digital

clinical lab12

input programs

digital equipment corporation

DEC-12-MCLIA-A-D

CLINICAL LAB-12

INPUT PROGRAMS

Version 7

First Printing August 1970
Revised July 1972
Revised June 1973

The "HOW TO OBTAIN SOFTWARE INFORMATION" page, located at the back of this document, explains the various services available to DIGITAL software users.

The postage prepaid "READER'S COMMENTS" form on the last page of this document requests the user's critical evaluation. All comments received are acknowledged and will be considered when subsequent documents are prepared.

The material in this document is for information purposes only and is subject to change without notice.

DIGITAL assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL. DIGITAL assumes no responsibility for any errors that may appear in this document.

PREFACE

This document is written for personnel who wish to become acquainted with the internal structure of the system. Complete understanding of the contents of this manual requires the reader to be familiar with the PDP-12 System Reference Manual (DEC-12-SRZC-D) and the CLINICAL LAB-12 System Programmers Manual (DEC-12-MRDC-D).

Each section of this manual deals with one program and its associated overlays. A section includes the user's guide, an internal description, a set of assembly instructions and a complete set of flow charts.

Associated documents include:

Operator's Handbook, DEC-12-MCLOA-A-D
User's Handbook, DEC-12-MCLUA-A-D
System Programmer's Manual, DEC-12-MRDC-D
Input Programs Manual, DEC-12-UIFB-D
Output Programs Manual, DEC-12-U2FB-D
On-Line Programs Manual, DEC-12-U3FB-D

CONTENTS

CONTENTS

CONTENTS

CHAPTER 1

INTRODUCTION

The programs described in this manual fall into two classes, the first set deals with administrative data and the second with test data. Also included is a description of DATA-PF, which is a system program called by other user programs to file patient data.

The first set of programs consists of Administrative Update (AD), Requisition Entry (RE) and Delete (DE). The AD Program permits addition, modification, or deletion of patient administrative data (name, patient number etc.) in the files on disk. RE is used to requisition tests and save space on the disk for the test results. DE permits full or partial deletion of patient data.

The second set of programs consists of Test Update (TE), Manual Calculations (CA), and Accession Number Entry (AC). TE is the program which allows the technician to modify test data or to enter results from an instrument not directly connected to the system. CA allows the technician to calculate a new result from one or more pieces of information already in the patients file. AC is used with on-line data to identify one or more results from an Auto-Analyzer sample with the correct patient.

CHAPTER 2
ADMINISTRATION UPDATE

The Administration Update (AD) program consists of two separate routines to perform the functions of entering and modifying patient administrative data. The technician controls these routines through a step-by-step cookbook procedure. The goal in coding each routine has been to make things quick and convenient for the experienced technician, while guiding the beginner who is attempting to alter the information in the patient's file without knowing the proper format.

2.1 ENTERING A NEW PATIENT

The technician, after typing AD and receiving the message E OR M*, types "E↵" which calls this routine. The conversation between the system and the technician which follows can best be shown in an example, such as the one given below. All system inspired typeouts in the example are underlined.

```
E OR M*                    E↵
n PATIENTS ON FILE
PATIENT #:                 1234↵
NAME (L F M):              JONES, JOHN A.↵
PAT. TYPE:                 SU↵
NEW TYPE. OK?              Y↵
N.S.:                      63E↵
DR.:                       JOE↵
ROOM #:                    C-25↵
SEX:                       M↵
BIRTHDATE:                 12/07/1941↵
CHANGES?                   N↵
PATIENT #:
```

For the experienced operator, the above procedure is sufficient to add any patient to the patient files.

An editing feature has been provided for the case in which a technician notices a mistake on a line(s) previously typed in. Rather than recalling the program by an "E↵" and retyping eight lines to correct the one errant line, all the technician need do is answer "YES" to the CHANGES? typed by the system. The enter routine then enters the correction mode, where a "↵" by the technician indicates the line is correct as it stands, and any other character indicates the technician is retyping the last line. (See the example below where the technician wishes to correct both the physician's code and the patient's name).

```
CHANGES?                      YES)
PATIENT #:                    1234*)
NAME (L F M):                 JONES, JOHN A. *BARROW CLYDE M.)
PAT. TYPE:                    SU)
N.S.:                         6BE*)
DR.:                          JOE*  BOB)
ROOM #:                       C-25 )
SEX:                          M*)
BIRTHDATE:                    12/07/1941*)
CHANGES?                      N)
```

If no changes are required (NO is typed after "CHANGES?"), the program asks for another patient name.

The technician can exit this program at any time by typing "STOP)" for any line. The program types "TTY IS FREE" and exits, with no information for this patient reaching the disk.

Before storing the information given it by the technician, the routine conducts certain tests on its own; if the information is not in the proper format, the routine prints an appropriate error message. The technician then retypes the offending line and, if all other tests check out, the information is stored in the patient files. The error messages listed below provide sufficient checks to catch any error.

```
1.   INVALID INPUT -
     TRY AGAIN  (AN INVALID CODE HAS BEEN TYPED)

2.   PATIENT # IN USE  (SOMEONE ELSE HAS THE PATIENT NUMBER
                        TECHNICIAN IS TRYING TO INSERT)

3.   FORMAT IS MONTH/DAY/YEAR  (WRONG DATE OF BIRTH FORMAT)

4.   ANSWER Y OR N   (IMPROPER ANSWER TO "CHANGE?" WAS GIVEN)

5.   NO MORE ROOM    (FILES ARE FULL)
```

In any of the above cases (except patient # and sex) the technician could strike a "?" or)for a given line and not receive an error message. The six character code for "?" (77 octal) followed by all spaces (character code 40) is then stored in the appropriate file.

All information is written out and read in from the disk by the routine before being permanently stored there to ensure the accuracy of the patient information on the disk. The routine searches the subfile directory "number" of the first location containing 7777 octal indicating an open file location. This "number" is the relative address of the location containing the 7777 octal. The routine uses this "number" to locate the starting address of the location where each separate line of patient information should be stored. It does this by multiplying the "number" by the number of locations necessary to define the given piece of information, and adds the result to the initial address of the appropriate file. The information is then stored in this resulting address. In addition, the number of empty slots is counted, and if fewer than 50 remain, an appropriate message is output.

2.2 MODIFYING INFORMATION ON A PATIENT FILE

After typing AD and receiving the message "E OR M*", the technician types "M " to call the modify routine. The message "FOR ITEM TYPE P, N, T, W, R, D, S, or B   PATIENT #:" is printed.

The technician types the patient number of the file to be modified and then approves the patient's name printed by the system to match the patient number. If the computer printed patient name does not match the patient number, INQUIRY (part of the summary program) can be used to find the problem.

To modify an item, the technician types the appropriate code (P, N, T, W, R, D, S, B) in response to ITEM* where:

P	=	Patient Number	R	=	Room Number
N	=	Name	D	=	Doctor
T	=	Type	S	=	Sex
W	=	Nursing Station	B	=	Birthdate

After the item code is typed, the correction procedure is the same as described in Section 2.1. An example in which the technician alters the patient's number is shown below:

```
E OR M*                                 M↵
FOR ITEM TYPE P,N,T,W,R,S,O,R,B
PATIENT #:                              1234↵
IS IS BARROW CLYDE M?:                  YES↵
ITEM*                                   P↵
PATIENT #: 1234*                        1236↵
```

2.3 ASSEMBLY INSTRUCTIONS

The AD and DE programs are listed as one large program so that the conditional assembly should indicate AD or DE. The assembler builds the appropriate chain. Blocks 1-15 of the binary are saved as AD.

2.4 INTERNAL DESCRIPTION

When started, AD performs an initialization procedure to determine the size of file 30 and file 26. The program reads the first 5 words of file 21 to establish constants for buffer sizes corresponding to the size of the patient number. This code is located in the disk buffer and is destroyed when the program starts. The program builds a mask of 77 or 00 bytes to mask out the reference field of a patient number when AD is concerned only with the patient number reference field. The mask code is located in the Teletype buffer and is destroyed when AD starts. AD then enters a routine to ask which option the user wants to run. Possible options include S, M, or E:

> "S" transfers control to a "show-me routine;
>
> "M" transfers control to the modify path;
>
> "E" transfers control to the COUNT routine which counts the number of patients on file and then transfers to the entry path.

Most of the code for the entry path is shared by the modify path. In addition, MODIFY sets flags, which affect the course of the program, so that when the patient file to be modified is specified, the old information is read from the administrative files. Then instead of running through a sequence of questions to ask, the modify path has a list of letters/addresses corresponding to items to be modified together with the address of the routine for this question.

The ENTRY routine merely consists of a sequence of JMS instruction to dispatch control to each of the question asking subroutines, or ASK handlers.

Once the user has specified no more changes in the entry path or typed a carriage return in the modify path after "ITEM*", control is transferred to the FILER routine. If in the entry mode, FILER looks for the first open slot in file 26 and writes a 7776 there. If the files are full, "NO MORE ROOM" is typed and the program exits. The sort file routines (see Section 2.4.1) are then called and when a carriage return is typed, a sequence of JMS instructions transfers control to subroutines for filing each piece of administrative data. The modify path skips the procedure of reading file 26 for the first open slot. The FILER routine then checks a flag to see whether it should return to the entry or modify subroutines.

The program maintains buffers in core for each piece of administrative data. For some data, like patient name or room number, there is only one buffer. The format is 6-bit ASCII preceded by a constant equal to the negative of the number of words in the character string. For some data, like birth date and sex, there are two buffers:

1. One represents the actual binary value to be filed;
2. The other represents a text string corresponding to the binary value to be typed out to the user.

"GET" handlers, the subroutines for getting old information out of the files for the modify path, also transform the binary value on file to the text to be typed to the user. "ASK" handlers will ask the appropriate question, parse the input for validity, and encode the input text as a new binary value, if necessary. In that case, the new text typed in will be saved as the new text string to be typed out in case the user's responses cause the ASK handler to be called again before the data is filed. "FILE" handlers merely file the data buffer in the right place and right file on disk. The handlers (GET 23, ASKQ5) for room number information exemplify this process well because there is no syntax checking or parsing involved. This represents the minimum amount of overhead necessary to handle a data item. Note that there is no FILE23 subroutine, since the code involved would be exactly the same as in the GET 23 subroutine. The only difference is that one reads and the other writes. This parameter is never under the control of an individual handler, however. A global switch (WRITE) on page 0 controls reading or writing and thus, the individual handler will perform the action dictated by the routine containing the GET sequence of FILE sequence. These handlers are designed so that they can be called without regard to order. These handlers are completely disjoint except for ward/Dr. code and sex/DOB because these items are combined into records in the same file.

All of the ask handlers begin by using ASKSB as a front end. ASKSB will put the patients' old data in the buffer before typing the information. If the program is using the modify option and if a carriage return is typed, the routine will branch back to call +3. The format in a handler then is:

```
ASKX,     0
          JMS I ASKSUB          /Jump to front end
          QX                    /Question text to type out
          BUFFX                 /Old info. text to type if in modify path
          JMP I ASKX            /Do nothing if modify path and C.R. input
          :                     /Check for input syntax (optional)
          :                     /Convert text to binary value (optional)
          :                     /Search for an occurrence of this
          :                     /Item already on file (optional)
          TAD ABUFFX            /Save new text
          JMS I PACK
          JMP I ASKX            /Done
BADQX,    JMS I TYPOUT
          BOOMS
          JMP ASKX + 1          /Try again
```

For further information about a specific ASK, GET or FILE handler see the flow charts or the program itself.

2.4.1 Sort File Routines

There are four main routines that maintain the sequenced directories. The routine for deleting a patient file from the directories is common to AD and DE. When a patient file is entered, the program checks that the patients file 26 position is not in the sort file. When patient data is modified, the patient data is removed and then reinserted into the sort files.

2.4.1.1 MSSORT - This subroutine runs through a list of addresses which point to parameter lists describing each type of sort to be done. For each such entry, the SORT routine is called to find the relative position at which the new file 26 key is to put in each subfile of the total sortfile. A list of the keys is built. After all keys are determined, the subroutine INSRT is called for each item in the list to insert the key into its corresponding subfile of the sort file.

2.4.1.2 SORT - This subroutine determines where to put the patient data in the first (next) subfile. An interval halving method is used to locate the highest placed key which has corresponding AD data less than or equal to the AD data of the patient to be admitted. The high and low boundaries of the subfile which are still under consideration are maintained. When their difference (SPAN) becomes less than 256, that segment of the subfile is read, and further fetches of keys are done from core rather from disk. When the next key is fetched, the parameter list passed by MSSORT is interpreted to find which AD data files to read to compare with which core buffer of new data and which bits or bytes of the records to mask when doing the comparison. If data from several AD files is to be considered, the first (next) entry is checked. Subsequent AD data files will not be checked unless there is a match on the previous data file. The output of the routine is the position where the patient data is inserted later.

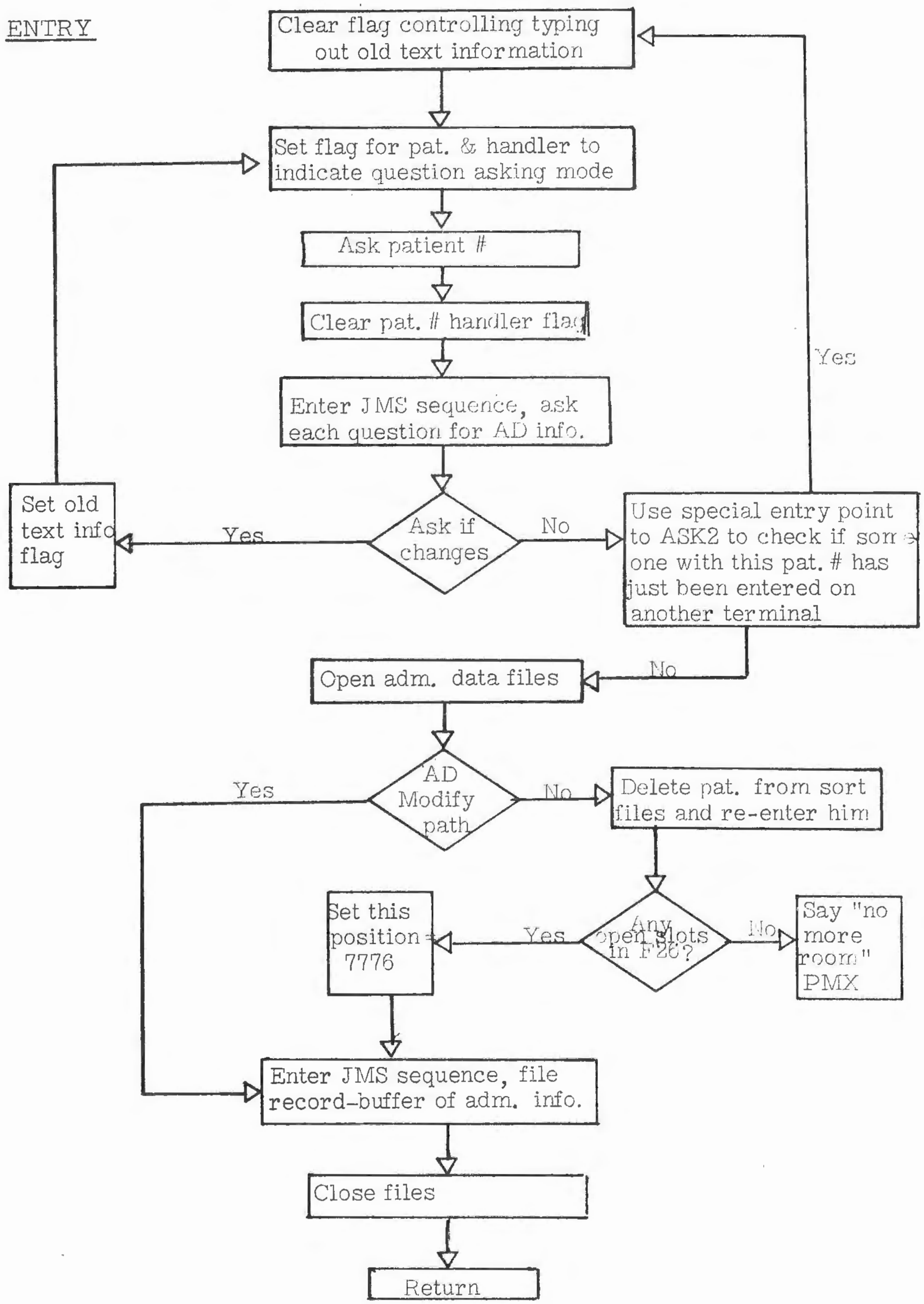
2.4.1.3 INSRT - This subroutine actually inserts a file 26 position word into a subfile directory. The proper subfile is read beginning with the position to insert the patient into the second through last words of the core buffer. The key to insert is put in the first word and the first to second from last word is written back out. The last word of the buffer is put in the first word and the process is repeated until the logical end of the subfile.

2.4.1.4 DESRT - This subroutine deletes occurrences of a specified key or keys which are not less than HOSIZE from all subfiles of the sort file. The program treats each subfile as a logical entity, and reads successive buffers of keys from disk. These buffers are then "collapsed" by removing the above mentioned keys. The buffer is then written back out only if there has been some change so far, either in the current buffer or in previous buffers. This process is continued until either the end of the subfile is reached or until a 7777 key is encountered. A total count of the number of words deleted has been kept and the buffer is end-filled with 7777 for this many words. Then this last buffer is written back out.

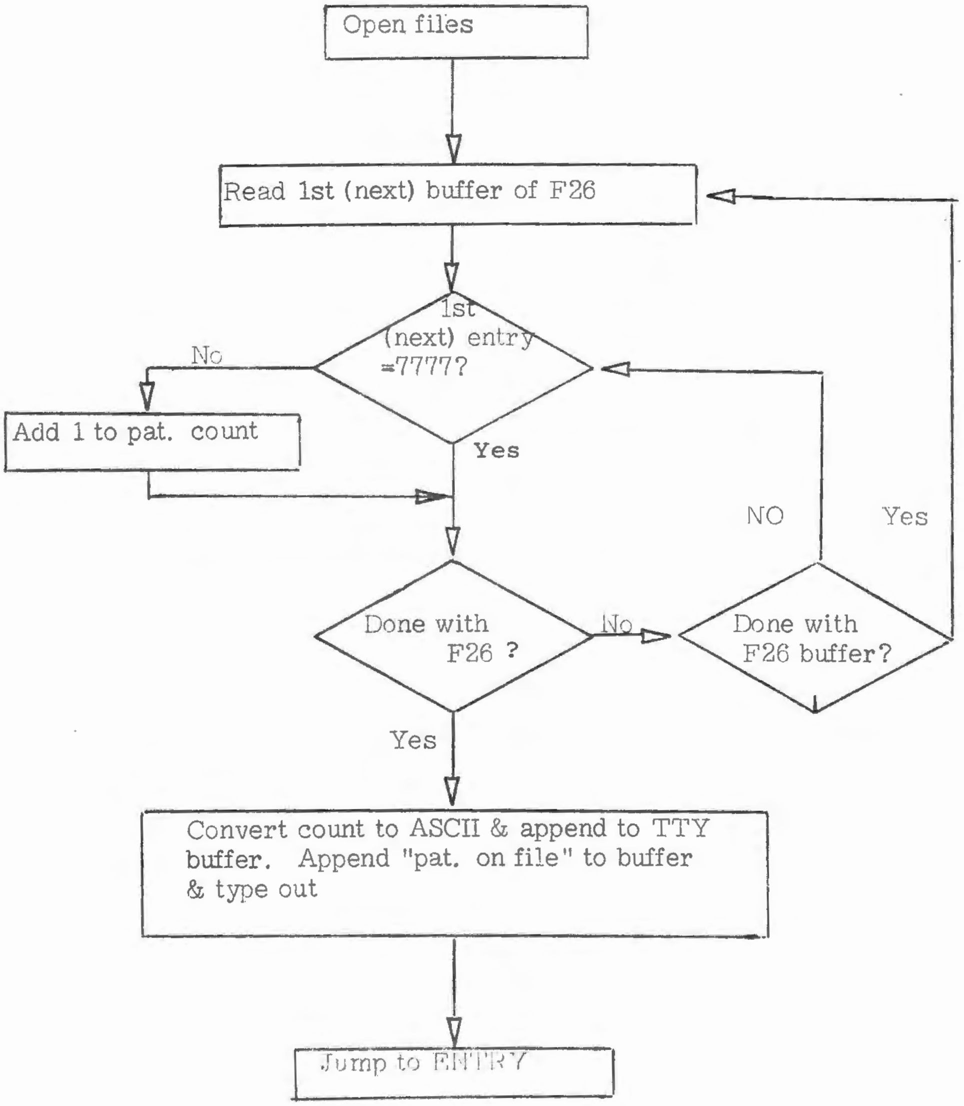
NOTE

> The FILER program (FI), which rebuilds directories, uses the AD sorting routines intact. When changing the number of directories, one must change the equates in the beginning of AD+DE and CH, and add appropriate code to FI for the new directory.

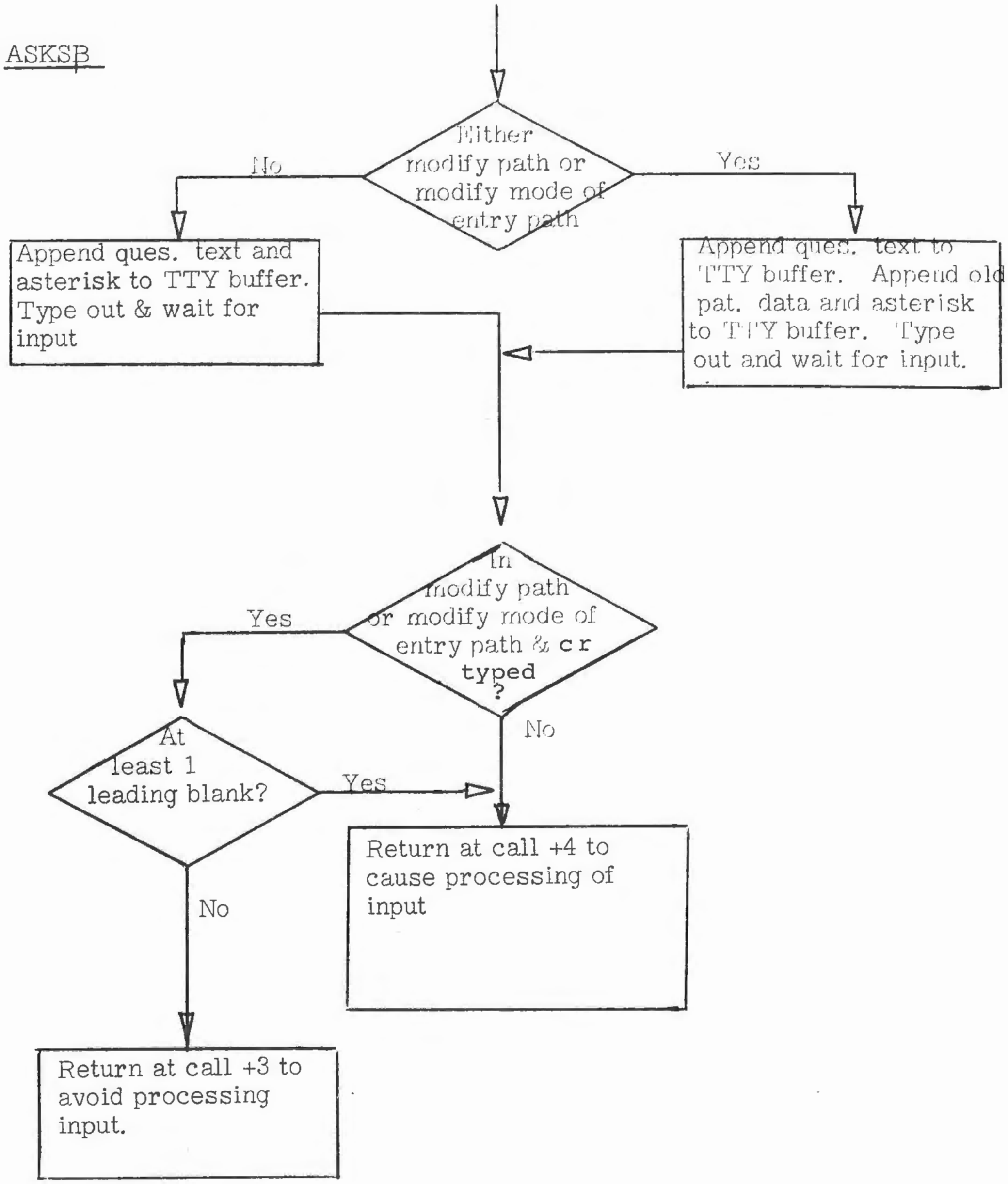
2.5 ADMINISTRATIVE UPDATE FLOWCHARTS


ENTRY
Clear flag controlling typing out old text information
Set flag for pat. & handler to indicate question asking mode
Ask patient #
Clear pat. # handler flag
Enter JMS sequence, ask each question for AD info.
Ask if changes
Yes
Set old text info flag
No
Use special entry point to ASK2 to check if some one with this pat. # has just been entered on another terminal
Yes
No
Open adm. data files
AD Modify path
Yes
No
Delete pat. from sort files and re-enter him
Any open slots in F26?
Yes
Set this position = 7776
No
Say "no more room" PMX
Enter JMS sequence, file record-buffer of adm. info.
Close files
Return

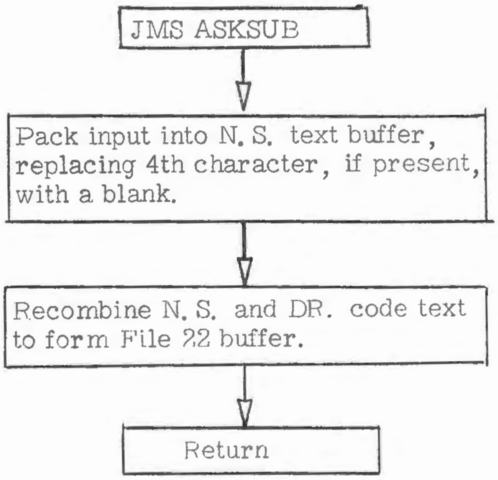
COUNT


Open files
Read 1st (next) buffer of F26
1st (next) entry =7777?
No
Add 1 to pat. count
Yes
NO
Yes
Done with F26 ?
No
Done with F26 buffer?
Yes
Convert count to ASCII & append to TTY buffer. Append "pat. on file" to buffer & type out
Jump to ENTRY

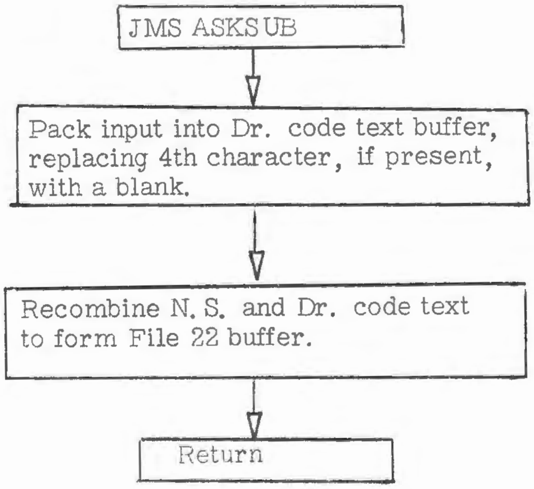
ASKSB


Either modify path or modify mode of entry path
No
Yes
Append ques. text and asterisk to TTY buffer. Type out & wait for input
Append ques. text to TTY buffer. Append old pat. data and asterisk to TTY buffer. Type out and wait for input.
In modify path or modify mode of entry path & cr typed ?
Yes
No
At least 1 leading blank?
Yes
No
Return at call +4 to cause processing of input
Return at call +3 to avoid processing input.

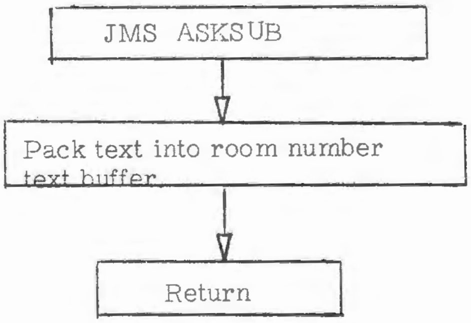
ASK3


JMS ASKSUB
Pack input into N. S. text buffer, replacing 4th character, if present, with a blank.
Recombine N. S. and DR. code text to form File 22 buffer.
Return

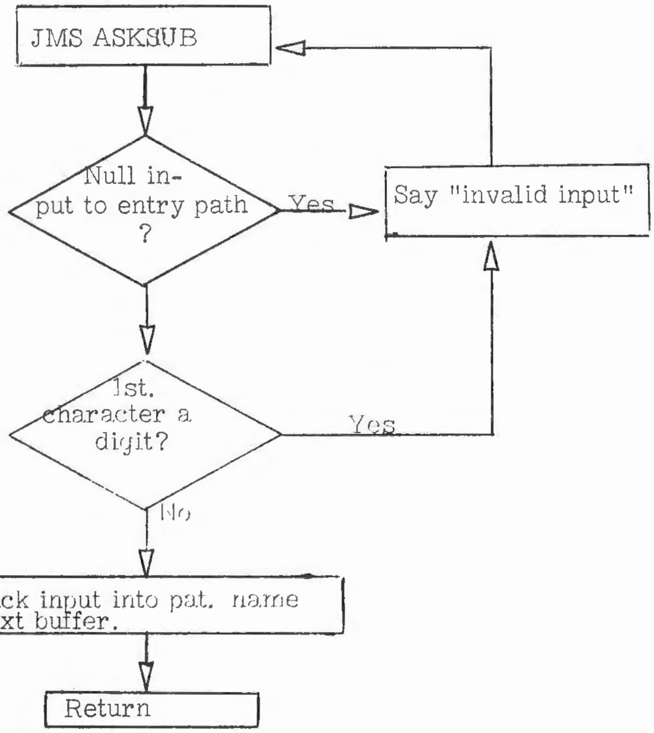
ASK4


JMS ASKSUB
Pack input into Dr. code text buffer, replacing 4th character, if present, with a blank.
Recombine N. S. and Dr. code text to form File 22 buffer.
Return

