DECUS
PROGRAM LIBRARY

DECUS NO.	11-307
TITLE	STAGE 2 For The PDP-11 Operating Under RT-11
AUTHOR	W. M. Waite University of Colorado
COMPANY	Australian National University
DATE	21 September 1976
SOURCE LANGUAGE	MACRO 11

GENERAL INFORMATION

Object Computer(s) PDP-11 Source Computer (if different) _____

File Name _____ Version No. _____

Title STAGE 2 For The PDP-11 Operating Under RT-11

Author W.M. Waite University of Colorado

Submitter (if other than author) DOS-11 Implementation by Peter H. Heinrich/ Modified for RT-11 by D.M. Nessett

Affiliation Australian National University

Address PO Box 4, Canberra. Act. Australia

Country _____

Monitor/Operating System RT-11/V2B DEC No. _____

Core Storage Required _____ Starting Address _____

Peripherals Required _____

Other Software Required _____ DEC or DECUS No. _____

Source Language MACRO 11 Category Programming System

Restrictions, Deficiencies, Problems STAGE2/RT11 will not run in less than 12K and is not really useful unless there is 16K (assuming the S/J Monitor) The STAGE2 Command "Change I/O channels and copy text" is modified.

Date of Planned or Possible Future Revisions _____

TAPES AVAILABLE

Paper Tapes Object Binary ☐ Object ASCII ☒ Source ☒ Other _____

DECtape ☐ LINCtape ☐ Format _____ Magtape: 7 Track ☐ 9 Track ☐ BPI _____

Object Files ☐ Source Files ☐ Documentation Files ☐ Other _____

ABSTRACT

This implementation of STAGE2 is a modification of the DOS-11 version (DECUS No. 11-158) by Peter H. Heinrich so that it will run under rt-11

TITLE: STAGE 2 FOR PDP-11 (RT-11)

AUTHOR: WILLIAM M. WAITE, UNIVERSITY OF COLORADO.
DOS-11 IMPLEMENTATION: PETER H. HEINRICH (INST. F. BIOKYBERNETIK AND BIOMED. TECHNIK UNIVERSITAET KARLSRUHE, W.-GERMANY) MODIFICATIONS FOR RT-11: D.M. NESSETT (AUSTRALIAN NATIONAL UNIVERSITY CANBERRA, A.C.T., AUSTRALIA)

DATE: SEPT. - 1976

SOURCE: MACRO-11

1. INTRODUCTION

This document describes an implementation of Stage 2 (William M. Waite, "Implementing Software for Non-Numeric Applications", Prentice-Hall N.J., 1973) for PDP-11 Computer Systems operating under RT-11. Stage 2 is a general purpose macro processor designed to port software written for abstract machines. The macro processor is itself protable being written for an abstract machine called "FLUB"

2. PROGRAMS

The dectape contains the following modules:

STG2.FLB	FLUB-CODE OF STAGE 2	(W.M. WAITE)
FLUB.ST2	FLUB MACRO DEFINITIONS	(P.H. HEINRICH)
OPT.ST2	OPTIMIZATION MACRO DEFINITIONS	(P.H. HEINRICH)
ST2CMD.MAC	COMMAND INPUT MODULE	(D.M. NESSETT)
ST2RDN.MAC	INPUT SUBROUTINES MODULE	(D.M. NESSETT)
ST2WTN.MAC	OUTPUT SUBROUTINES MODULE	(D.M. NESSETT)
ST2MSC.MAC	MISC. SUBROUTINES MODULE	(D.M. NESSETT)
ST2DAT.MAC	DATA SECTION MODULE	(D.M. NESSETT)
ST2DFN.MAC	DEFINITIONS MODULE	(D.M. NESSETT)
BOOT.MAC	STAGE 2 IN EXPANDED FORM	
BOOT.OBJ	STAGE 2 OBJECT MODULE	
ST2BLD.BAT	BATCH STREAM FOR BUILDING STAGE 2	(D.M. NESSETT)

3. IMPLEMENTATIONS

This implementation of Stage 2 was achieved by modifying Peter Heinrich's DOS-11 implementation so that it could operate under RT-11. Heinrich's abstract machine implementation did not require any modification to run under RT-11, but the supporting subroutines (to handle I/O and command dialog) were completely re-written. This involved the development of record I/O subroutines, since all I/O in RT-11 is physical I/O. The detailed operation of these routines is discussed below.

4. PREREQUISITES

RT-11/V2. If processor is not a PDP-11/40 or 45, EAE must exist.

5. RESTRICTIONS

Since all Input channels are treated as if they were concentrated into a single file, the Stage 2 command "change I/O channels and Copy Text" cannot be used to change the input channel number.

6. INSTALLATION

The installation of Stage 2/RT-11 is accomplished by running the batch stream "ST2BLD.BAT". This batch stream must first be transferred to the system device by mounting the distribution dectape onto DTØ and executing:

```
. R PIP
* SY:ST2BLD.BAT = DT:ST2BLD.BAT
*<CNTL C>
.
```

The batch monitor must then be linked into RT-11 (see RT-11 System Reference Manual (DEC-11-ORUGA-C-D) Chapter 12) and the batch stream executed by typing:

```
. R BATCH
* ST2BLD
```

If the RT-11 installation cannot run batch for some reason (e.g. the PDP-11 has less than 12K of core). The file ST2BLD.BAT should be listed and the commands which are contained therein followed manually.

7. STAGE 2 VØ3-3/RT-11

Stage 2 uses all core available. The I/O is double buffered (at the block level) and the error messages are those standard for Stage 2. Input to Stage 2/RT-11 may consist of from one to three output files and from one to six input files. Input files are treated as though they have been concatenated into one file in the same order as they appear on the command line. Output files are available on Stage 2 Channels 3, 4, and 5.

For example:

A. The following defines one output file (for Stage 2 Channel 3) and three input files (which will be read by Stage 2 as if they had been concatenated together).

```
* OUTPUT.MAC = MACRO.DFN, TEXT1.ST2, TEXT2.ST2
```

B. The following defines two output files (on Stage 2 Channels 3 and 5).

```
* OUTPUT.MAC,, AUX.LST = MACRO.DFN, TEXT1.ST2, TEXT2.ST2
```

C. The following uses the default extension capability of Stage 2 (output file 1 = MAC, output file 2 = LOG, output file 3 = LST, all input files = ST2).

```
* OUTPUT, LISTNG, ERRORS = MACRO, TEXT1, TEXT2
```

This is equivalent to:

```
* OUTPUT. MAC, LISTNG.LOG, ERRORS.LST = MACRO.ST2,
  TEXT1.ST2, TEXT2.ST2
```

8. IMPLEMENTATION GUIDE FOR STAGE 2/RT-11

A. INTRODUCTION

This document contains a written description of the RT-11 implementation of Stage 2 (flowcharts are to be found in Section 9). Since the RT-11 implementation is a modification of Peter H. Heinrich's DOS-11 implementation (DECUS NO. 11-158), some

of the information contained herein describes Heinrich's design of the FLUB abstract machine for the PDP-11. In order that credit is given where credit is due, the FLUB implementation will be briefly discussed in Section B and the RT-11 dependant sections of Stage 2/RT-11 will be covered in Section C.

B. PDP-11 FLUB IMPLEMENTATION

The material in this section briefly describes the FLUB implementation designed by Peter H. Heinrich for PDP-11 Systems. It is assumed that the reader is thoroughly familiar with the architecture of FLUB and with Stage 2's implementation on the FLUB abstract machine. Those unfamiliar with these should read the relevant sections of William M. Waite's book ("Implementing Software for Non-Numeric Applications", Prentice-Hall, N.J., 1973.) before proceeding.

The major design decision that must be made when implementing FLUB is how to represent FLUB words. Each FLUB word consists of three fields:

1) The VAL field;
2) The PTR field;

and

3) The FLG field.

