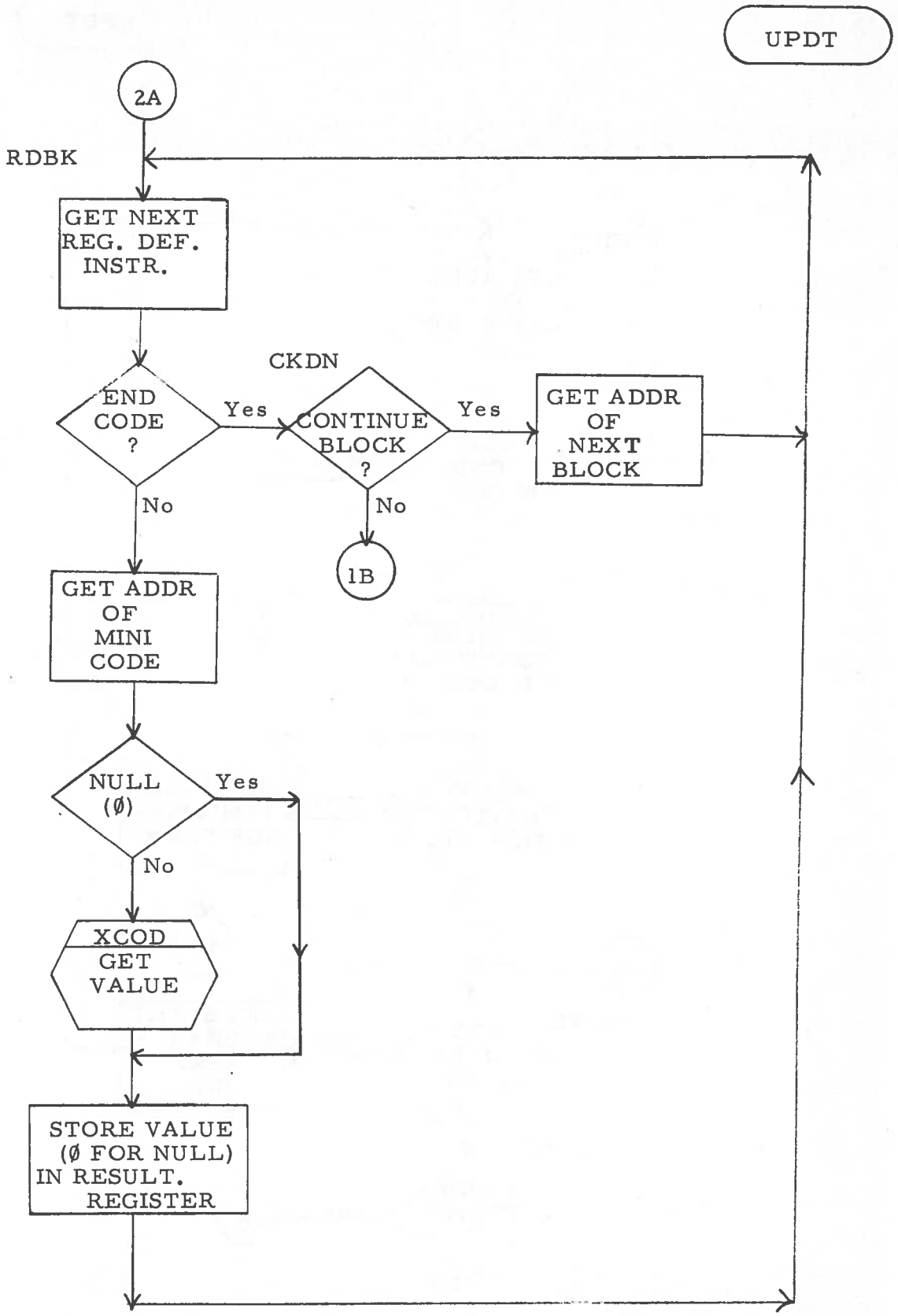
6. EXTERNAL REFERENCES:

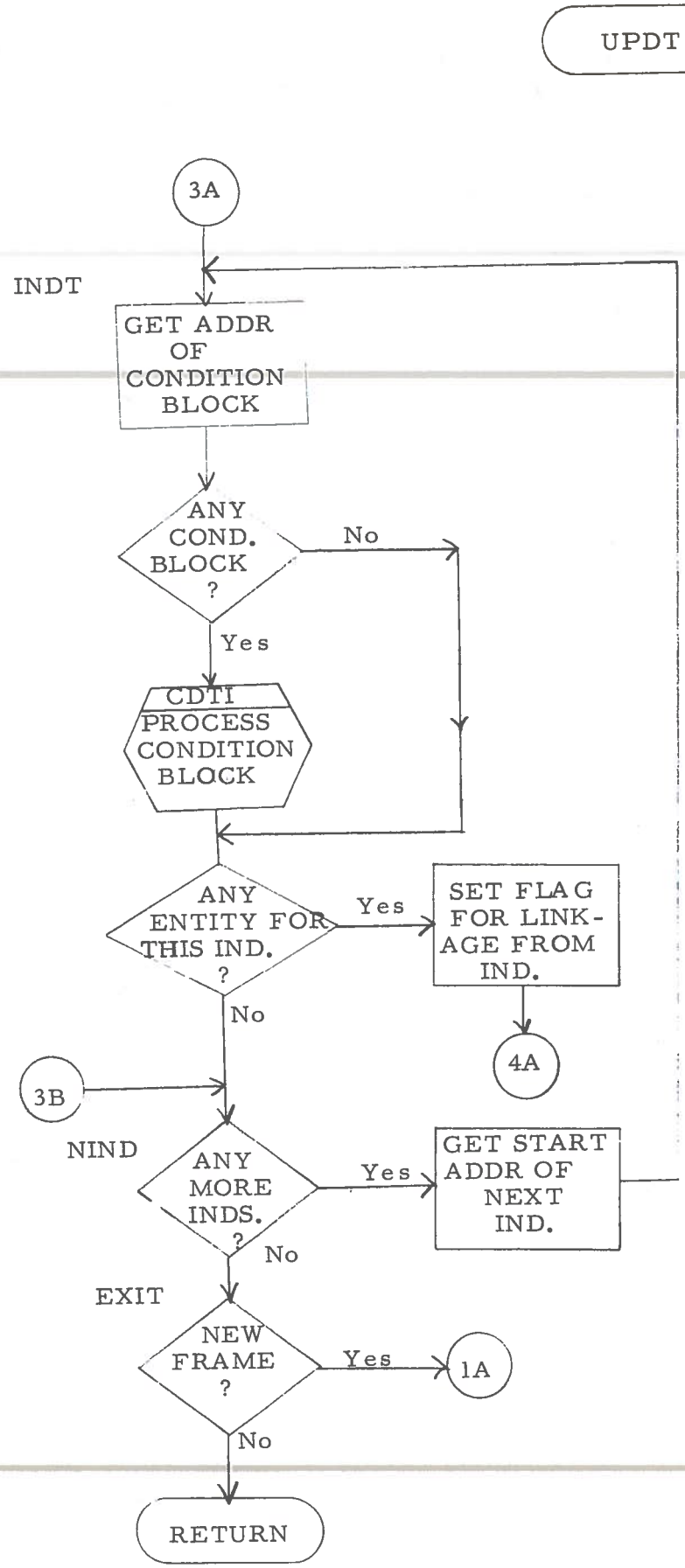
None

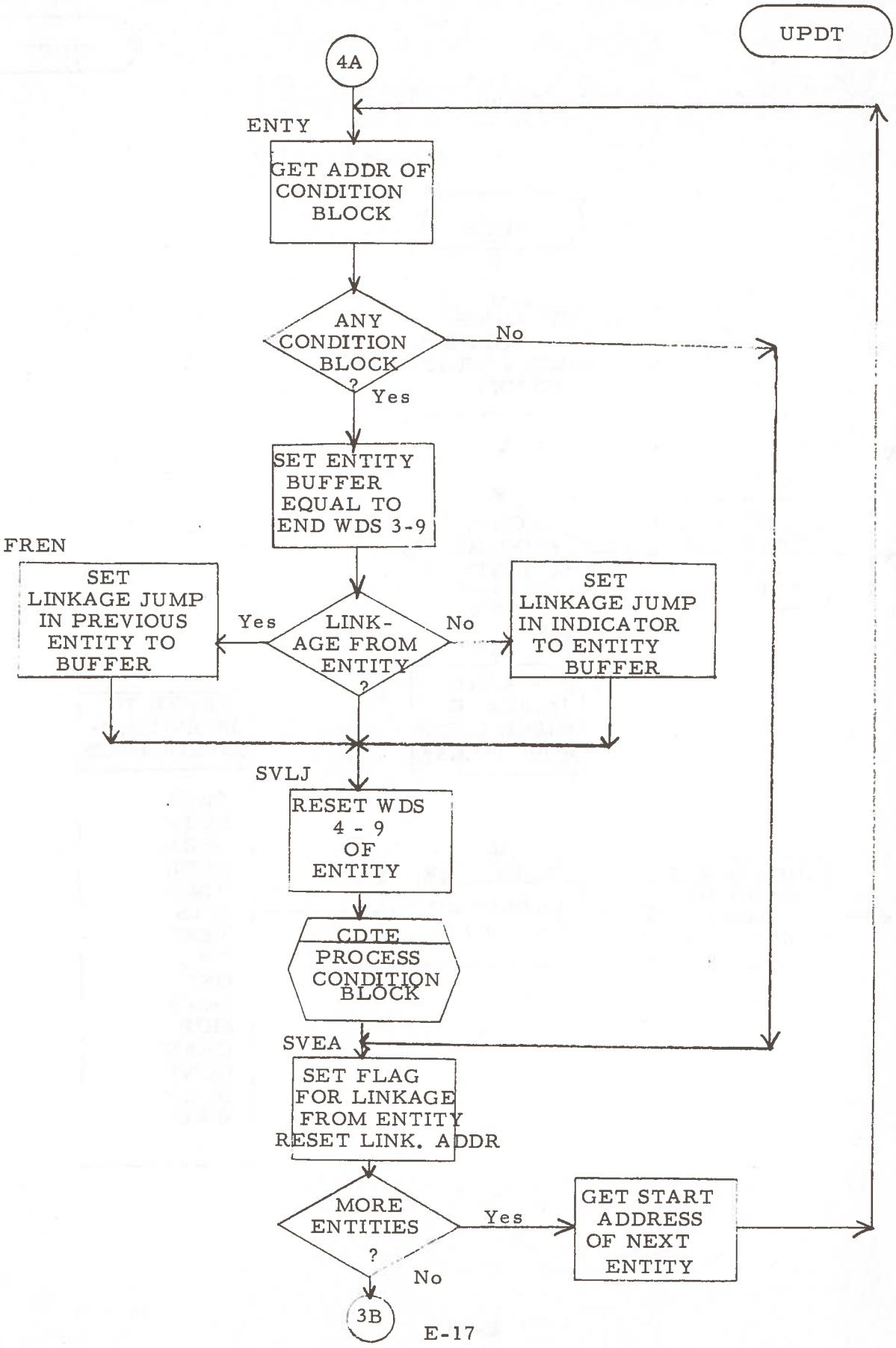
7. CORE USED:

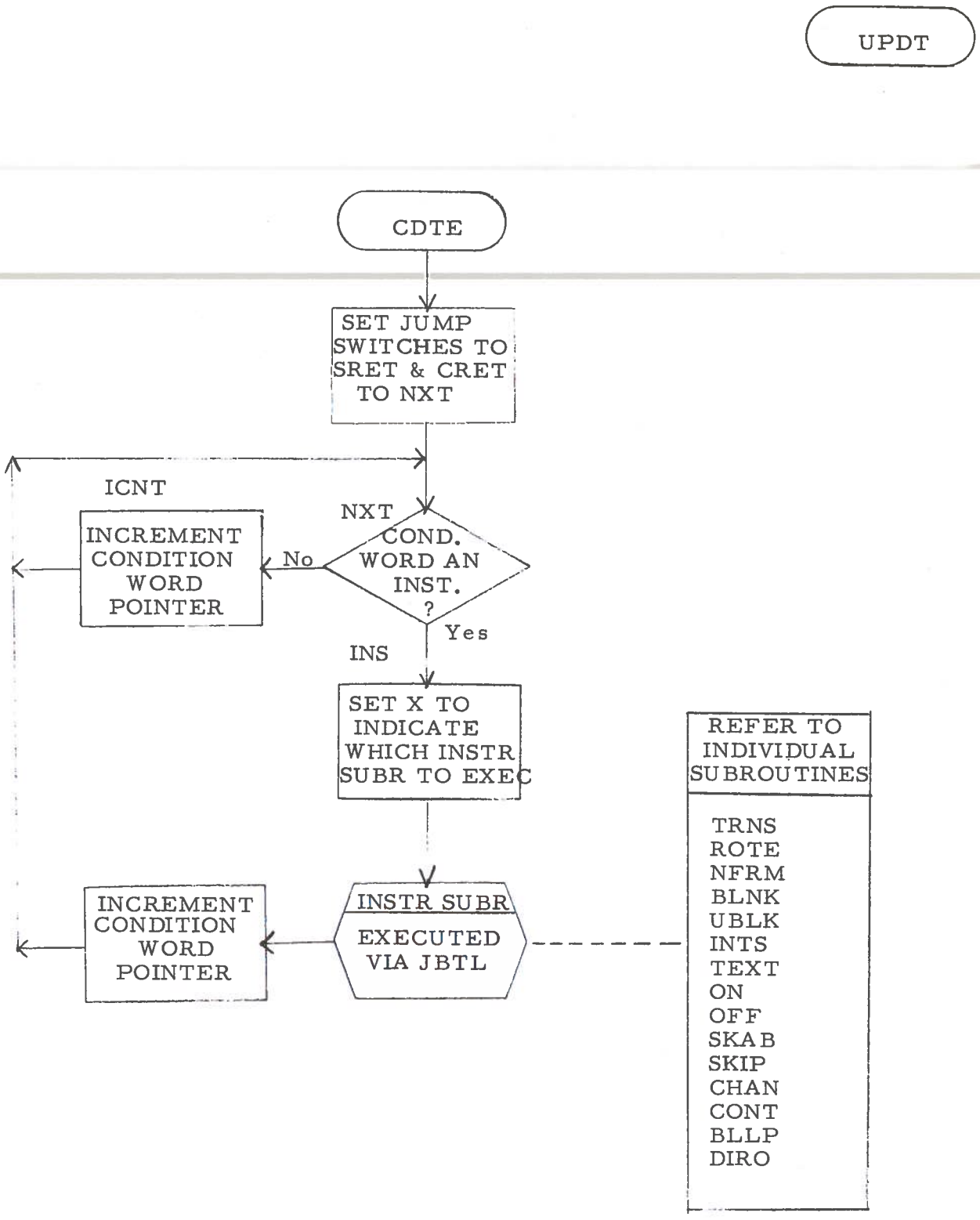
1266_8

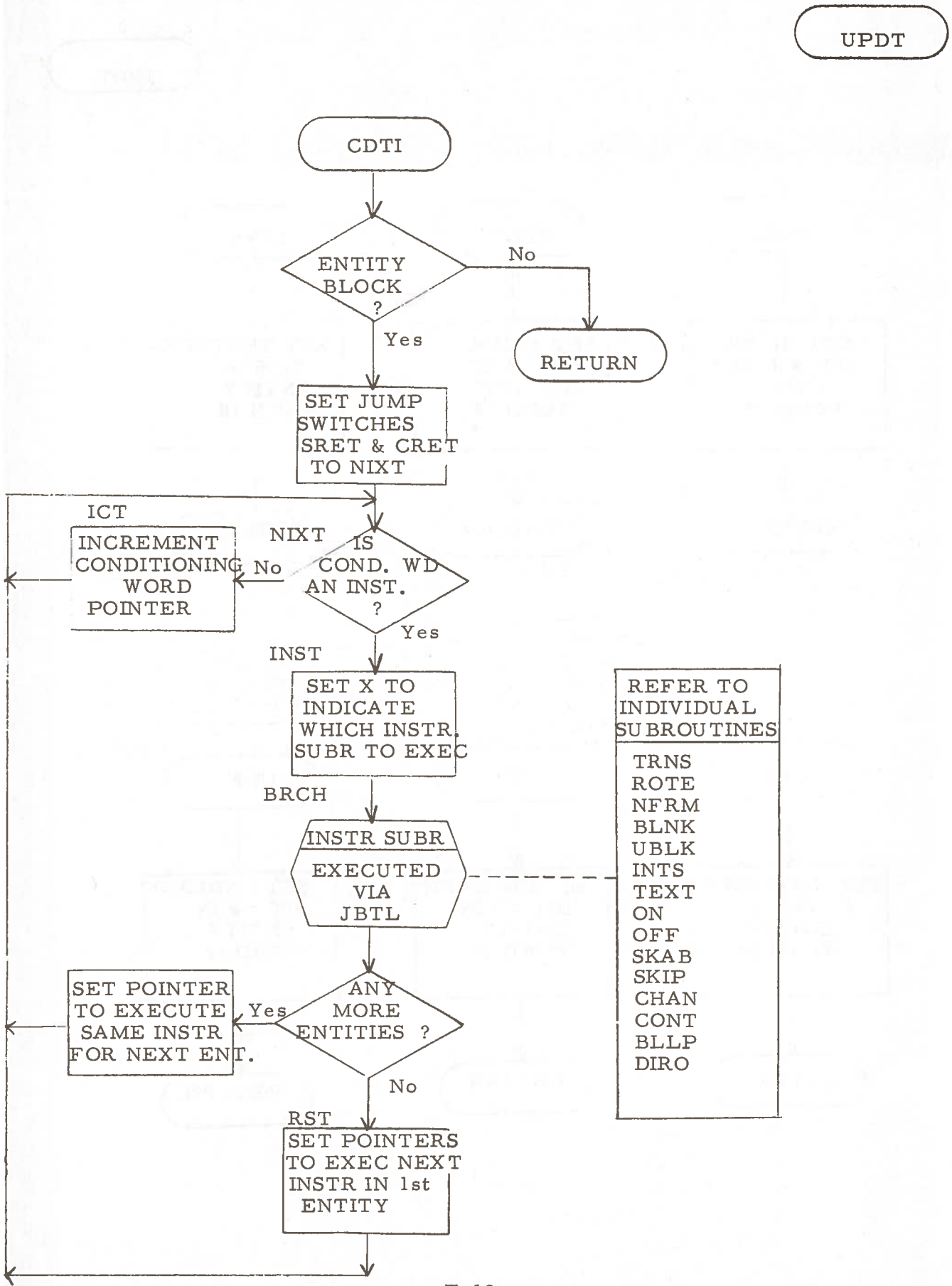

UPDT
1A
GET ADDR OF 100 REGS. & FRAME START
ANY REGISTER DEF. ?
Yes
2A
No
1B
RDDN
GET IND. START ADDR c(FA4)
ANY INDICATORS ?
Yes
3A
No
RETURN