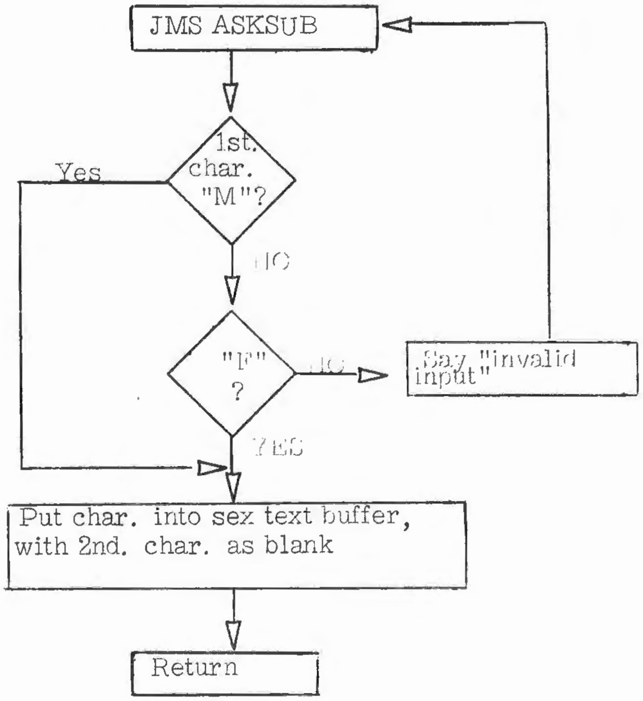
ASK5



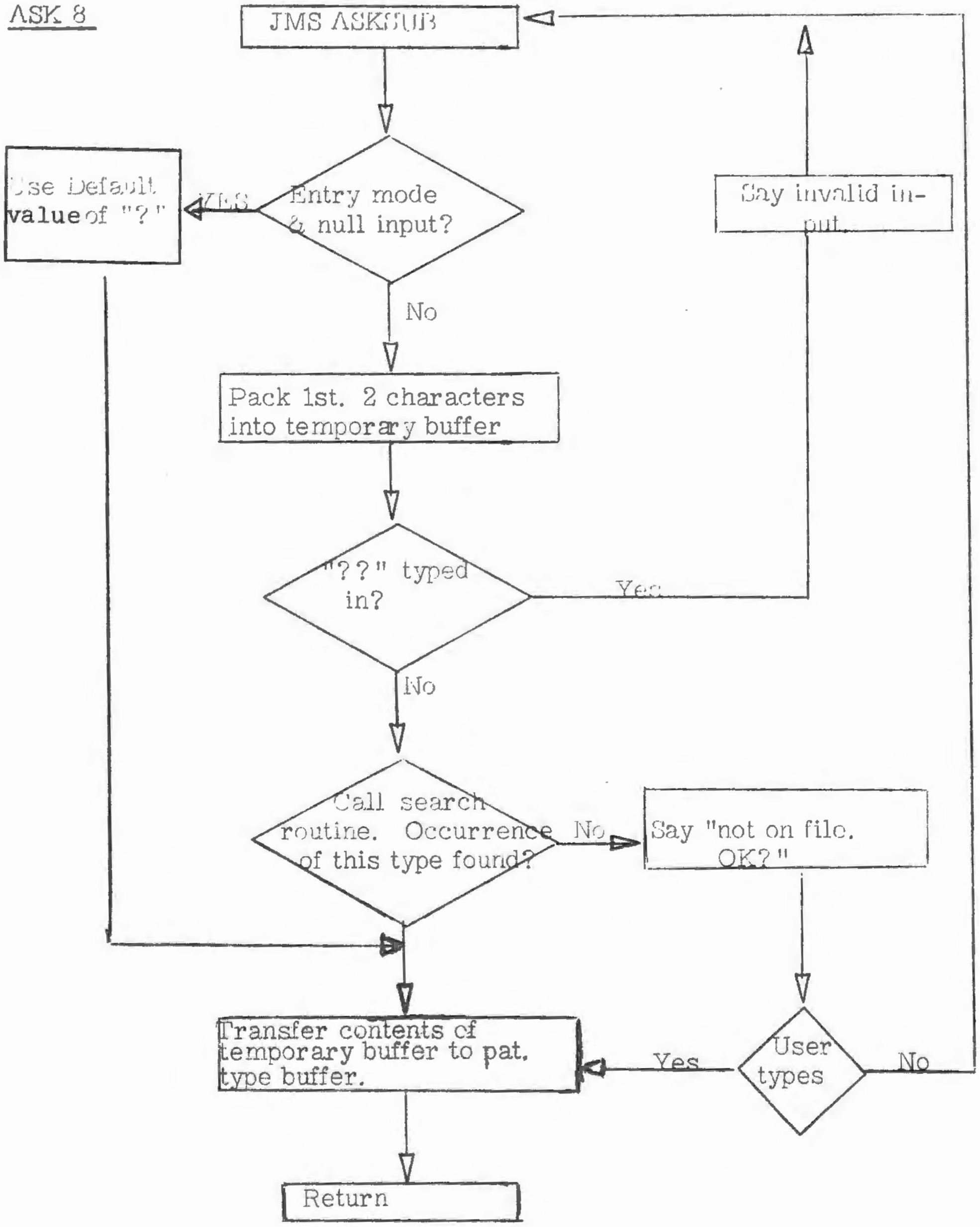
ASK 1


JMS ASKSUB
Null input to entry path ?
Yes
Say "invalid input"
1st. character a digit?
Yes
No
Pack input into pat. name text buffer.
Return

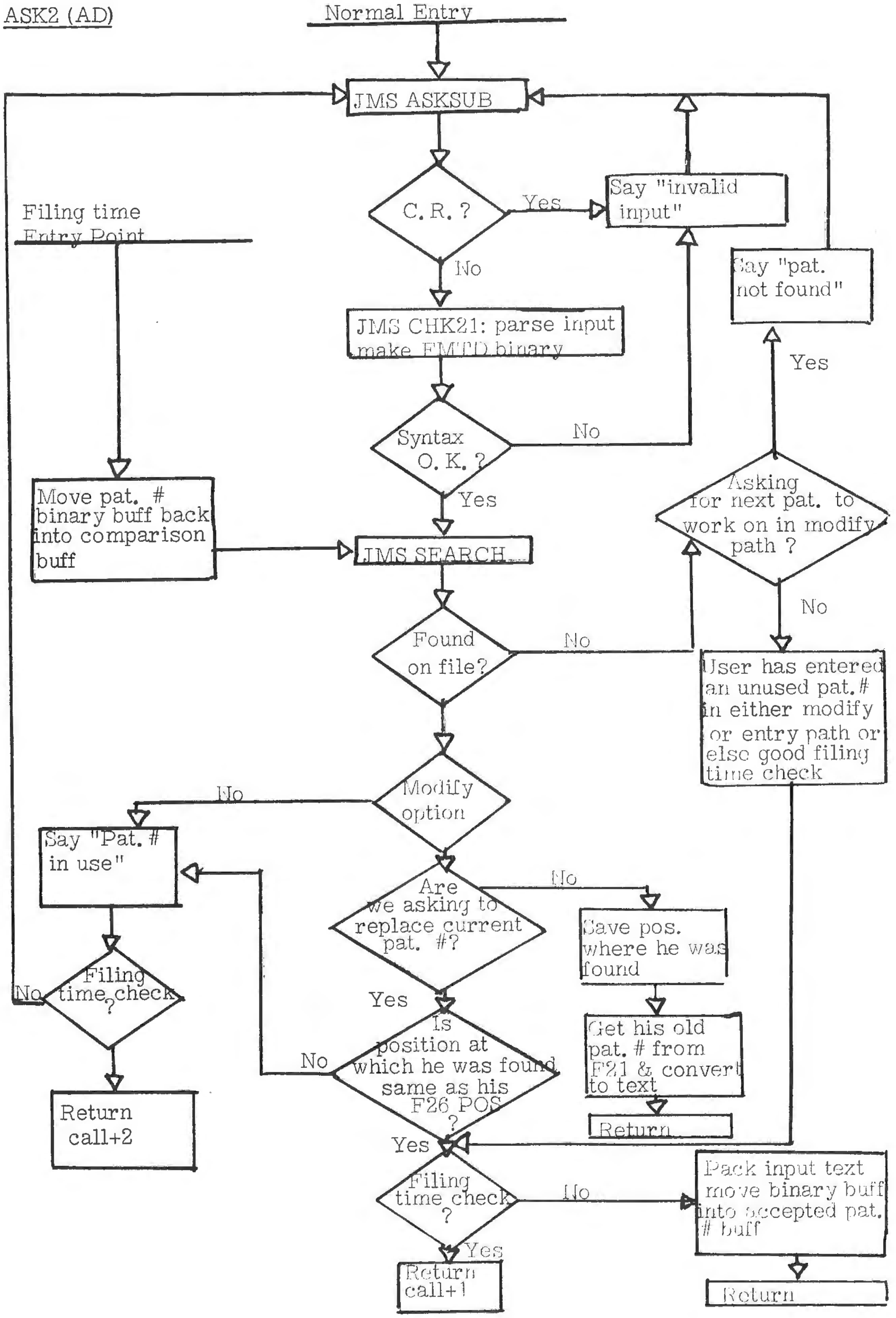
ASK 6


JMS ASKSUB
1st. char. "M"?
Yes
NO
"F" ?
NO
Say "invalid input"
YES
Put char. into sex text buffer, with 2nd. char. as blank
Return

ASK 8

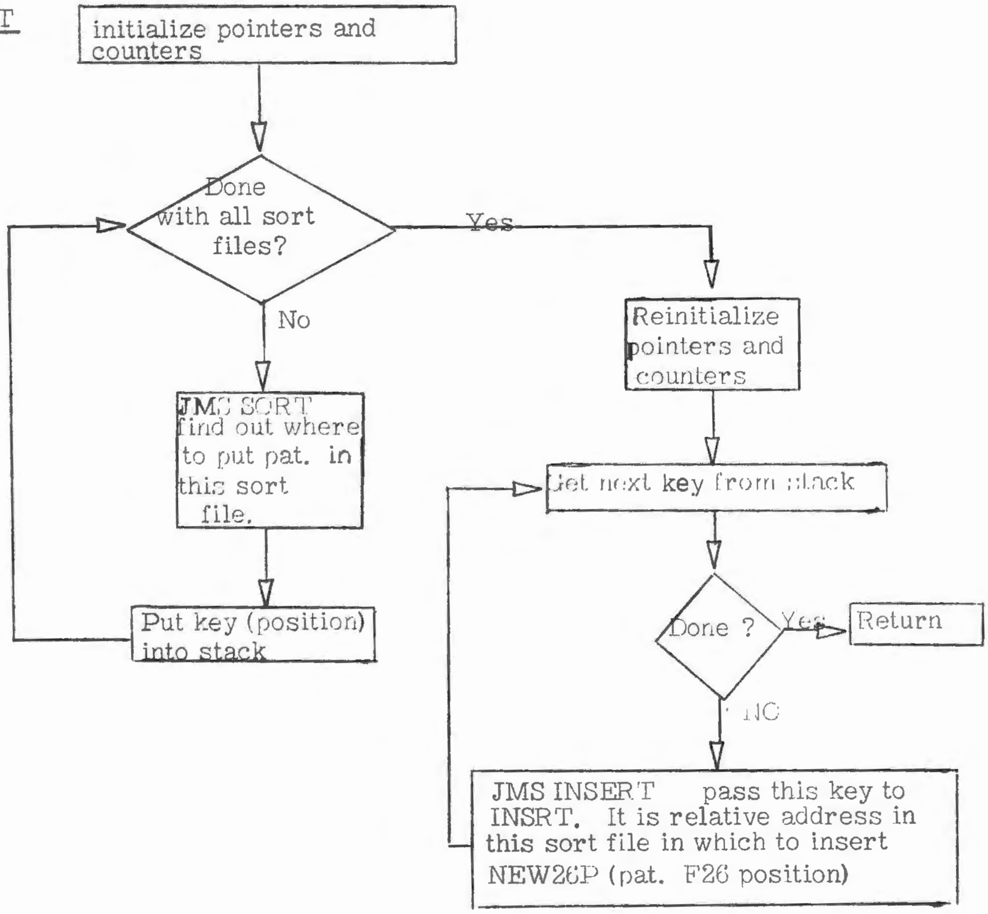

JMS ASKSUB
Entry mode & null input?
YES
Use Default value of "?"
No
Pack 1st. 2 characters into temporary buffer
"??" typed in?
Yes
Say invalid input.
No
Call search routine. Occurrence of this type found?
No
Say "not on file. OK?"
User types
Yes
No
Transfer contents of temporary buffer to pat. type buffer.
Return

ASK2 (AD)

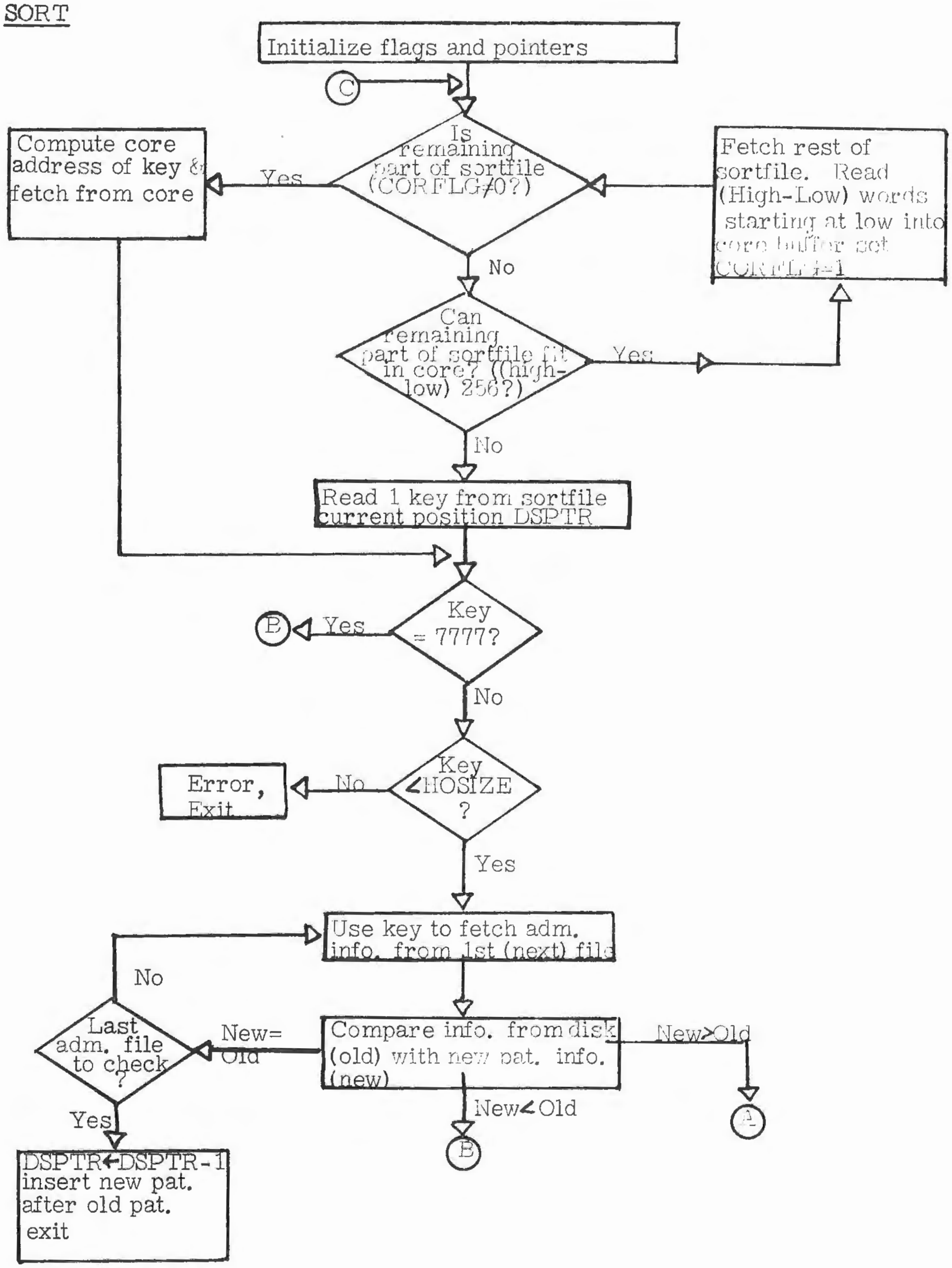


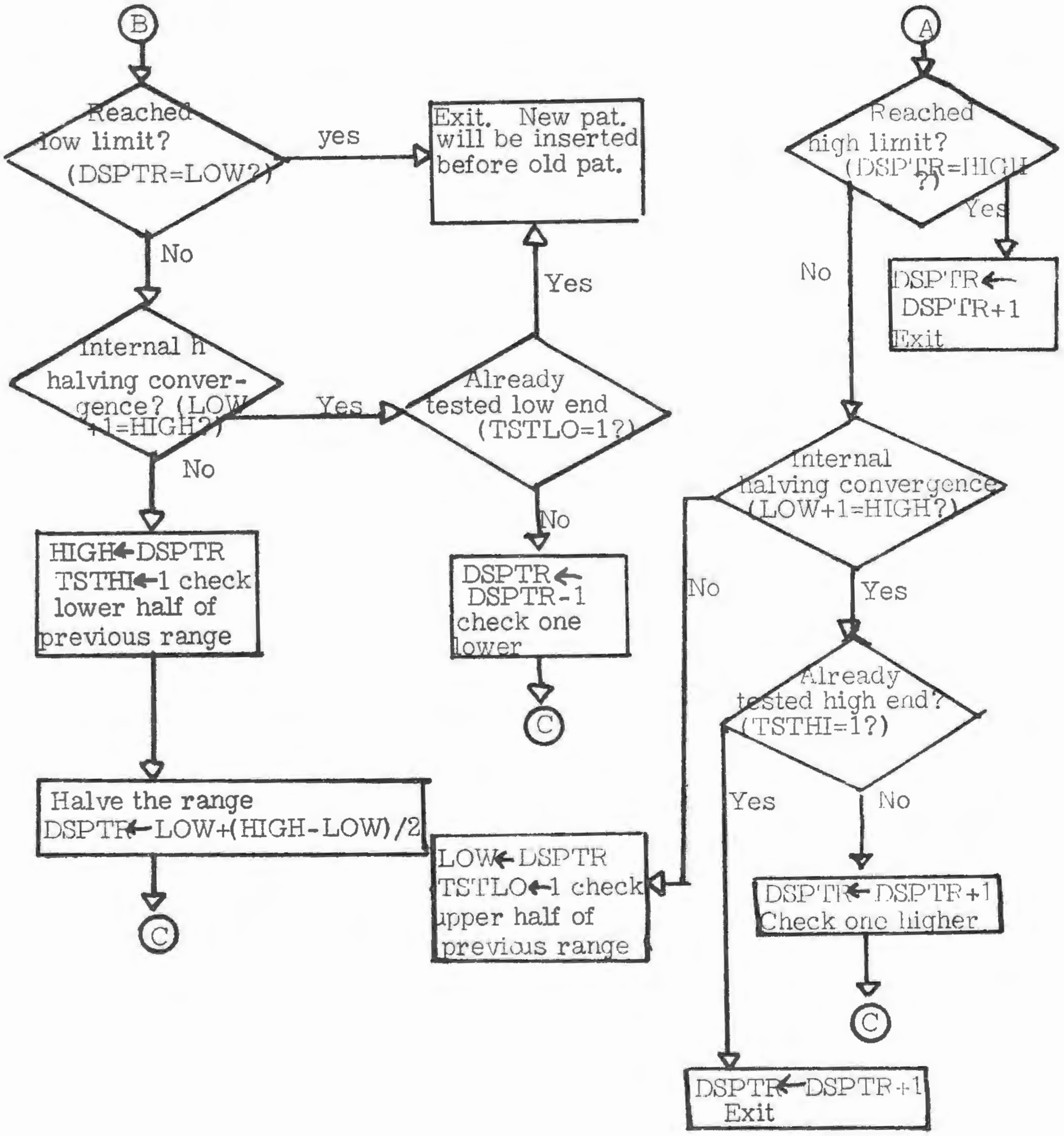
Normal Entry
JMS ASKSUB
C. R. ?
Yes
Say "invalid input"
No
JMS CHK21: parse input make FMTD binary
Syntax O. K. ?
No
Yes
Filing time Entry Point
Move pat. # binary buff back into comparison buff
JMS SEARCH
Say "pat. not found"
Yes
Asking for next pat. to work on in modify path ?
No
Found on file?
No
User has entered an unused pat.# in either modify or entry path or else good filing time check
Modify option
No
Say "Pat. # in use"
Are we asking to replace current pat. #?
No
Save pos. where he was found
Get his old pat. # from F21 & convert to text
Return
Filing time check ?
No
Yes
Is position at which he was found same as his F26 POS ?
No
Return call+2
Yes
Filing time check ?
No
Yes
Return call+1
Pack input text move binary buff into accepted pat. # buff
Return

MSSORT

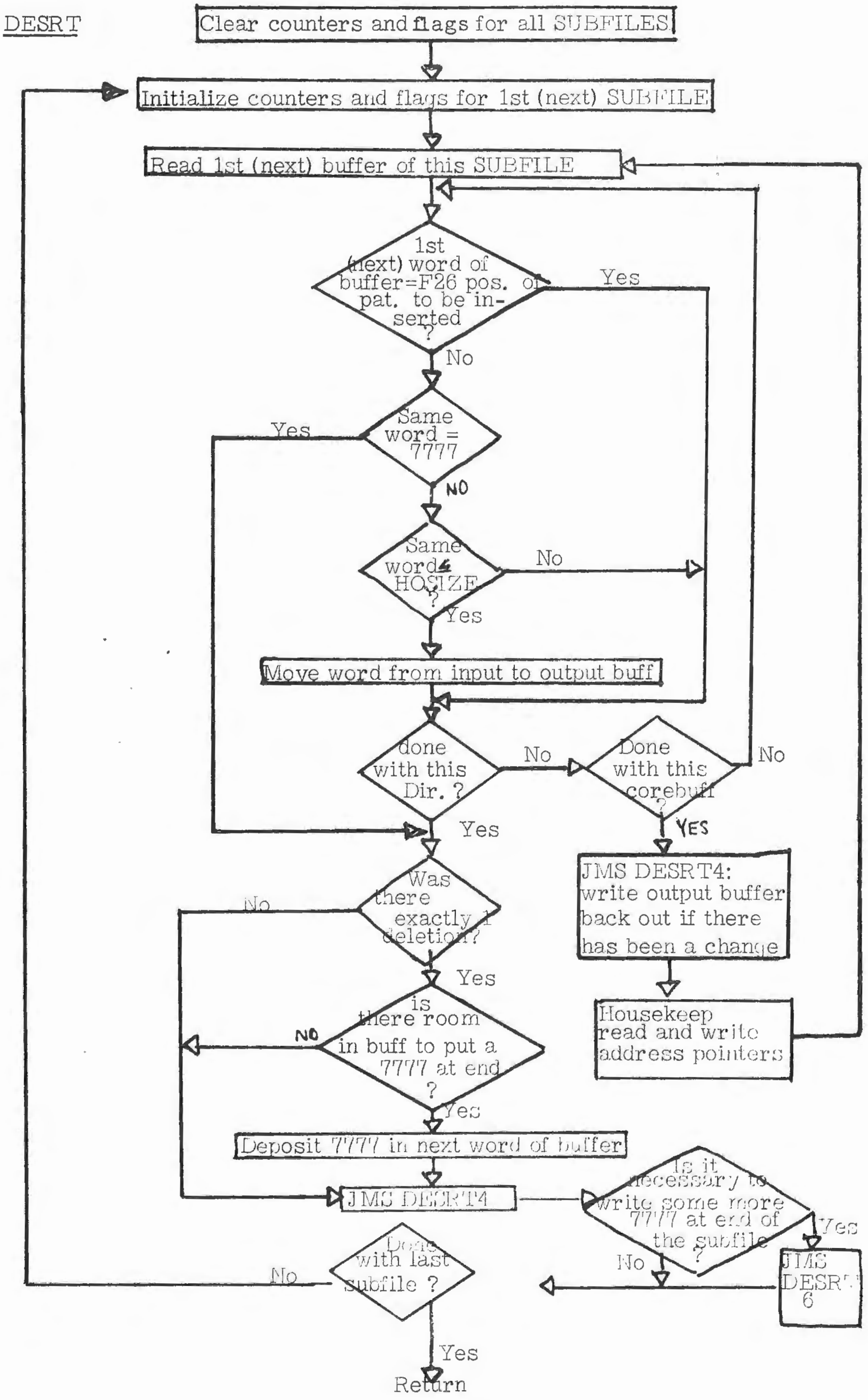

initialize pointers and counters
Done with all sort files?
Yes
No
JMS SORT find out where to put pat. in this sort file.
Put key (position) into stack
Reinitialize pointers and counters
Get next key from stack
Done ?
Yes
Return
NO
JMS INSERT pass this key to INSRT. It is relative address in this sort file in which to insert NEW26P (pat. F26 position)

SORT

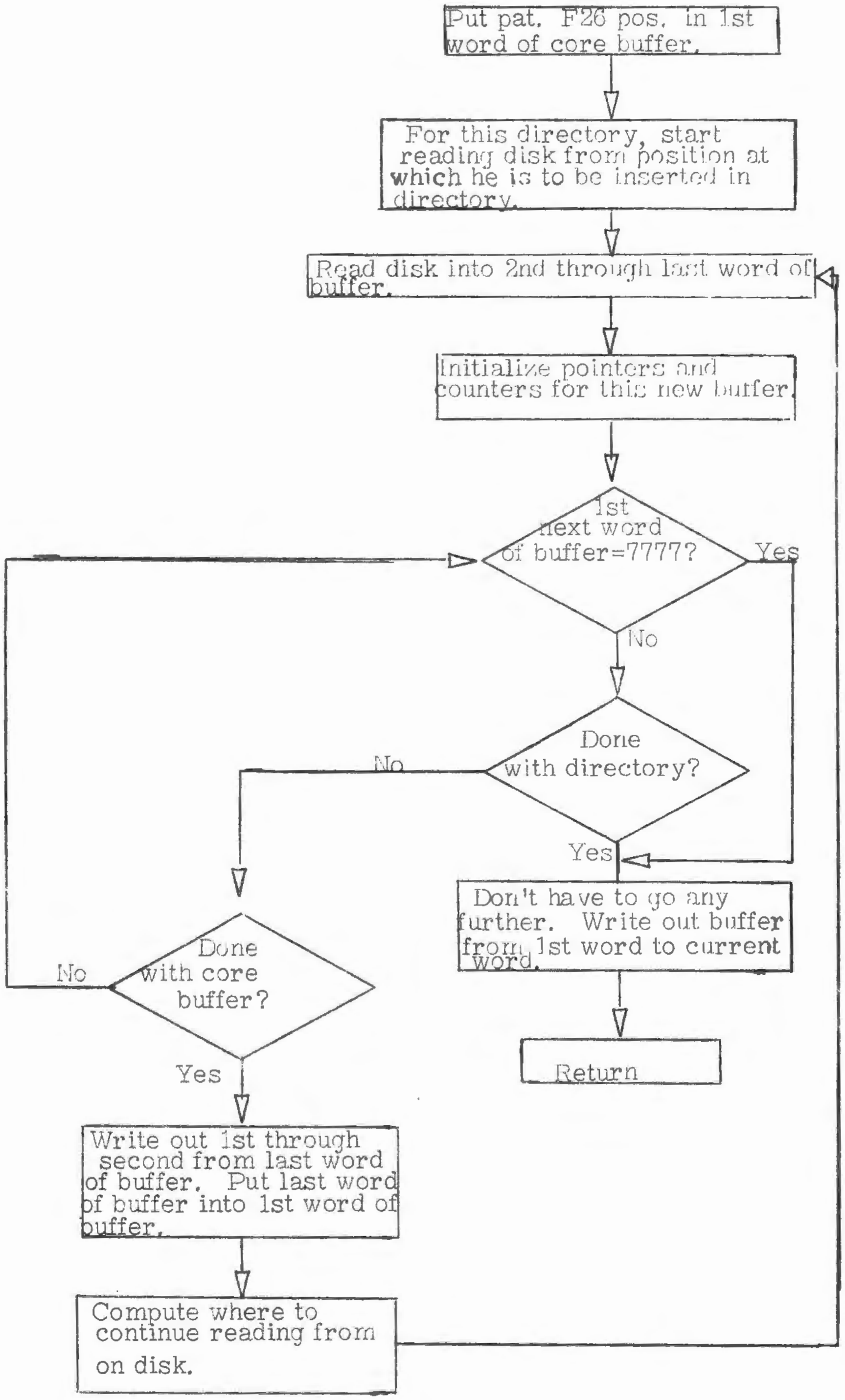

Initialize flags and pointers
C
Is remaining part of sortfile (CORFLG≠0?)
Yes
Compute core address of key & fetch from core
Fetch rest of sortfile. Read (High-Low) words starting at low into core buffer set CORFLG=1
No
Can remaining part of sortfile fit in core? ((high-low) 256?)
Yes
No
Read 1 key from sortfile current position DSPTR
B
Yes
Key = 7777?
No
Error, Exit
No
Key <HOSIZE ?
Yes
Use key to fetch adm. info. from 1st (next) file
No
Last adm. file to check ?
New= Old
Compare info. from disk (old) with new pat. info. (new)
New>Old
A
Yes
New<Old
DSPTR←DSPTR-1 insert new pat. after old pat. exit
B


B
Reached low limit? (DSPTR=LOW?)
yes
Exit. New pat. will be inserted before old pat.
No
Yes
Internal h halving convergence? (LOW +1=HIGH?)
Yes
Already tested low end (TSTLO=1?)
No
No
HIGH←DSPTR TSTHI←1 check lower half of previous range
DSPTR← DSPTR-1 check one lower
C
Halve the range DSPTR←LOW+(HIGH-LOW)/2
C
LOW←DSPTR TSTLO←1 check upper half of previous range
A
Reached high limit? (DSPTR=HIGH ?)
Yes
No
DSPTR← DSPTR+1 Exit
Internal halving convergence (LOW+1=HIGH?)
No
Yes
Already tested high end? (TSTHI=1?)
Yes
No
DSPTR←DSPTR+1 Check one higher
C
DSPTR←DSPTR+1 Exit

DESRT


Clear counters and flags for all SUBFILES
Initialize counters and flags for 1st (next) SUBFILE
Read 1st (next) buffer of this SUBFILE
1st (next) word of buffer=F26 pos. of pat. to be inserted ?
Yes
No
Same word = 7777
Yes
NO
Same word≤ HOSIZE ?
No
Yes
Move word from input to output buff
done with this Dir. ?
No
Yes
Done with this corebuff ?
No
YES
JMS DESRT4: write output buffer back out if there has been a change
Housekeep read and write address pointers
Was there exactly 1 deletion?
No
Yes
is there room in buff to put a 7777 at end ?
NO
Yes
Deposit 7777 in next word of buffer
JMS DESRT4
Is it necessary to write some more 7777 at end of the subfile ?
Yes
No
JMS DESRT 6
Done with last subfile ?
No
Yes
Return

INSRT


Put pat. F26 pos. in 1st word of core buffer.
For this directory, start reading disk from position at which he is to be inserted in directory.
Read disk into 2nd through last word of buffer.
Initialize pointers and counters for this new buffer.
1st next word of buffer=7777?
Yes
No
Done with directory?
No
Yes
Don't have to go any further. Write out buffer from 1st word to current word.
Return
Done with core buffer?
No
Yes
Write out 1st through second from last word of buffer. Put last word of buffer into 1st word of buffer.
Compute where to continue reading from on disk.

CHAPTER 3

REQUISITION ENTRY

The functions of the REQUISITION ENTRY (RE) program are summarized below:

1. Accept keyboard input including an identifying patient number, test types, and their identifying accession number, and the time of day the sample was (or perhaps will be) drawn.
2. Create entries in the patient files for test results to be supplied later.
3. Create new blocks of test data as needed.
4. Maintain a "running count" of active tests[1] in the patient files.
5. Assign new "day headers" and "pointers to today" in the patient files as needed.
6. Indicate active accession numbers[2] in the Requisition Index as new numbers are entered.
7. Sound an alarm to the hospital personnel in the form of a typewritten message on the calling terminal when the maximum number of blocks in the Patient Files is being approached.

3.1 GENERAL DESCRIPTION

Input is received from either a terminal or a card reader.

Output will be to the patient files on the disk and in the case of card reader input a verification sheet on either terminal or line printer.

I/O USED--Teletype and disk. The disk files used are the following:

File No.	20	PATIENT NAMES	Read only
	21	PATIENT NUMBERS	Read only
	26	SUBFILE DIRECTORY	Read and Write

[1] Active Test - a test which has been requested but whose results have not been entered into the patient files.

[2] Active accession number - an accession number which has been requested but whose corresponding test result(s) has not been entered into the patient files.

File No.	27	REQUISITION INDEX	Read and Write
	30	PATIENT TEST DATA	Read and Write
	34	DAD MAP	Read and Write
	35	ALPHA TEST TYPES	Read only
	36	TEST TYPE PARAMETERS	Read only
	42	BATTERY TABLE	Read only
	43	PACKAGE TABLE	Read only
	46	CARD POINTERS	Read only
	50	CARD FORMATS	Read only

The REQUISITION ENTRY program consists of two essentially disjointed parts, one for input from a terminal and the other for input from a card reader. The operator calls in RE and the terminal will print:

```
OUTPUT DEVICE, 1 TERMINAL, 2 LINE PRINTER
```

Typing 1 will cause overlay RF to be read in and the input will be from the calling terminal. RF will call overlays R1, R2, R3, R4, R5 and R6 in the course of operation.

Typing 2 will cause RE to continue and input will be from a card reader with interaction from the calling terminal. RE will call overlays R7, R8, R0, RG, RH and RI in the course of operation.

The operation of the two parts of the program is completely different although they have the same results and functions. See section 3.2 for terminal input and section 3.8 for card reader input.

3.2 FUNCTIONAL DESCRIPTION OF TERMINAL INPUT

After selection of terminal input, the computer will respond with the following message on the calling terminal:

```
ENTER REQUISITIONS AS:
TIME, PATIENT #, TEST TYPE(S), ACC #
```

Requisitions may now be typed. Examples of the Teletype input are given below. Computer responses are underscored.

```
*8:45AM, 12345, GLUC, BUN, 3245)
 JONES SAM R* YES)

*2:00P,2369P,BUN)
 #3457 SMITH SAM* )
```

```
*234567, ELEC, MA12, 570↵
 BROWN GEORGE* NO↵
```

Commas are the delimiters which separate each field. Redundant blanks are ignored[1]. A line of input is terminated by RETURN. An "*" typed by the computer signals that it is ready for a line of input.

Two options exist for the "time of day" entry:

1. The time may be typed in the format (h)h:mmT where hh is a decimal number in the range 1-12, mm is a decimal number in the range 00-59, and T=A(M) or P(M)[2].
2. The time may be ommited. Omission of the "time of day" field is assumed if all characters in the first field are decimal digits.

The number of test types which may be entered on one line is restricted only by the physical length of the line. If the accession number is omitted[3], the next available accession number is automatically assigned by the computer and printed.

Following the accession number, the computer responds by typing the patient name. At this point, two options exist:

1. The line may be accepted by typing Y(ES)
2. The line may be rejected by typing N(O)
   The computer responds by typing "** INPUT REJECTED" and ignores the line of input.