Heinrich chose to represent these fields as bytes for the FLAG and VAL fields and as a 16-bit word for the PTR field. Since operations upon FLUB words cannot be executed directly but must first be loaded into a FLUB "register", the representation of these registers also needs to be described. Each register field corresponds to one PDP-11 16-bit word. Each register field is not contiguous with the other two

fields of the register, but is contained in an array of fields of its own type. Thus FLG.7 (the FLG field for register 7) is contiguous with FLG.8 not with VAL.7 (this is in keeping with Waite's suggested implementation technique for FLUB's registers). The absolute physical address where FLUB memory begins is kept in R3 (which is called "BASE" in the assembly code). R4 and R5 are used to hold respectively the line buffer read pointer (LBR) and the line buffer write pointer (LBW).

C. RT-11 DEPENDENT SECTIONS OF STAGE 2/RT-11

Anyone examining the macro definitions for State 2/RT-11 will notice that there are subroutine calls imbedded into five of the macro bodies. These five subroutines and their uses are:

1) IWRCH - Write a character to the line buffer;
2) CLOSE - Close a Stage 2 input channel;
3) READ - Read the next line buffer from the input stream;
4) WRITE - Write the current line buffer to an output channel; and
5) ERRMSG - Output an error message.

The first two of these are fairly simple and are functionally identical to the routines provided in Heinrich's DOS-11 implementation. 1WRCH simply performs a "MOVB" to insert the given character into the line buffer. CLOSE does nothing useful, since in the RT-11 version of Stage 2 (as in the DOS-11 version) the capability of rewinding input channels is not provided (CLOSE appears only in the macro definition of the FLUB command "REWIND"). The routine ERRMSG is also fairly simple and its operation should be

obvious from its listing. The two remaining subroutines - READ and WRITE - are not as simple, however, and will be discussed in detail.

The read software module is composed of four sections:

1) The main subroutine (READ);

2) The next line subroutine (NXTLNE);

3) The next block subroutine (NXTBLK);

and

4) The next file subroutine (NXTFIL).

The main routine - READ - checks the input channel number. If this is zero, an automatic EOF is returned. If the channel number is equal to one, NXTLNE is called to get the next line from the current input block. If the channel number is neither zero nor one, an error code of two is returned to the caller.

NXTLNE tries to obtain a new line buffer from the current input block. If this is possible, NXTLNE returns with the C bit clear to indicate success. If this is not possible, NXTBLK is called to return the next block and initiate the transfer of the following block from the input device. NXTLNE determines it has a complete line when it transfers a carriage return (line feeds are ignored - not transferred to line buffer - by NXTLNE). The operation of NXTLNE is dependent on a number of pointers into various character arrays:

(i) ILNEPT - This pointer keeps track of the position in the line buffer where the next character from the current block is to be inserted.

(ii) IBLKPT - This pointer is used to keep track of where in the current block the next line begins.

(iii) ICBKEN - This location contains the address of the end of the current block.

(iv) IENDBF - The address of the end of the line buffer.

As was said previously, NXTBLK returns the next block in the input stream. Ideally this block has already been transferred from the current input device. If that transfer has not yet finished, NXTBLK waits for its completion. When the block becomes available, two pointers (FILBUF and ICRBUF) are exchanged to indicate which buffer is being used and which is being filled. IBLKPT is then loaded to contain the first character address of the current block and ICBKEN is updated. After these operations have been accomplished, NXTBLK attempts to read the next block from the input file. If there are no more blocks in this file, NXTFIL is called in an attempt to locate another input channel. Otherwise, NXTBLK returns to its caller with the C bit set to indicate success.

The first time it is called, NXTFIL sequences through the input channel numbers 3-8 in an attempt to find a file which was opened by command dialog (see below). If this search is successful, NXTFIL updates the variable "INPCHN" so that it contains the current input channel number. Each subsequent call to NXTFIL begins searching from this number

for another defined input channel. If this search is successful, "INPCHN" is updated and the routine returns to its caller. If the search is unsuccessful, bit Ø in "FLBFLG" is set and NXTFIL returns control to its caller.

The write module is composed of three routines:

1) The main subroutine (WRITE);

2) The next line subroutine (NXTLNE);

and

3) The next block subroutine (NXTBLK).

"WRITE" first tests the Stage 2 channel number to see if it is zero. If so, the routine returns immediately to its caller after indicating a successful write. If the Stage 2 channel number is non-zero, it is decremented by 3 to transform the Stage 2 channel numbers 3, 4, and 5 into their corresponding RT-11 numbers 0, 1, and 2. If the transformed channel number is equal to neither 0, 1, nor 2, "WRITE" returns to its caller with an error condition code. If the channel number is in the proper range, the minus one used by FLUB to indicate end-of-line is replaced by a carriage return/line feed and the subroutine NXTLNE is called.

NXTLNE first transforms the output channel number into an index by shifting it one left. This index is used when accessing the output pointers OLNEPT, OBLKPT and OLBKEN. This is necessary since simultaneous output to different channels is allowed in Stage 2. NXTLNE begins moving characters into the output block currently being constructed for the specified channel. If the line fits into this block,

NXTLNE returns indicating success. If the line does not fit into the current block, NXTLNE places as much of the line as will fit into the current block and then calls NXTBLK. After NXTBLK returns, OBLKPT points to the first character position in the next block to be constructed. NXTLNE places the remainder of the line to be written into this new block and returns to its caller.

NXTBLK tests to see if the last block write operation has completed. If it hasn't, NXTBLK waits for this to happen. It then rotates the two block buffers (by exchanging the pointers DRNBUF and OCRBUF) and updates pointers OBLKPT (which keeps track of where the next character is to be written into the current block) and OCBKEN (which holds the address of the current block's end). A .WRITE is then initiated on the block passed to NXTBLK by NXTLNE and after updating the block number for the channel, control passes back to NXTLNE.

D. INITIATING A STAGE 2 RUN - COMMAND DIALOG

Since Stage 2 runs under RT-11, some means must exist for executing the routine and for setting up the files which Stage 2 will manipulate. Initiating Stage 2 execution is accomplished simply by the RT-11 command:

```
.R STAGE2
```

Input and output files are specified by an RT-11 command string. Interpretation of this string and I/O setup and shut-down is the responsibility of the command DIALOG module.

The command dialog module is composed of three routines:

1) The command dialog input routine - ST2CMD;

2) The prime input channel subroutine - PRMCHN;

and

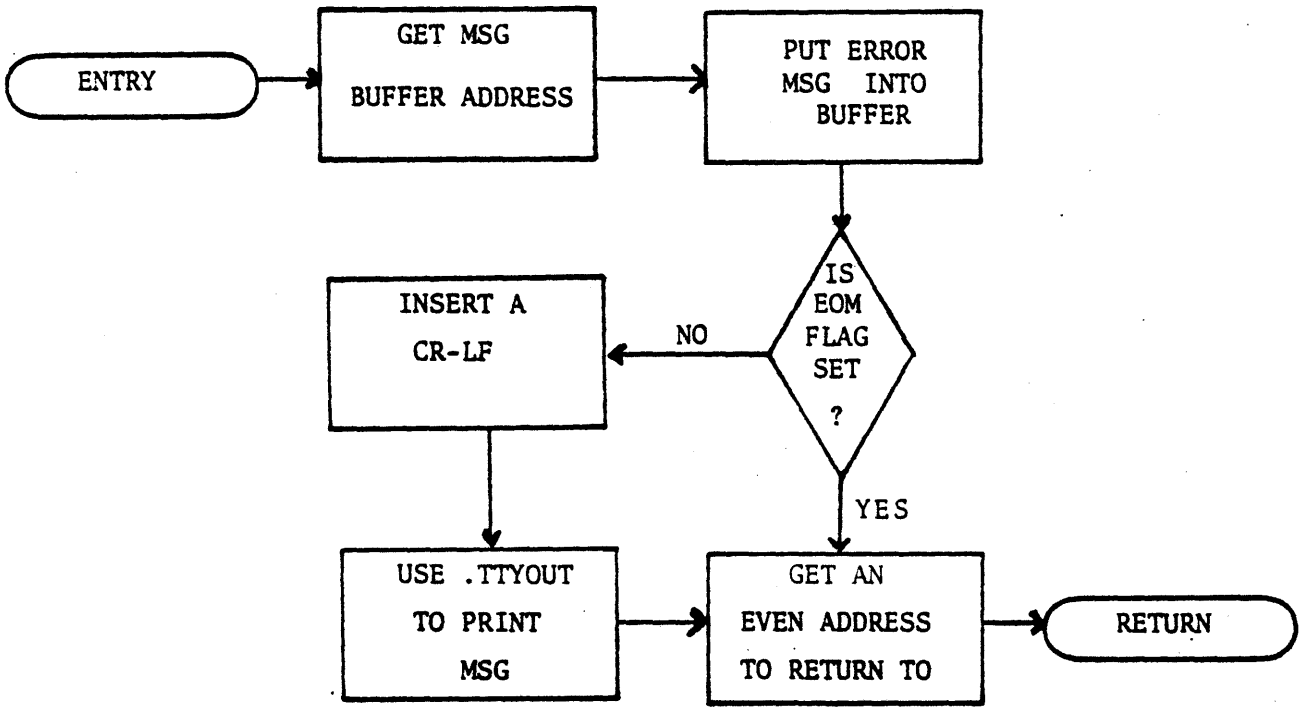
3) The drain output channels subroutine - DRNCHN.