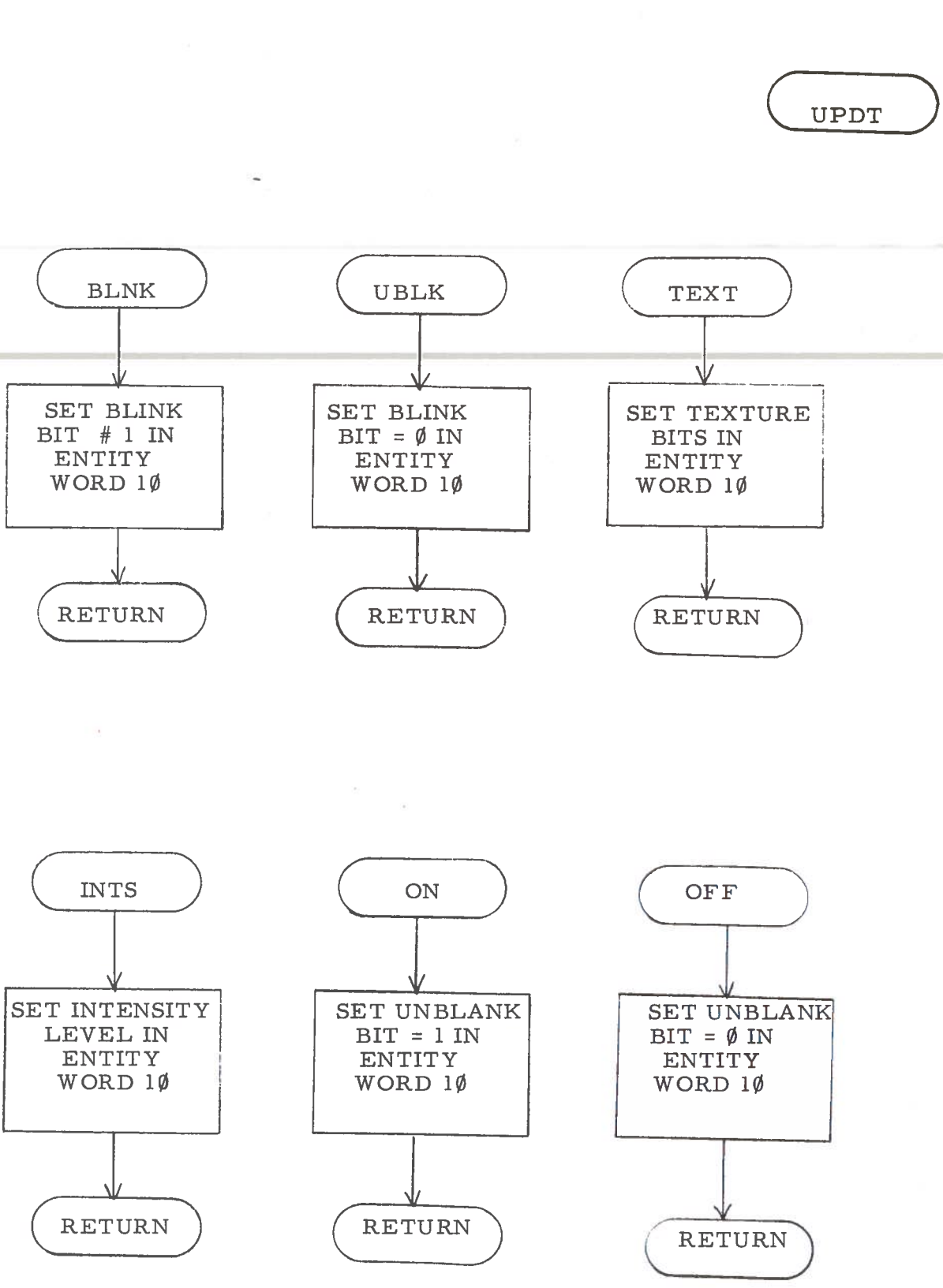

UPDT
2A
RDBK
GET NEXT REG. DEF. INSTR.
END CODE ?
Yes
CKDN
CONTINUE BLOCK ?
Yes
GET ADDR OF NEXT BLOCK
No
No
1B
GET ADDR OF MINI CODE
NULL (Ø)
Yes
No
XCOD
GET VALUE
STORE VALUE (Ø FOR NULL) IN RESULT. REGISTER

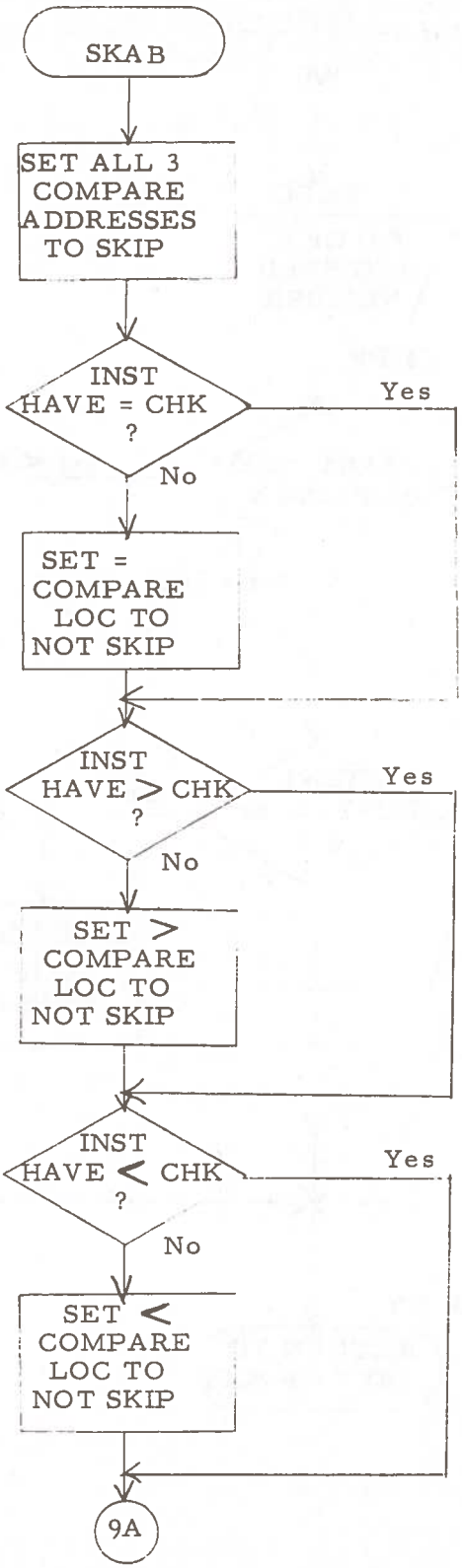

UPDT
3A
INDT
GET ADDR OF CONDITION BLOCK
ANY COND. BLOCK ?
No
Yes
CDTI
PROCESS CONDITION BLOCK
ANY ENTITY FOR THIS IND. ?
Yes
SET FLAG FOR LINK-AGE FROM IND.
4A
No
3B
NIND
ANY MORE INDS. ?
Yes
GET START ADDR OF NEXT IND.
No
EXIT
NEW FRAME ?
Yes
1A
No
RETURN