[1] Redundant blank = a blank before or after a field, but not within a field. Thus:

```
*ΔΔΔ1234ΔΔ,ΔΔΔΔGLUCΔΔΔ    is legal
*12Δ34,GLUC    is illegal
```

[2] Characters enclosed in parentheses are optional and may be omitted.

[3] If a test battery includes, say, 12 tests, the count is increased by 12_{10}.

Input is terminated when the first four non-redundant characters of a line are "STOP". The computer then prints the message "RE DONE? TTY IS FREE".

3.3 TEST PACKAGE

A "test type" is synonymous with the term "battery". The set of tests which make up a battery are all intended to be identified with the same accession number in the patient test data. A battery made up of N test types is said to be of length N. Each test type is itself a battery of length 1. E.g., the battery (test type) SMA-12 has a length of 15 and consists of 15 batteries (test types) all of length 1.

Two separate batteries typed on the same line have the same accession number in the test data. However, the same accession number typed on two different lines is not accepted so that a duplicate battery-accession number combination does not enter into the files.

A test package is a set of test batteries, with no duplicate accession numbers, which is placed in the patient test data for billing purposes.

A package is identified in the Test Data by a unique header followed by the set of batteries which combine to form the package.

Test batteries within packages are exceptions to the file structure since space may be allocated for a battery, although the battery has not yet been requested. A battery is flagged as unrequested by setting the two high order bits of the accession number to 1.

A package may be requested by typing the package name and the identifying patient number. The computer will then "lead the typist by the hand" in identifying each battery within the package and accepting the corresponding inputs.

The format for a package requisition is shown in the following example, where "ADM" is the name of an admittance package.

```
    i.       *12345, ADM↵
              <PKG> JONES SAM R.*Y ↵
   ii.         PKG REQ. ON 12/31 ↵
  iii.       TEST, TIME, ACC #;  TYPE I TO IGNORE A TEST ↵
   iv.       CBC  *1:15AM, 1234 ↵
             OK? YES↵
    v.       GLUC * I↵
   vi.       BUN * ↵
             #1235 OK? ↵
  vii.       VDRL  <2:35PM #6789> ↵
             .
             .
             .
 viii.       END OF PACKAGE. ↵
```

Explanation:

line i   The format for packages is similar to that for batteries, with the "time" and "accession number" fields omitted. The computer, upon detecting a code as a package name, signals it as such by typing "<PKG>" followed by the patient name. The line may accepted or rejected as usual. An initial search through the appropriate patient file for the requested package header is made. If the header exists, and at least one battery within the package has not been requested, the computer proceeds to line ii. If the header does not exist, space is allocated in the patient file for each battery in the package at this time; then the computer proceeds to line ii.

line ii A new package of the same name cannot be requested until all batteries within the old package have been requested for that patient, because a package cannot be uniquely recalled from a patient's file by the typist.

So long as there remains at least one unrequested battery within a package, the program recalls that package when requested and allows the typist to complete the remaining requisitions.

If a package is recalled from a file (that is, at least one battery remains to be requested for that package), the computer prints PKG REQ. ON XX/XX, where XX/XX is the date on which the package was first requested.

If it is necessary to lay out a new package, the program obtains operator approval before proceeding. Instead of the message printed in line ii, the following message is typed: NEW PKG. OK? (Y OR N)*

If Y is typed, the program then lays out the new package in the file and proceeds to line iii. If N is typed, the program prints **INPUT REJECTED, cycles back to the beginning, then prints an *.

Line iii The first half of line iii shows the format for entering data. The second half of the line presents the I (Ignore) option. If a requisition is not yet available, type I and a Carriage RETURN; the computer proceeds to the next line.

line iv. The computer types the first battery name and waits for input. The time and accession number for this test may now be typed in. Both fields are optional and both may be omitted simply by typing RETURN. This signals the computer to assign an accession number, as in line v. Each line of input is terminated by RETURN and accepted or rejected as usual after "OK?" is typed by the computer. Rejection of a line causes the computer to retype the line, preceded by the message "INPUT REJECTED." If the line is accepted, the computer places the entered information in the test data; it then proceeds to the next line.

line v. The GLUC test is not available. I is typed and the computer proceeds to the next line.

line vi. Here, the accession number has been assigned by the computer and printed. After acceptance of the number, the computer proceeds to the next line.

line vii. A test which has previously been requested within the package causes the computer to type the time and accession number enclosed by the characters "<" and ">" thus distinguishing it from a line of input. It then proceeds to the next line.

line viii. This line is typed after the last battery in the package. The computer is then ready to receive another line of input.

3.4 ERROR MESSAGES

Appropriate error messages are also printed. They are summarized below.

**ILLEGAL TIME FORMAT — Self-explanatory.

**NON-EXISTENT PATIENT # — Patient number not found in PATIENT NUMBER file, or non-numeric characters. May also indicate format for the time of day is incorrect.

**ILLEGAL TEST TYPE FORMAT

Either more than four characters were typed in the test type field or the test type field was omitted.

**NON-EXISTENT TEST TYPE XXXX

The test type format was legal, but XXXX could not be found in the ALPHA TEST TYPE file.

**DUPLICATION OF TEST TYPE XXXX

The program checks for duplication of test types which were entered on one line with the same accession number. Each test within a battery, as well as the battery itself, is checked for duplicate entries. Thus, if both the batteries MA12 and BGLU contain GLUC as one of the tests within the battery, all of the following entries are illegal:

```
...MA12, BGLU,...
...MA12, GLUC,...
...BGLU, GLUC,...
...GLUC, GLUC,...
...MA12, MA12,...
```

**ILLEGAL PACKAGE FORMAT

May indicate:

1. More than one package code on a line.
2. Intermixing of package codes and batteries (or single test types).
3. "Time of day" field or "accession number" field or both were entered. Both fields are illegal in the package format. (See section "Test Packages")

**ACC. # NOT UNIQUE

A pointer for the entered accession number already exists in the REQUISITION INDEX.

**DISK ERROR. TTY IS FREE.

A non-recoverable disk error was detected. The program then exits.

**SORRY

THERE ARE NO AVAILABLE ACC. NUMBERS AT THIS TIME. COMPUTER MUST EXIT

In attempting to assign an accession number, no available numbers were found. The program then exits.

**WARNING. DISK FILE IS NEARLY FULL

This message is printed when there are less than 20_{10} blocks available in file 30.

**DISK IS FULL. RE IS TERMINATED

There are no more available blocks in file 30. This message is printed only when the program tries to assign another block to a subfile. Thus, it is possible to enter more requisitions for other patients so long as there is available space in the subfiles for those patients.

**CANNOT ACCEPT THE LARGE VOLUME OF INPUT. SUGGEST ASSIGNING DIFFERENT ACC. NUMBERS TO SAMPLE

This is an improbable but possible circumstance. Three blocks of data must be created as a result of one line of requisitions for the circumstance to occur.

**TEST HAS BEEN DELETED

This message is printed if a package has been deleted during the time between a package request entry and the printing of the first battery of the package.

Except as noted above, "**INPUT REJECTED" will be printed after every message.

3.5 INPUT PROCESSING

Each line of input is processed as it is received. The steps in the processing are summarized as follows:

1. The time of day is coded to the format in which it will appear in the test data. If no time of day, the time is coded as binary 1's. An error message is typed if the time is out of range.

2. The patient number is converted to binary and an error message is typed if necessary. A search through the PATIENT NUMBER file is made for the number in question. If the number typed is not found, an error message is typed. If the number is found, the corresponding name is retrieved from the PATIENT NAME file by noting the relative position of the number in the PATIENT NUMBER file.

3. A table lookup is made for the binary equivalent of each test type. If the search is unsuccessful, an error message results. If a package code is typed, the computer proceeds in the manner described above.

4. If the accession number was entered, the corresponding pointer is read from the REQUISITION INDEX. If the pointer equals 7777, it is replaced temporarily by 7776. If the pointer is not 7777, an error message is typed.

   If the accession number was not entered, a search for the next available number in the REQUISITION INDEX is made. Then 7776 is placed into the REQUISITION INDEX.

   If the pointer to the SUBFILE DIRECTORY equals 7776, a block must be assigned to the patient. This requires a search through the DAD file for an available block. A pointer to the assigned block is then placed into the REQUISITION INDEX and SUBFILE DIRECTORY.

5. If no errors are detected, the patient name is printed. If NO is typed following the name, the line is rejected, and the pointer in the REQUISITION INDEX is again set to 7777. If the line is accepted the information may now be placed in the PATIENT TEST DATA. The "current day" header is found by referencing the pointer in the first block. "Today's" header is placed in the subfile if necessary, and the pointer to the current day is updated. The "active test" count is increased accordingly.[1] Each battery of length N which was typed results in 3 + (Nx3) words to be created in the test data. The length of each battery is found by examining the TEST TYPE TABLE. The test type (battery number), accession number, and time are placed in words N, N+1, and N+2. Each subsequent three-word entry, consisting of test status, technician code, and results, is set to zero, except that the "last test in battery" bit is set to one in the last three-word entry. Word 2 of the first block is set to the relative position of its pointer in the SUBFILE DIRECTORY. The pointer in the REQUISITION INDEX, temporarily set to 7776, is now replaced by the pointer to the first block of the patient's subfile.

The program assigns new blocks to the patient's subfiles as needed. Data is placed into each block up to the 256th word, and a pointer to the next block is placed there if a new block must be created. The last word in the file is followed by a logical end of file marker (7777), and the last word of that block is set to 7777. Note that the logical end of file may not exist if the last word happens to fall in the 255th word of the block. The DAD file is then updated and written back onto the disk. If the number of available blocks is less than or equal to 20, a message is printed on the terminal. If no available blocks remain on the disk, an appropriate message follows.

CALLING PROGRAMS - The program is called by operator request only.

PROGRAMS CALLED - Except for program overlays, no other program is called.

[1] If a test battery includes 12 tests, the count is increased by 12_{10}.

ERRORS - Input cannot be edited with the Requisition Entry program. If the computer detects a syntax error, an error message is typed, and the current line of input is ignored. Errors detected by the user can be corrected by calling the "DELETE DATA" program, and deleting the offending tests, then recalling the Requisition Entry program.

3.6 ASSEMBLY INSTRUCTIONS

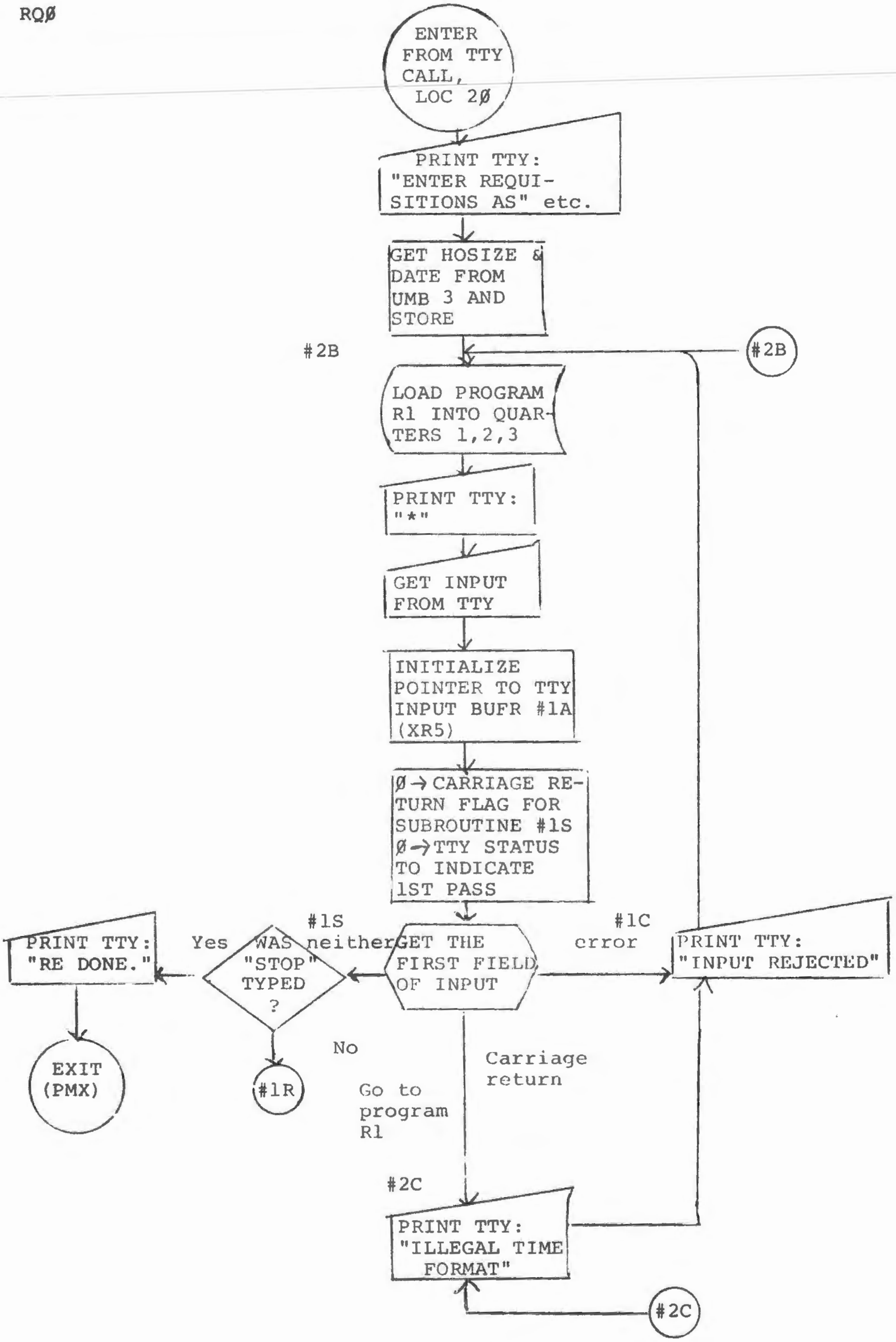
The source exists in seven DIAL sections: REQENT, REQENT-1, REQENT-2, REQENT-3, REQENT-4, REQENT-5, REQTXT. REQENT-1, -2, -3, -4, and -5 are first added to REQENT using the AP DIAL command. These combined sources chain to REQTXT. The binaries are saved in the following manner.

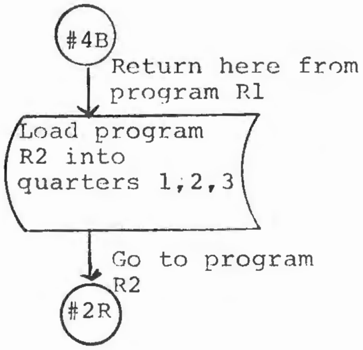
+ = add
→ = chain

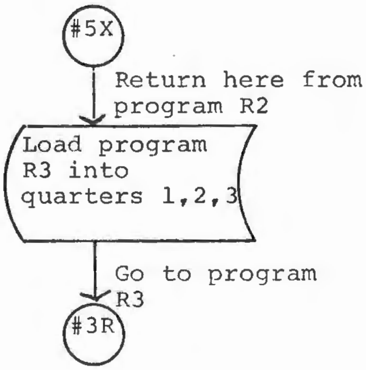
	first	number	
	binary	of	name on
Source	block	blocks	start up tape
REQENT+REQENT1→REQTXT	2	3	R1
REQENT+REQENT2→REQTXT	2	2	R2
REQENT+REQENT3→REQTXT	2	3	R3
REQENT+REQENT4→REQTXT	2	3	R4
REQENT+REQENT5→REQTXT	2	3	R5
REQENT+REQENT5→REQTXT	1	1Ø	RE

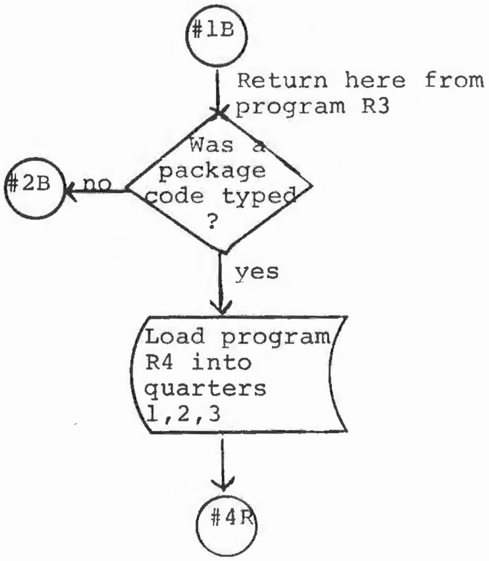
3.7 FLOW CHARTS FOR REQUISITION ENTRY (TERMINAL INPUT)

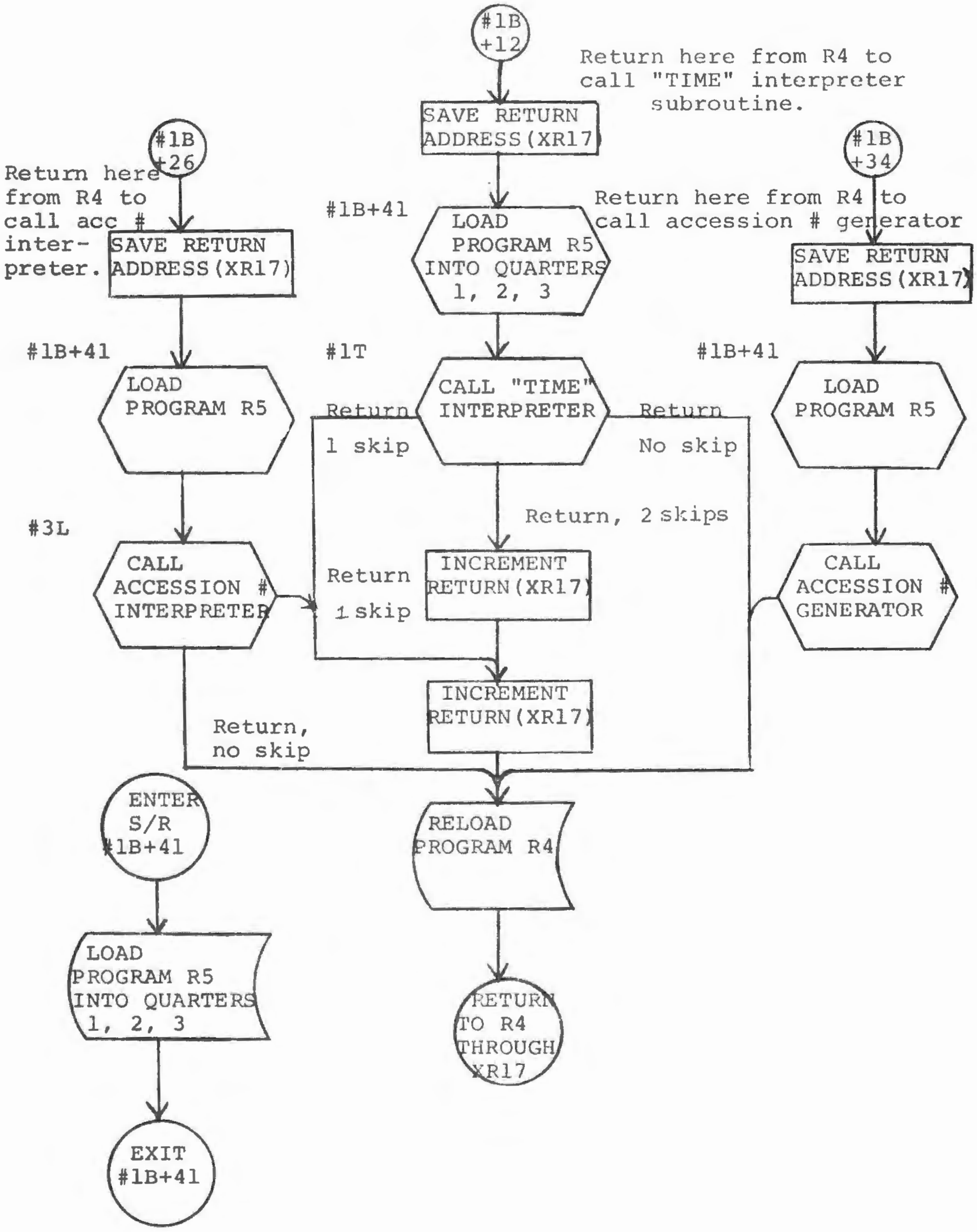
RQØ

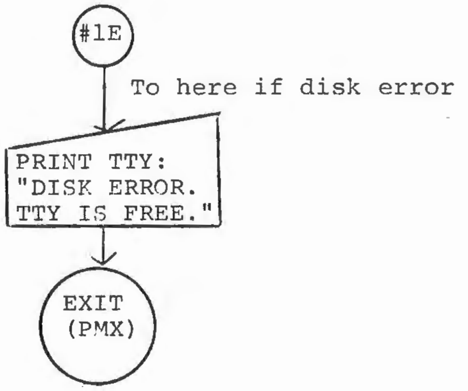

ENTER FROM TTY CALL, LOC 2Ø
PRINT TTY: "ENTER REQUISITIONS AS" etc.
GET HOSIZE & DATE FROM UMB 3 AND STORE
#2B
#2B
LOAD PROGRAM R1 INTO QUARTERS 1,2,3
PRINT TTY: "*"
GET INPUT FROM TTY
INITIALIZE POINTER TO TTY INPUT BUFR #1A (XR5)
Ø→CARRIAGE RETURN FLAG FOR SUBROUTINE #1S Ø→TTY STATUS TO INDICATE 1ST PASS
#1S
GET THE FIRST FIELD OF INPUT
#1C
error
PRINT TTY: "INPUT REJECTED"
neither
WAS "STOP" TYPED ?
Yes
PRINT TTY: "RE DONE."
EXIT (PMX)
No
#1R
Go to program R1
Carriage return
#2C
PRINT TTY: "ILLEGAL TIME FORMAT"
#2C

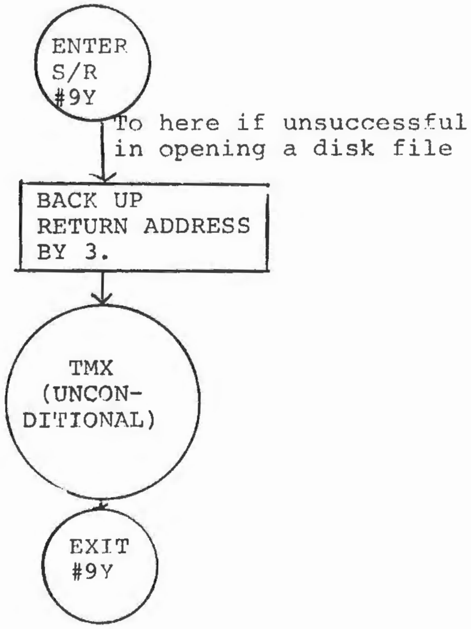

#4B
Return here from program R1
Load program R2 into quarters 1,2,3
Go to program R2
#2R

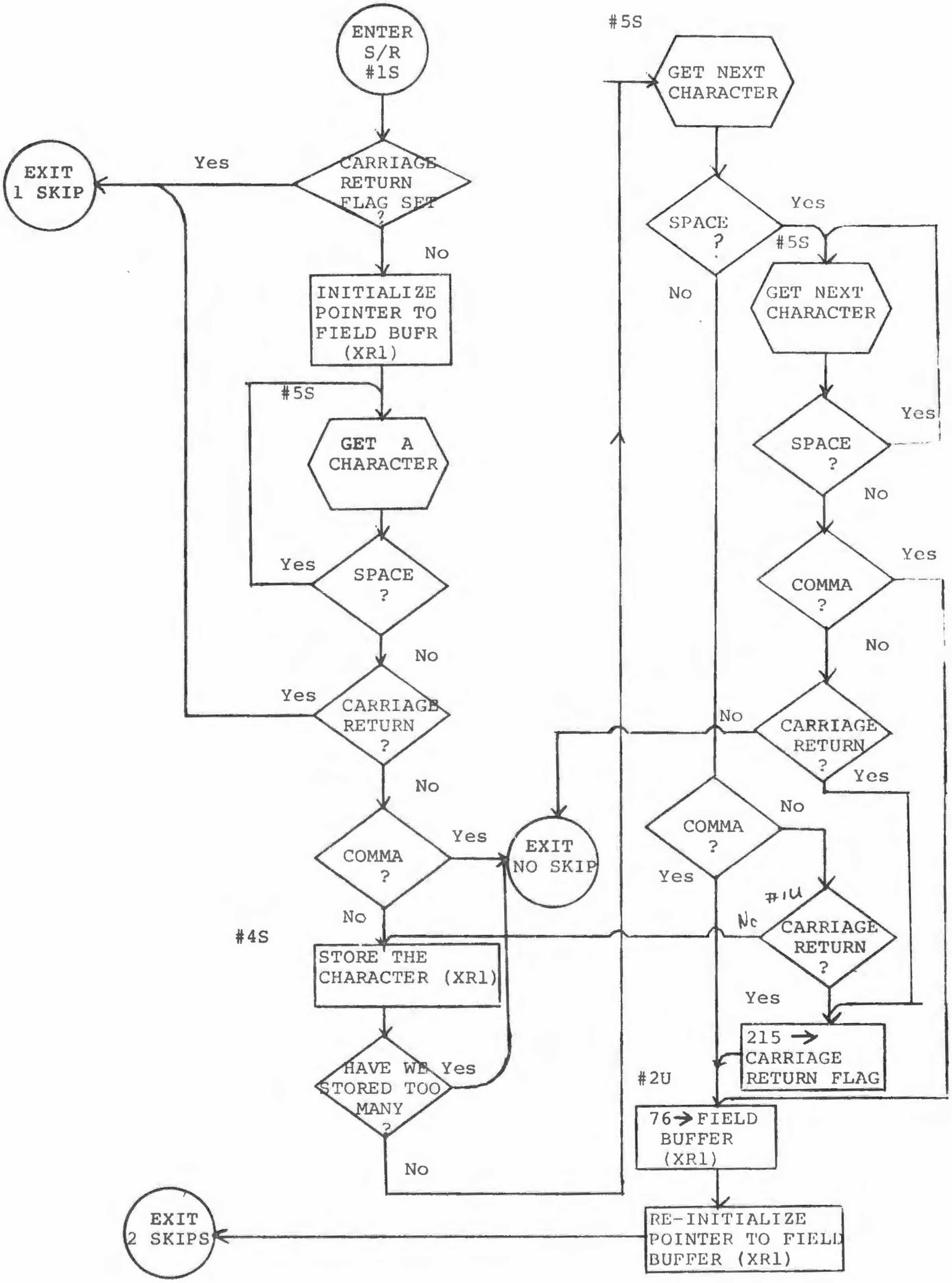

#5X
Return here from program R2
Load program R3 into quarters 1,2,3
Go to program R3
#3R

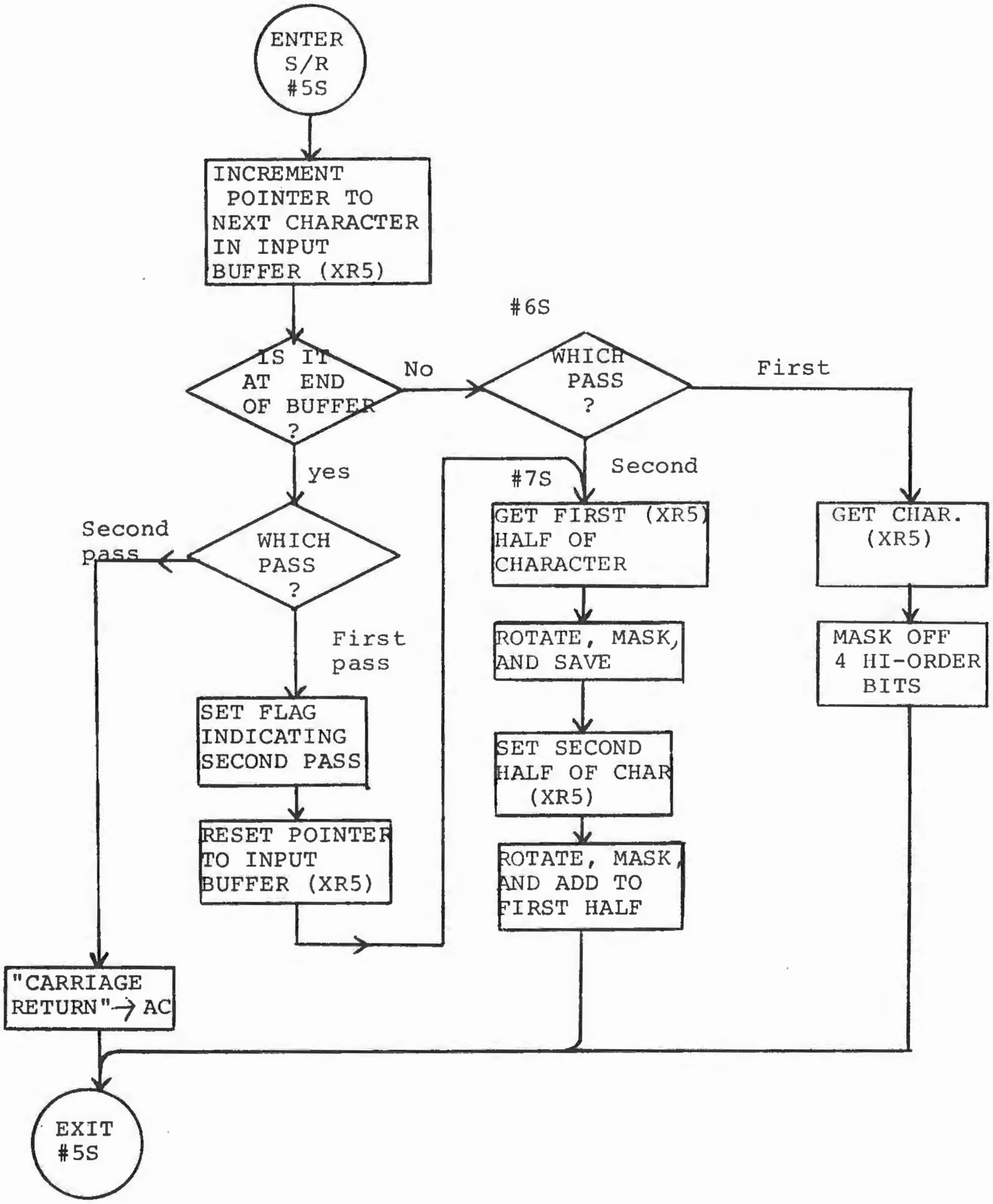

#1B
Return here from program R3
Was a package code typed ?
no
#2B
yes
Load program R4 into quarters 1,2,3
#4R

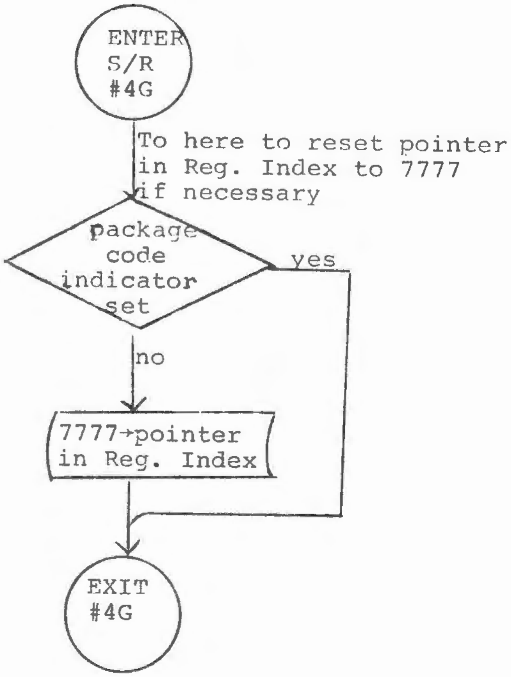

#1B +12
Return here from R4 to call "TIME" interpreter subroutine.
SAVE RETURN ADDRESS(XR17)
#1B +26
Return here from R4 to call acc # inter-preter.
SAVE RETURN ADDRESS(XR17)
#1B +34
Return here from R4 to call accession # generator
SAVE RETURN ADDRESS(XR17)
#1B+41
LOAD PROGRAM R5 INTO QUARTERS 1, 2, 3
#1B+41
LOAD PROGRAM R5
#1T
CALL "TIME" INTERPRETER
Return 1 skip
Return No skip
Return, 2 skips
#1B+41
LOAD PROGRAM R5
#3L
CALL ACCESSION # INTERPRETER
Return 1 skip
INCREMENT RETURN(XR17)
CALL ACCESSION # GENERATOR
INCREMENT RETURN(XR17)
Return, no skip
ENTER S/R #1B+41
RELOAD PROGRAM R4
LOAD PROGRAM R5 INTO QUARTERS 1, 2, 3
RETURN TO R4 THROUGH XR17
EXIT #1B+41


#1E
To here if disk error
PRINT TTY: "DISK ERROR. TTY IS FREE."
EXIT (PMX)