ST2CMD begins by reserving all available memory for Stage 2 by executing a .SETTOP. It saves the stack position, outputs the Stage 2 title, calls the subroutine INIT to initialize some pointers, and then calls .CSIGEN to input and interpret a command string from the console. After the command string has been processed (see description of .CSIGEN in RT-11 System Reference Manual), the stack is restored, FLUB's pointers are initialized, and the routine PRMCHN is called. After control returns to ST2CMD, the Stage 2 macro processor is called to process the files specified in the command input string. When this is complete, ST2CMD calls DRNCHN and then loops to accept a new command string.

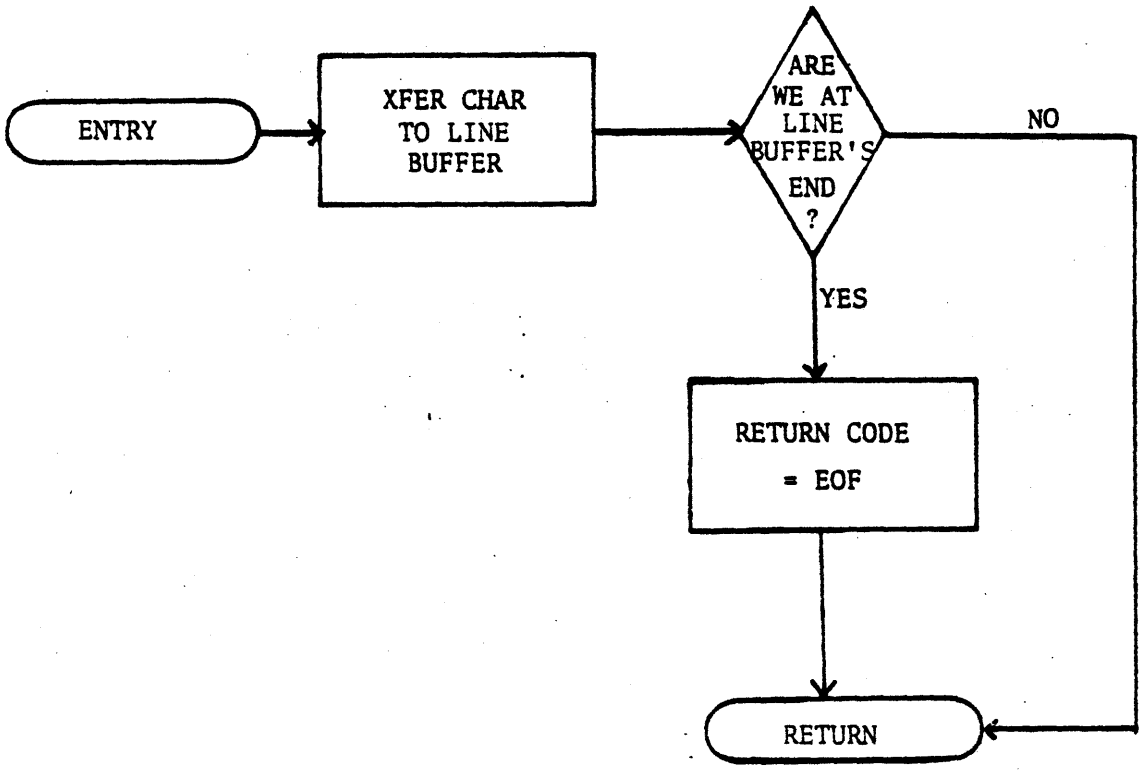
The routines PRMCHN and DRNCHN are used basically for I/O startup and shutdown. PRMCHN calls FSTFIL (which is an equivalent entry point of NXTFIL) to set up the first input file, twice calls RDBLK (which is an equivalent entry point of READ NXTBLK) to set up the first two read block buffers, and returns. DRNCHN writes out the last blocks of the three output channels.