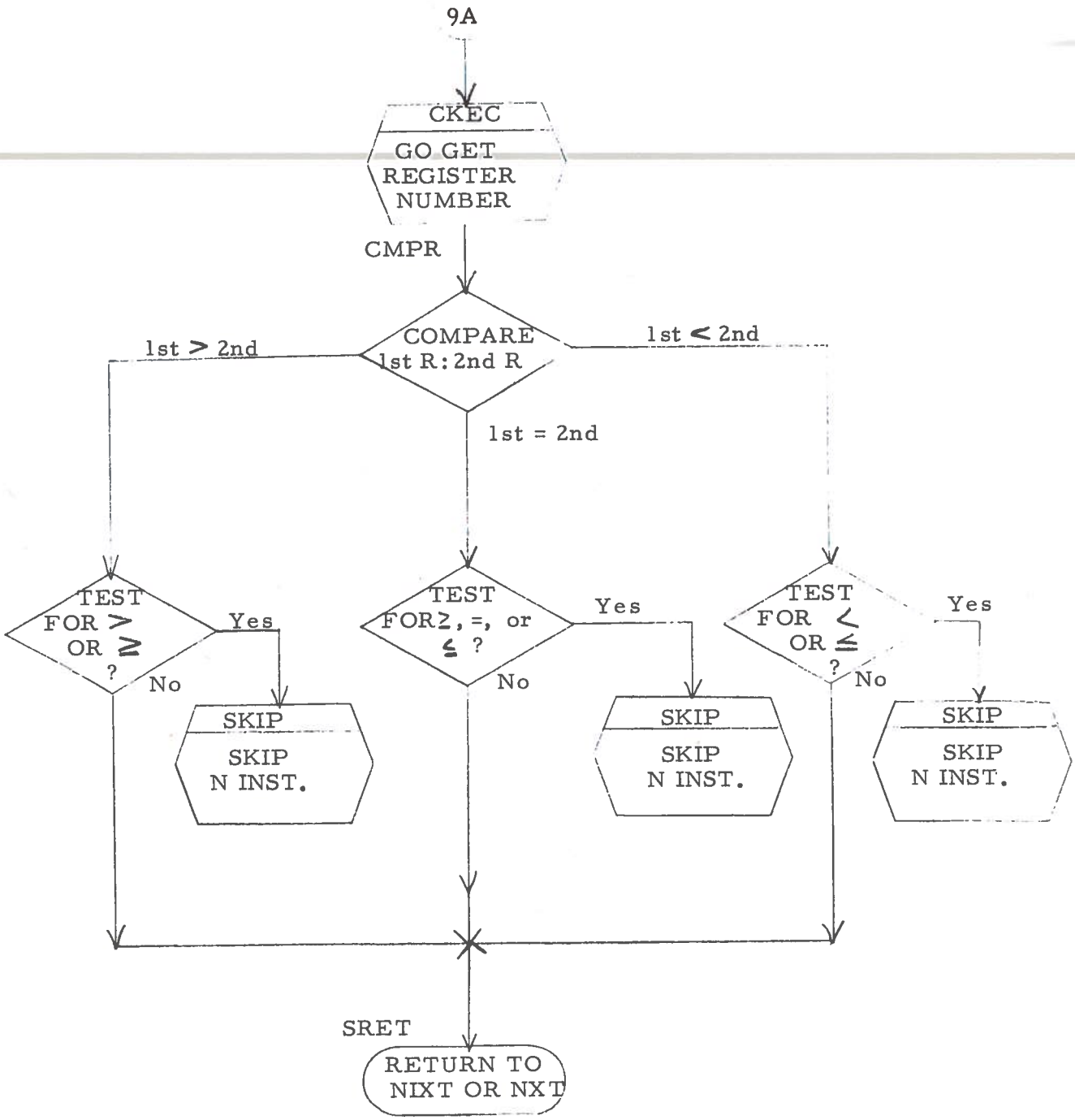

UPDT
4A
ENTY
GET ADDR OF CONDITION BLOCK
ANY CONDITION BLOCK ?
No
Yes
SET ENTITY BUFFER EQUAL TO END WDS 3-9
FREN
SET LINKAGE JUMP IN PREVIOUS ENTITY TO BUFFER
Yes
LINK-AGE FROM ENTITY ?
No
SET LINKAGE JUMP IN INDICATOR TO ENTITY BUFFER
SVLJ
RESET WDS 4 - 9 OF ENTITY
CDTE
PROCESS CONDITION BLOCK
SVEA
SET FLAG FOR LINKAGE FROM ENTITY RESET LINK. ADDR
MORE ENTITIES ?
Yes
GET START ADDRESS OF NEXT ENTITY
No
3B

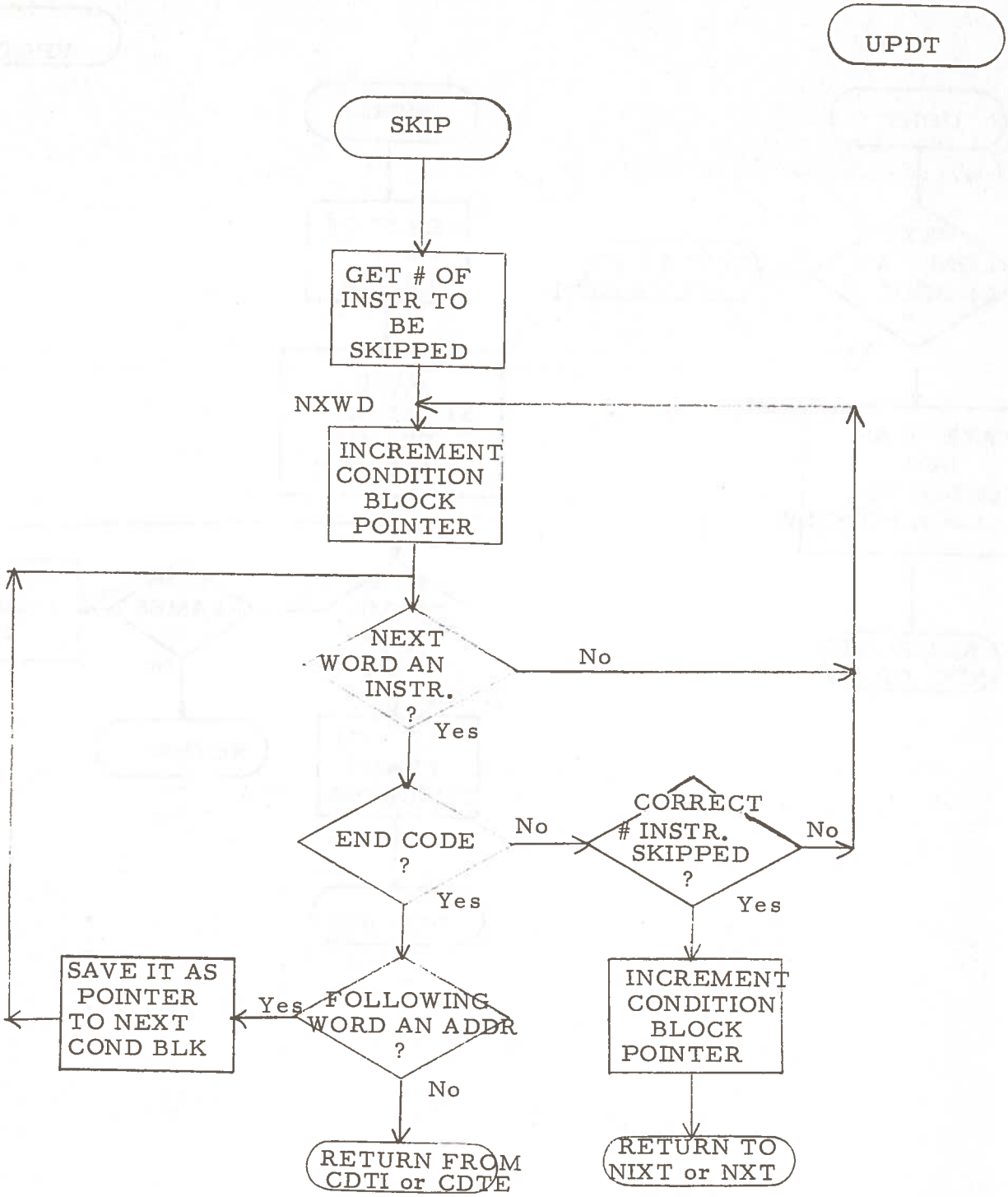

UPDT
CDTE
SET JUMP SWITCHES TO SRET & CRET TO NXT
ICNT
NXT
COND. WORD AN INST. ?
No
INCREMENT CONDITION WORD POINTER
Yes
INS
SET X TO INDICATE WHICH INSTR SUBR TO EXEC
REFER TO INDIVIDUAL SUBROUTINES
TRNS
ROTE
NFRM
BLNK
UBLK
INTS
TEXT
ON
OFF
SKAB
SKIP
CHAN
CONT
BLLP
DIRO
INSTR SUBR EXECUTED VIA JBTL
INCREMENT CONDITION WORD POINTER