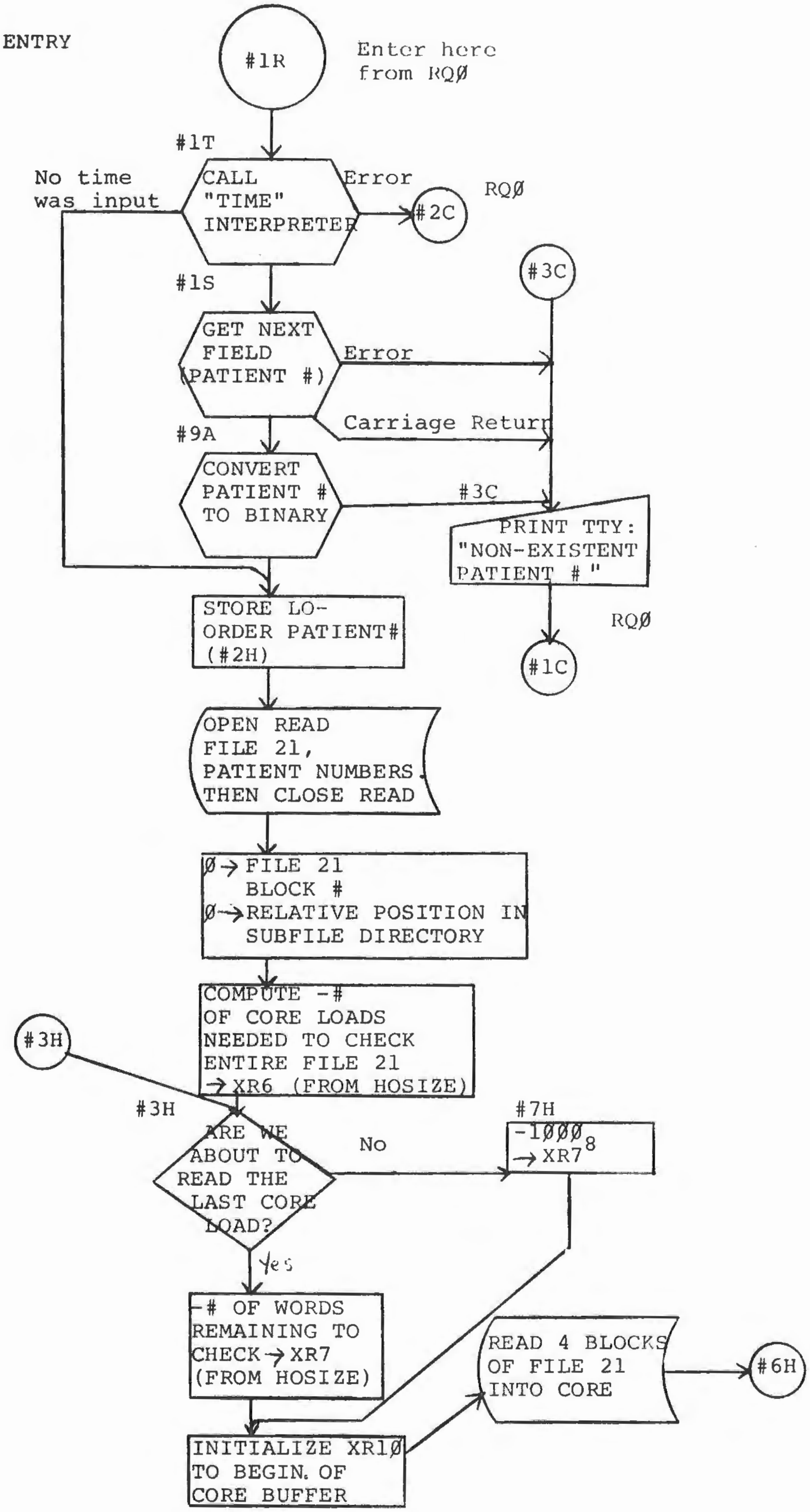

ENTER S/R #9Y
To here if unsuccessful in opening a disk file
BACK UP RETURN ADDRESS BY 3.
TMX (UNCONDITIONAL)
EXIT #9Y

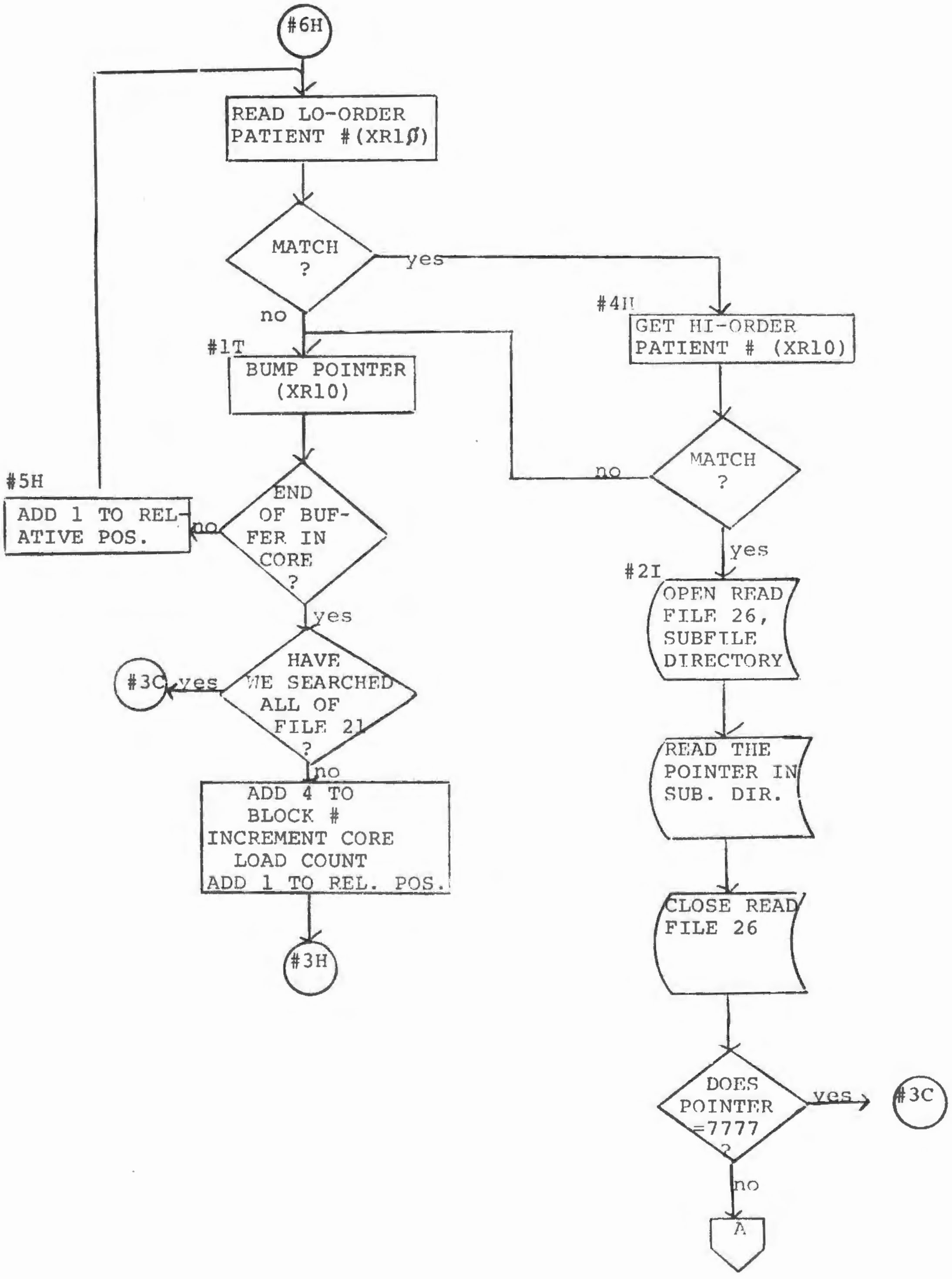

ENTER S/R #1S
CARRIAGE RETURN FLAG SET ?
Yes
EXIT 1 SKIP
No
INITIALIZE POINTER TO FIELD BUFR (XR1)
#5S
GET A CHARACTER
SPACE ?
Yes
No
CARRIAGE RETURN ?
Yes
No
COMMA ?
Yes
EXIT NO SKIP
No
#4S
STORE THE CHARACTER (XR1)
HAVE WE STORED TOO MANY ?
Yes
No
EXIT 2 SKIPS
#5S
GET NEXT CHARACTER
SPACE ?
Yes
No
#5S
GET NEXT CHARACTER
SPACE ?
Yes
No
COMMA ?
Yes
No
CARRIAGE RETURN ?
No
Yes
COMMA ?
No
Yes
#1U
No
CARRIAGE RETURN ?
Yes
215 → CARRIAGE RETURN FLAG
#2U
76→FIELD BUFFER (XR1)
RE-INITIALIZE POINTER TO FIELD BUFFER (XR1)

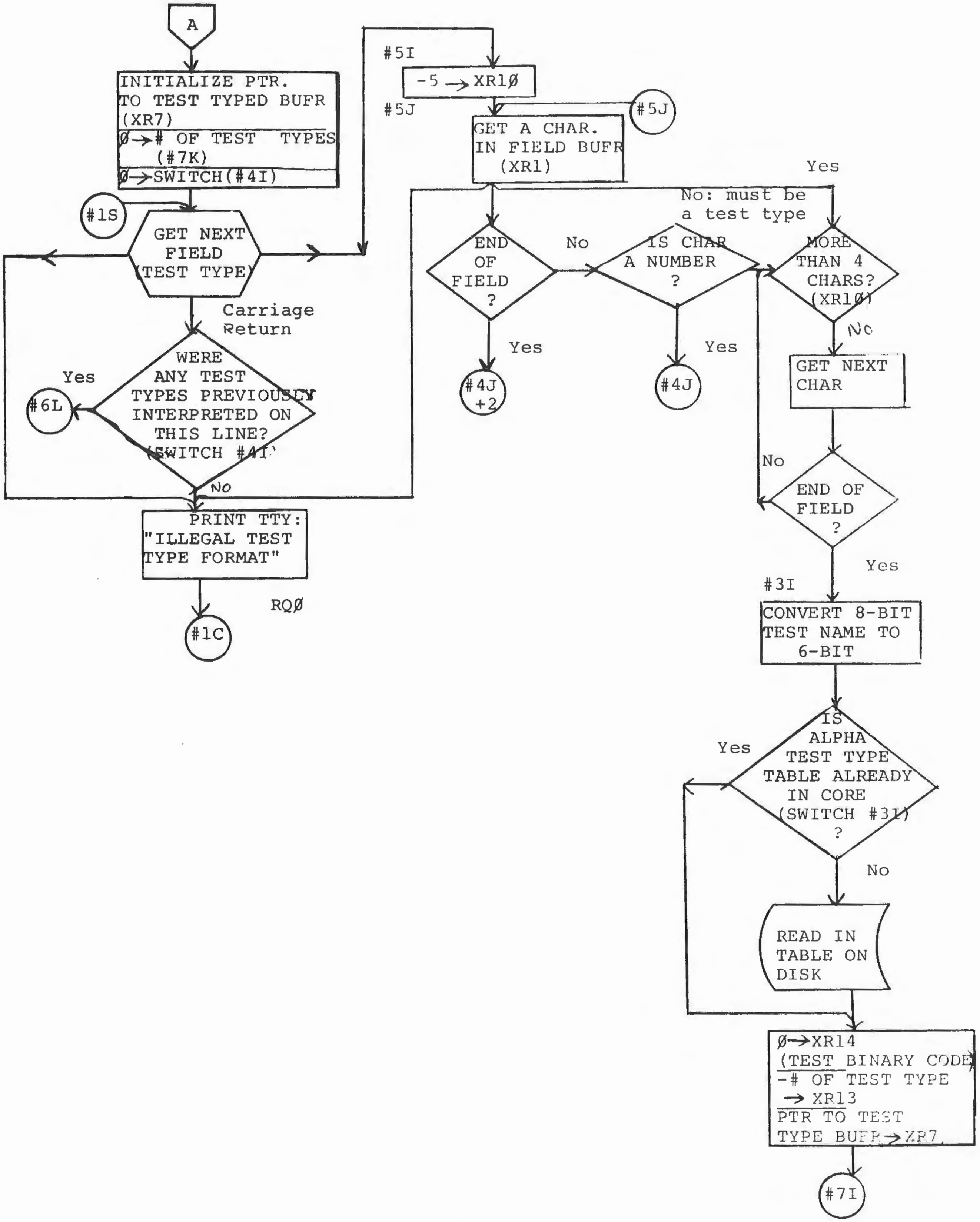

ENTER S/R #5S
INCREMENT POINTER TO NEXT CHARACTER IN INPUT BUFFER (XR5)
#6S
IS IT AT END OF BUFFER ?
No
WHICH PASS ?
First
yes
Second
#7S
Second pass
WHICH PASS ?
GET FIRST (XR5) HALF OF CHARACTER
GET CHAR. (XR5)
First pass
ROTATE, MASK, AND SAVE
MASK OFF 4 HI-ORDER BITS
SET FLAG INDICATING SECOND PASS
SET SECOND HALF OF CHAR (XR5)
RESET POINTER TO INPUT BUFFER (XR5)
ROTATE, MASK, AND ADD TO FIRST HALF
"CARRIAGE RETURN"→ AC
EXIT #5S

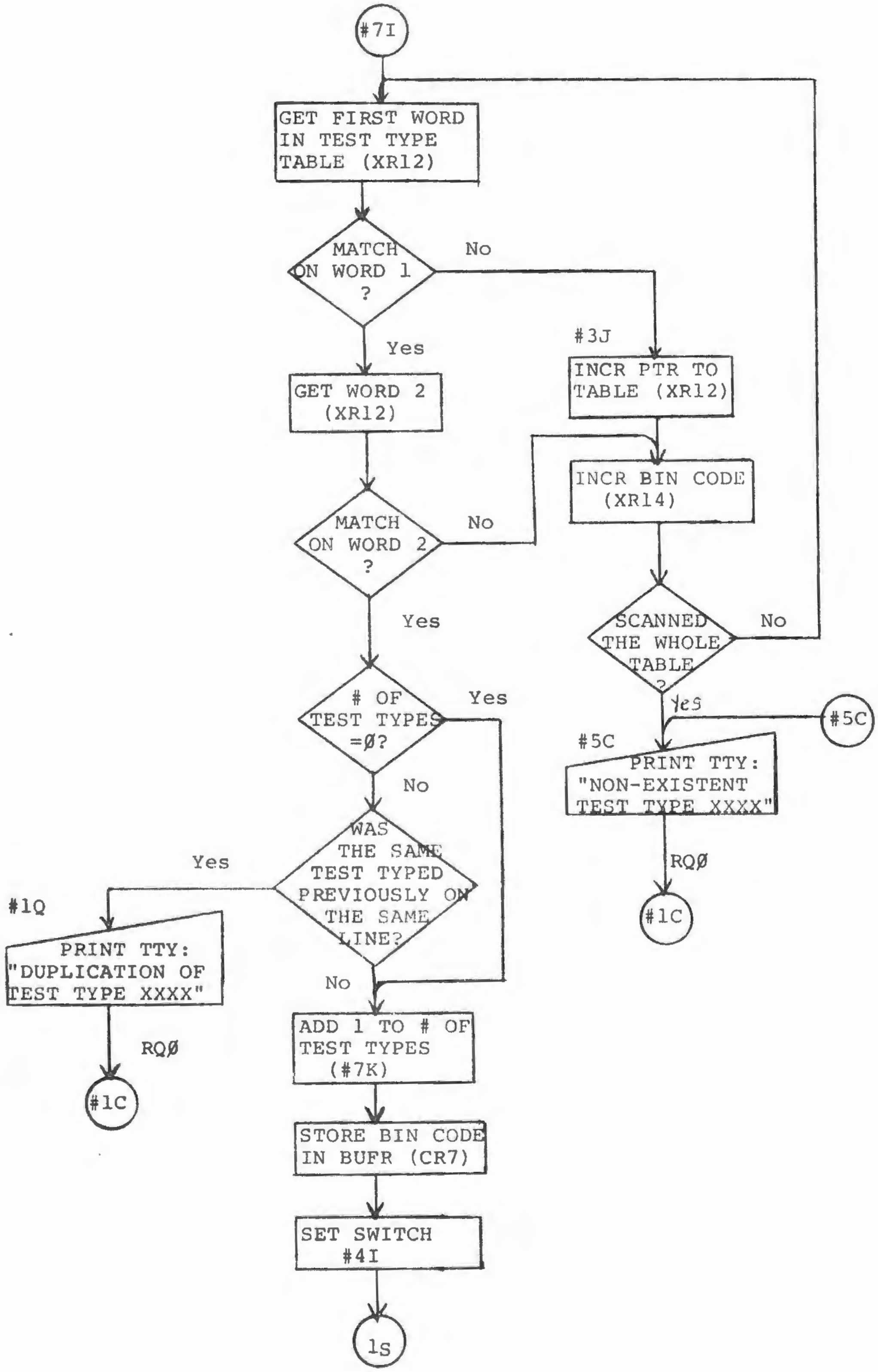

ENTER
S/R
#4G
To here to reset pointer
in Reg. Index to 7777
if necessary
package
code
indicator
set
yes
no
7777→pointer
in Reg. Index
EXIT
#4G

REQUISITION ENTRY
R1

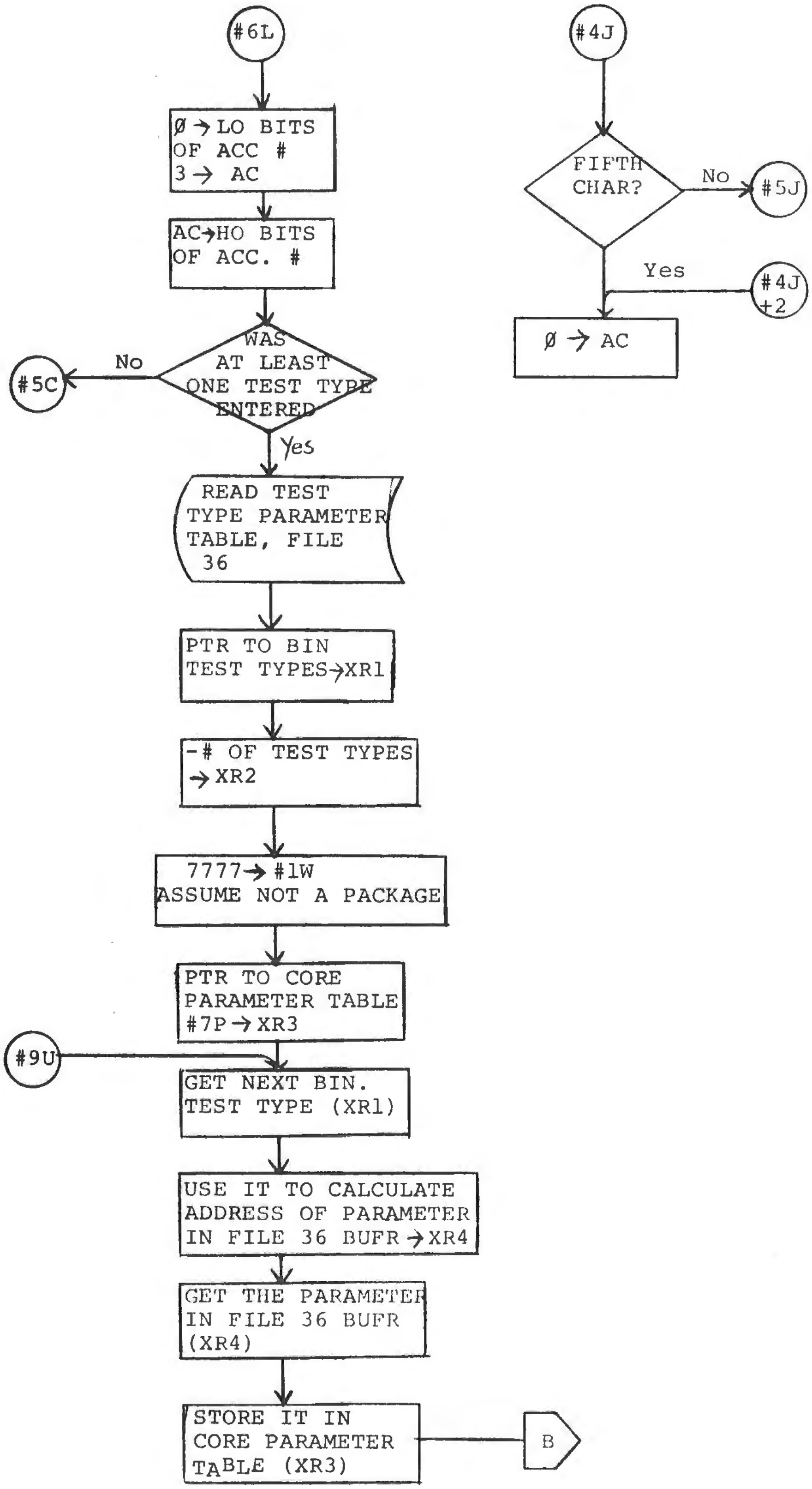


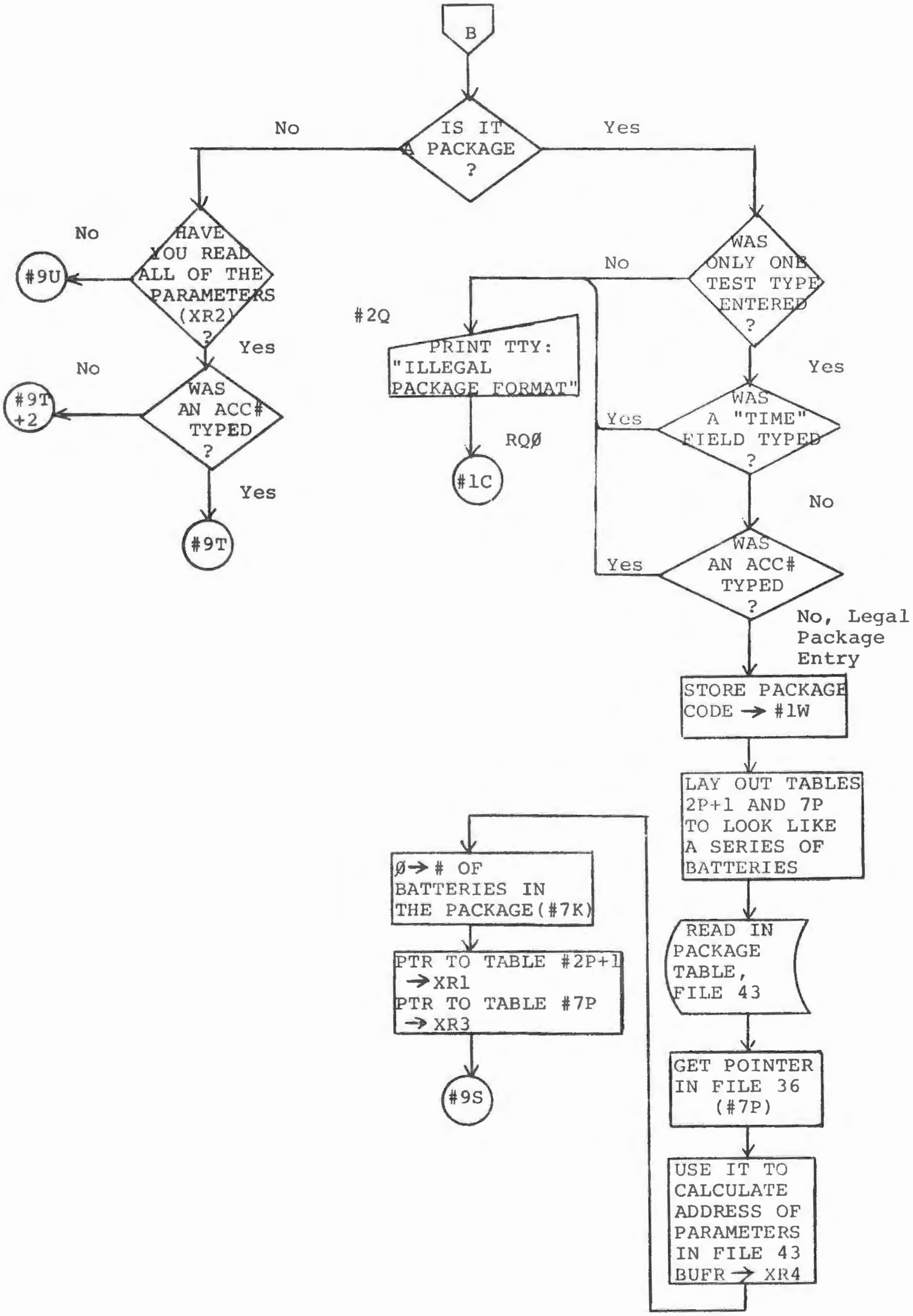

#6H
READ LO-ORDER PATIENT #(XR1Ø)
MATCH ?
yes
no
#4H
GET HI-ORDER PATIENT # (XR10)
#1T
BUMP POINTER (XR10)
MATCH ?
no
yes
#5H
ADD 1 TO RELATIVE POS.
no
END OF BUFFER IN CORE ?
yes
#2I
OPEN READ FILE 26, SUBFILE DIRECTORY
#3C
yes
HAVE WE SEARCHED ALL OF FILE 21 ?
no
READ THE POINTER IN SUB. DIR.
ADD 4 TO BLOCK # INCREMENT CORE LOAD COUNT ADD 1 TO REL. POS.
CLOSE READ FILE 26
#3H
DOES POINTER =7777 ?
yes
#3C
no
A

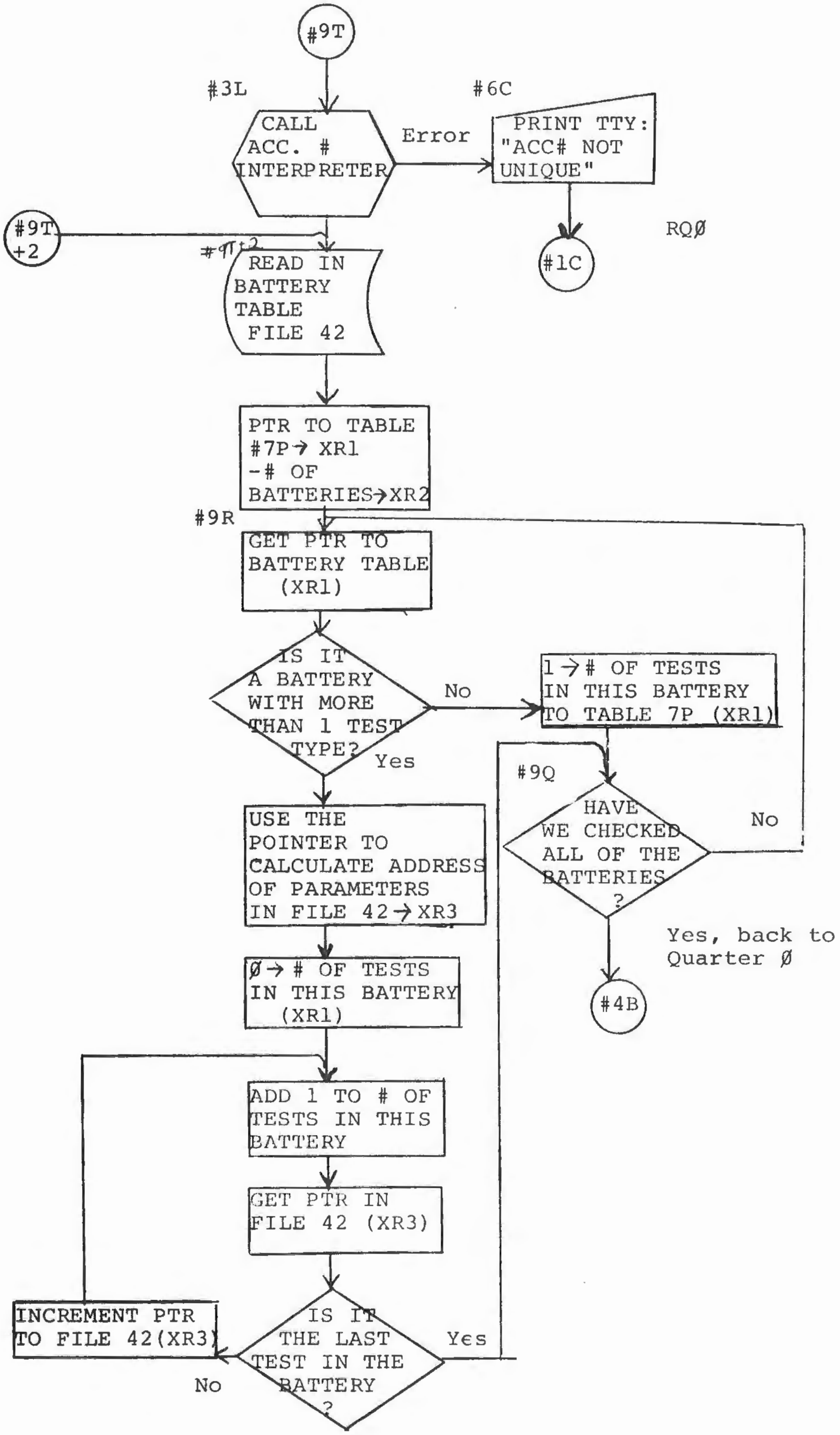

A
INITIALIZE PTR. TO TEST TYPED BUFR (XR7)
Ø→# OF TEST TYPES (#7K)
Ø→SWITCH(#4I)
#1S
GET NEXT FIELD (TEST TYPE)
Carriage Return
WERE ANY TEST TYPES PREVIOUSLY INTERPRETED ON THIS LINE? (SWITCH #4I)
Yes
#6L
No
PRINT TTY: "ILLEGAL TEST TYPE FORMAT"
RQØ
#1C
#5I
-5 → XR1Ø
#5J
#5J
GET A CHAR. IN FIELD BUFR (XR1)
END OF FIELD ?
Yes
#4J +2
No
IS CHAR A NUMBER ?
Yes
#4J
No: must be a test type
Yes
MORE THAN 4 CHARS? (XR1Ø)
No
GET NEXT CHAR
No
END OF FIELD ?
Yes
#3I
CONVERT 8-BIT TEST NAME TO 6-BIT
IS ALPHA TEST TYPE TABLE ALREADY IN CORE (SWITCH #3I) ?
Yes
No
READ IN TABLE ON DISK
Ø→XR14 (TEST BINARY CODE)
-# OF TEST TYPE → XR13
PTR TO TEST TYPE BUFR→XR7
#7I