9. FLOW CHARTS

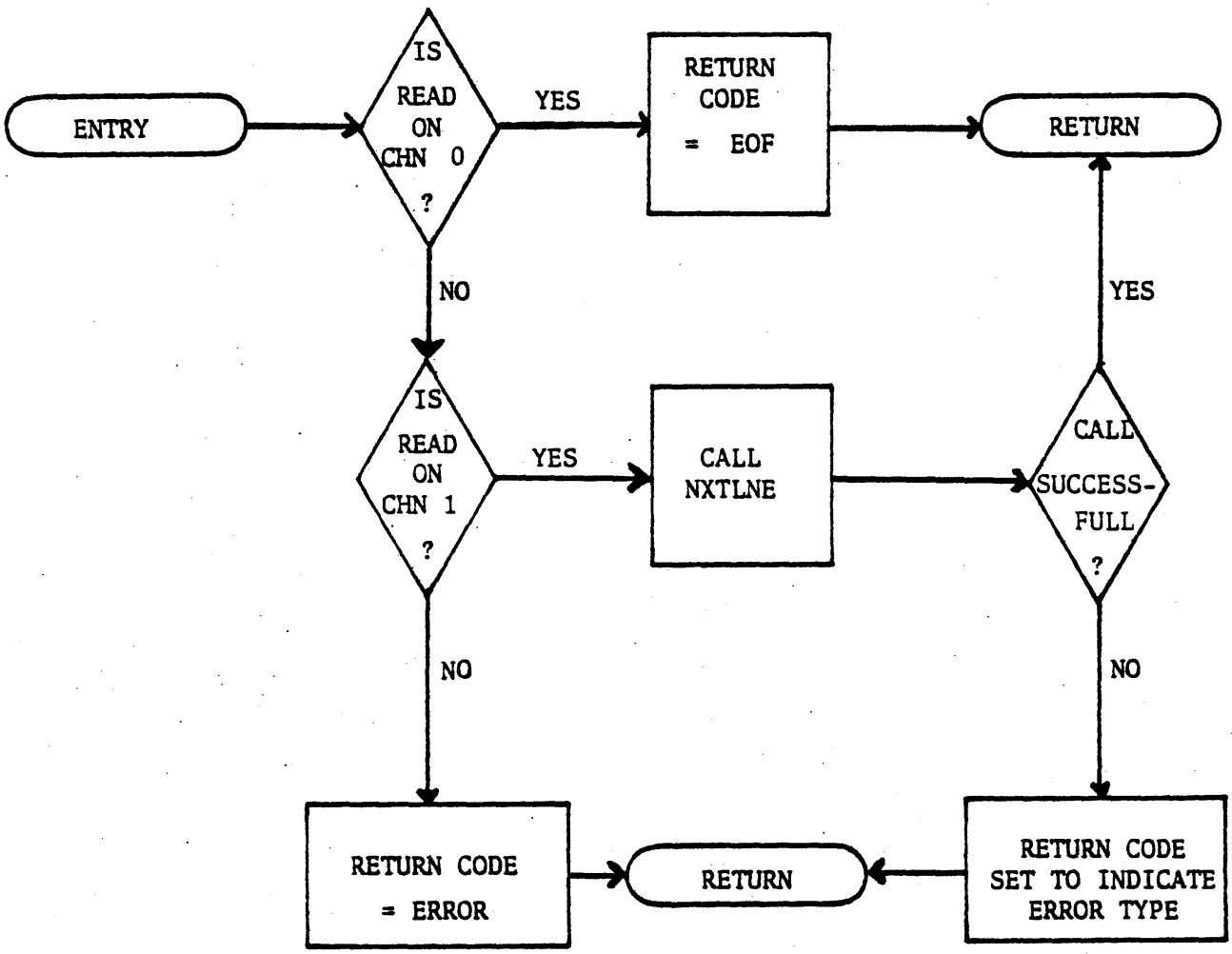
A. ERRMSG



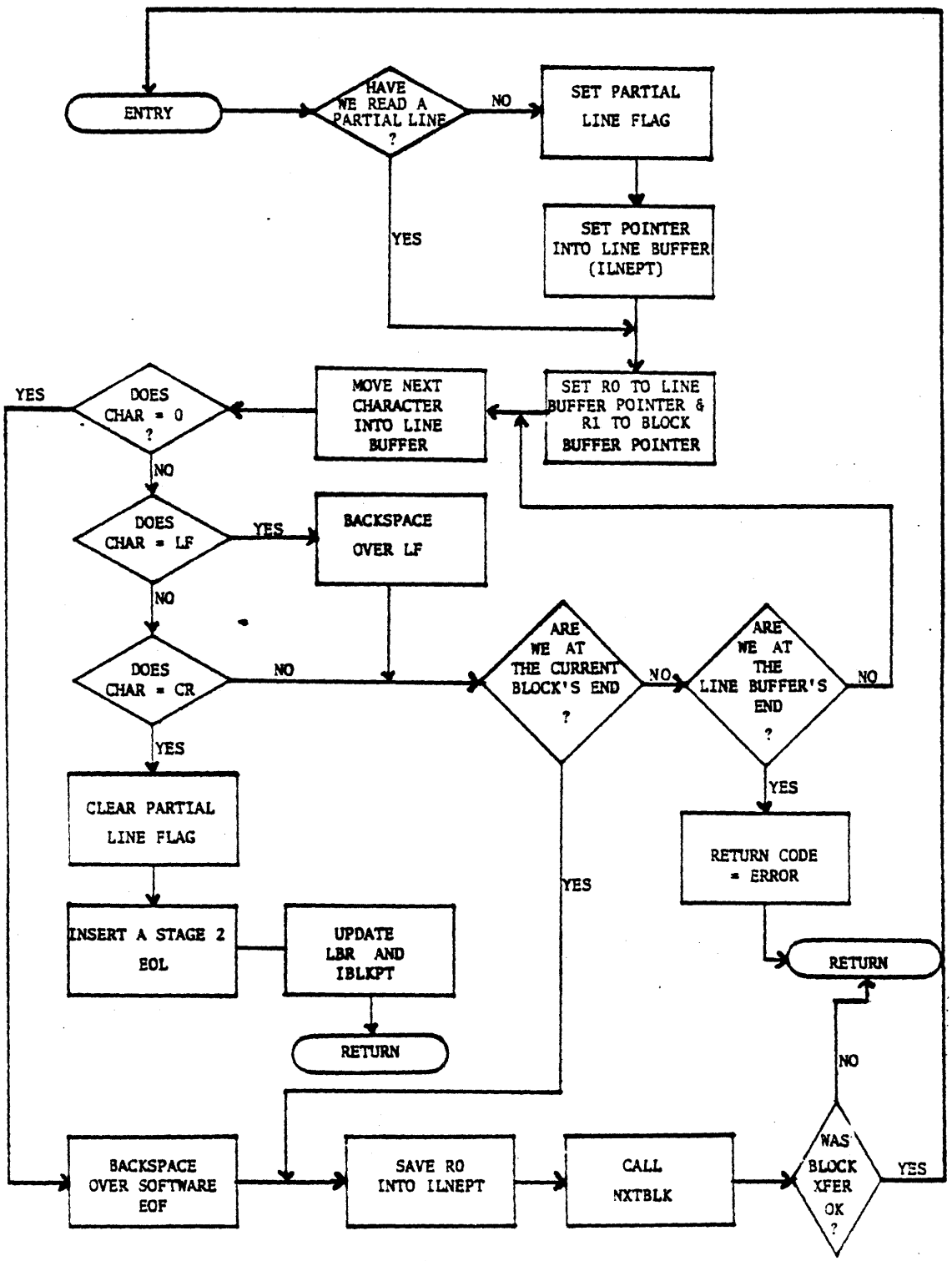
B. IWRCH



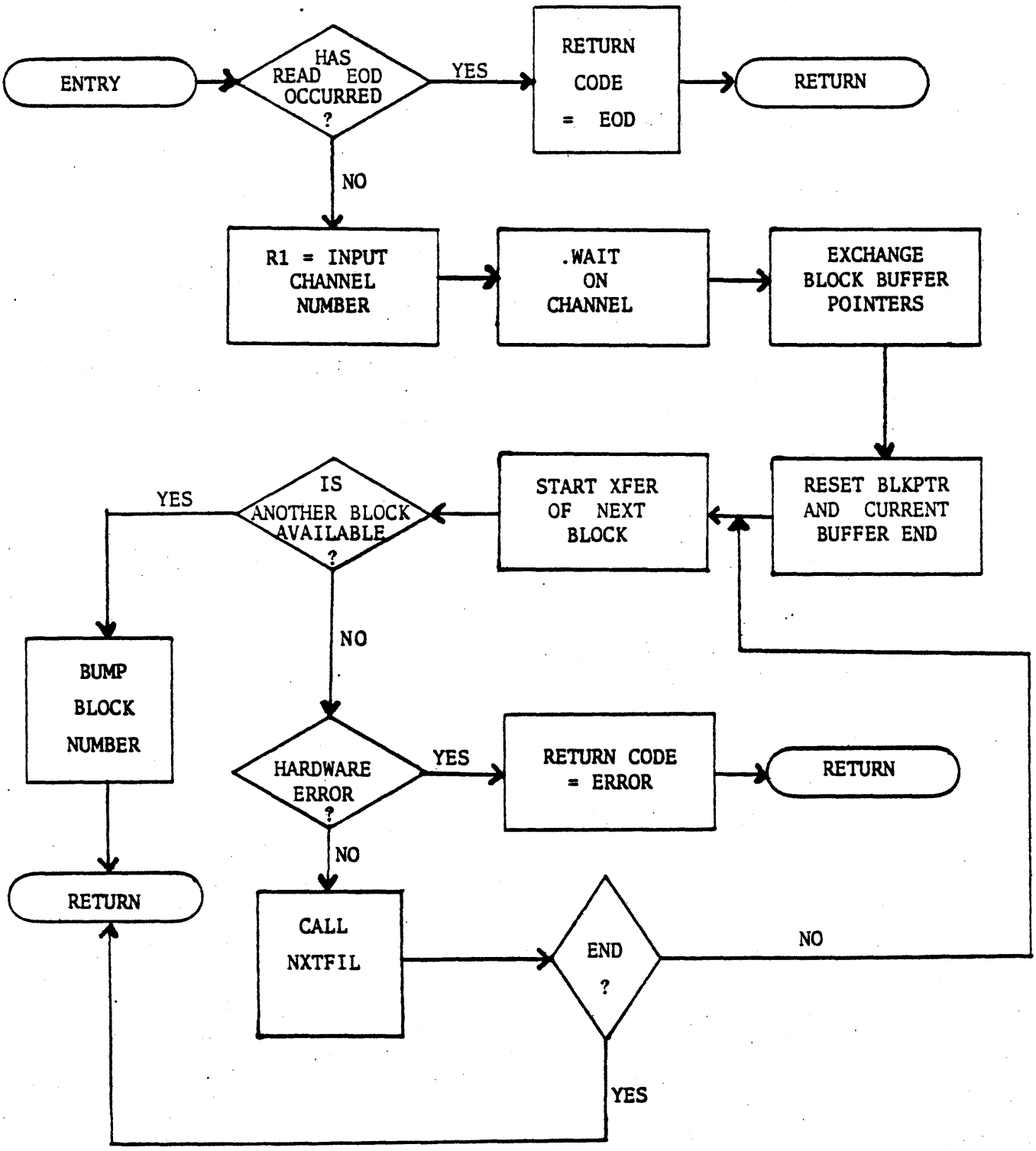
C. (i) READ


ENTRY
IS READ ON CHN 0 ?
YES
RETURN CODE = EOF
RETURN
NO
YES
IS READ ON CHN 1 ?
YES
CALL NXTLNE
CALL SUCCESS-FULL ?
NO
NO
RETURN CODE = ERROR
RETURN
RETURN CODE SET TO INDICATE ERROR TYPE

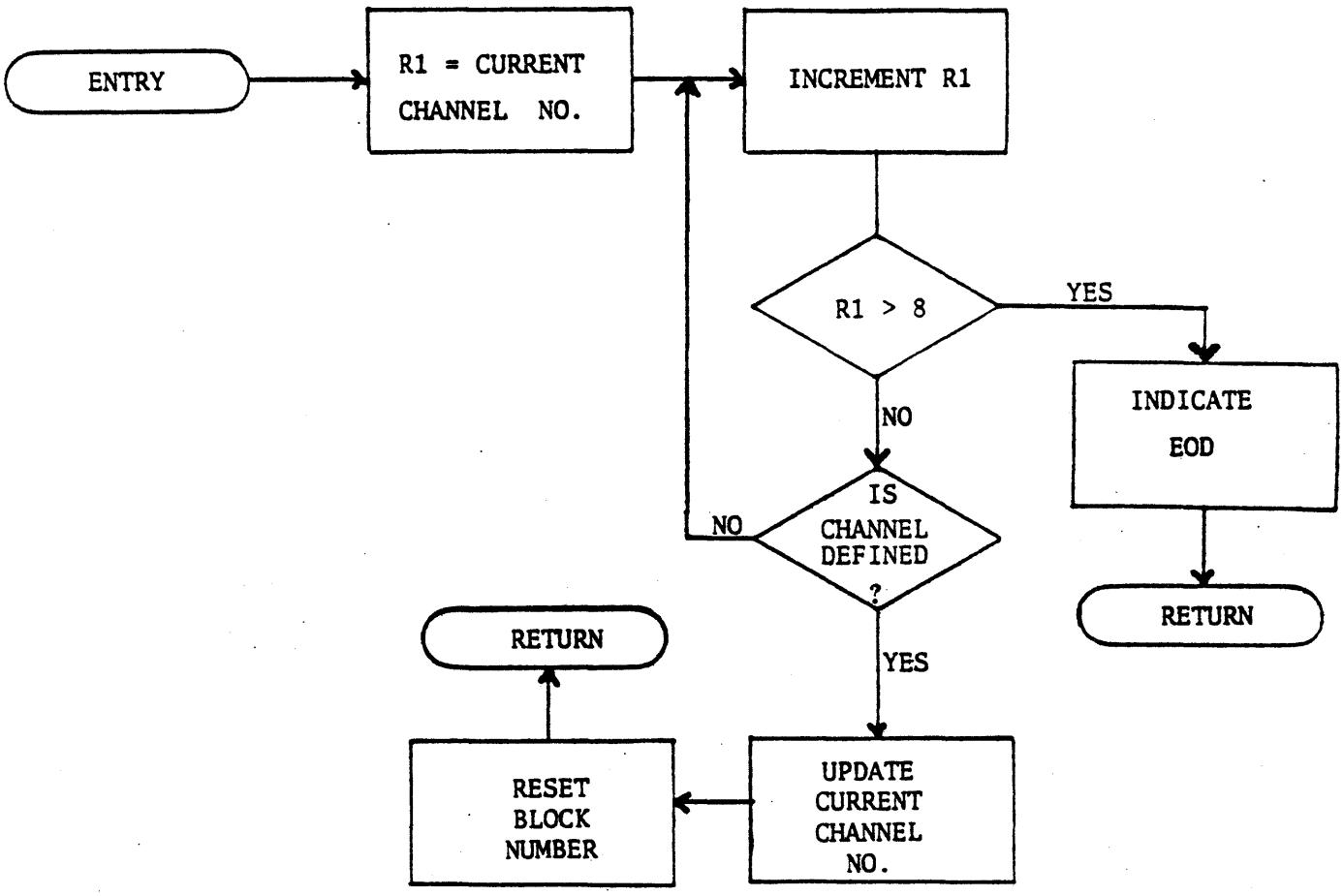
(ii) NXTLNE


ENTRY
HAVE WE READ A PARTIAL LINE ?
NO
SET PARTIAL LINE FLAG
YES
SET POINTER INTO LINE BUFFER (ILNEPT)
SET R0 TO LINE BUFFER POINTER & R1 TO BLOCK BUFFER POINTER
MOVE NEXT CHARACTER INTO LINE BUFFER
DOES CHAR = 0 ?
YES
NO
DOES CHAR = LF
YES
BACKSPACE OVER LF
NO
DOES CHAR = CR
NO
YES
ARE WE AT THE CURRENT BLOCK'S END ?
NO
ARE WE AT THE LINE BUFFER'S END ?
NO
YES
CLEAR PARTIAL LINE FLAG
RETURN CODE = ERROR
INSERT A STAGE 2 EOL
UPDATE LBR AND IBLKPT
RETURN
YES
RETURN
BACKSPACE OVER SOFTWARE EOF
SAVE R0 INTO ILNEPT
CALL NXTBLK
WAS BLOCK XFER OK ?
NO
YES