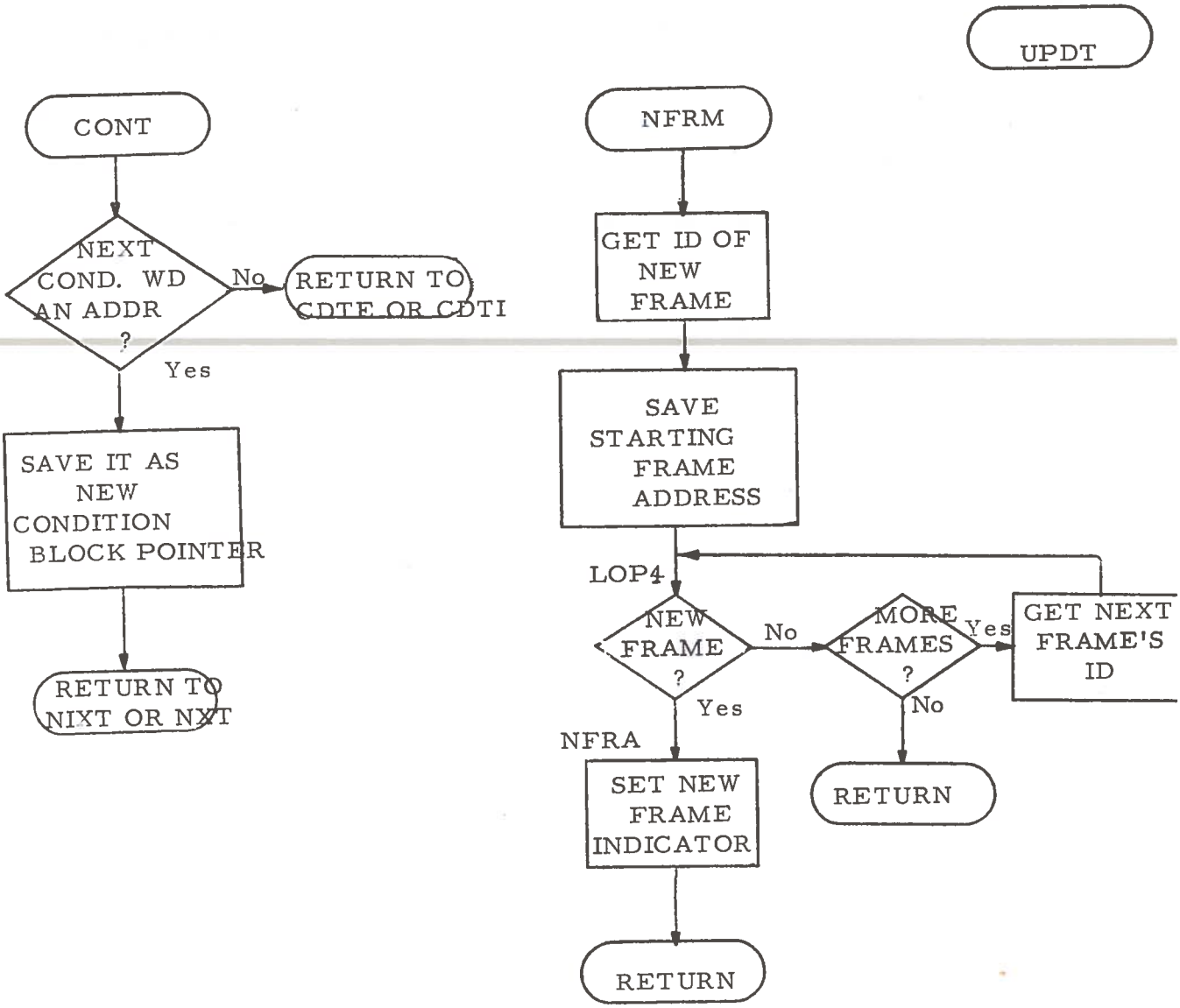

UPDT
CDTI
ENTITY BLOCK ?
No
Yes
RETURN
SET JUMP SWITCHES SRET & CRET TO NIXT
ICT
INCREMENT CONDITIONING WORD POINTER
NIXT
IS COND. WD AN INST. ?
No
Yes
INST
SET X TO INDICATE WHICH INSTR. SUBR TO EXEC
BRCH
INSTR SUBR EXECUTED VIA JBTL
REFER TO INDIVIDUAL SUBROUTINES
TRNS
ROTE
NFRM
BLNK
UBLK
INTS
TEXT
ON
OFF
SKAB
SKIP
CHAN
CONT
BLLP
DIRO
SET POINTER TO EXECUTE SAME INSTR FOR NEXT ENT.
Yes
ANY MORE ENTITIES ?
No
RST
SET POINTERS TO EXEC NEXT INSTR IN 1st ENTITY

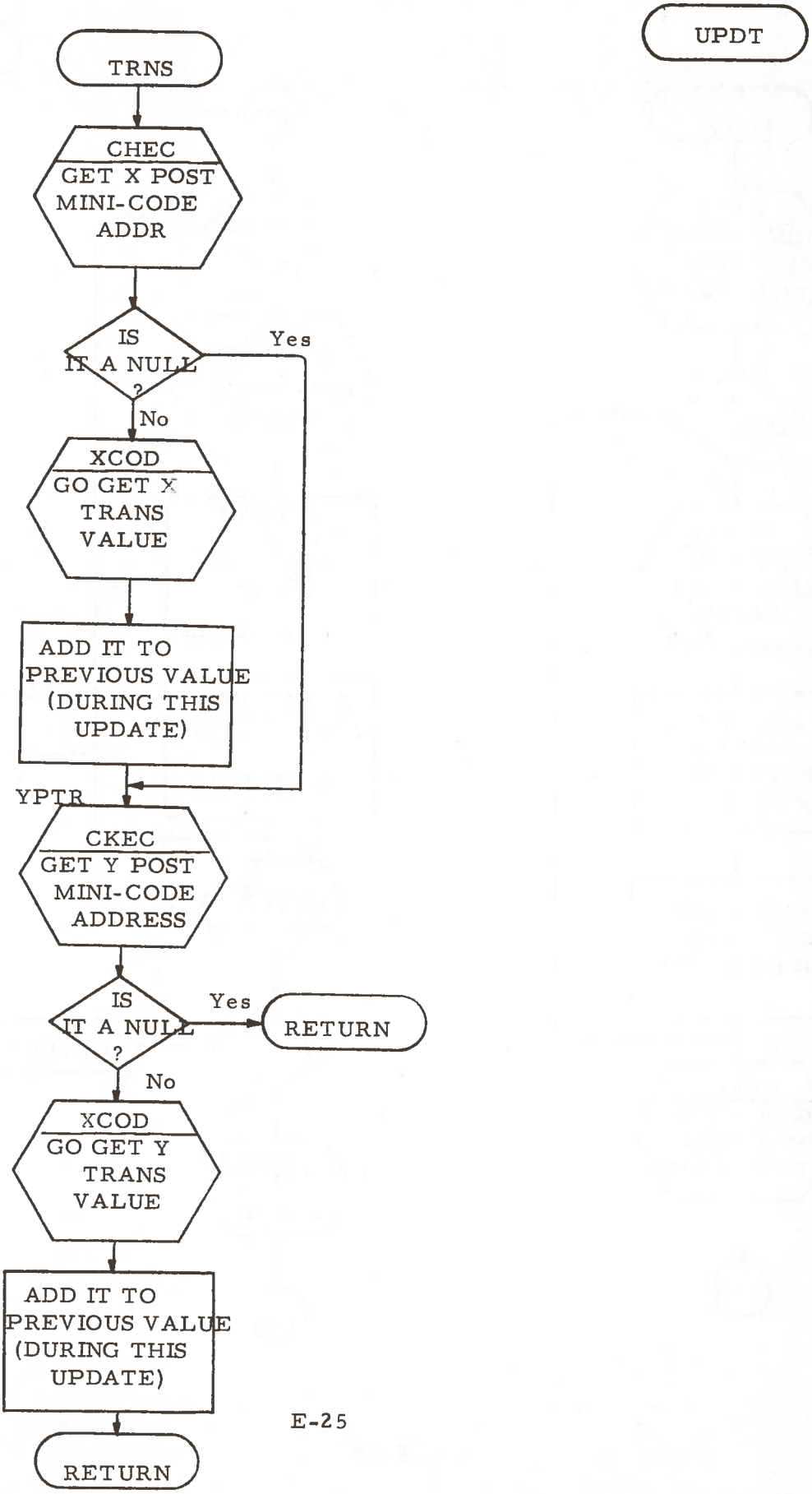

UPDT
BLNK
SET BLINK BIT # 1 IN ENTITY WORD 1Ø
RETURN
UBLK
SET BLINK BIT = Ø IN ENTITY WORD 1Ø
RETURN
TEXT
SET TEXTURE BITS IN ENTITY WORD 1Ø
RETURN
INTS
SET INTENSITY LEVEL IN ENTITY WORD 1Ø
RETURN
ON
SET UNBLANK BIT = 1 IN ENTITY WORD 1Ø
RETURN
OFF
SET UNBLANK BIT = Ø IN ENTITY WORD 1Ø
RETURN

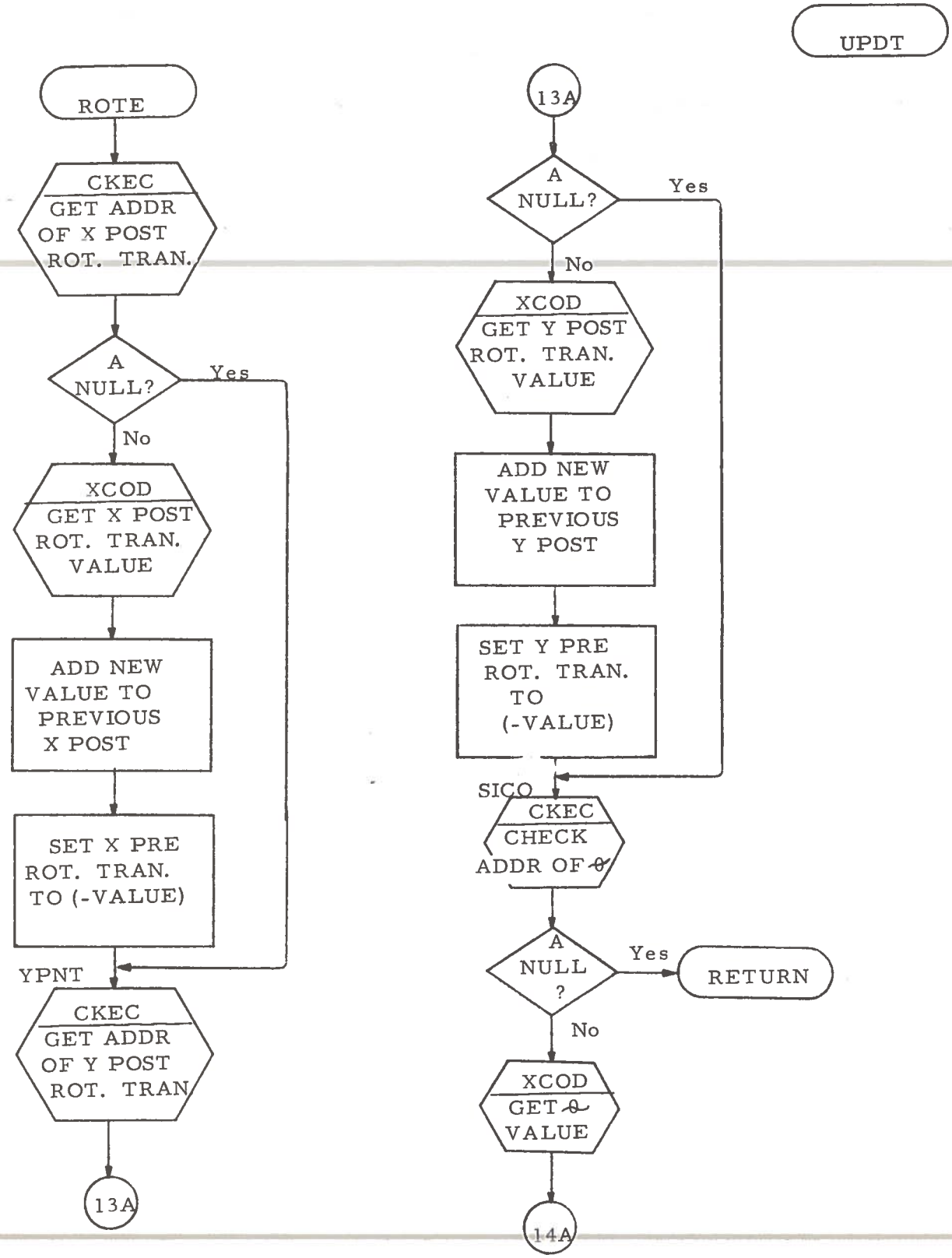

SKAB
SET ALL 3 COMPARE ADDRESSES TO SKIP
INST HAVE = CHK ?
Yes
No
SET = COMPARE LOC TO NOT SKIP
INST HAVE > CHK ?
Yes
No
SET > COMPARE LOC TO NOT SKIP
INST HAVE < CHK ?
Yes
No
SET < COMPARE LOC TO NOT SKIP
9A