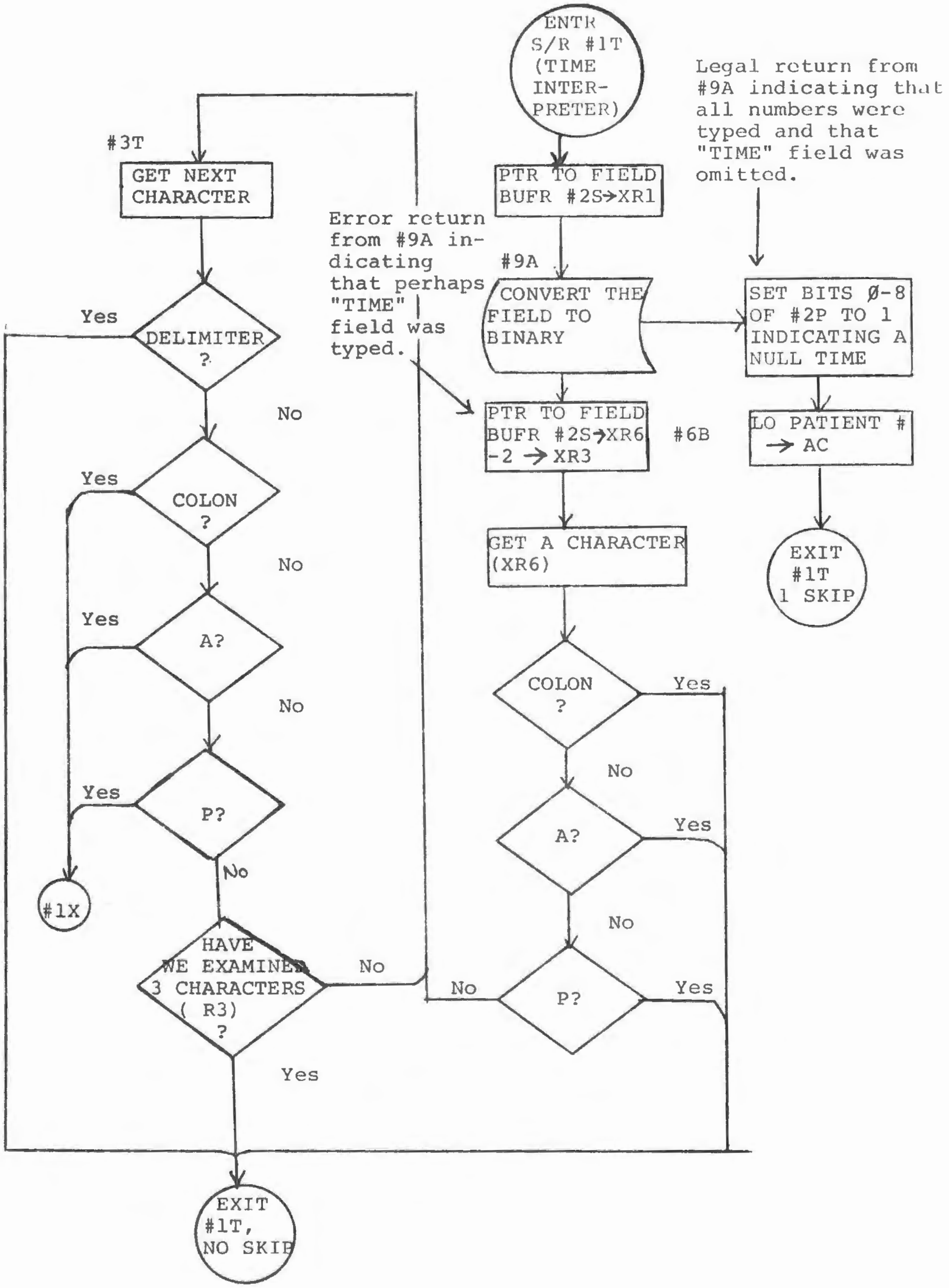


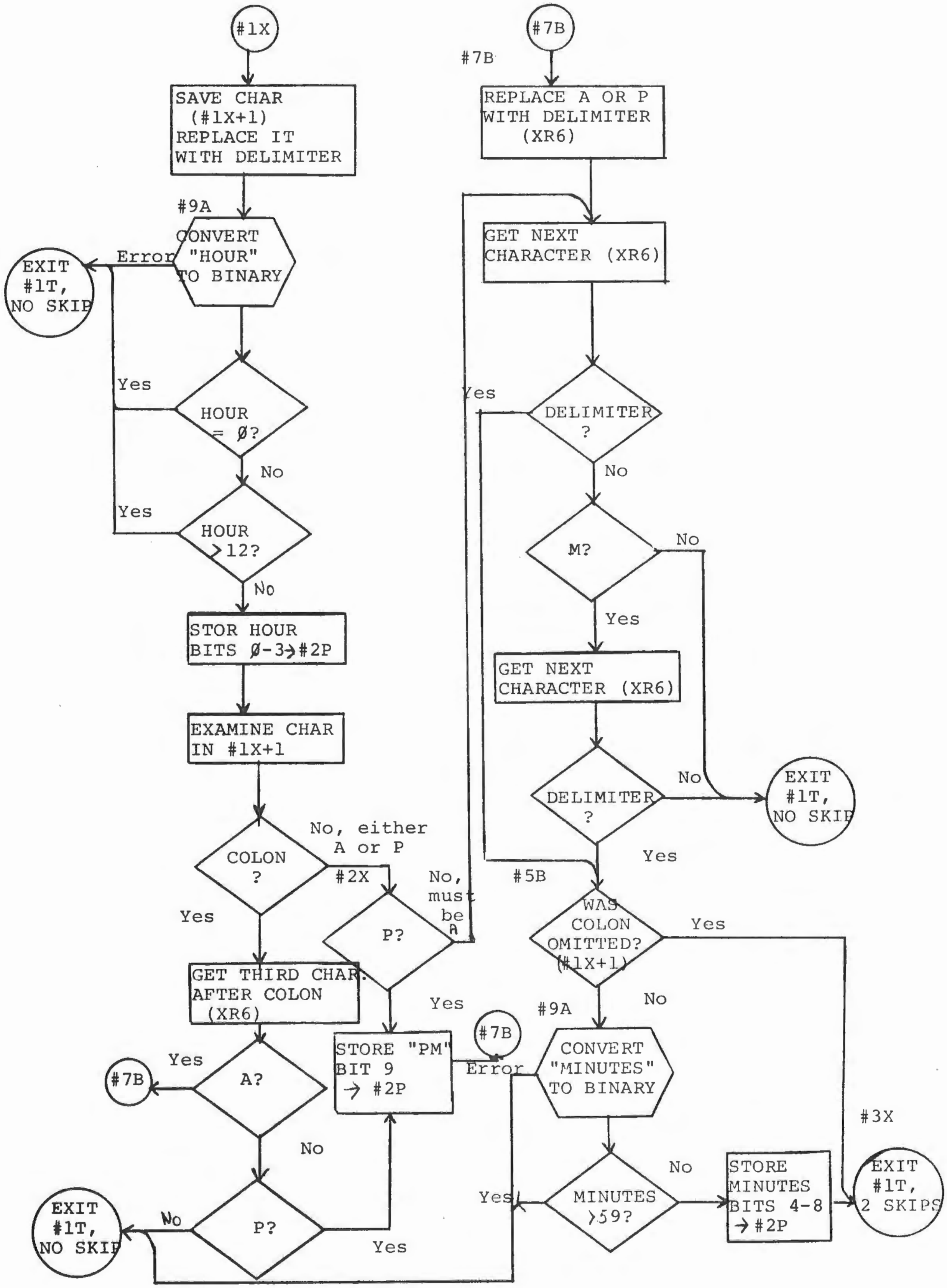
#7I
GET FIRST WORD IN TEST TYPE TABLE (XR12)
MATCH ON WORD 1 ?
No
Yes
#3J
INCR PTR TO TABLE (XR12)
GET WORD 2 (XR12)
INCR BIN CODE (XR14)
MATCH ON WORD 2 ?
No
Yes
SCANNED THE WHOLE TABLE ?
No
Yes
#5C
#5C
PRINT TTY: "NON-EXISTENT TEST TYPE XXXX"
RQØ
#1C
# OF TEST TYPES =Ø?
Yes
No
WAS THE SAME TEST TYPED PREVIOUSLY ON THE SAME LINE?
Yes
No
#1Q
PRINT TTY: "DUPLICATION OF TEST TYPE XXXX"
RQØ
#1C
ADD 1 TO # OF TEST TYPES (#7K)
STORE BIN CODE IN BUFR (CR7)
SET SWITCH #4I
1S

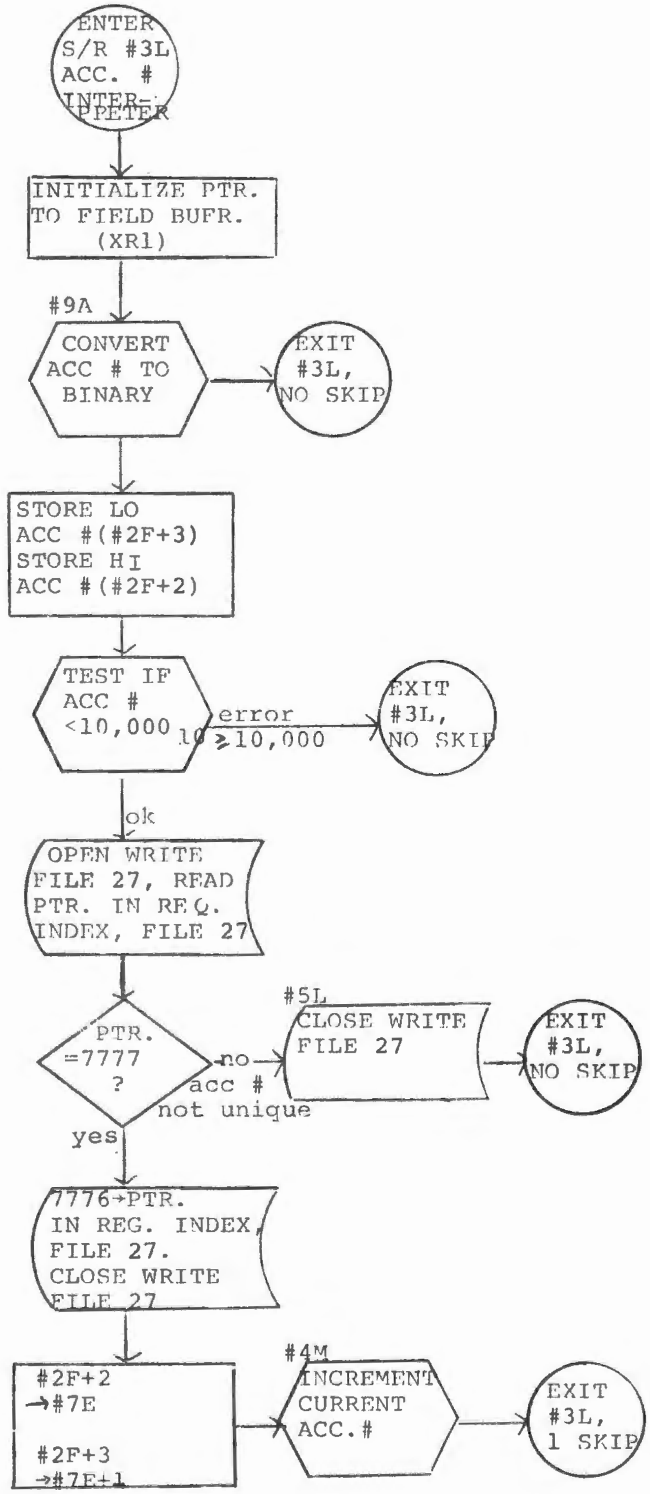

#6L
Ø → LO BITS OF ACC # 3 → AC
AC → HO BITS OF ACC. #
WAS AT LEAST ONE TEST TYPE ENTERED
No
#5C
Yes
READ TEST TYPE PARAMETER TABLE, FILE 36
PTR TO BIN TEST TYPES → XR1
-# OF TEST TYPES → XR2
7777 → #1W ASSUME NOT A PACKAGE
PTR TO CORE PARAMETER TABLE #7P → XR3
#9U
GET NEXT BIN. TEST TYPE (XR1)
USE IT TO CALCULATE ADDRESS OF PARAMETER IN FILE 36 BUFR → XR4
GET THE PARAMETER IN FILE 36 BUFR (XR4)
STORE IT IN CORE PARAMETER TABLE (XR3)
B
#4J
FIFTH CHAR?
No
#5J
Yes
#4J +2
Ø → AC

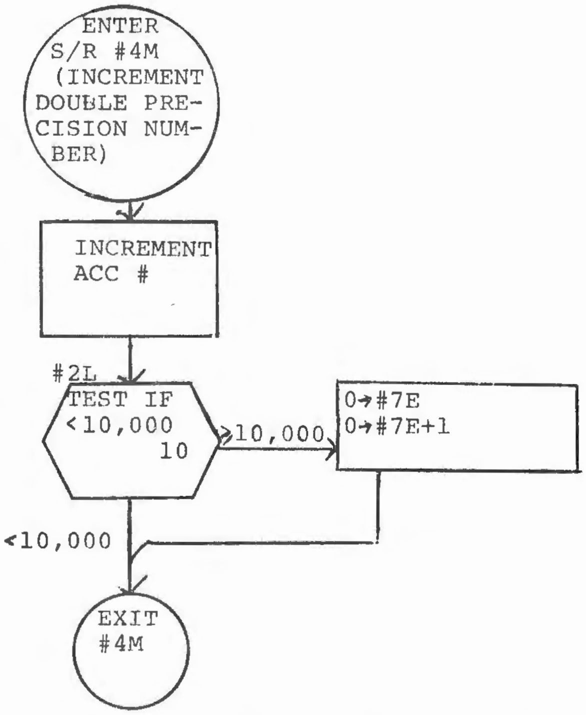

B
IS IT A PACKAGE ?
No
Yes
HAVE YOU READ ALL OF THE PARAMETERS (XR2) ?
No
#9U
Yes
WAS AN ACC# TYPED ?
No
#9T +2
Yes
#9T
WAS ONLY ONE TEST TYPE ENTERED ?
No
#2Q
PRINT TTY: "ILLEGAL PACKAGE FORMAT"
RQØ
#1C
Yes
WAS A "TIME" FIELD TYPED ?
Yes
No
WAS AN ACC# TYPED ?
Yes
No, Legal Package Entry
STORE PACKAGE CODE → #1W
LAY OUT TABLES 2P+1 AND 7P TO LOOK LIKE A SERIES OF BATTERIES
READ IN PACKAGE TABLE, FILE 43
GET POINTER IN FILE 36 (#7P)
USE IT TO CALCULATE ADDRESS OF PARAMETERS IN FILE 43 BUFR → XR4
Ø→ # OF BATTERIES IN THE PACKAGE(#7K)
PTR TO TABLE #2P+1 →XR1
PTR TO TABLE #7P →XR3
#9S

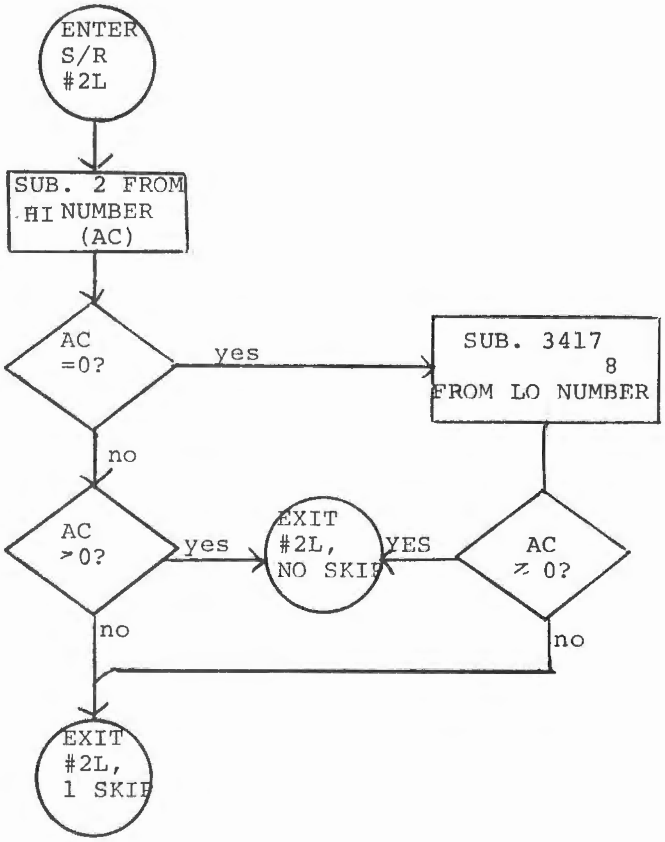

#9T
#3L
CALL
ACC. #
INTERPRETER
Error
#6C
PRINT TTY:
"ACC# NOT
UNIQUE"
RQØ
#1C
#9T
+2
#9T+2
READ IN
BATTERY
TABLE
FILE 42
PTR TO TABLE
#7P→ XR1
-# OF
BATTERIES→XR2
#9R
GET PTR TO
BATTERY TABLE
(XR1)
IS IT
A BATTERY
WITH MORE
THAN 1 TEST
TYPE?
No
Yes
1→# OF TESTS
IN THIS BATTERY
TO TABLE 7P (XR1)
#9Q
HAVE
WE CHECKED
ALL OF THE
BATTERIES
?
No
Yes, back to
Quarter Ø
#4B
USE THE
POINTER TO
CALCULATE ADDRESS
OF PARAMETERS
IN FILE 42→XR3
Ø→# OF TESTS
IN THIS BATTERY
(XR1)
ADD 1 TO # OF
TESTS IN THIS
BATTERY
GET PTR IN
FILE 42 (XR3)
INCREMENT PTR
TO FILE 42(XR3)
IS IT
THE LAST
TEST IN THE
BATTERY
?
Yes
No

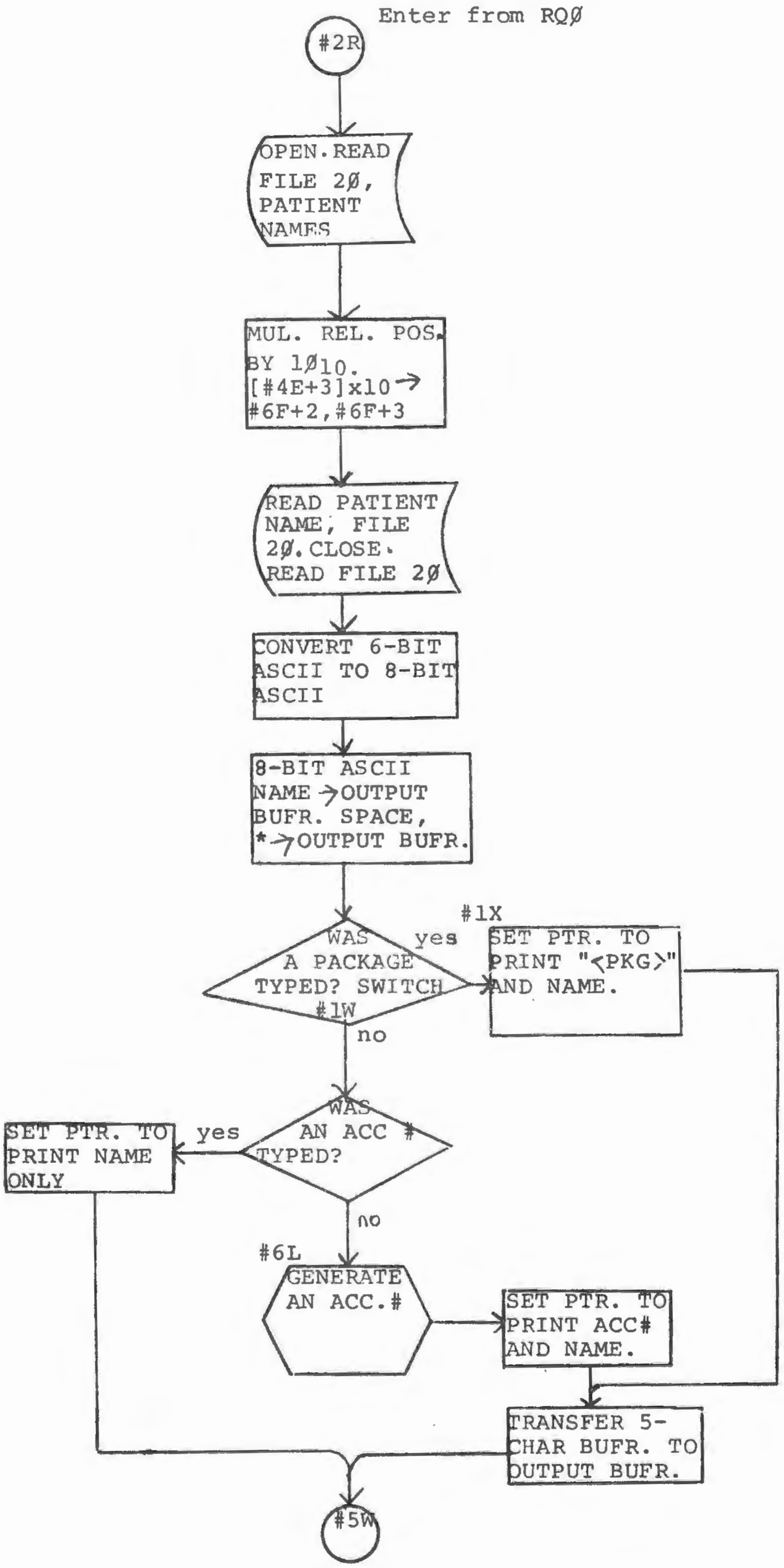

ENTR S/R #1T (TIME INTER-PRETER)
PTR TO FIELD BUFR #2S→XR1
#9A
CONVERT THE FIELD TO BINARY
Legal return from #9A indicating that all numbers were typed and that "TIME" field was omitted.
SET BITS Ø-8 OF #2P TO 1 INDICATING A NULL TIME
LO PATIENT # → AC
EXIT #1T 1 SKIP
Error return from #9A in-dicating that perhaps "TIME" field was typed.
PTR TO FIELD BUFR #2S→XR6 -2 → XR3
#6B
GET A CHARACTER (XR6)
COLON ?
Yes
No
A?
Yes
No
P?
Yes
No
#3T
GET NEXT CHARACTER
DELIMITER ?
Yes
No
COLON ?
Yes
No
A?
Yes
No
P?
Yes
No
#1X
HAVE WE EXAMINED 3 CHARACTERS (R3) ?
No
Yes
EXIT #1T, NO SKIP

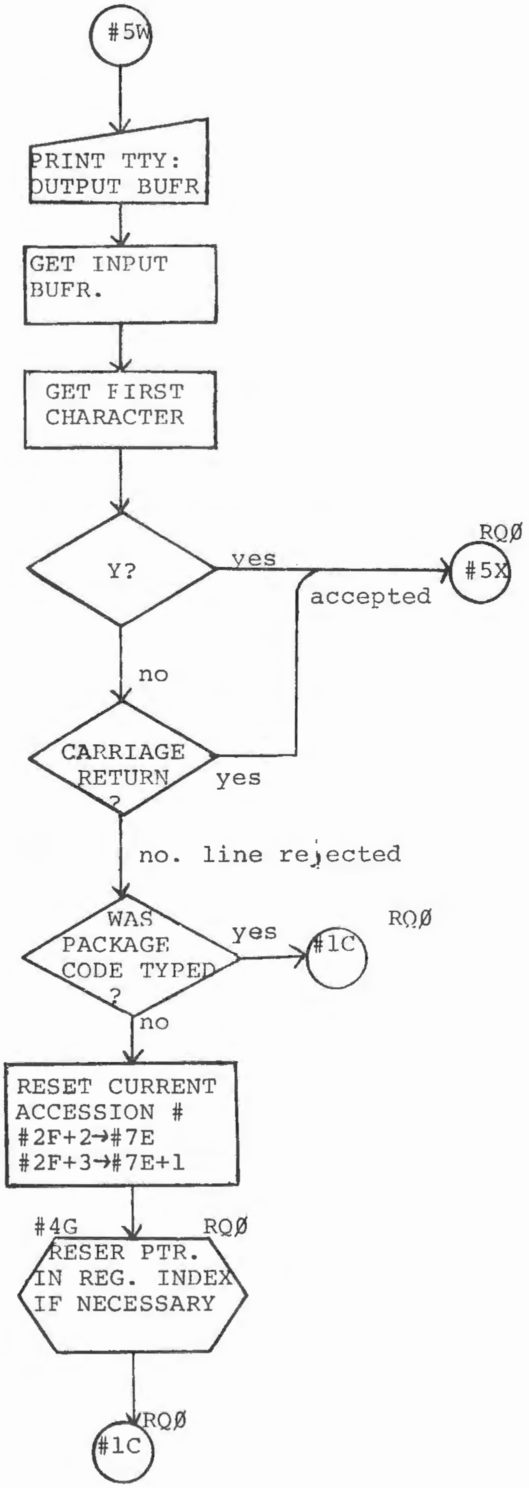

#1X
SAVE CHAR (#1X+1) REPLACE IT WITH DELIMITER
#9A
CONVERT "HOUR" TO BINARY
Error
EXIT #1T, NO SKIP
Yes
HOUR = Ø?
No
Yes
HOUR >12?
No
STOR HOUR BITS Ø-3→#2P
EXAMINE CHAR IN #1X+1
COLON ?
No, either A or P
#2X
P?
No, must be A
Yes
GET THIRD CHAR. AFTER COLON (XR6)
Yes
A?
#7B
No
P?
No
EXIT #1T, NO SKIP
Yes
STORE "PM" BIT 9 → #2P
#7B
#7B
REPLACE A OR P WITH DELIMITER (XR6)
GET NEXT CHARACTER (XR6)
Yes
DELIMITER ?
No
M?
No
Yes
GET NEXT CHARACTER (XR6)
DELIMITER ?
No
EXIT #1T, NO SKIP
Yes
#5B
WAS COLON OMITTED? (#1X+1)
Yes
No
#9A
#7B
Error
CONVERT "MINUTES" TO BINARY
#3X
Yes
MINUTES >59?
No
STORE MINUTES BITS 4-8 →#2P
EXIT #1T, 2 SKIPS

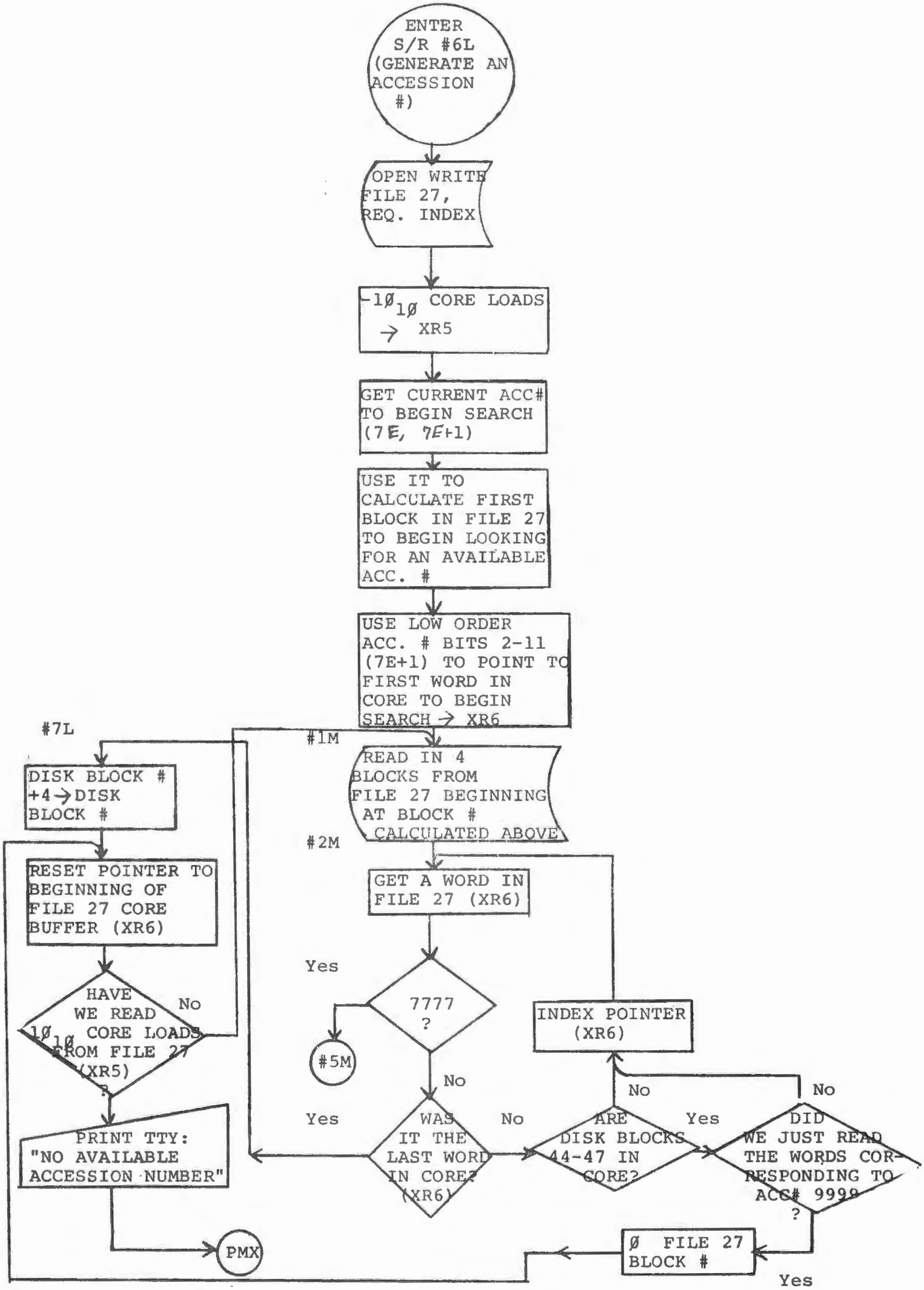

ENTER S/R #3L ACC. # INTER-PRETER
INITIALIZE PTR. TO FIELD BUFR. (XR1)
#9A
CONVERT ACC # TO BINARY
EXIT #3L, NO SKIP
STORE LO ACC #(#2F+3) STORE HI ACC #(#2F+2)
TEST IF ACC # <10,000
error
≥10,000
EXIT #3L, NO SKIP
ok
OPEN WRITE FILE 27, READ PTR. IN REQ. INDEX, FILE 27
PTR. =7777 ?
no
acc # not unique
#5L
CLOSE WRITE FILE 27
EXIT #3L, NO SKIP
yes
7776→PTR. IN REG. INDEX, FILE 27. CLOSE WRITE FILE 27
#2F+2 →#7E
#2F+3 →#7E+1
#4M
INCREMENT CURRENT ACC.#
EXIT #3L, 1 SKIP

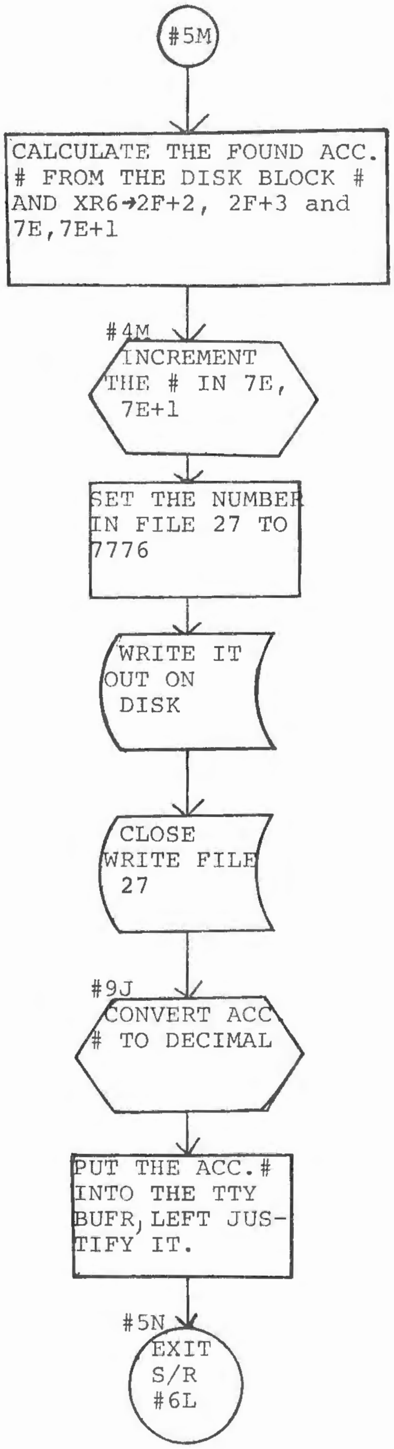

ENTER S/R #4M (INCREMENT DOUBLE PRECISION NUMBER)
INCREMENT ACC #
#2L
TEST IF <10,000 10
>10,000
0→#7E
0→#7E+1
<10,000
EXIT #4M


ENTER S/R #2L
SUB. 2 FROM HI NUMBER (AC)
AC =0?
yes
SUB. 3417 8 FROM LO NUMBER
no
AC >0?
yes
EXIT #2L, NO SKIP
YES
AC ≥ 0?
no
no
EXIT #2L, 1 SKIP