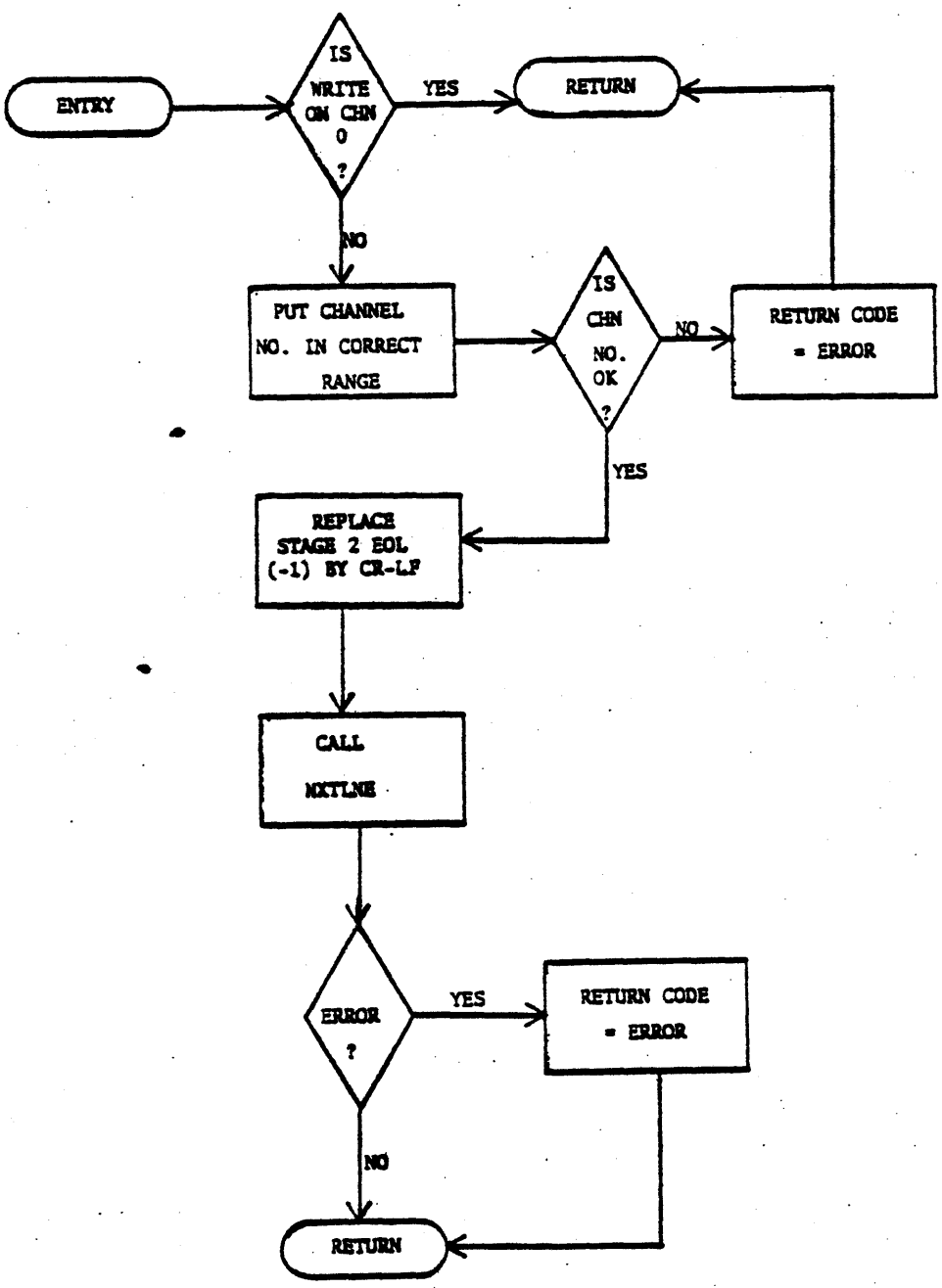
(iii) NXTBLK (RDBLK)



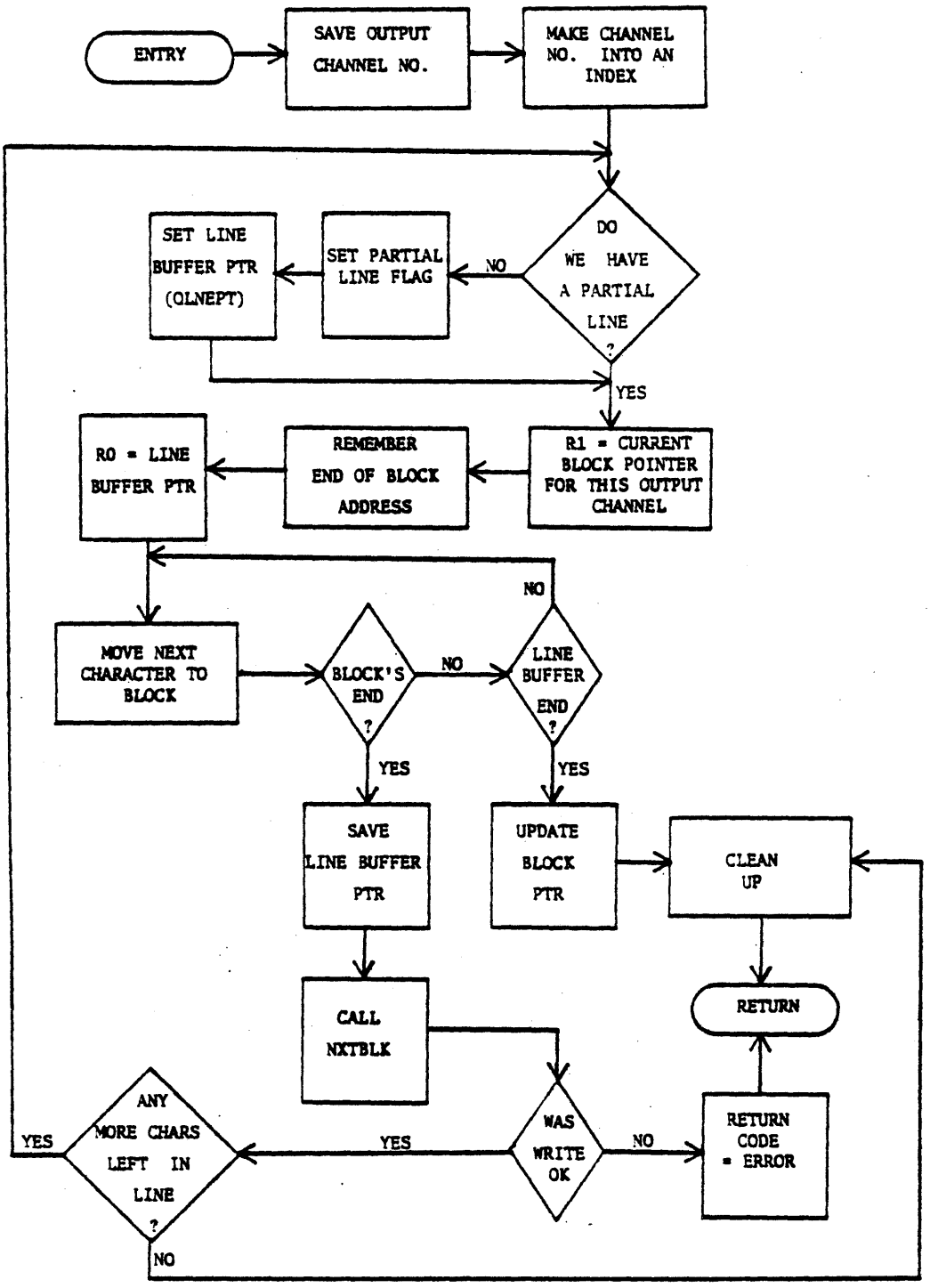
(iv) NXTFIL (FSTFIL)