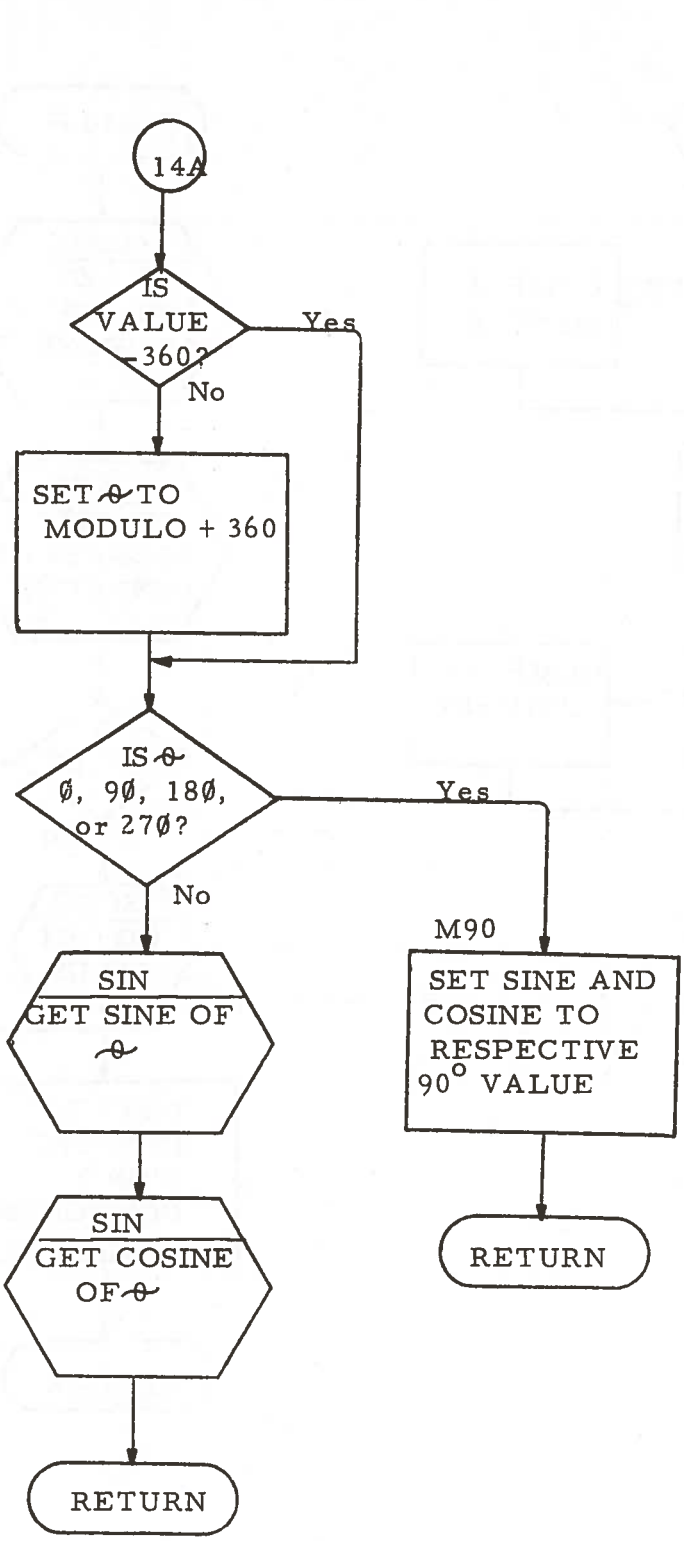

9A
CKEC
GO GET REGISTER NUMBER
CMPR
COMPARE 1st R:2nd R
1st > 2nd
1st = 2nd
1st < 2nd
TEST FOR > OR ≥ ?
Yes
No
SKIP
SKIP N INST.
TEST FOR ≥, =, or ≤ ?
Yes
No
SKIP
SKIP N INST.
TEST FOR < OR ≤ ?
Yes
No
SKIP
SKIP N INST.
SRET
RETURN TO NIXT OR NXT

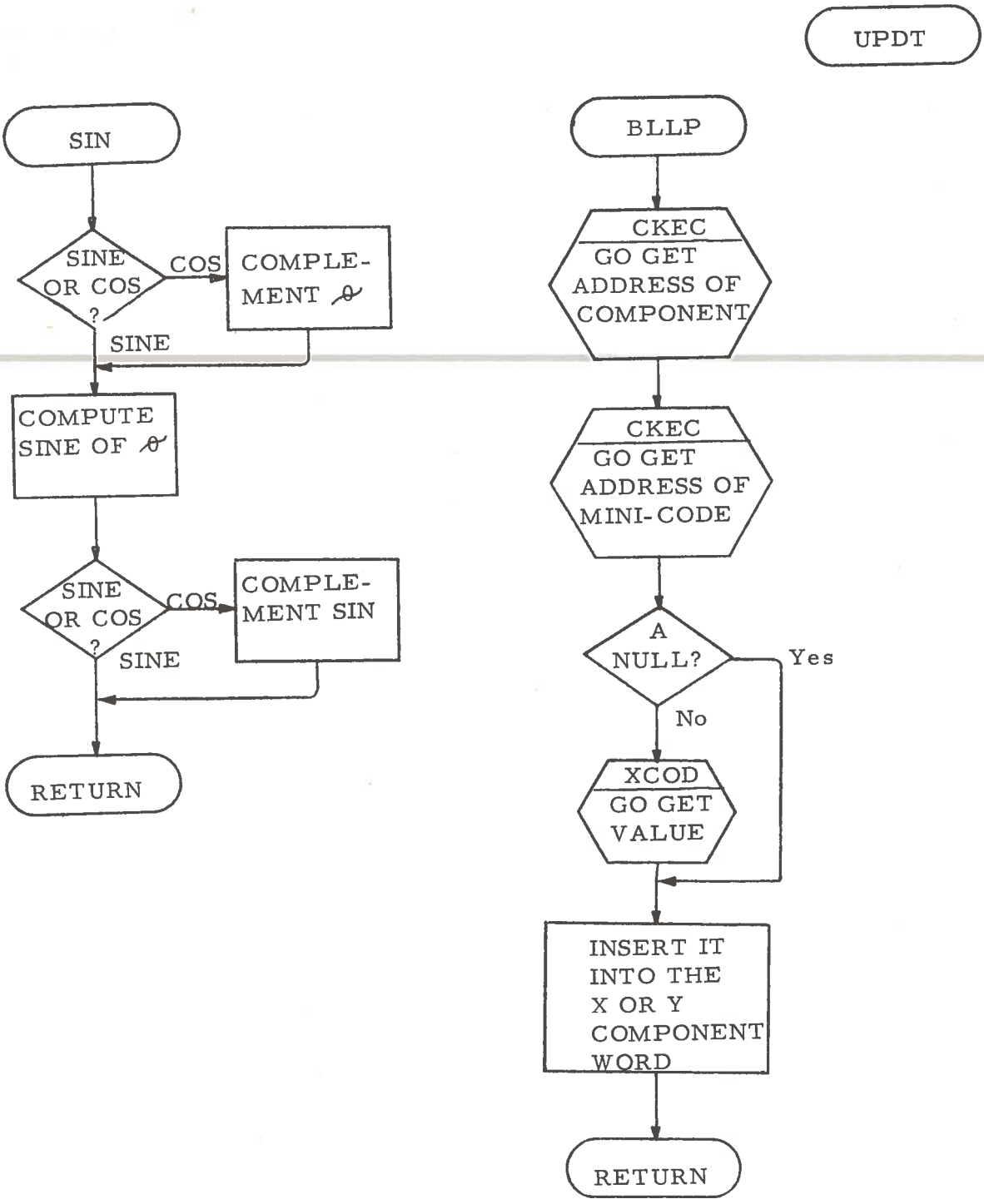

UPDT
SKIP
GET # OF INSTR TO BE SKIPPED
NXWD
INCREMENT CONDITION BLOCK POINTER
NEXT WORD AN INSTR. ?
No
Yes
END CODE ?
No
CORRECT # INSTR. SKIPPED ?
No
Yes
Yes
SAVE IT AS POINTER TO NEXT COND BLK
Yes
FOLLOWING WORD AN ADDR ?
No
INCREMENT CONDITION BLOCK POINTER
RETURN FROM CDTI or CDTE
RETURN TO NIXT or NXT