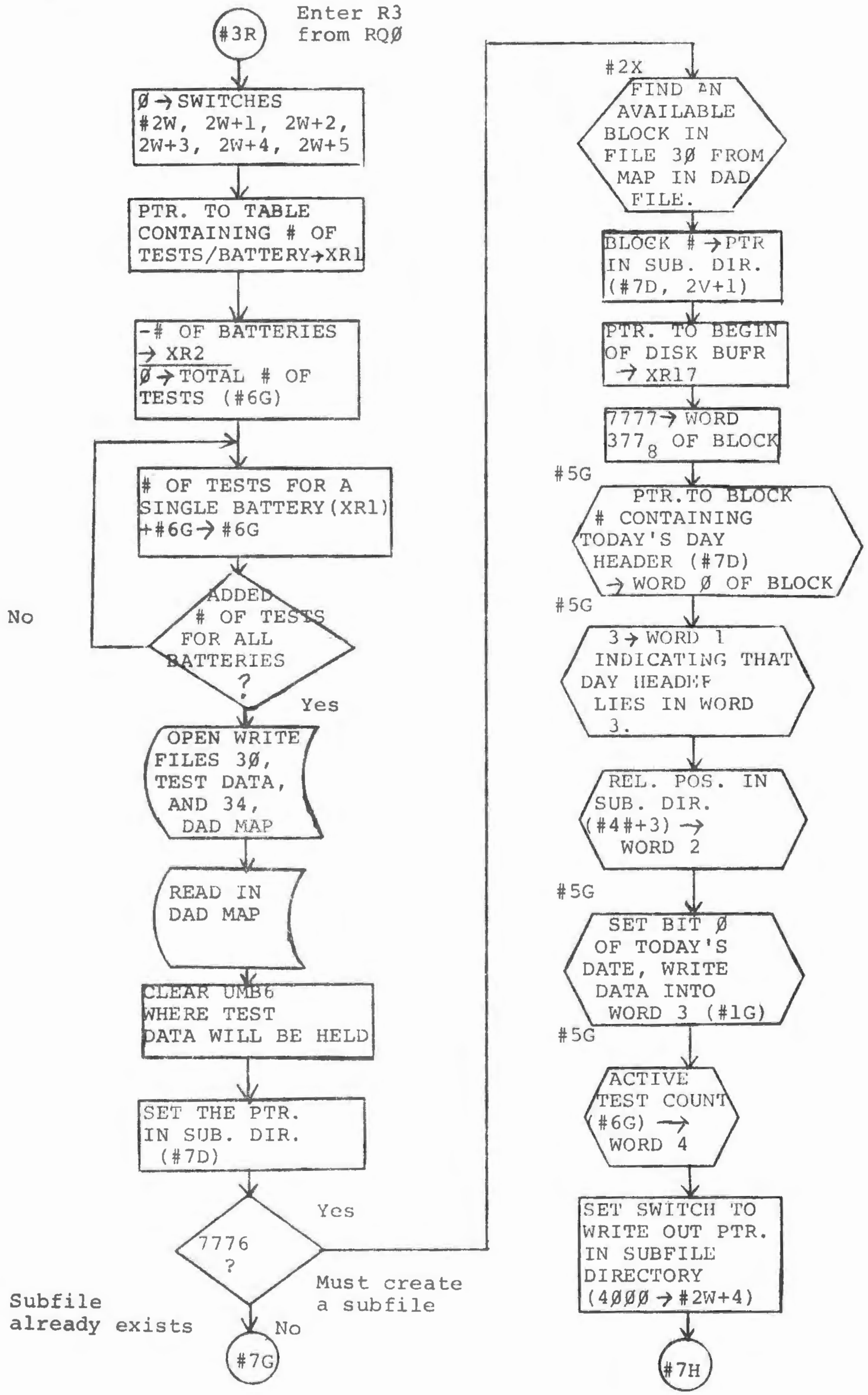

Enter from RQØ
#2R
OPEN.READ FILE 2Ø, PATIENT NAMES
MUL. REL. POS. BY 1Ø10. [#4E+3]x10 → #6F+2,#6F+3
READ PATIENT NAME, FILE 2Ø. CLOSE. READ FILE 2Ø
CONVERT 6-BIT ASCII TO 8-BIT ASCII
8-BIT ASCII NAME → OUTPUT BUFR. SPACE, * → OUTPUT BUFR.
WAS A PACKAGE TYPED? SWITCH #1W
yes
#1X
SET PTR. TO PRINT "<PKG>" AND NAME.
no
WAS AN ACC # TYPED?
yes
SET PTR. TO PRINT NAME ONLY
no
#6L
GENERATE AN ACC.#
SET PTR. TO PRINT ACC# AND NAME.
TRANSFER 5-CHAR BUFR. TO OUTPUT BUFR.
#5W

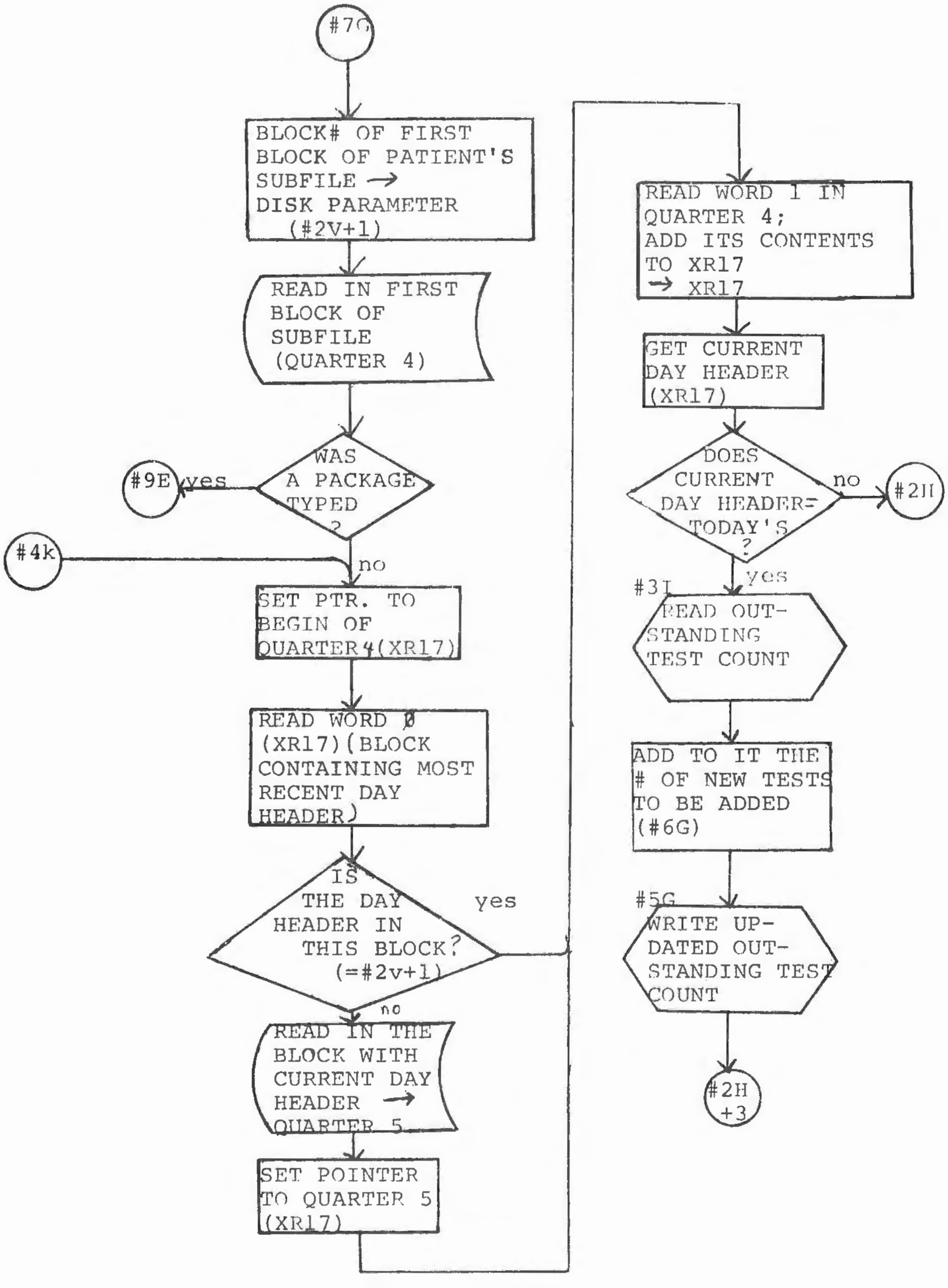

#5W
PRINT TTY:
OUTPUT BUFR
GET INPUT
BUFR.
GET FIRST
CHARACTER
Y?
yes
RQØ
#5X
accepted
no
CARRIAGE
RETURN
?
yes
no. line rejected
WAS
PACKAGE
CODE TYPED
?
yes
#1C
RQØ
no
RESET CURRENT
ACCESSION #
#2F+2→#7E
#2F+3→#7E+1
#4G
RQØ
RESER PTR.
IN REG. INDEX
IF NECESSARY
RQØ
#1C

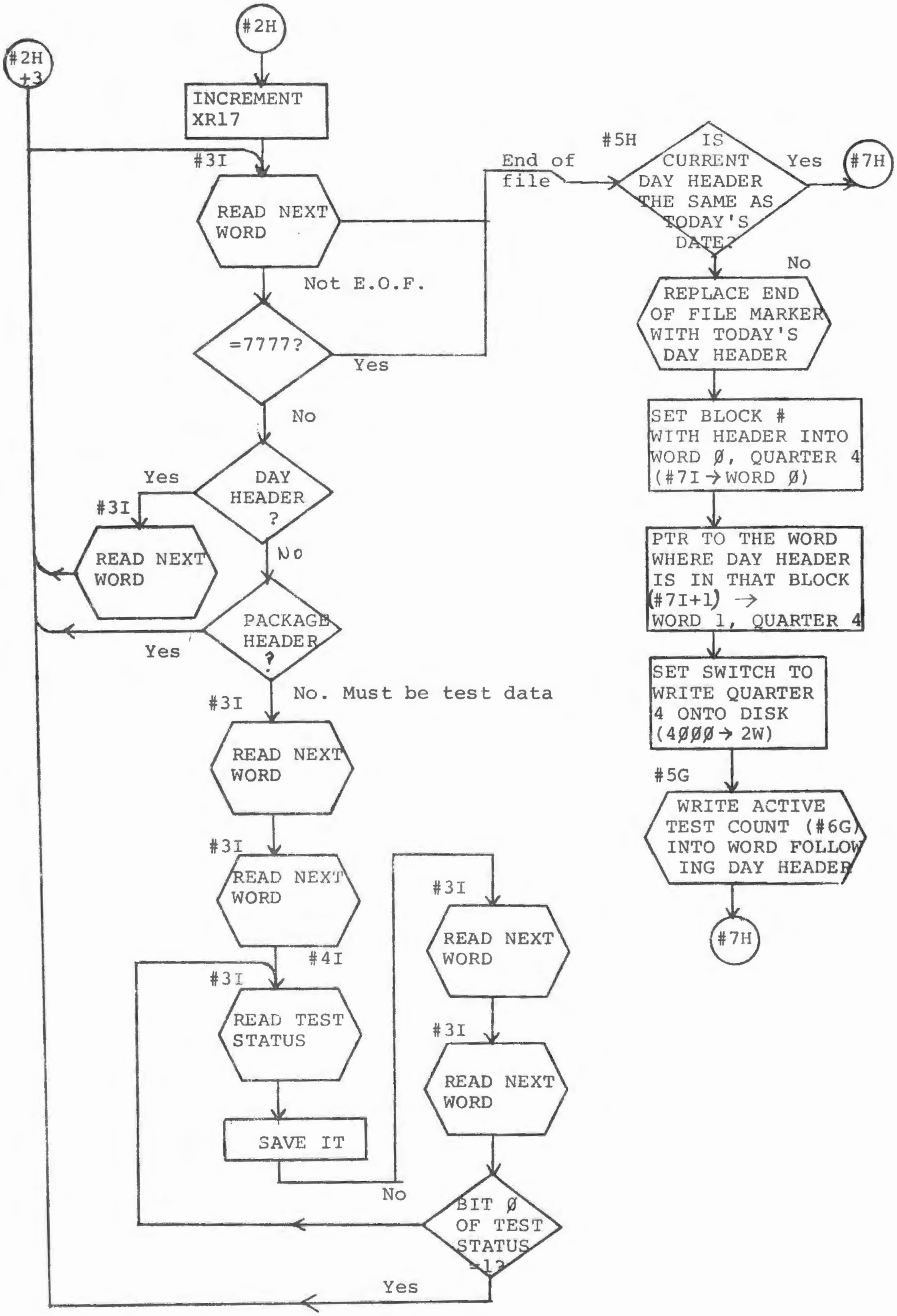

ENTER S/R #6L (GENERATE AN ACCESSION #)
OPEN WRITE FILE 27, REQ. INDEX
-10₁₀ CORE LOADS → XR5
GET CURRENT ACC# TO BEGIN SEARCH (7E, 7E+1)
USE IT TO CALCULATE FIRST BLOCK IN FILE 27 TO BEGIN LOOKING FOR AN AVAILABLE ACC. #
USE LOW ORDER ACC. # BITS 2-11 (7E+1) TO POINT TO FIRST WORD IN CORE TO BEGIN SEARCH → XR6
#1M
READ IN 4 BLOCKS FROM FILE 27 BEGINNING AT BLOCK # CALCULATED ABOVE
#2M
GET A WORD IN FILE 27 (XR6)
7777 ?
Yes
#5M
No
WAS IT THE LAST WORD IN CORE? (XR6)
Yes
No
ARE DISK BLOCKS 44-47 IN CORE?
No
Yes
DID WE JUST READ THE WORDS CORRESPONDING TO ACC# 9999 ?
No
Yes
INDEX POINTER (XR6)
Ø FILE 27 BLOCK #
#7L
DISK BLOCK # +4 → DISK BLOCK #
RESET POINTER TO BEGINNING OF FILE 27 CORE BUFFER (XR6)
HAVE WE READ 10₁₀ CORE LOADS FROM FILE 27 (XR5) ?
No
PRINT TTY: "NO AVAILABLE ACCESSION NUMBER"
PMX

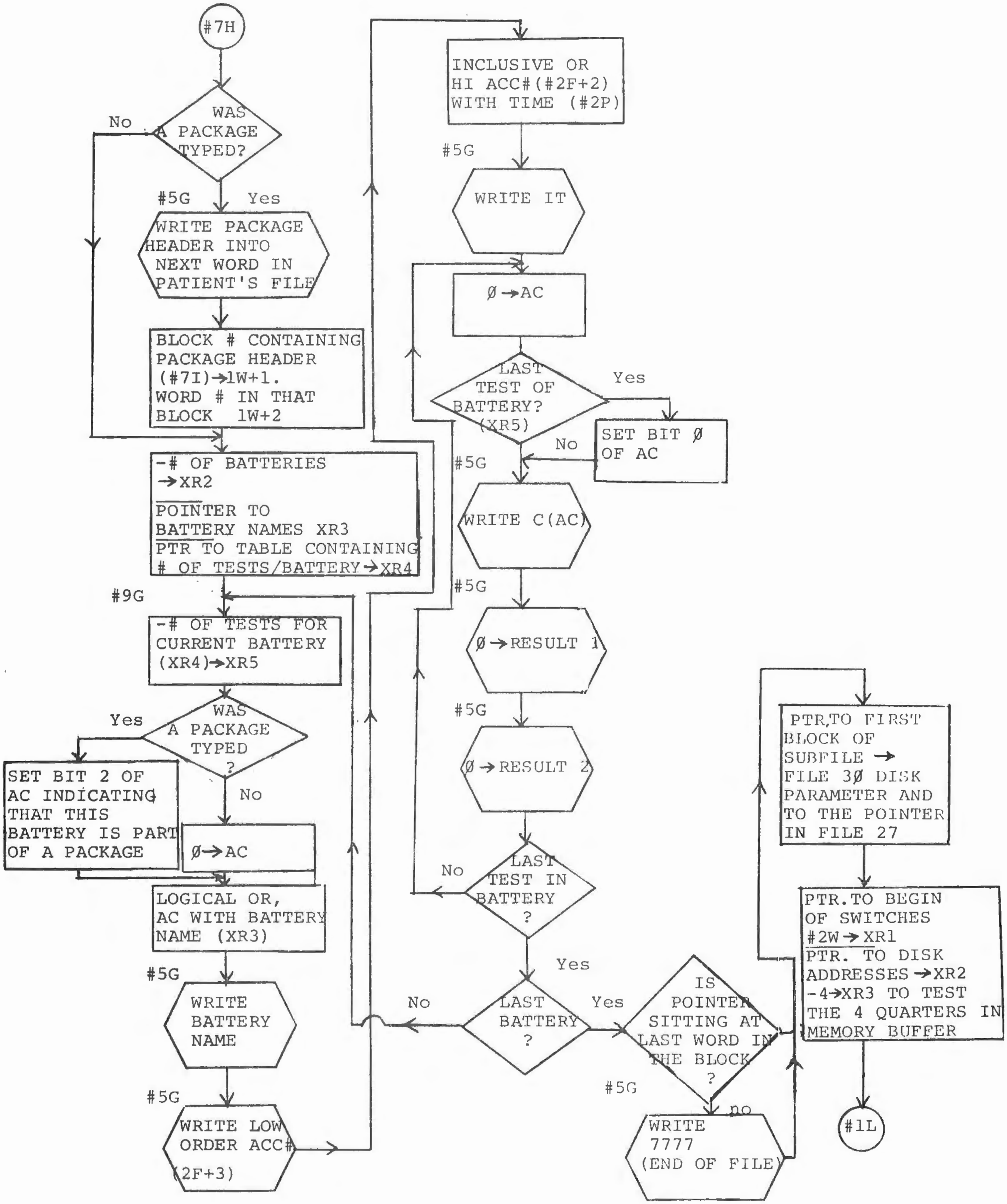

#5M
CALCULATE THE FOUND ACC. # FROM THE DISK BLOCK # AND XR6→2F+2, 2F+3 and 7E,7E+1
#4M
INCREMENT THE # IN 7E, 7E+1
SET THE NUMBER IN FILE 27 TO 7776
WRITE IT OUT ON DISK
CLOSE WRITE FILE 27
#9J
CONVERT ACC. # TO DECIMAL
PUT THE ACC.# INTO THE TTY BUFR, LEFT JUSTIFY IT.
#5N
EXIT S/R #6L

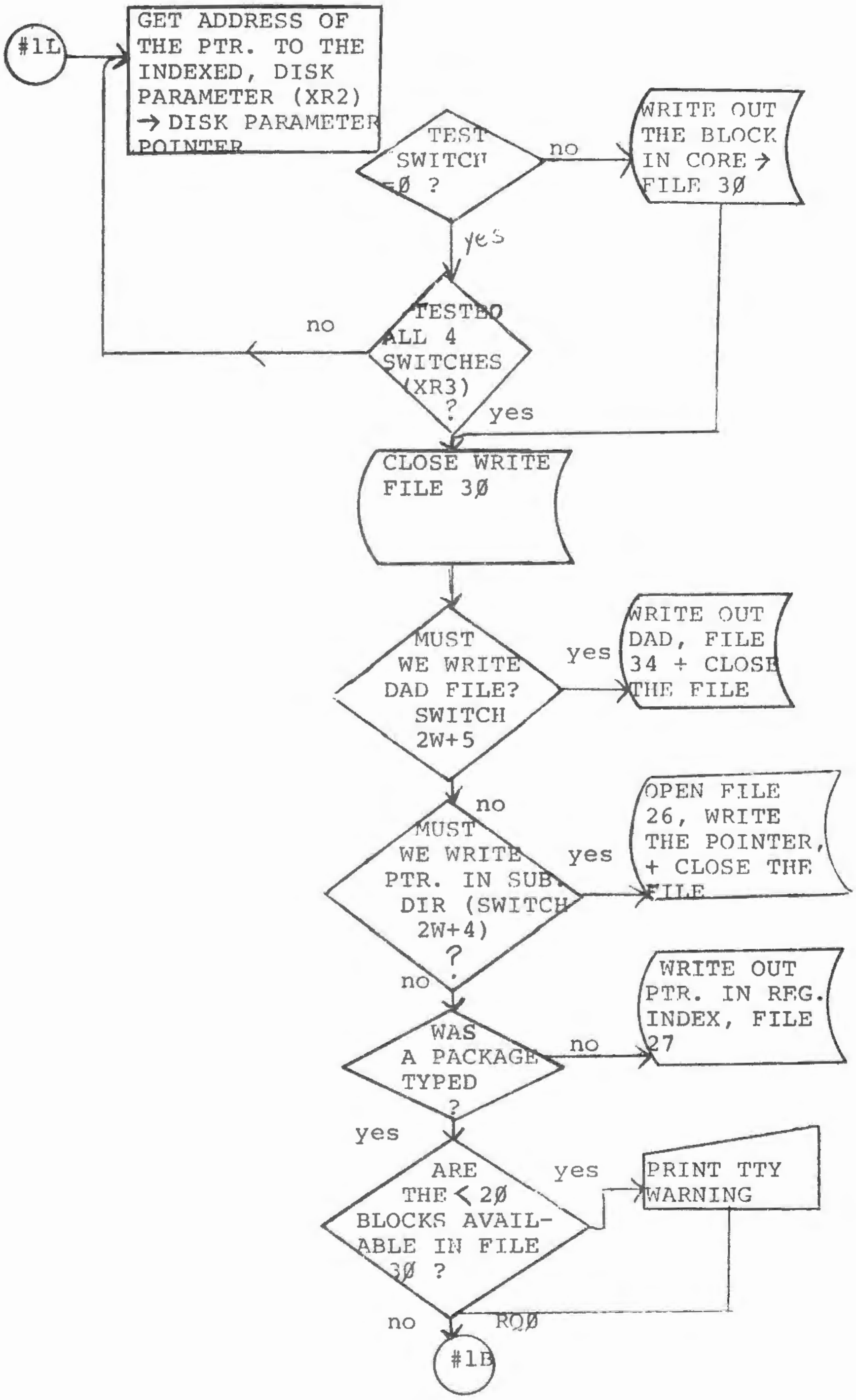
REQUISITION ENTRY
R3

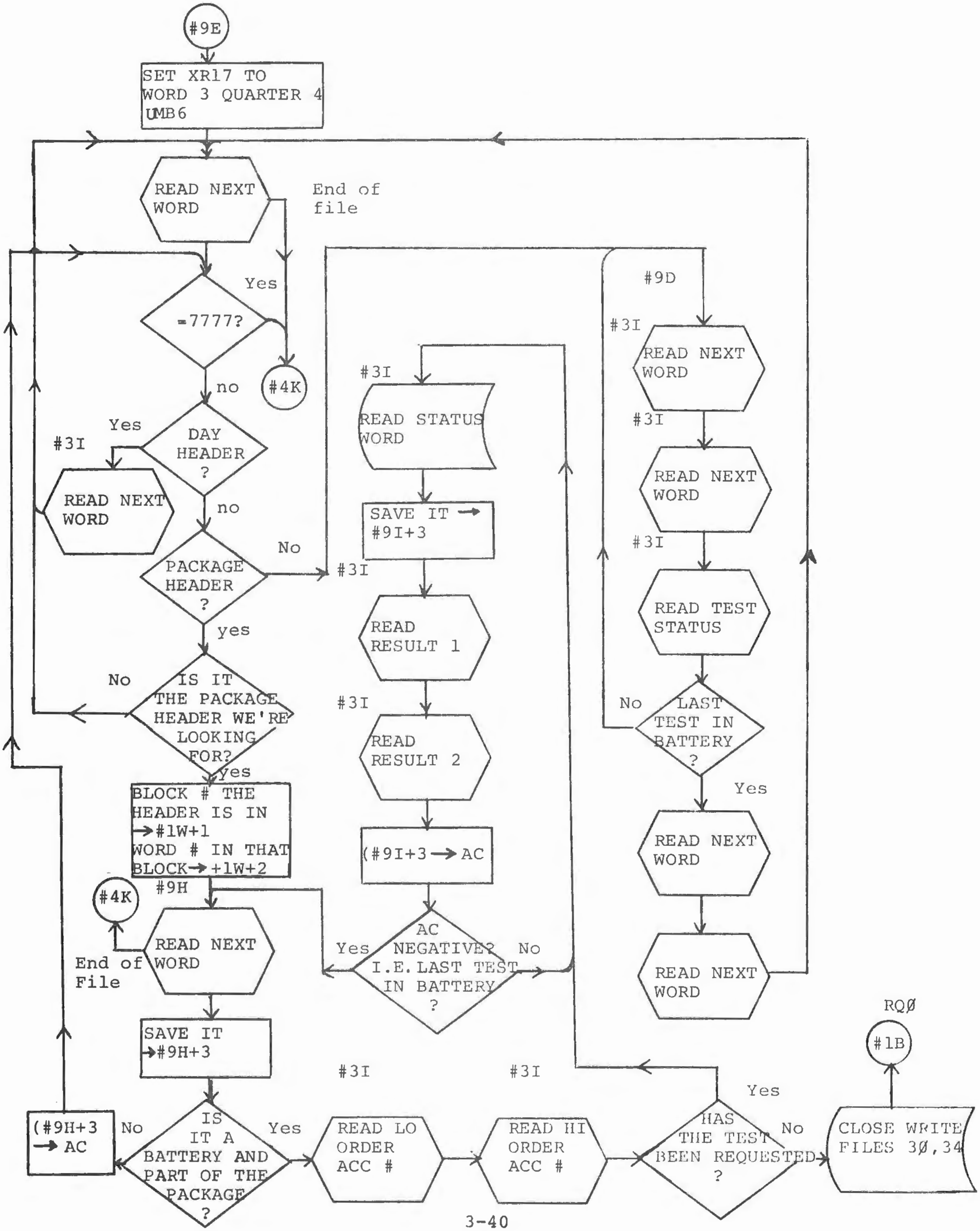

#3R
Enter R3 from RQØ
Ø → SWITCHES #2W, 2W+1, 2W+2, 2W+3, 2W+4, 2W+5
PTR. TO TABLE CONTAINING # OF TESTS/BATTERY→XR1
-# OF BATTERIES → XR2
Ø → TOTAL # OF TESTS (#6G)
# OF TESTS FOR A SINGLE BATTERY(XR1) +#6G → #6G
ADDED # OF TESTS FOR ALL BATTERIES ?
No
Yes
OPEN WRITE FILES 3Ø, TEST DATA, AND 34, DAD MAP
READ IN DAD MAP
CLEAR UMB6 WHERE TEST DATA WILL BE HELD
SET THE PTR. IN SUB. DIR. (#7D)
7776 ?
Yes
Must create a subfile
Subfile already exists
No
#7G
#2X
FIND AN AVAILABLE BLOCK IN FILE 3Ø FROM MAP IN DAD FILE.
BLOCK # → PTR IN SUB. DIR. (#7D, 2V+1)
PTR. TO BEGIN OF DISK BUFR → XR17
7777 → WORD 377_8 OF BLOCK
#5G
PTR. TO BLOCK # CONTAINING TODAY'S DAY HEADER (#7D) → WORD Ø OF BLOCK
#5G
3 → WORD 1 INDICATING THAT DAY HEADER LIES IN WORD 3.
REL. POS. IN SUB. DIR. (#4#+3) → WORD 2
#5G
SET BIT Ø OF TODAY'S DATE, WRITE DATA INTO WORD 3 (#1G)
#5G
ACTIVE TEST COUNT (#6G) → WORD 4
SET SWITCH TO WRITE OUT PTR. IN SUBFILE DIRECTORY (4ØØØ → #2W+4)
#7H

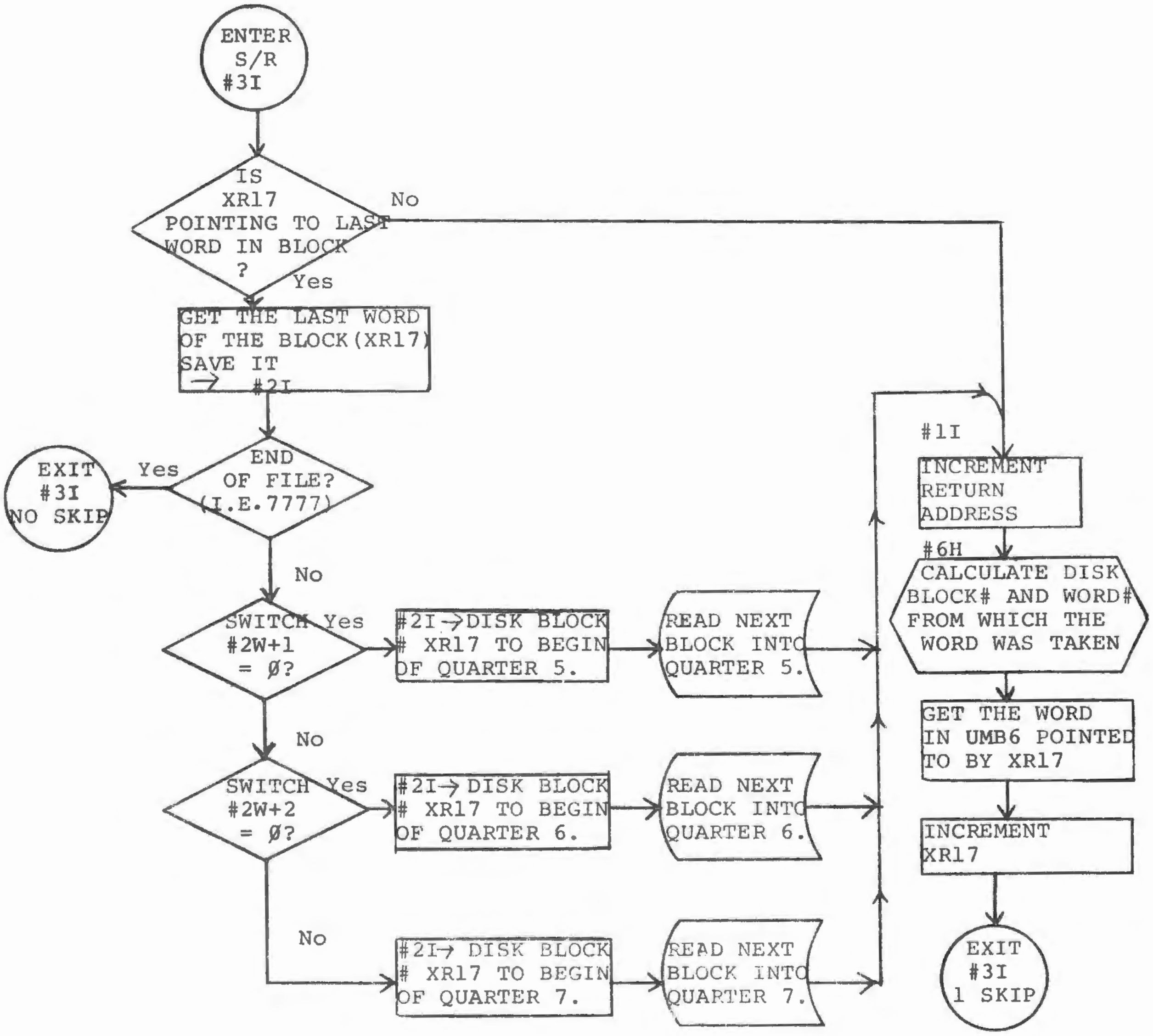

#7G
BLOCK# OF FIRST BLOCK OF PATIENT'S SUBFILE → DISK PARAMETER (#2V+1)
READ IN FIRST BLOCK OF SUBFILE (QUARTER 4)
WAS A PACKAGE TYPED ?
yes
#9E
no
#4k
SET PTR. TO BEGIN OF QUARTER 4(XR17)
READ WORD Ø (XR17) (BLOCK CONTAINING MOST RECENT DAY HEADER)
IS THE DAY HEADER IN THIS BLOCK? (=#2v+1)
yes
no
READ IN THE BLOCK WITH CURRENT DAY HEADER → QUARTER 5
SET POINTER TO QUARTER 5 (XR17)
READ WORD 1 IN QUARTER 4; ADD ITS CONTENTS TO XR17 → XR17
GET CURRENT DAY HEADER (XR17)
DOES CURRENT DAY HEADER= TODAY'S ?
no
#2H
yes
#3I
READ OUT-STANDING TEST COUNT
ADD TO IT THE # OF NEW TESTS TO BE ADDED (#6G)
#5G
WRITE UP-DATED OUT-STANDING TEST COUNT
#2H +3