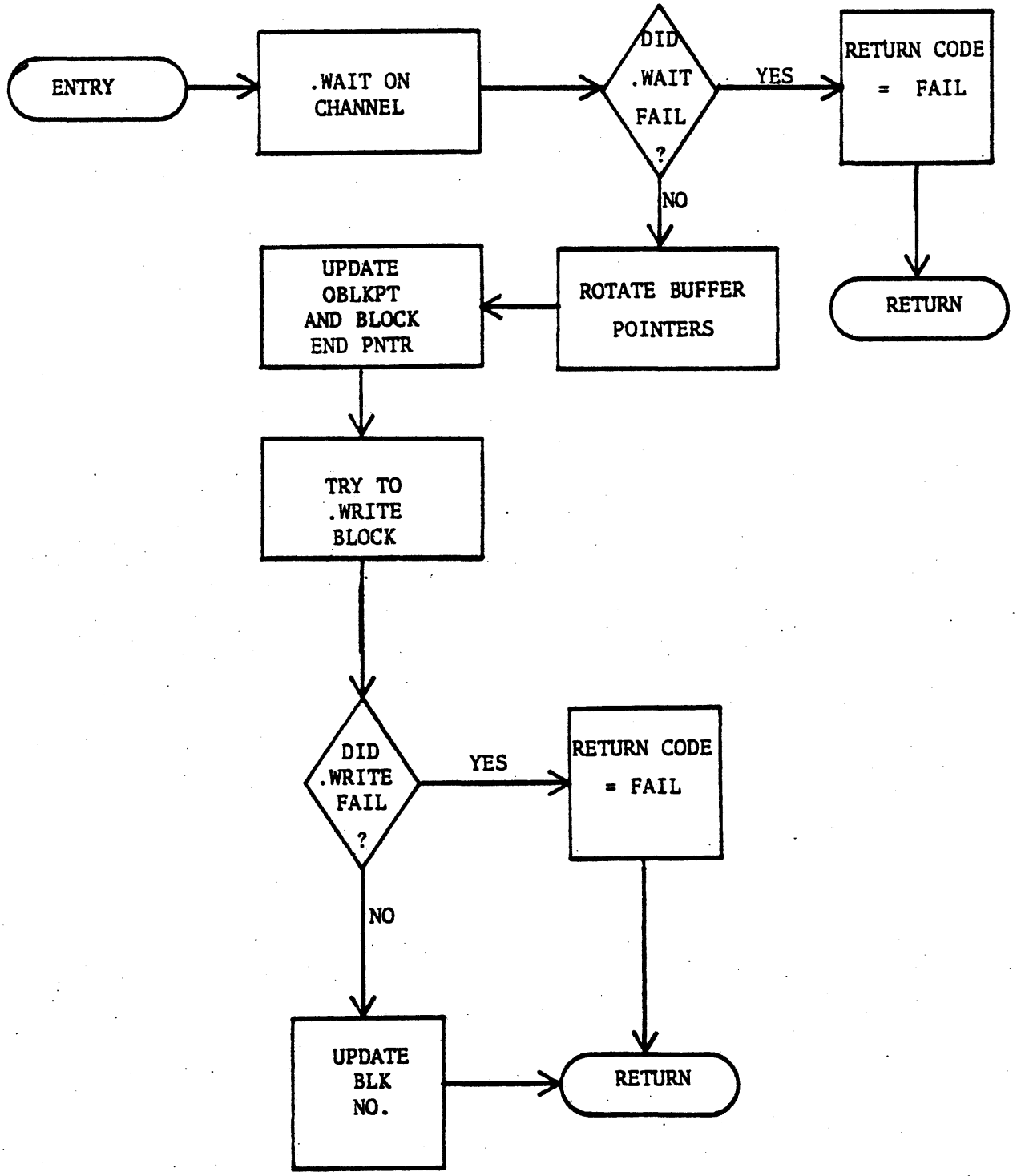
D. (i) WRITE


ENTRY
IS WRITE ON CHN 0 ?
YES
RETURN
NO
PUT CHANNEL NO. IN CORRECT RANGE
IS CHN NO. OK ?
NO
RETURN CODE = ERROR
YES
REPLACE STAGE 2 EOL (-1) BY CR-LF
CALL NXTLNE
ERROR ?
YES
RETURN CODE = ERROR
NO
RETURN

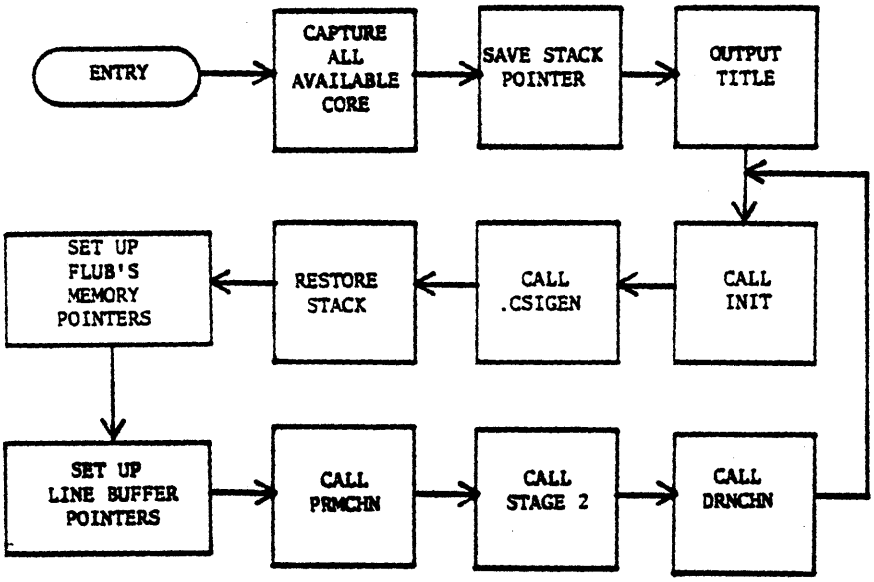
(ii) NXTLNE


ENTRY
SAVE OUTPUT CHANNEL NO.
MAKE CHANNEL NO. INTO AN INDEX
DO WE HAVE A PARTIAL LINE ?
NO
SET PARTIAL LINE FLAG
SET LINE BUFFER PTR (OLNEPT)
YES
R1 = CURRENT BLOCK POINTER FOR THIS OUTPUT CHANNEL
REMEMBER END OF BLOCK ADDRESS
R0 = LINE BUFFER PTR
NO
LINE BUFFER END ?
NO
BLOCK'S END ?
MOVE NEXT CHARACTER TO BLOCK
YES
YES
SAVE LINE BUFFER PTR
UPDATE BLOCK PTR
CLEAN UP
RETURN
CALL NXTBLK
WAS WRITE OK
NO
RETURN CODE = ERROR
YES
ANY MORE CHARS LEFT IN LINE ?
YES
NO

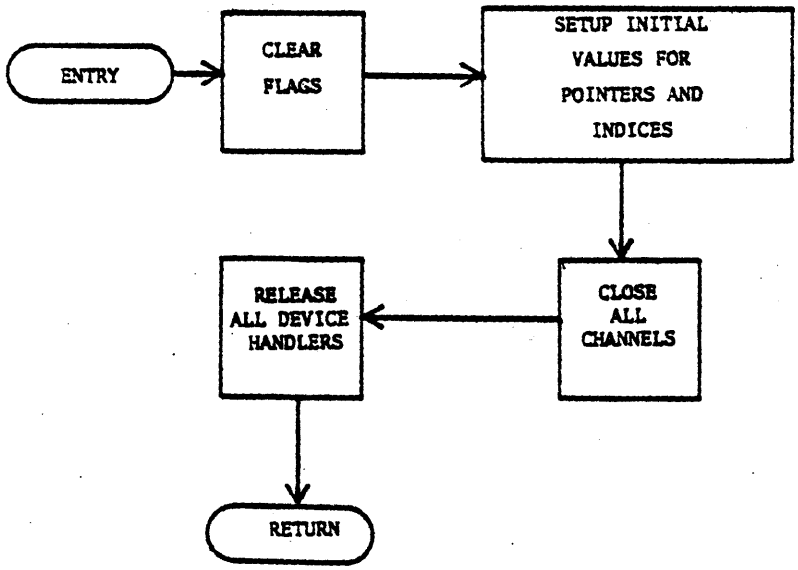
(iii) NXTBLK (WRTBLK)