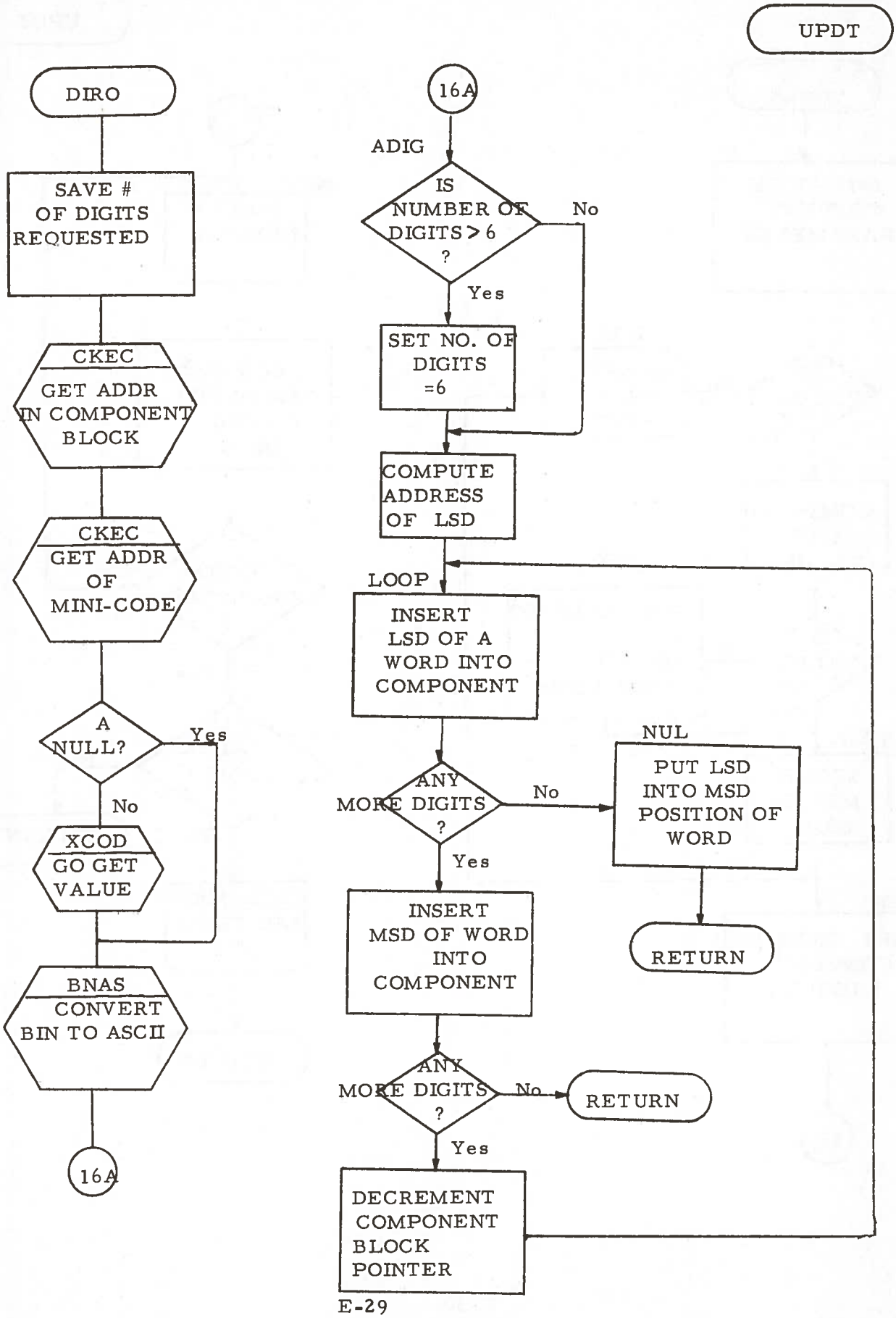

UPDT
CONT
NEXT COND. WD AN ADDR ?
No
RETURN TO CDTE OR CDTI
Yes
SAVE IT AS NEW CONDITION BLOCK POINTER
RETURN TO NIXT OR NXT
NFRM
GET ID OF NEW FRAME
SAVE STARTING FRAME ADDRESS
LOP4
NEW FRAME ?
No
MORE FRAMES ?
Yes
GET NEXT FRAME'S ID
Yes
No
NFRA
SET NEW FRAME INDICATOR
RETURN
RETURN

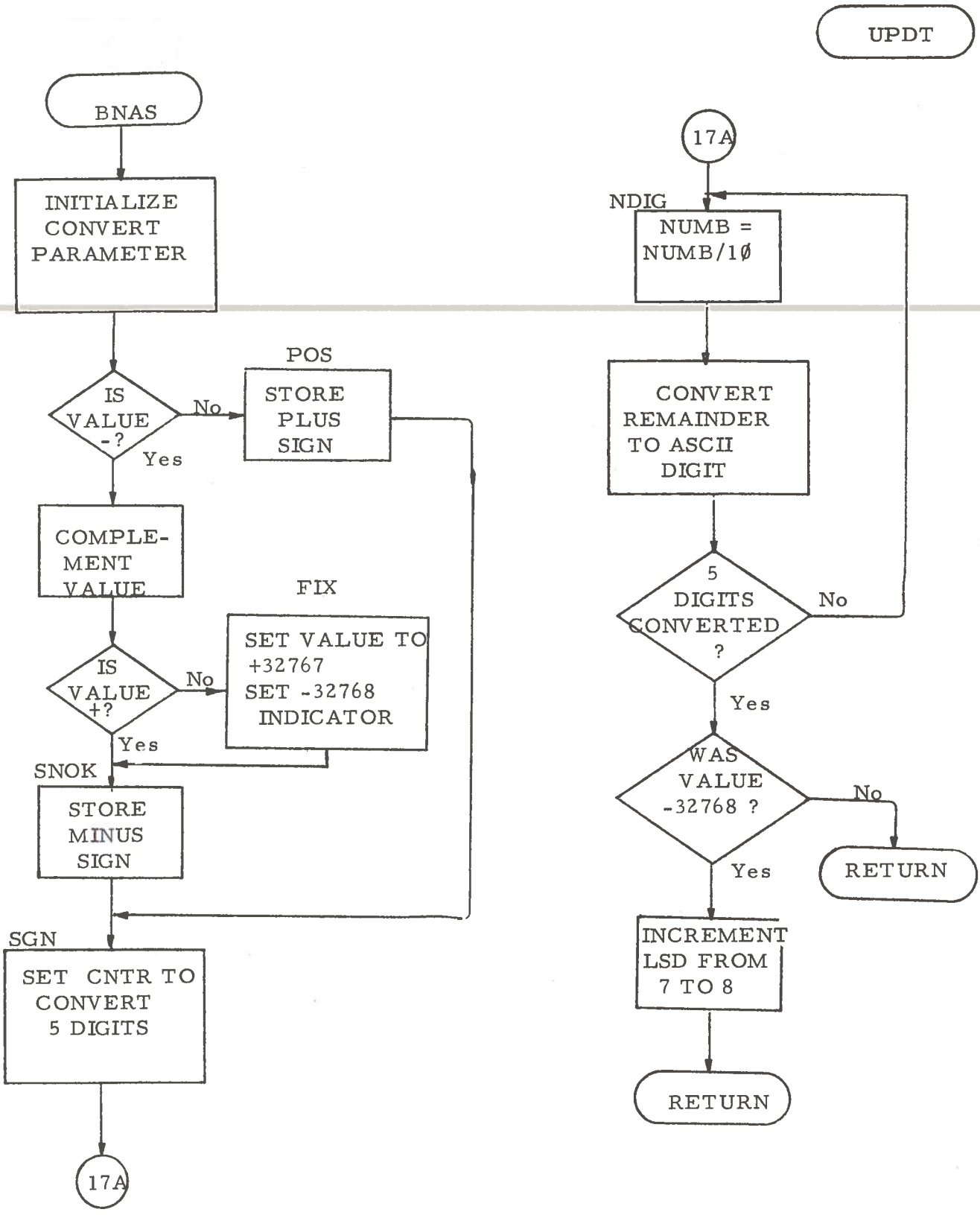

UPDT
TRNS
CHEC
GET X POST MINI-CODE ADDR
IS IT A NULL ?
Yes
No
XCOD
GO GET X TRANS VALUE
ADD IT TO PREVIOUS VALUE (DURING THIS UPDATE)
YPTR
CKEC
GET Y POST MINI-CODE ADDRESS
IS IT A NULL ?
Yes
RETURN
No
XCOD
GO GET Y TRANS VALUE
ADD IT TO PREVIOUS VALUE (DURING THIS UPDATE)
RETURN