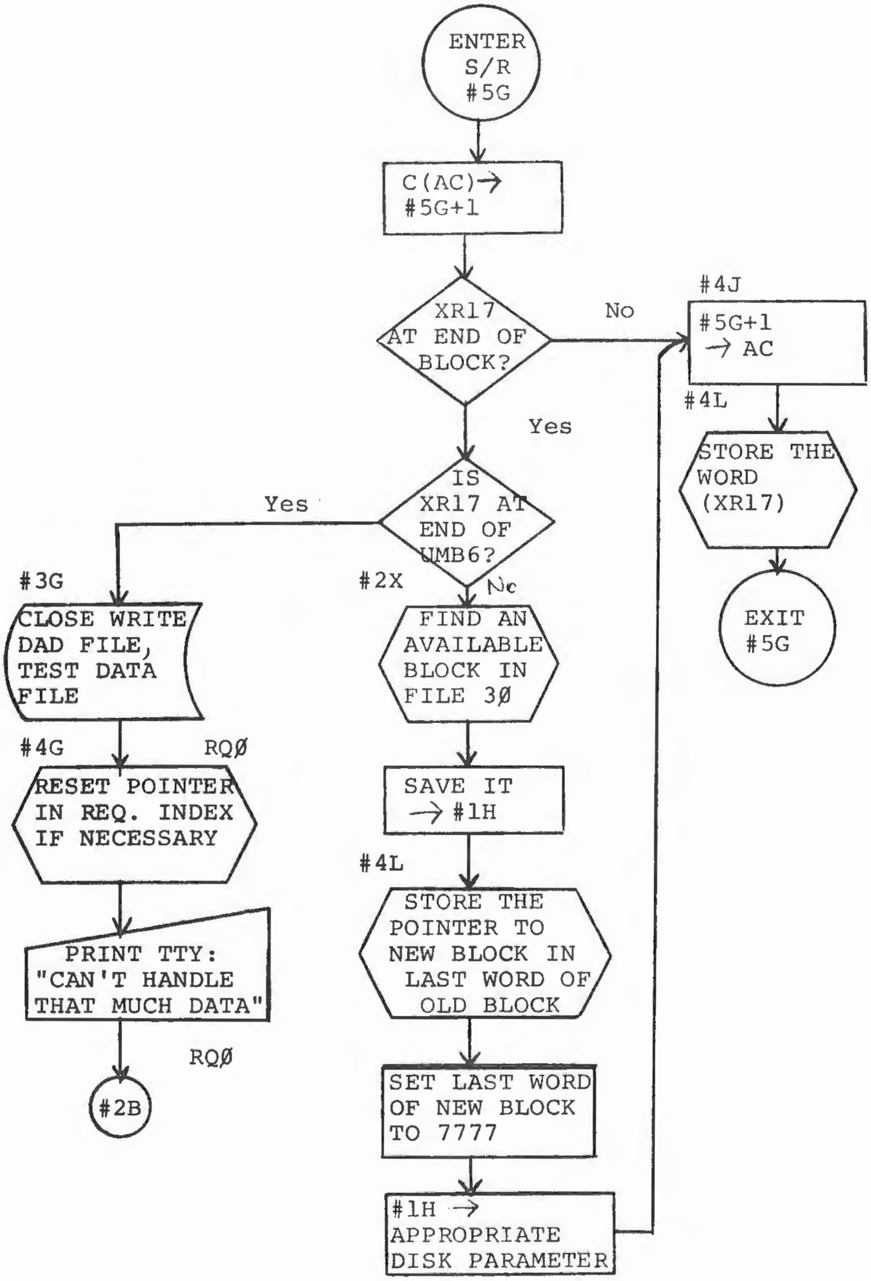

#2H
#2H +3
INCREMENT XR17
#3I
READ NEXT WORD
End of file
Not E.O.F.
=7777?
Yes
No
Yes
DAY HEADER ?
#3I
READ NEXT WORD
No
PACKAGE HEADER ?
Yes
No. Must be test data
#3I
READ NEXT WORD
#3I
READ NEXT WORD
#4I
#3I
READ TEST STATUS
SAVE IT
No
#3I
READ NEXT WORD
#3I
READ NEXT WORD
BIT Ø OF TEST STATUS =1?
Yes
#5H
IS CURRENT DAY HEADER THE SAME AS TODAY'S DATE?
Yes
#7H
No
REPLACE END OF FILE MARKER WITH TODAY'S DAY HEADER
SET BLOCK # WITH HEADER INTO WORD Ø, QUARTER 4 (#7I → WORD Ø)
PTR TO THE WORD WHERE DAY HEADER IS IN THAT BLOCK (#7I+1) → WORD 1, QUARTER 4
SET SWITCH TO WRITE QUARTER 4 ONTO DISK (4ØØØ → 2W)
#5G
WRITE ACTIVE TEST COUNT (#6G) INTO WORD FOLLOWING DAY HEADER
#7H

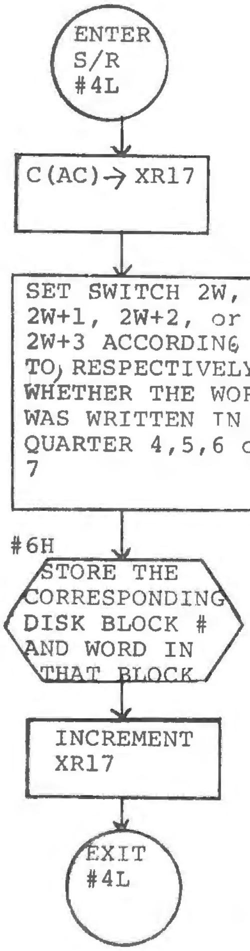

#7H
WAS A PACKAGE TYPED?
No
Yes
#5G
WRITE PACKAGE HEADER INTO NEXT WORD IN PATIENT'S FILE
BLOCK # CONTAINING PACKAGE HEADER (#7I)→1W+1.
WORD # IN THAT BLOCK 1W+2
-# OF BATTERIES →XR2
POINTER TO BATTERY NAMES XR3
PTR TO TABLE CONTAINING # OF TESTS/BATTERY→XR4
#9G
-# OF TESTS FOR CURRENT BATTERY (XR4)→XR5
WAS A PACKAGE TYPED ?
Yes
No
SET BIT 2 OF AC INDICATING THAT THIS BATTERY IS PART OF A PACKAGE
Ø→AC
LOGICAL OR, AC WITH BATTERY NAME (XR3)
#5G
WRITE BATTERY NAME
#5G
WRITE LOW ORDER ACC# (2F+3)
INCLUSIVE OR HI ACC#(#2F+2) WITH TIME (#2P)
#5G
WRITE IT
Ø→AC
LAST TEST OF BATTERY? (XR5)
Yes
No
SET BIT Ø OF AC
#5G
WRITE C(AC)
#5G
Ø→RESULT 1
#5G
Ø→RESULT 2
LAST TEST IN BATTERY ?
No
Yes
LAST BATTERY ?
No
Yes
IS POINTER SITTING AT LAST WORD IN THE BLOCK ?
#5G
no
WRITE 7777 (END OF FILE)
PTR.TO FIRST BLOCK OF SUBFILE → FILE 3Ø DISK PARAMETER AND TO THE POINTER IN FILE 27
PTR.TO BEGIN OF SWITCHES #2W→XR1
PTR. TO DISK ADDRESSES→XR2
-4→XR3 TO TEST THE 4 QUARTERS IN MEMORY BUFFER
#1L

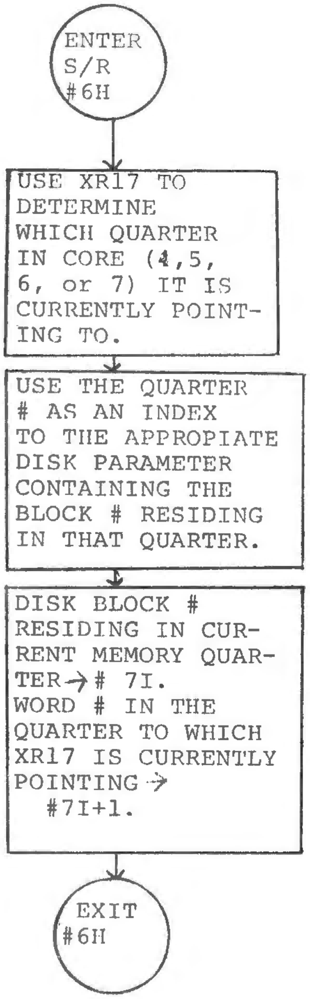

#1L
GET ADDRESS OF THE PTR. TO THE INDEXED, DISK PARAMETER (XR2) → DISK PARAMETER POINTER
TEST SWITCH =Ø ?
no
WRITE OUT THE BLOCK IN CORE → FILE 3Ø
yes
TESTED ALL 4 SWITCHES (XR3) ?
no
yes
CLOSE WRITE FILE 3Ø
MUST WE WRITE DAD FILE? SWITCH 2W+5
yes
WRITE OUT DAD, FILE 34 + CLOSE THE FILE
no
MUST WE WRITE PTR. IN SUB. DIR (SWITCH 2W+4) ?
yes
OPEN FILE 26, WRITE THE POINTER, + CLOSE THE FILE
no
WAS A PACKAGE TYPED ?
no
WRITE OUT PTR. IN REG. INDEX, FILE 27
yes
ARE THE < 2Ø BLOCKS AVAIL-ABLE IN FILE 3Ø ?
yes
PRINT TTY WARNING
no
RQØ
#1B

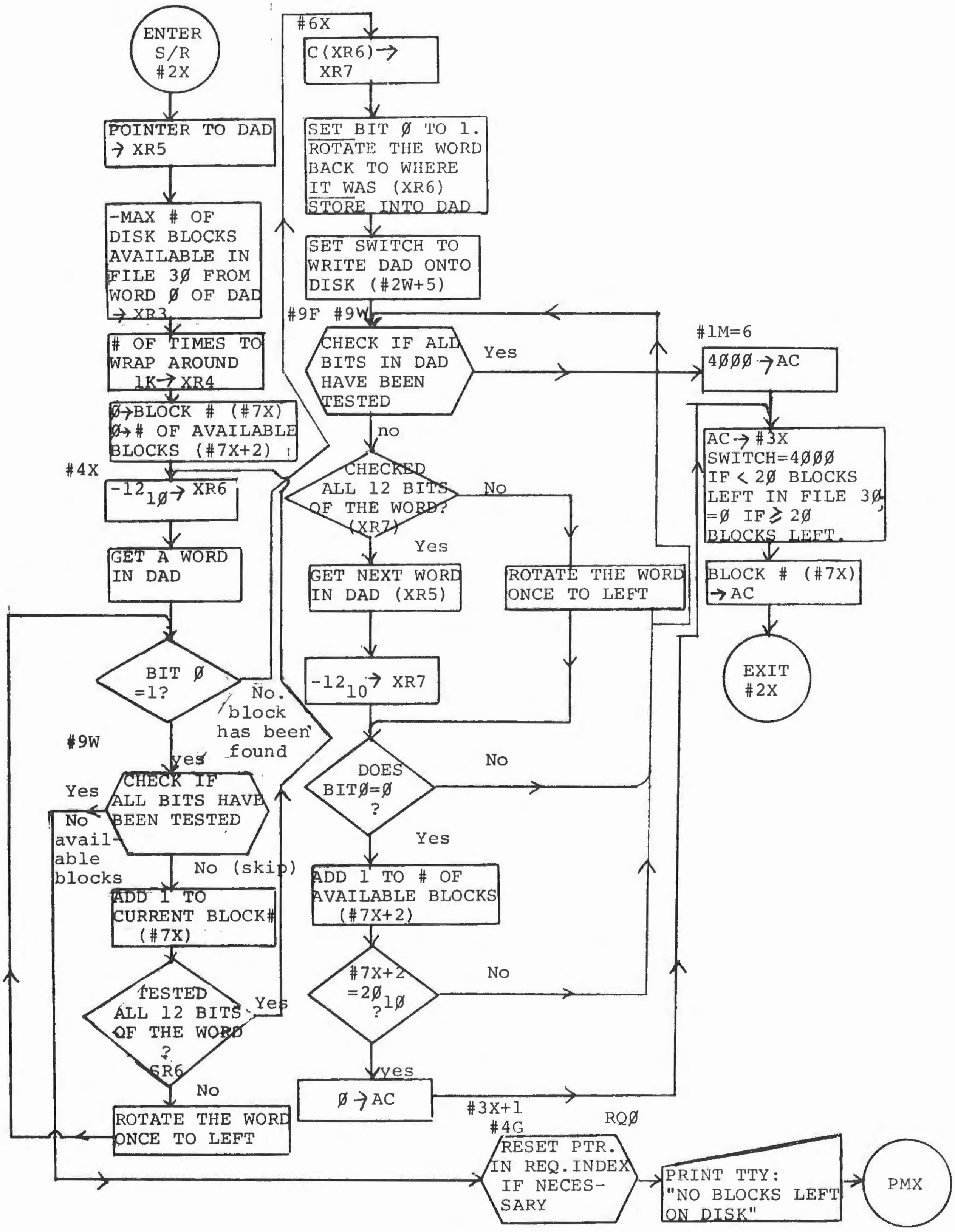

#9E
SET XR17 TO WORD 3 QUARTER 4 UMB6
READ NEXT WORD
End of file
Yes
=7777?
no
#4K
DAY HEADER ?
Yes
#3I
READ NEXT WORD
no
PACKAGE HEADER ?
No
#3I
yes
IS IT THE PACKAGE HEADER WE'RE LOOKING FOR?
No
yes
BLOCK # THE HEADER IS IN →#1W+1
WORD # IN THAT BLOCK→ +1W+2
#9H
#4K
End of File
READ NEXT WORD
SAVE IT →#9H+3
(#9H+3 → AC
No
IS IT A BATTERY AND PART OF THE PACKAGE ?
Yes
#3I
READ STATUS WORD
SAVE IT → #9I+3
READ RESULT 1
#3I
READ RESULT 2
(#9I+3 → AC
Yes
AC NEGATIVE? I.E. LAST TEST IN BATTERY ?
No
#3I
READ LO ORDER ACC #
#3I
READ HI ORDER ACC #
HAS THE TEST BEEN REQUESTED ?
Yes
No
#9D
#3I
READ NEXT WORD
#3I
READ NEXT WORD
#3I
READ TEST STATUS
No
LAST TEST IN BATTERY ?
Yes
READ NEXT WORD
READ NEXT WORD
RQØ
#1B
CLOSE WRITE FILES 3Ø,34

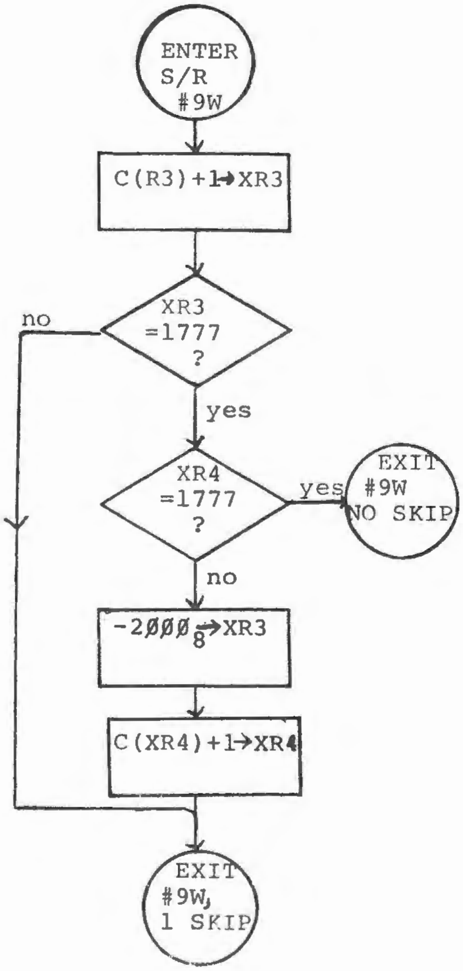

ENTER S/R #3I
IS XR17 POINTING TO LAST WORD IN BLOCK ?
No
Yes
GET THE LAST WORD OF THE BLOCK(XR17) SAVE IT → #2I
END OF FILE? (I.E.7777)
Yes
EXIT #3I NO SKIP
No
SWITCH #2W+1 = Ø?
Yes
#2I→DISK BLOCK # XR17 TO BEGIN OF QUARTER 5.
READ NEXT BLOCK INTO QUARTER 5.
No
SWITCH #2W+2 = Ø?
Yes
#2I→DISK BLOCK # XR17 TO BEGIN OF QUARTER 6.
READ NEXT BLOCK INTO QUARTER 6.
No
#2I→ DISK BLOCK # XR17 TO BEGIN OF QUARTER 7.
READ NEXT BLOCK INTO QUARTER 7.
#1I
INCREMENT RETURN ADDRESS
#6H
CALCULATE DISK BLOCK# AND WORD# FROM WHICH THE WORD WAS TAKEN
GET THE WORD IN UMB6 POINTED TO BY XR17
INCREMENT XR17
EXIT #3I 1 SKIP

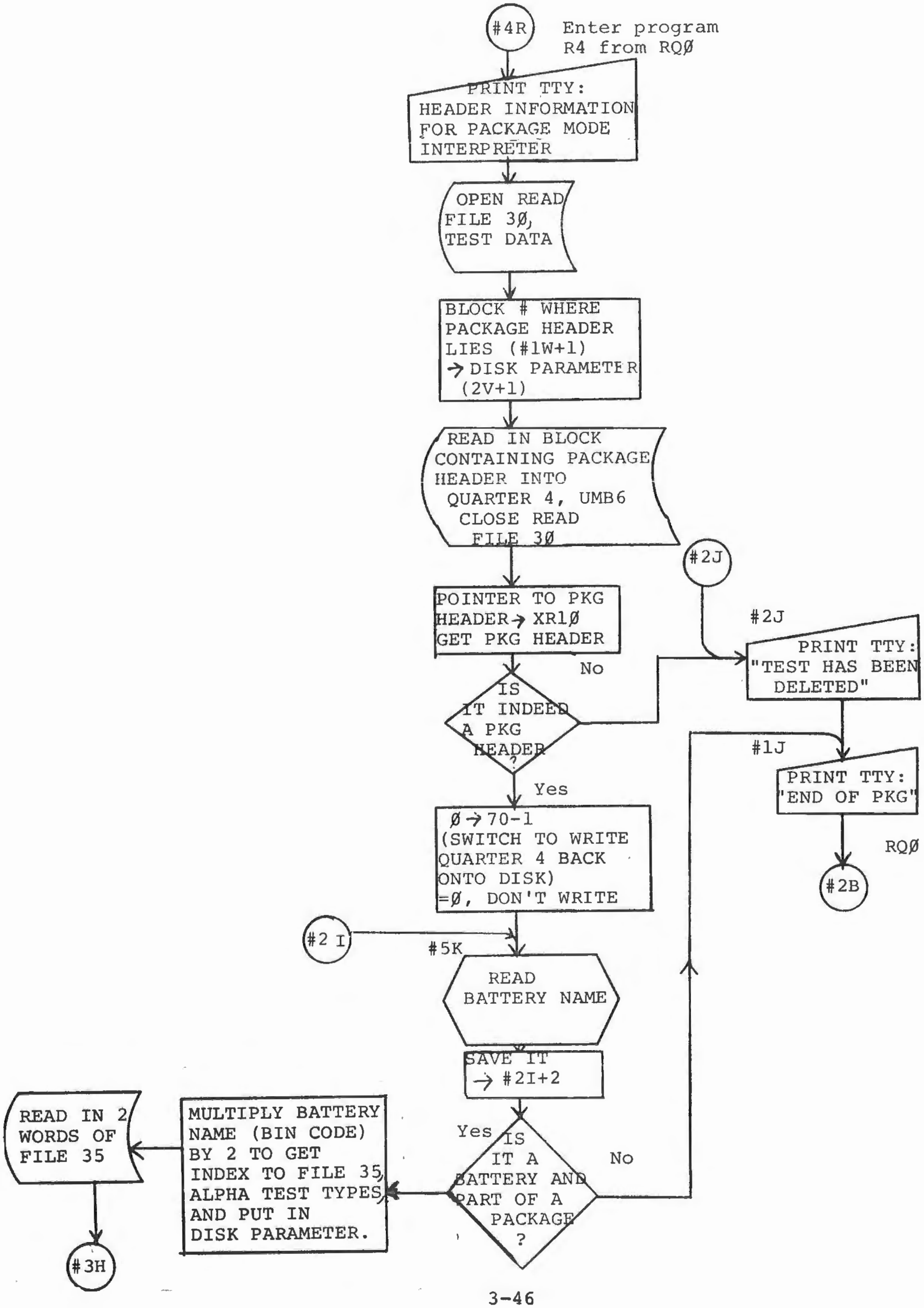

ENTER S/R #5G
C(AC)→ #5G+1
XR17 AT END OF BLOCK?
No
#4J
#5G+1 → AC
#4L
STORE THE WORD (XR17)
EXIT #5G
Yes
IS XR17 AT END OF UMB6?
Yes
#3G
CLOSE WRITE DAD FILE, TEST DATA FILE
#4G
RQØ
RESET POINTER IN REQ. INDEX IF NECESSARY
PRINT TTY: "CAN'T HANDLE THAT MUCH DATA"
RQØ
#2B
#2X
No
FIND AN AVAILABLE BLOCK IN FILE 3Ø
SAVE IT → #1H
#4L
STORE THE POINTER TO NEW BLOCK IN LAST WORD OF OLD BLOCK
SET LAST WORD OF NEW BLOCK TO 7777
#1H → APPROPRIATE DISK PARAMETER

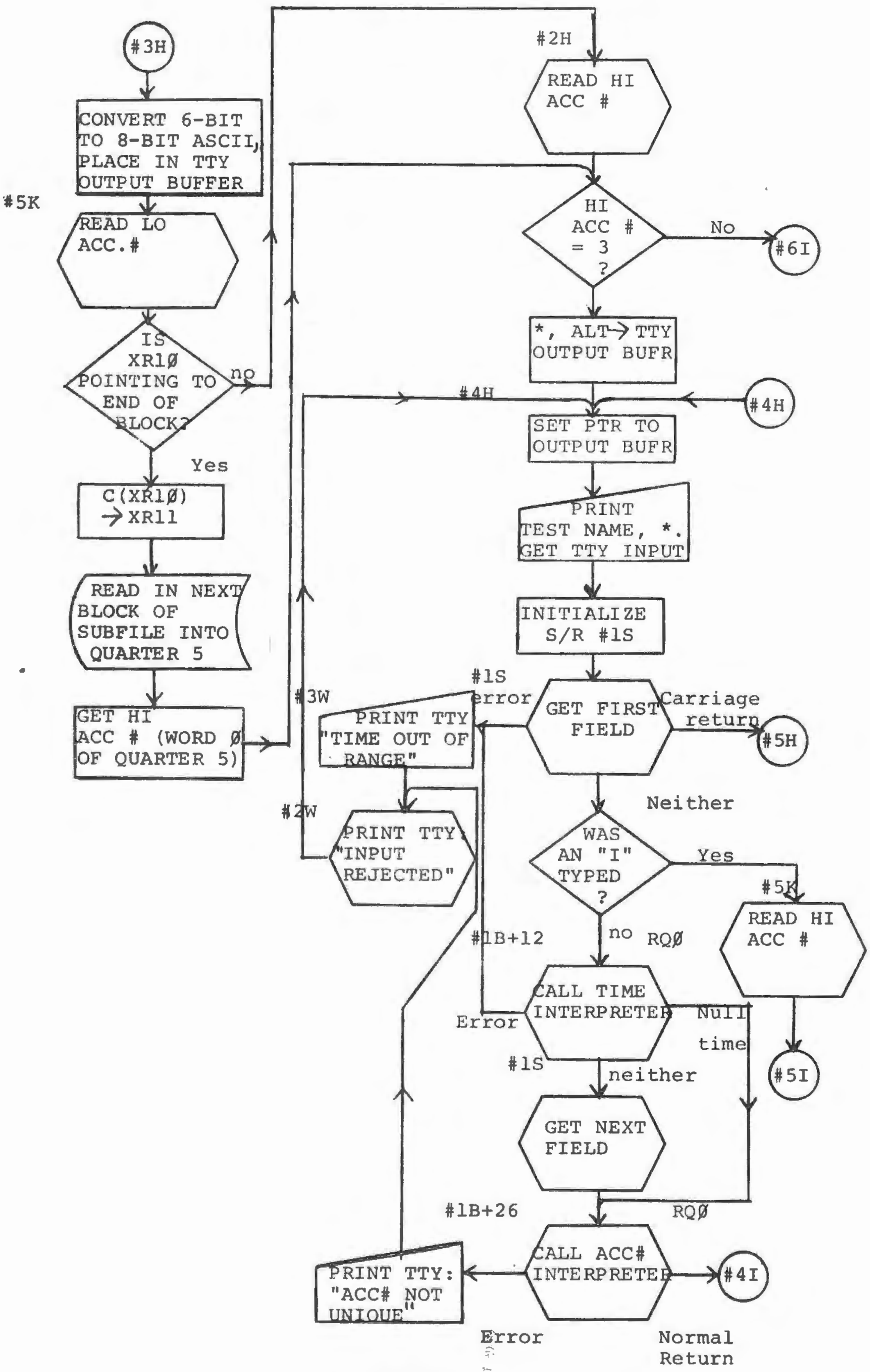

ENTER S/R #4L
C(AC)→XR17
SET SWITCH 2W, 2W+1, 2W+2, or 2W+3 ACCORDING TO, RESPECTIVELY, WHETHER THE WORD WAS WRITTEN TN QUARTER 4,5,6 or 7
#6H
STORE THE CORRESPONDING DISK BLOCK # AND WORD IN THAT BLOCK
INCREMENT XR17
EXIT #4L

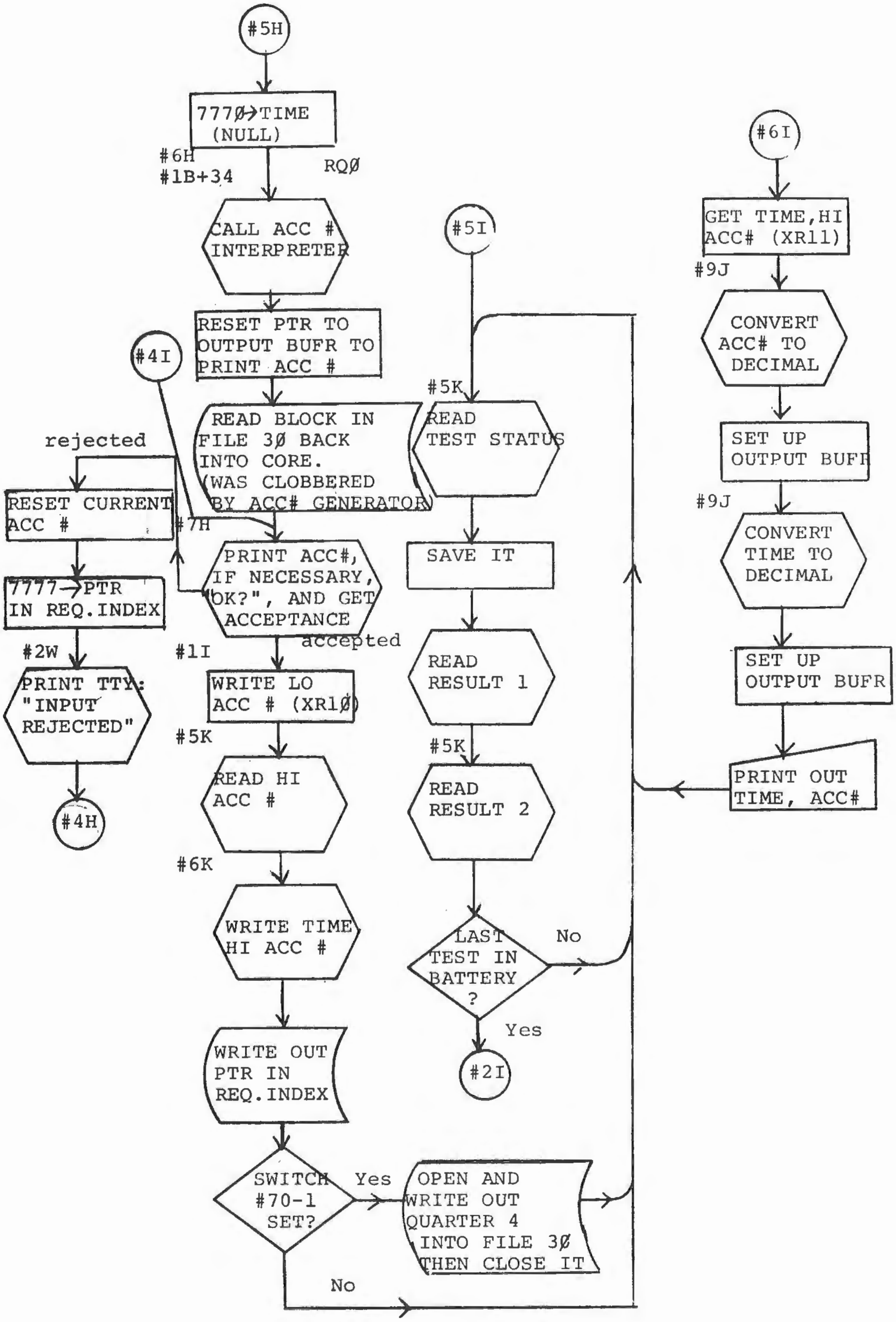

ENTER S/R #6H
USE XR17 TO DETERMINE WHICH QUARTER IN CORE (4,5, 6, or 7) IT IS CURRENTLY POINTING TO.
USE THE QUARTER # AS AN INDEX TO THE APPROPIATE DISK PARAMETER CONTAINING THE BLOCK # RESIDING IN THAT QUARTER.
DISK BLOCK # RESIDING IN CURRENT MEMORY QUARTER→# 7I. WORD # IN THE QUARTER TO WHICH XR17 IS CURRENTLY POINTING → #7I+1.
EXIT #6H

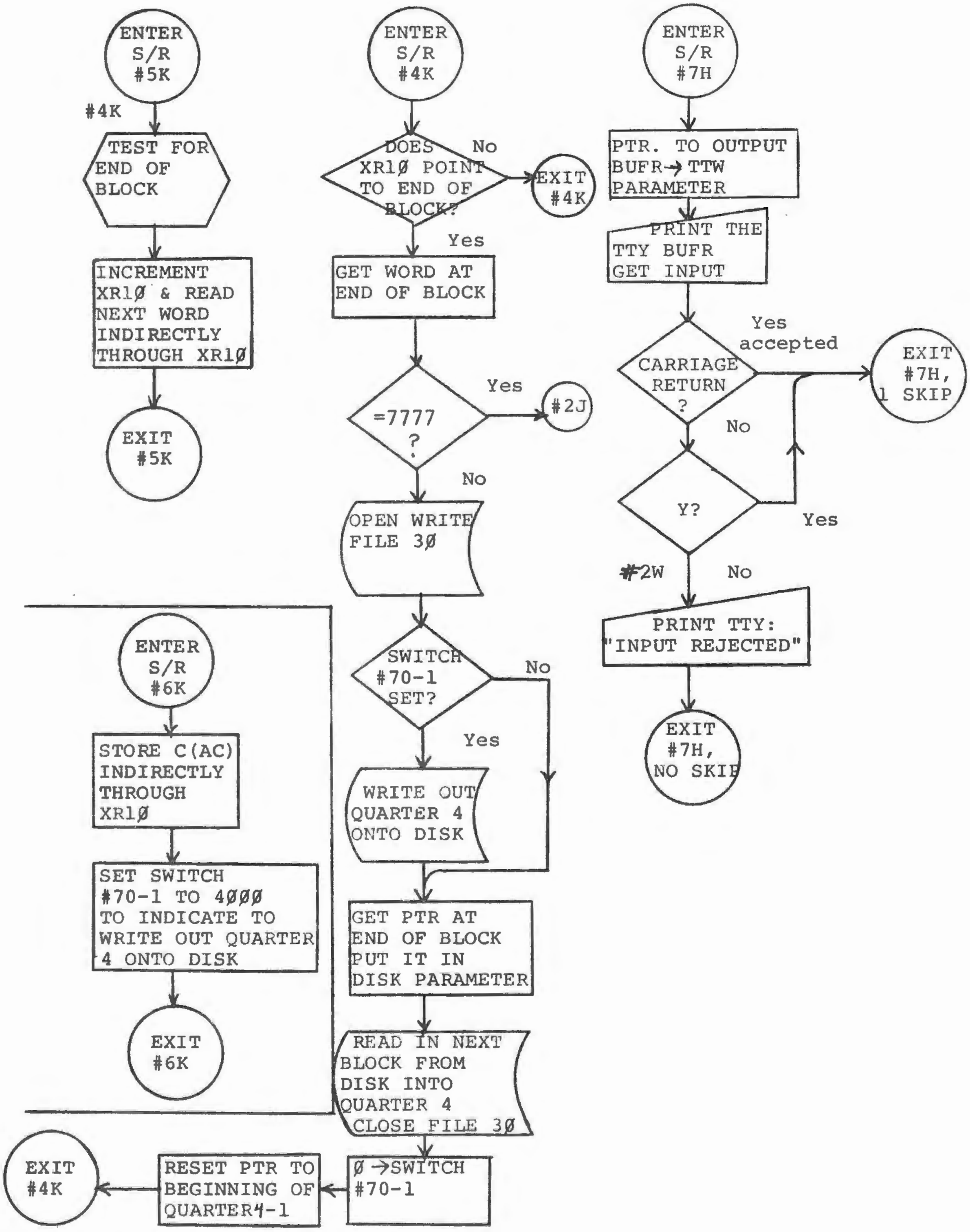

ENTER S/R #2X
POINTER TO DAD → XR5
-MAX # OF DISK BLOCKS AVAILABLE IN FILE 3Ø FROM WORD Ø OF DAD → XR3
# OF TIMES TO WRAP AROUND 1K → XR4
Ø→BLOCK # (#7X)
Ø→# OF AVAILABLE BLOCKS (#7X+2)
#4X
-12₁Ø → XR6
GET A WORD IN DAD
BIT Ø =1?
No. block has been found
#9W
yes
CHECK IF ALL BITS HAVE BEEN TESTED
Yes
No available blocks
No (skip)
ADD 1 TO CURRENT BLOCK# (#7X)
TESTED ALL 12 BITS OF THE WORD ? GR6
Yes
No
ROTATE THE WORD ONCE TO LEFT
#6X
C(XR6) → XR7
SET BIT Ø TO 1. ROTATE THE WORD BACK TO WHERE IT WAS (XR6) STORE INTO DAD
SET SWITCH TO WRITE DAD ONTO DISK (#2W+5)
#9F #9W
CHECK IF ALL BITS IN DAD HAVE BEEN TESTED
Yes
no
CHECKED ALL 12 BITS OF THE WORD? (XR7)
No
Yes
GET NEXT WORD IN DAD (XR5)
ROTATE THE WORD ONCE TO LEFT
-12₁Ø → XR7
DOES BITØ=Ø ?
No
Yes
ADD 1 TO # OF AVAILABLE BLOCKS (#7X+2)
#7X+2 =2Ø₁Ø ?
No
yes
Ø → AC
#3X+1
#1M=6
4ØØØ → AC
AC → #3X
SWITCH=4ØØØ IF < 2Ø BLOCKS LEFT IN FILE 3Ø, =Ø IF ≥ 2Ø BLOCKS LEFT.
BLOCK # (#7X) → AC
EXIT #2X
#4G
RQØ
RESET PTR. IN REQ.INDEX IF NECESSARY
PRINT TTY: "NO BLOCKS LEFT ON DISK"
PMX