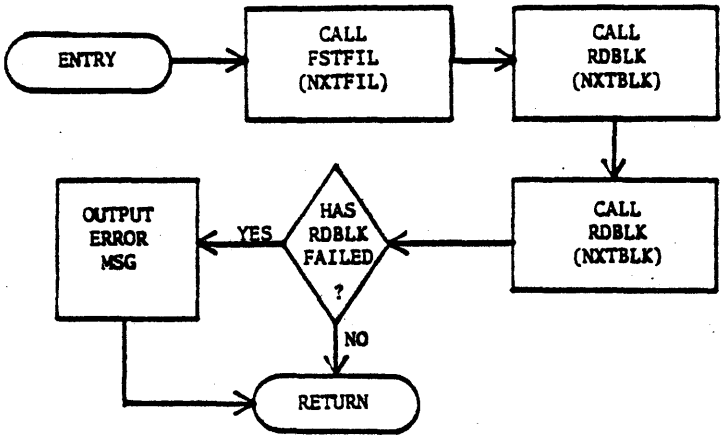
E. (i) ST2CMD



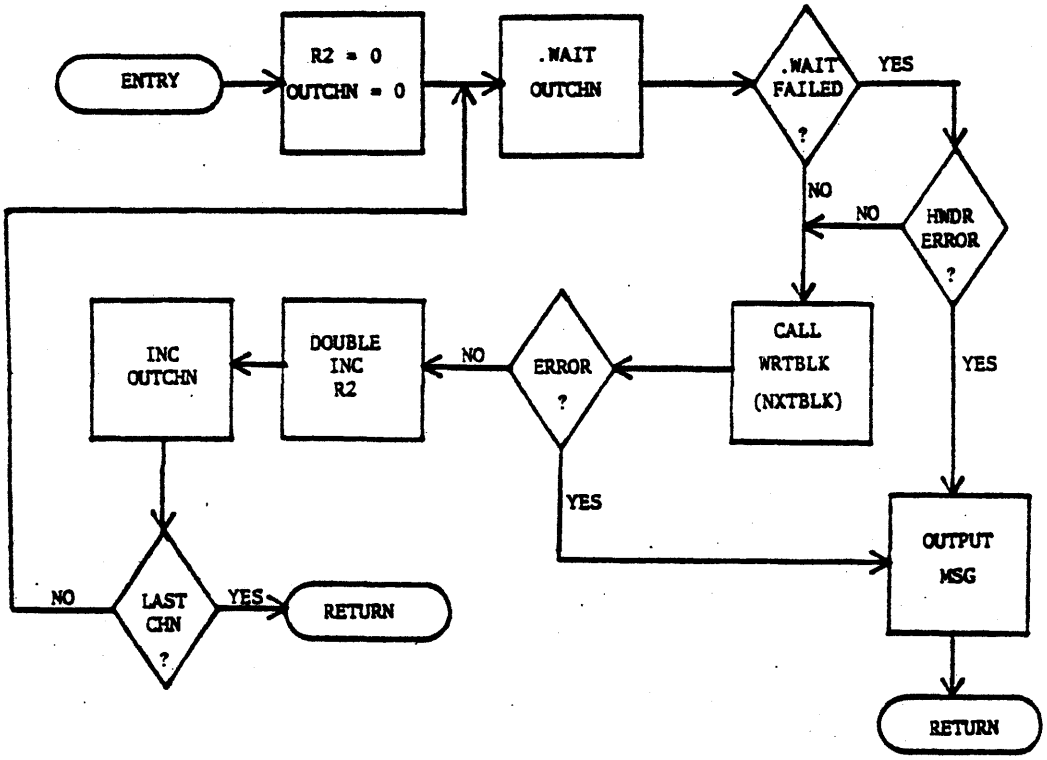
(ii) INIT



(iii) PRMCHN



(iv) DRNCHN



10. BATCH STREAM LISTINGS

```
$JOB/RT11
        !
        !!!!!!!!!!!!
        !           !
        !ST2BLD.BAT!
        !           !
        !!!!!!!!!!!!
        !
$MESSAGE/WAIT PLEASE MOUNT DISTRIBUTION DECTAPE ON UNIT 0
.R PIP
*SY:*.*/K=DT:ST2T.ST2,STG2.FLB,FLUB.ST2,OPT.ST2/X
*SY:*.*/K=DT:*.BAT/K
*SY:*.*/K=DT:ST2DFN.MAC
$MESSAGE/NOWAIT IF YOU WISH TO ASSEMBLE THE RT-11 DEPENDANT
$MESSAGE/NOWAIT MODULES AND LIST THEM BEFORE LINKING STAGE2,
$MESSAGE/NOWAIT TYPE:
$MESSAGE/NOWAIT                    BLDLST.BAT
$MESSAGE/NOWAIT
$MESSAGE/NOWAIT IF YOU WISH TO ASSEMBLE THE RT-11 DEPENDANT
$MESSAGE/NOWAIT MODULES WITHOUT LISTINGS BEFORE LINKING STAGE2
$MESSAGE/NOWAIT TYPE:
$MESSAGE/NOWAIT                    BLDASM.BAT
$MESSAGE/NOWAIT
$MESSAGE/NOWAIT IF YOU SIMPLY WISH TO LINK THE STAGE2 OBJECT
$MESSAGE/NOWAIT MODULES WHICH HAVE BEEN PROVIDED
$MESSAGE/NOWAIT TYPE:
$MESSAGE/NOWAIT                    BLDLNK.BAT
.'FF'
.R BATCH
*'CTY'
$EOJ
```

```
$JOB/RT11
        !
        !!!!!!!!!!!!
        !           !
        !BLDLST.BAT!
        !           !
        !!!!!!!!!!!!
        !
.R PIP
*SY:*.*/K=DT:*.MAC/K
.ASSIGN '"PLEASE TYPE LIST DEVICE NAME"''CTY'LST
.R MACRO
*ST2CMD,LST:/C=ST2DFN,ST2CMD
*ST2RDN,LST:/C=ST2DFN,ST2RDN
*ST2WIN,LST:/C=ST2DFN,ST2WIN
*ST2MSC,LST:/C=ST2DFN,ST2MSC
*ST2DAT,LST:/C=ST2DFN,ST2DAT
*BOOT,LST:/C=ST2DFN,BOOT
.R LINK
*STAGE2,LST:=ST2CMD,ST2RDN,ST2WIN,ST2MSC,ST2DAT,BOOT
.'FF'
$CHAIN TEST.BAT
$EOJ
```

```
$JOB/RT11
        !
        !!!!!!!!!!!!
        !           !
        !BLDASM.BAT!
        !           !
        !!!!!!!!!!!!
        !
.R PIP
*SY:*.*/X=DT:*.MAC/X
.R MACRO
*ST2CMD=ST2DFN,ST2CMD
*ST2RDN=ST2DFN,ST2RDN
*ST2WIN=ST2DFN,ST2WIN
*ST2MSG=ST2DFN,ST2MSG
*ST2DAT=ST2DFN,ST2DAT
*BOOT=ST2DFN,BOOT
.'FF'
$CHAIN LINKUM.BAT
$EOJ
```

```
$JOB/RT11
        !
        !!!!!!!!!!!!
        !           !
        !BLDLNK.BAT!
        !           !
        !!!!!!!!!!!!
        !
.R PIP
*SY:*.*/X=DT:*.OBJ/X
$CHAIN LINKUM.BAT
$EOJ
```

```
$JOB/RT11
        !
        !!!!!!!!!!!!
        !           !
        !LINKUM.BAT!
        !           !
        !!!!!!!!!!!!
        !
.R LINK
*STAGE2=ST2CMD,ST2RDN,ST2WIN,ST2MSG,ST2DAT,BOOT
.'FF'
$CHAIN TEST.BAT
$EOJ
```

```
$JOB/RT11
        !
        !!!!!!!!!!!!
        !           !
        !TEST.BAT   !
        !           !
        !!!!!!!!!!!!
        !
$MESSAGE/NOWAIT THE STAGE2 TEST WILL BE RUN TWICE, ONCE
$MESSAGE/NOWAIT FOR THE STAGE2 PROVIDED AND ONCE FOR THE
$MESSAGE/NOWAIT STAGE2 CONSTRUCTED IN THE TEST
.R STAGE2
*TEMP,TT:=FLUB.ST2,STG2.FLB
*STG2,TT:=OPT.ST2,TEMP.MAC
.R MACRO
*STG2=ST2DFN,STG2
.R LINK
*TEST=ST2CMD,ST2RDN,ST2WTN,ST2MSC,ST2DAT,STG2
.R TEST
*OUT1,ERR1=ST2T
.R STAGE2
*OUT2,ERR2=ST2T
.'FF'
.R PIP
*LST:=OUT1.MAC
*LST:=ERR1.LOG
*LST:=OUT2.MAC
*LST:=ERR2.LOG
*SY:ST2CMD.*,ST2RDN.*,ST2WTN.*,ST2MSC.*,ST2DAT.*/D
*SY:ST2DFN.*,BOOT.*,TEMP.*,STG2.*/D
*SY:OPT.ST2,FLUB.ST2,ST2T.ST2/D
*SY:*.BAT,*.CTL,TEST.SAV/D
*OUT1.MAC,ERR1.LOG,OUT2.MAC,ERR2.LOG/D
*SY:/S
$EOJ
```