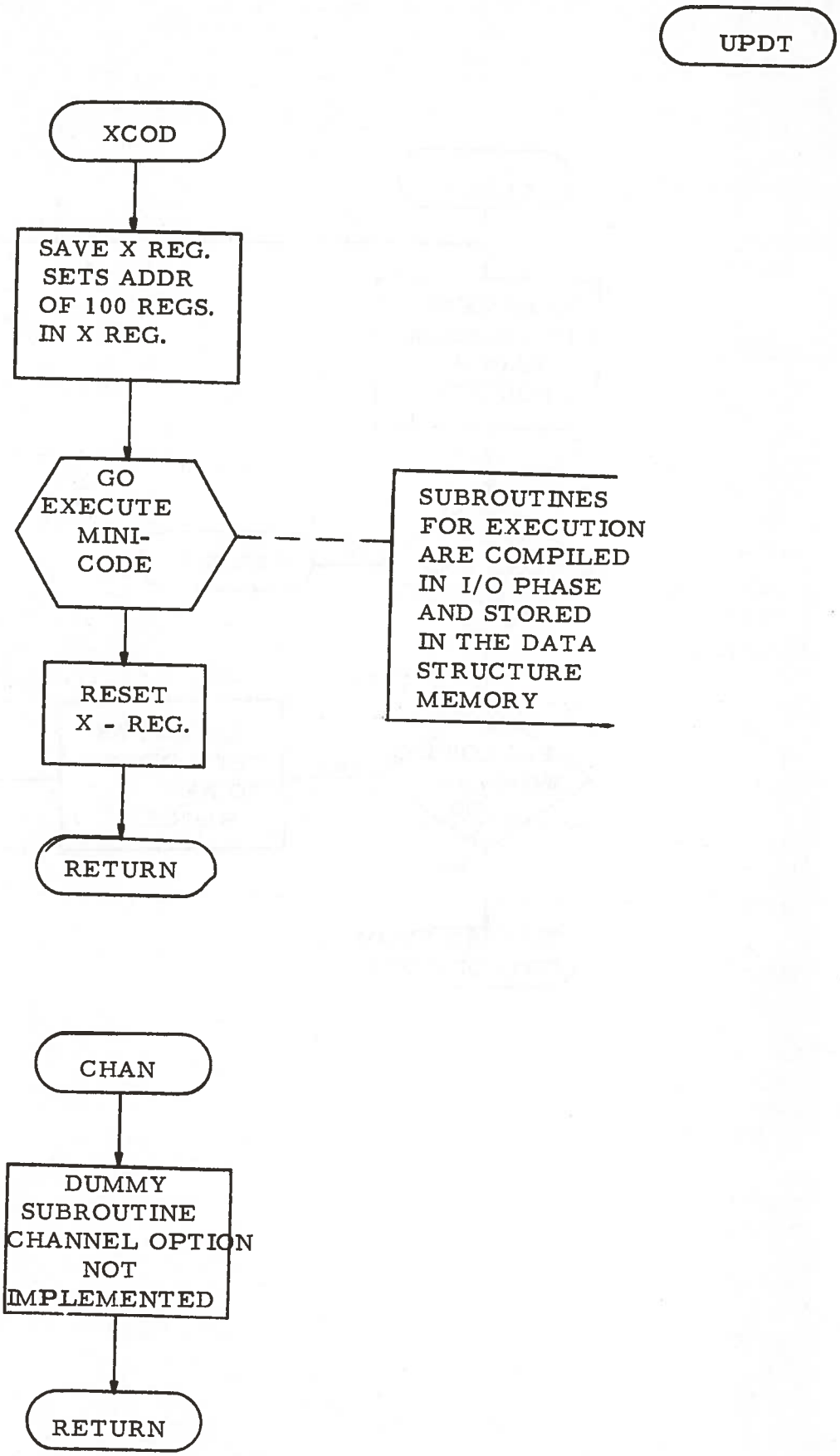

UPDT
ROTE
CKEC
GET ADDR OF X POST ROT. TRAN.
A NULL?
Yes
No
XCOD
GET X POST ROT. TRAN. VALUE
ADD NEW VALUE TO PREVIOUS X POST
SET X PRE ROT. TRAN. TO (-VALUE)
YPNT
CKEC
GET ADDR OF Y POST ROT. TRAN
13A
13A
A NULL?
Yes
No
XCOD
GET Y POST ROT. TRAN. VALUE
ADD NEW VALUE TO PREVIOUS Y POST
SET Y PRE ROT. TRAN. TO (-VALUE)
SICO
CKEC
CHECK ADDR OF 0
A NULL ?
Yes
RETURN
No
XCOD
GET 0 VALUE
14A

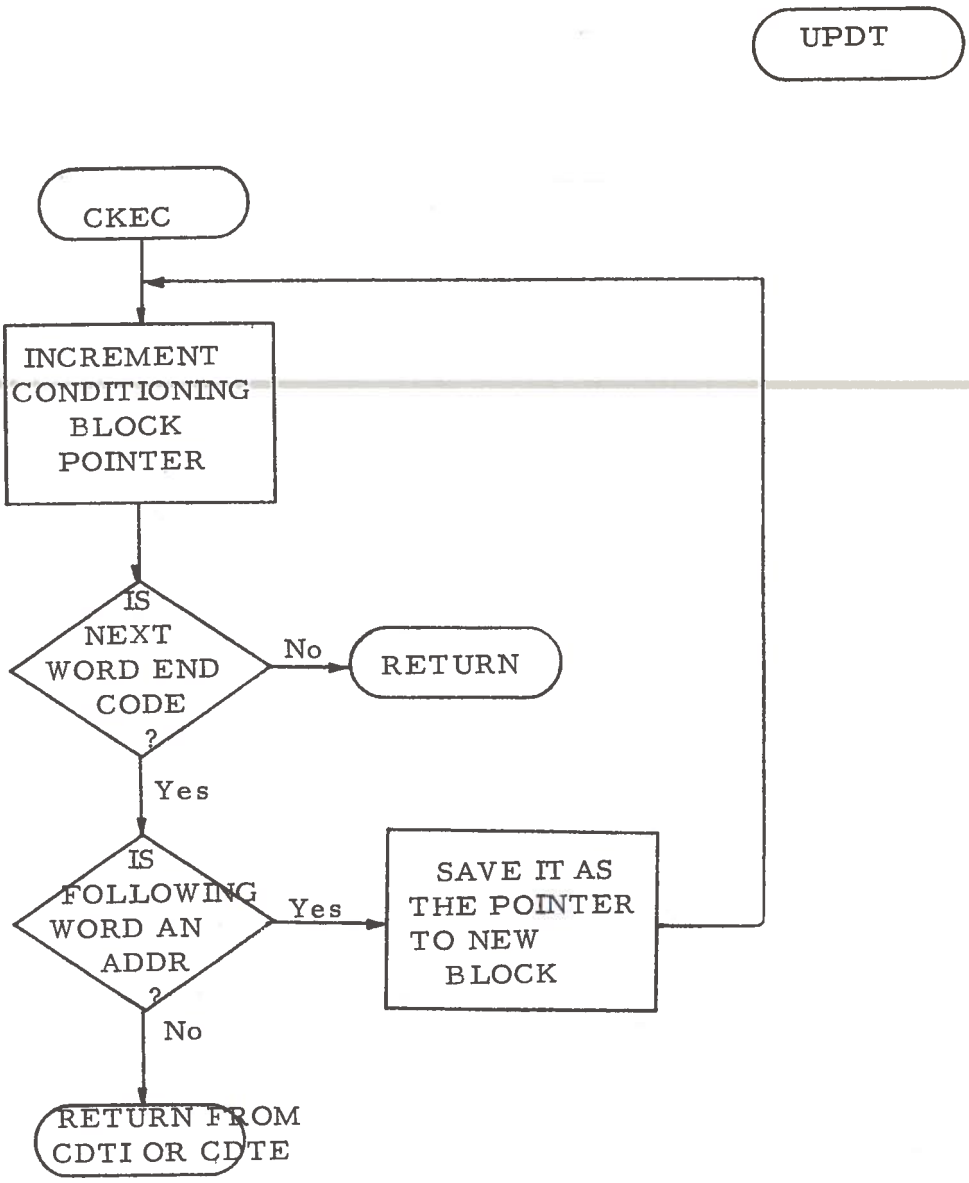

UPDT
14A
IS VALUE −360?
Yes
No
SET θ TO MODULO + 360
IS θ Ø, 9Ø, 18Ø, or 27Ø?
Yes
No
M90
SET SINE AND COSINE TO RESPECTIVE 90° VALUE
SIN
GET SINE OF θ
SIN
GET COSINE OF θ
RETURN
RETURN


UPDT
SIN
SINE OR COS ?
COS
COMPLE-MENT θ
SINE
COMPUTE SINE OF θ
SINE OR COS ?
COS
COMPLE-MENT SIN
SINE
RETURN
BLLP
CKEC
GO GET ADDRESS OF COMPONENT
CKEC
GO GET ADDRESS OF MINI-CODE
A NULL?
Yes
No
XCOD
GO GET VALUE
INSERT IT INTO THE X OR Y COMPONENT WORD
RETURN


UPDT
DIRO
SAVE # OF DIGITS REQUESTED
CKEC
GET ADDR IN COMPONENT BLOCK
CKEC
GET ADDR OF MINI-CODE
A NULL?
Yes
No
XCOD
GO GET VALUE
BNAS
CONVERT BIN TO ASCII
16A
16A
ADIG
IS NUMBER OF DIGITS > 6 ?
No
Yes
SET NO. OF DIGITS =6
COMPUTE ADDRESS OF LSD
LOOP
INSERT LSD OF A WORD INTO COMPONENT
ANY MORE DIGITS ?
No
NUL
PUT LSD INTO MSD POSITION OF WORD
RETURN
Yes
INSERT MSD OF WORD INTO COMPONENT
ANY MORE DIGITS ?
No
RETURN
Yes
DECREMENT COMPONENT BLOCK POINTER


UPDT
BNAS
INITIALIZE CONVERT PARAMETER
IS VALUE -?
No
Yes
POS
STORE PLUS SIGN
COMPLE-MENT VALUE
IS VALUE +?
No
Yes
FIX
SET VALUE TO +32767 SET -32768 INDICATOR
SNOK
STORE MINUS SIGN
SGN
SET CNTR TO CONVERT 5 DIGITS
17A
17A
NDIG
NUMB = NUMB/1Ø
CONVERT REMAINDER TO ASCII DIGIT
5 DIGITS CONVERTED ?
No
Yes
WAS VALUE -32768 ?
No
Yes
RETURN
INCREMENT LSD FROM 7 TO 8
RETURN


UPDT
XCOD
SAVE X REG.
SETS ADDR
OF 100 REGS.
IN X REG.
GO
EXECUTE
MINI-
CODE
SUBROUTINES
FOR EXECUTION
ARE COMPILED
IN I/O PHASE
AND STORED
IN THE DATA
STRUCTURE
MEMORY
RESET
X - REG.
RETURN
CHAN
DUMMY
SUBROUTINE
CHANNEL OPTION
NOT
IMPLEMENTED
RETURN


UPDT
CKEC
INCREMENT CONDITIONING BLOCK POINTER
IS NEXT WORD END CODE ?
No
RETURN
Yes
IS FOLLOWING WORD AN ADDR ?
Yes
SAVE IT AS THE POINTER TO NEW BLOCK
No
RETURN FROM CDTI OR CDTE