ENTER S/R #9W
C(R3)+1→XR3
XR3 =1777 ?
no
yes
XR4 =1777 ?
yes
EXIT #9W NO SKIP
no
-2ØØØ$_8$→XR3
C(XR4)+1→XR4
EXIT #9W, 1 SKIP


#4R
Enter program R4 from RQØ
PRINT TTY: HEADER INFORMATION FOR PACKAGE MODE INTERPRETER
OPEN READ FILE 3Ø, TEST DATA
BLOCK # WHERE PACKAGE HEADER LIES (#1W+1) → DISK PARAMETER (2V+1)
READ IN BLOCK CONTAINING PACKAGE HEADER INTO QUARTER 4, UMB6 CLOSE READ FILE 3Ø
#2J
POINTER TO PKG HEADER → XR1Ø GET PKG HEADER
IS IT INDEED A PKG HEADER ?
No
#2J
PRINT TTY: "TEST HAS BEEN DELETED"
#1J
PRINT TTY: "END OF PKG"
RQØ
#2B
Yes
Ø → 70-1 (SWITCH TO WRITE QUARTER 4 BACK ONTO DISK) =Ø, DON'T WRITE
#2 I
#5K
READ BATTERY NAME
SAVE IT → #2I+2
IS IT A BATTERY AND PART OF A PACKAGE ?
Yes
No
MULTIPLY BATTERY NAME (BIN CODE) BY 2 TO GET INDEX TO FILE 35, ALPHA TEST TYPES, AND PUT IN DISK PARAMETER.
READ IN 2 WORDS OF FILE 35
#3H


#3H
CONVERT 6-BIT TO 8-BIT ASCII, PLACE IN TTY OUTPUT BUFFER
#5K
READ LO ACC.#
IS XR1Ø POINTING TO END OF BLOCK?
no
Yes
C(XR1Ø) → XR11
READ IN NEXT BLOCK OF SUBFILE INTO QUARTER 5
GET HI ACC # (WORD Ø OF QUARTER 5)
#2H
READ HI ACC #
HI ACC # = 3 ?
No
#6I
*, ALT→ TTY OUTPUT BUFR
#4H
#4H
SET PTR TO OUTPUT BUFR
PRINT TEST NAME, *. GET TTY INPUT
INITIALIZE S/R #1S
#1S
error
GET FIRST FIELD
Carriage return
#5H
#3W
PRINT TTY "TIME OUT OF RANGE"
Neither
#2W
PRINT TTY: "INPUT REJECTED"
WAS AN "I" TYPED ?
Yes
#5K
READ HI ACC #
#1B+12
no
RQØ
CALL TIME INTERPRETER
Error
Null time
#1S
neither
#5I
GET NEXT FIELD
#1B+26
RQØ
CALL ACC# INTERPRETER
PRINT TTY: "ACC# NOT UNIQUE"
#4I
Error
Normal Return


#5H
7770→TIME (NULL)
#6H
#1B+34
RQØ
CALL ACC # INTERPRETER
RESET PTR TO OUTPUT BUFR TO PRINT ACC #
READ BLOCK IN FILE 3Ø BACK INTO CORE. (WAS CLOBBERED BY ACC# GENERATOR)
#7H
PRINT ACC#, IF NECESSARY, "OK?", AND GET ACCEPTANCE
accepted
rejected
#4I
RESET CURRENT ACC #
7777→PTR IN REQ.INDEX
#2W
PRINT TTY: "INPUT REJECTED"
#4H
#1I
WRITE LO ACC # (XR1Ø)
#5K
READ HI ACC #
#6K
WRITE TIME HI ACC #
WRITE OUT PTR IN REQ.INDEX
SWITCH #70-1 SET?
Yes
No
OPEN AND WRITE OUT QUARTER 4 INTO FILE 3Ø THEN CLOSE IT
#5I
#5K
READ TEST STATUS
SAVE IT
READ RESULT 1
#5K
READ RESULT 2
LAST TEST IN BATTERY ?
No
Yes
#2I
#6I
GET TIME,HI ACC# (XR11)
#9J
CONVERT ACC# TO DECIMAL
SET UP OUTPUT BUFR
#9J
CONVERT TIME TO DECIMAL
SET UP OUTPUT BUFR
PRINT OUT TIME, ACC#


ENTER S/R #5K
#4K
TEST FOR END OF BLOCK
INCREMENT XR1Ø & READ NEXT WORD INDIRECTLY THROUGH XR1Ø
EXIT #5K
ENTER S/R #4K
DOES XR1Ø POINT TO END OF BLOCK?
No
EXIT #4K
Yes
GET WORD AT END OF BLOCK
=7777 ?
Yes
#2J
No
OPEN WRITE FILE 3Ø
SWITCH #70-1 SET?
No
Yes
WRITE OUT QUARTER 4 ONTO DISK
GET PTR AT END OF BLOCK PUT IT IN DISK PARAMETER
READ IN NEXT BLOCK FROM DISK INTO QUARTER 4 CLOSE FILE 3Ø
Ø →SWITCH #70-1
RESET PTR TO BEGINNING OF QUARTER4-1
EXIT #4K
ENTER S/R #7H
PTR. TO OUTPUT BUFR→TTW PARAMETER
PRINT THE TTY BUFR GET INPUT
CARRIAGE RETURN ?
Yes accepted
EXIT #7H, 1 SKIP
No
Y?
Yes
#2W
No
PRINT TTY: "INPUT REJECTED"
EXIT #7H, NO SKIP
ENTER S/R #6K
STORE C(AC) INDIRECTLY THROUGH XR1Ø
SET SWITCH #70-1 TO 4ØØØ TO INDICATE TO WRITE OUT QUARTER 4 ONTO DISK
EXIT #6K

3.8 FUNCTIONAL DESCRIPTION OF CARD READER INPUT

After selection of Card Reader input, the computer will respond with the following message on the calling terminal:

```
     OUTPUT ON 1 TERMINAL, 2 LINE PRINTER
```

A response of 1 will set location OUTDEV to 0000. A response of 2 will set location OUTDEV to 7777. The value of OUTDEV is transferred between overlays for use by overlay R8.

The program then searches the card type files and forms a list of RE card codes at TYPTAB in bank 6. The program then opens a scratch file in which to put the card information. The program then opens the card reader and prints the following message on the calling terminal:

```
     LOAD CARDS, HIT STOP, HIT READY, HIT RETURN*
```

After a key is struck, the program causes three cards to be read into the buffer CRDBUF in bank 7. The program then initiates a read of three or more cards and processes the first three cards while the next are being read (0.9 seconds).

The process continues until 150 cards have been read or the hopper is empty (a read with no errors occurred and 0 cards were read, and hopper empty flag set).

The information from the cards is partially processed and stored in a buffer in bank 6. The buffer is in the following format:

1. Card number in binary, 1 word.

2. Card type in 2-character, 6-bit trimmed ASCII. (Second character is always 22(R), 1 word.)

3. Patient number in 6-bit trimmed ASCII. (If an odd number of characters, the last half-word is 00.) The number of words will vary as to the length of the patient number in the system.

4. Accession number in 2-word binary. If accession number is not entered, the value 7403 7777 is put in. If an error occurs (internal blanks or illegal characters), the value 4207 7777 is put in.

5. Time in 3-word format. First word is the 2-digit trimmed ASCII value for the hour. Second word is the 2-digit trimmed ASCII value for the minutes. Minutes are truncated to be even if an odd value is entered. Third word is 2000 for a.m., or 4000 for p.m. Null time is entered as all zeros.

6. Date in the monitor format, 1 word. Error in date is entered as 4000. No date entered will assume today's date.

1	m1	m2	m2	m2	m2	d1	d1	d2	d2	d2	d2

7. Tests requested 1 word per test. Bits 0-7 give the column number and bits 8-11 are the row number. Columns are 1-80, rows 1-12.

8. End of card word, 1 word (7777). No disk reads are required to process the card information. All of the processing coding is located in bank 4.

After information from all three cards has been placed in the buffer in bank 6, the program executes the routine PROBK6 in bank 6 to process the information further and place it into the scratch file. The routine looks at the card type. If the card type is illegal, bit 0 of the card number is set. Otherwise, the card number is added to the scratch file, the card type omitted, the patient number added, a temporary subfile pointer of 7777 added, the accession number, time and date transferred. The column-row numbers for the tests are then compared to the list in file 50 for that particular card and the values which correspond are replaced by the pointer to the relative position in file 36. The end of file 7777 is then transferred.

The information in the scratch file for each card is in the format described above and in the following order:

1. Card number, 1 word, bit 0 set if invalid.
2. Patient number, length will vary.
3. Downpointer, 1 word (7777)
4. Accession number, 2 words
5. Time, 3 words
6. Date, 1 word
7. Test pointers, 1 word each
8. End of card, 1 word (7777)

When all of the information for all of the cards has been placed in the scratch file, the program jumps to bank 5 location 20 for error checking.

The first part of the error checking uses routine PATNUM located in the last 2 blocks of bank 5 to check the patient number for validity. The scratch file is read and the patient numbers are listed with the relative position (downpointer) in a table in banks 6 and 7. The tape buffer is opened if the number of words in the patient number is sufficient to require this space. The patient numbers are read in 4 blocks at a time and all the numbers from the cards are compared each time. When a match occurs, the relative position is stored in the downpointer word. The program continues until all cards are checked. Then, the program makes another pass through the scratch file and replaces the word following the patient number with the correct downpointer or if no match is found, 7776 is stored in the downpointer word.

The second part of error checking uses routine TEST to process the test requests. Overlay RG is read into the last two blocks of bank 5 for the processing. The routine reads in a card information, sets up a bit map for the tests requested today on the particular patient in BITMP1. Another scratch file is opened and the information is transferred to the new file because the new file may be longer. Each test requested is read in and a check is made as to whether it was already requested today (a repeat) in BITMP1 or whether the test was previously requested on this card (a duplicate) in BITMP2. Parts of packages are only checked for repeats as they are assured against duplications. For single tests and batteries, the format in the list remains one word. Bit 0 is set if the test is a repetition, bit 1 is set if a duplication. For a package, the one word format is expanded. Bit 2 of the original word is set to show that a package appears. The following two words are a dummy 7777 7777. Following these, each of the codes for the batteries within the package appear with bit 2 set and followed by 0000 0000 as dummy words for future accession number. If consecutive cards are for the same patient, the program will include the previous cards in determining repetitions.

The routine then closes the first scratch file and proceeds to read in R0 for the last part of error checking.

The third part of the error checking is ACCNUM, which is located in overlay R0 as read into bank 4. The routine makes a pass through the scratch file and makes a list of the card numbers and the accession numbers assigned to the cards. The accession numbers, originally in the form XXXXXXXXXAB CDEFGHIJKLMN are transformed into the form ABCDXXXXXXAB CDEFGHIJKLMN where the first four bits indicate the disk read in which the requested accession number can be found. When a number is found, bit 4 is set in the two word format, the appropriate word in file 27 is set to 7776 and the number is stored in a list starting at word 1000 of bank 4. After all of the requests have been checked, the null values (originally 7403 7777) are assigned an available accession number. The errors 4207 7777 will not be checks as bit 4 is already set. The routine then makes a pass through the scratch file and places the appropriate value in the accession number words and then files through the test requests for packages. In the two words allocated after each part of a package (except for the first package words which remain 7777 7777) an assigned accession number is placed. If no tests other than packages are found, the accession number is set to 7600 0000. The program then reads in overlay R7 into bank 5 and processing has been completed.

After the initial processing has been completed, the operating sequence remains fairly stable. The options for handling are controlled through overlay R7 in bank 5 and all other programs return after completion. The last two blocks of bank 4 always contain the list of accession numbers saved. The option programs are located in bank 6 and, if necessary, in bank 7. Any programs needing storage space use the first two blocks of bank 4 and the available space in bank 7. If additional space is needed in bank 5, program R7 is restored. A return without disturbing R7 returns to location 21 of bank 5. A return after rereading R7 must be to location 20 of bank 5 with the following values in bank 4. The output device code (OUTDEV) in location 4, the number of words in the patient number (COUNT) in location 5, the disk read parameter for the scratch file (DISK17) in location 6 and the number of cards in the scratch file NUMCRD in location 7. These four values (OUTDEV, COUNT,DISK17 and NUMCRD) are common between overlays except that some may be complemented.

After R7 is read in,the following message will be printed on the calling terminal:

```
ENTER CODE*
```

The operator should type the proper code for one of the options below:

A -- to file all cards without errors with the option of typing in exceptions.

N -- to file all cards without error as typed in.

L -- to print mnemonics for error codes appearing on cards.

P -- to print the card information on the appropriate output device.

SHOW ME -- to print an explanation of the options.

STOP -- to terminate the program.

When option A is selected, the program calls routine GOODLT to leave a list of card numbers for cards without errors at LISTAB in bank 4. The first word of the list is the negative of the number of card numbers in the list. GOODLT accomplishes its task by calling routine GETERR to form a list (ERRTAB in bank 7) of card numbers and their appropriate errors. The error word is as follows:

Bit Set	Meaning
0	Invalid card type
1	Illegal patient number
2	Illegal accession number
3	Illegal time
4	Illegal date
5	Duplication of test type
6	No tests requested

GOODLT then only selects the card numbers whose error word is zero. The program then asks the operator to type in a list of excluded cards. This list should be legal card numbers, separated by commas. Cards with errors need not be entered as they will be rejected anyway. The numbers typed are read and placed in numerical order by routine GETLST. The exceptions are then removed from the LISTAB list and R9 is read into bank 6 in order to file the cards listed.

When option N is selected, the program calls routine GOODLT for a list of the cards without errors. Then routine GETLST is called to input a list of cards to be included in the list to be filed. The two lists are compared and any card that is input and is not on the list of cards without errors will be typed out as not able to be filed. The others will be placed in LISTAB and overlay R9 read into bank 6.

When option L is selected, routine GETERR is called and then the program calls overlay RH into bank 6. Program RH types the card number and the single letter code corresponding to any bit set in the error word. The codes are separated by commas if there are more than one per card and are as follows:

Code	Error Bit Set	Error Type
I	0	Card type
N	1	Patient number
A	2	Accession number
T	3	Time
D	4	Date
R	5	Duplication of test type
M	6	No requests

The calling terminal will request whether the operator desires an explanation of the codes. Entering Y will cause this explanation to be printed, anything else will cause a return to R7.

When option P is selected, the program loads overlay R8 into bank 6. R8 reads the scratch file and prints the information on the selected output device (OUTDEV). After the cards are printed, the program returns to R7 in bank 5.

When SH or SHOW ME is typed, an explanation of the options is typed on the calling terminal.

When STOP is typed, the program uses the list in bank 4 of the accession numbers and restores the 7776 in file 27 to the original 7777 so that others may use those accession numbers. The program then closes the scratch file and exits.

Both the A and the N options call R9 the requisition filing routine. R9 is read into bank 6 and the first block of bank 7. The routine uses LISTAB to file cards in the list, if possible. The program first calls routine NUMGET which finds the card in the scratch file, forms in the correct format the accession number, time and date, and sets up a list of the number of three-word slots to be allocated for each test requested in table SLOTTB in bank 7.

The program then calls routine PRELIN which searches the patient's file and determines the number of blocks of data for the patient NMBLK, the first block of data for the patient FSTBLK, the block and word for the end of file word NEDBLK and ENDWRD. A flag, HEADFL, is set if the current day header is in the file. When HEADFL is set, locations HEADBL and HEADWD contain the block and word for the current day header. The program then calls routine

CHKLEN which verifies that the added tests will not exceed the 16 block limit as the length of a patient's file. CHKLEN adds the number of words determined by (NUMBLK X 256) + ENDWRD + (sum of values in SLOTTB) X 3 + 16. If this value is greater than 4096, the program continues with the next card. If the value is less than 4096, the program calls routine PROCES to add the tests to the patient data in file 30.

NUMBLK is checked and if there is no data in the file, a block is secured by setting a bit in file 34 by routine FILE34. The correct header and test count is set up, and then routine FILLIT puts in the new requests. An end of file is added, completing the file. Routine FILLIT removes the accession numbers from the list in bank 4 by changing the value to 7777 7777 and places the proper value in file 27.

If there is data in the file and the current day is not in the file, the routine adds the new requisitions at the end, using FILLIT and the pointers in the first block are modified to reflect the change. When the current day header is in the file, the new tests are added to the end of existing tests for that day, with new blocks of data created. The remainder of the file is then transferred into the new block chain. Any blocks that are no longer in use will be released for use. When a card is filed, bit 0 of the word in LISTAB is set to 1. AFter all of the requisitions are entered, the program jumps to routine DELCRD in bank 7 which removes all of the cards filed from the scratch file.

DELCRD then loads overlay RI into bank 6. RI searches through the LISTAB list and first prints the list of cards filed, and then prints the list of the cards not filed. RI then rereads overlay R7 into bank 5, sets up the initialization of the pointers for R7 and jumps to bank 5.

3.8.1 Error Messages (Card Reader)

The following error messages will be printed during the operation of the program during card reader input.

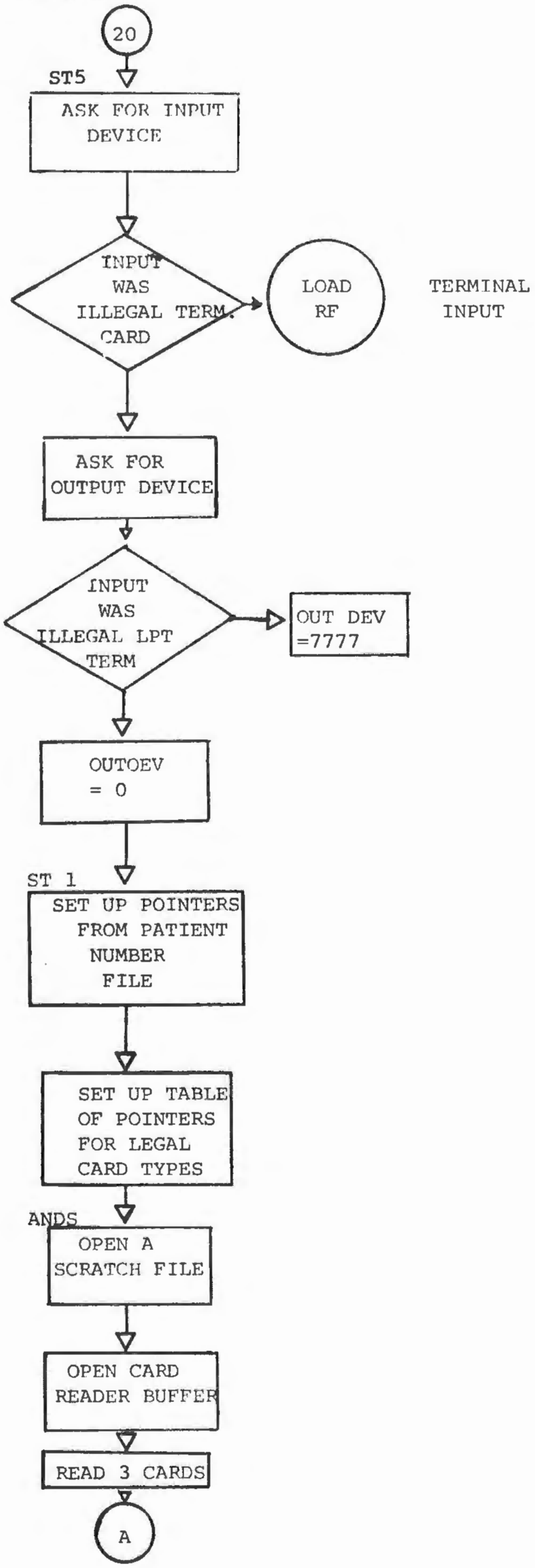
MESSAGE	MEANING
TYPE 1 OR 2	A response of 1 or 2 was expected and was not received.
FEED ERROR, CHECK CARD DECK	A feed error was found in reading cards. Reload deck and press RETURN.
CARD READER NOT READY	The card reader was not turned on or the ready button was not lit.
WAITING FOR SCRATCH FILE	A scratch file counld not be obtained. The program will wait until a file is available.
DISK ERROR	A read or write error was found when working with the disk. Program will exit.
NO GOOD ENTER CODE	An illegal code was typed and rejected. Only P, L, N and A are valid.
ILLEGAL CHARACTER IN STRING	An illegal code was typed when entering lists of excluded or included cards.
NUMBER TOO LARGE	A number larger than 150 was entered in a list of excluded or included cards.
NO CARDS IN LIST	All cards read have been filed. Program exits.

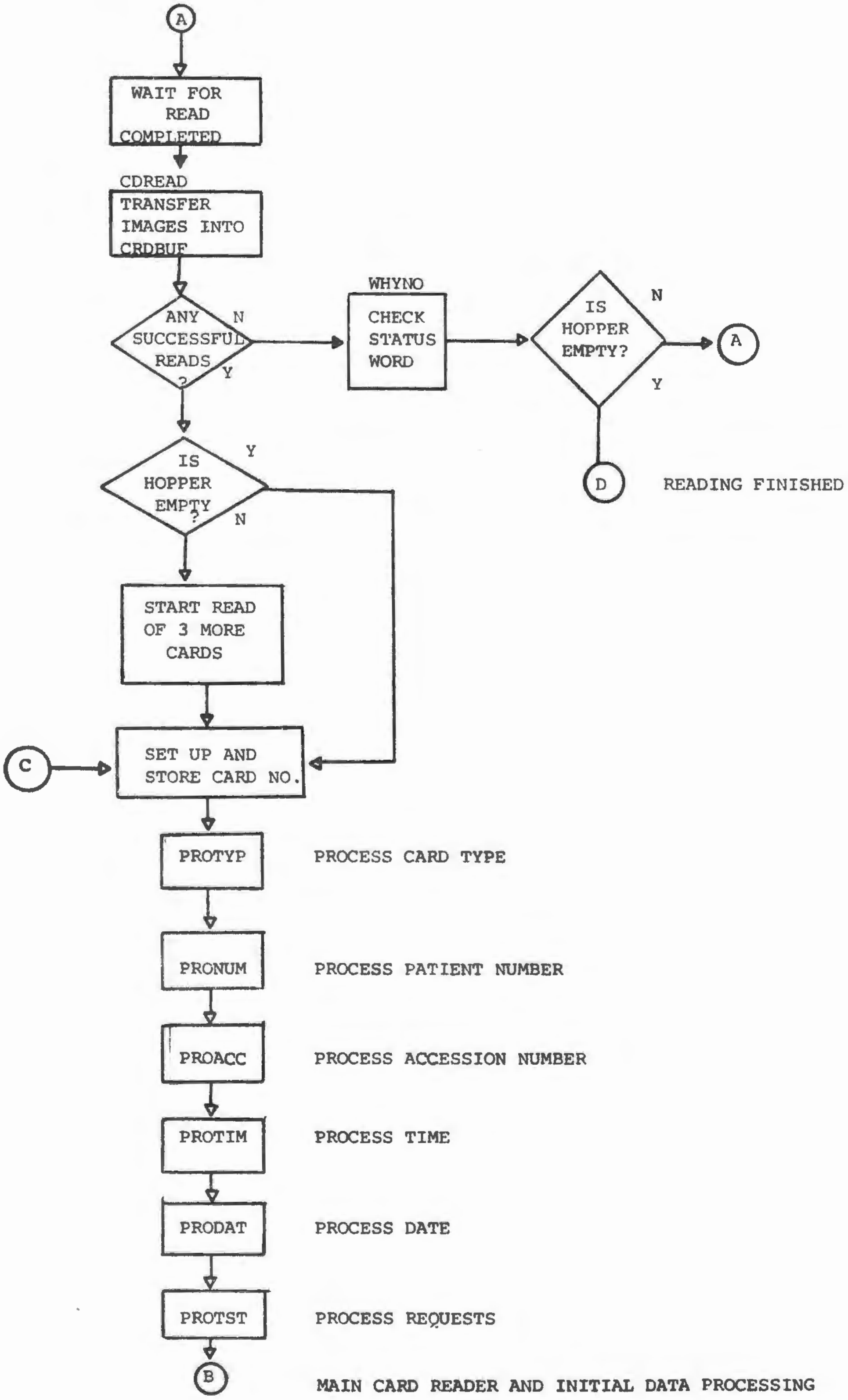
3.8.2 Card Requisition Entry Assembly Instructions

All numbers are octal.

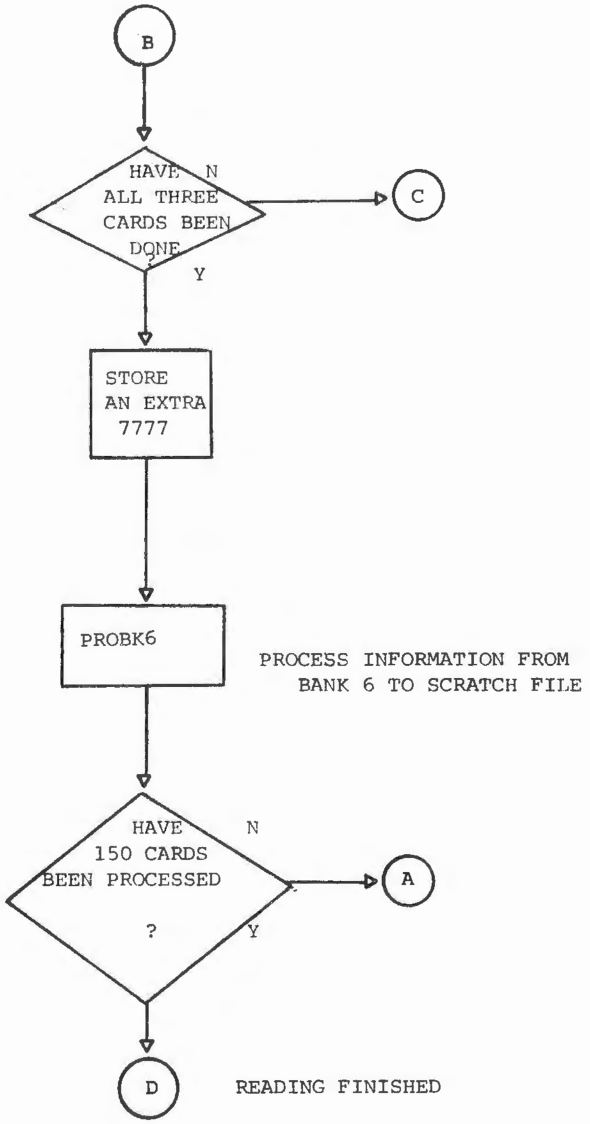
ASSEMBLY PROGRAM	CHAINS TO PROGRAMS	PRODUCES BINARY BLOCKS	START AT BLOCK	NUMBER OF BLOCKS	ON STARTUP TAPE AS
RE	RE-1	15	1	12	RE
	RE-2		13	2	RG
	RE-3				
R7	None	5	1	4	R7
R8	None	6	1	5	R8
R9	None	7	1	6	R9
R0	None	3	1	2	R0
RH	None	3	1	2	RH
RI	None	3	1	2	RI

3.9 FLOW CHARTS FOR CARD REQUISITION ENTRY

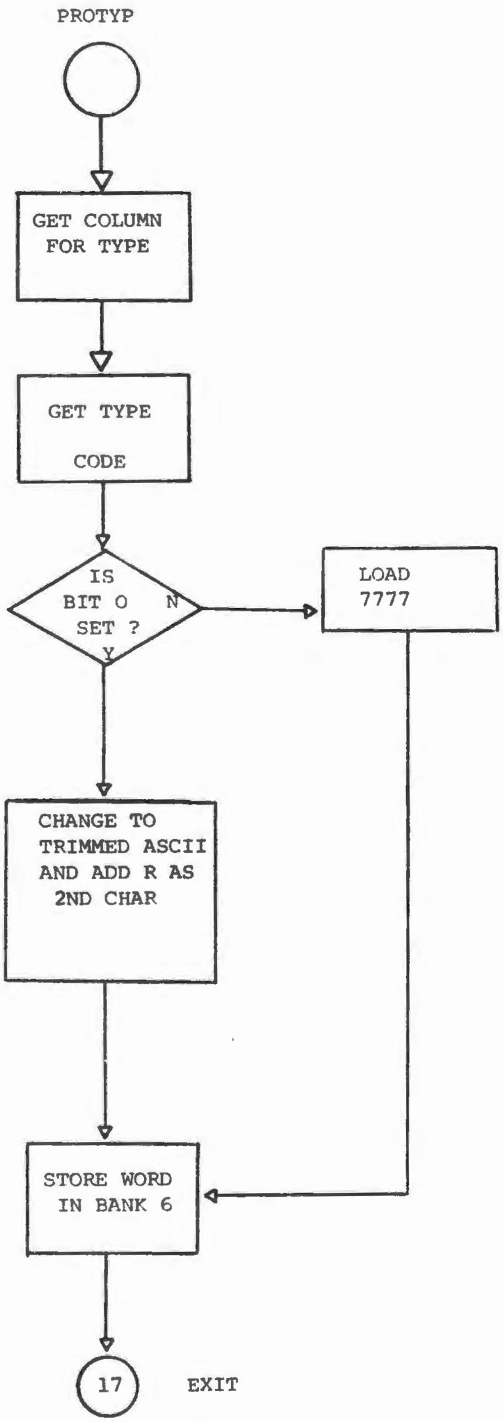




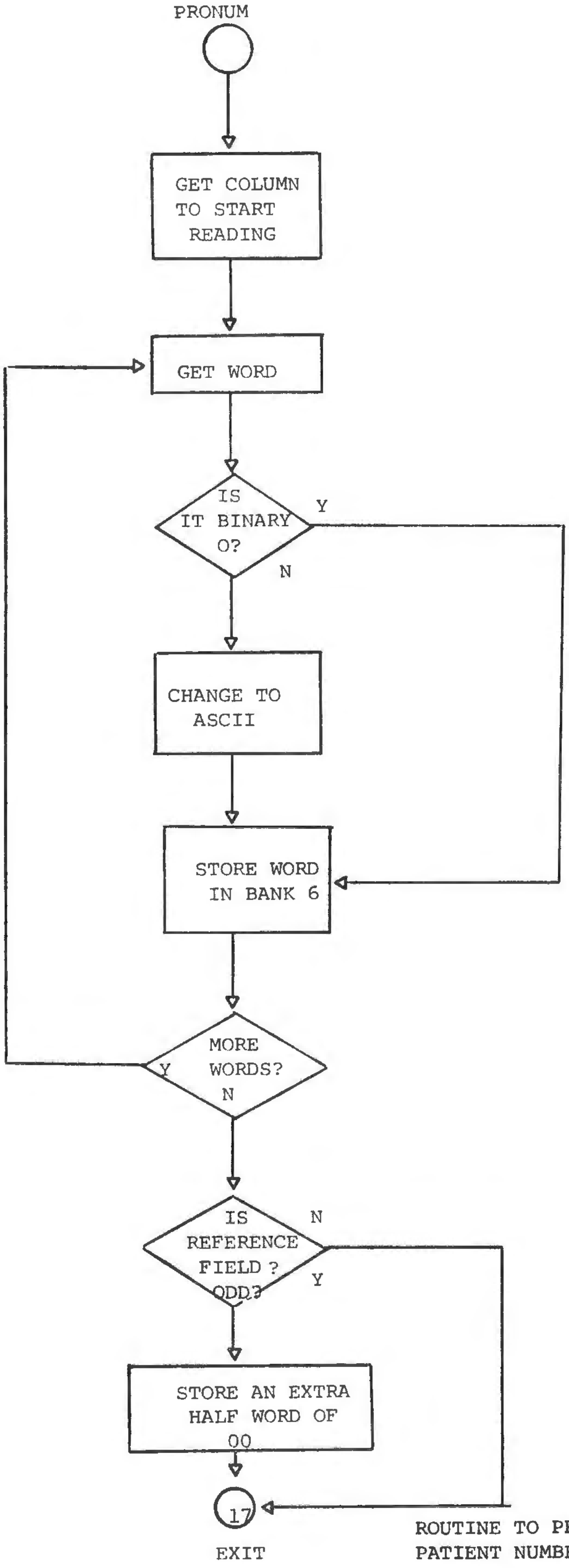
MAIN CARD READER AND INITIAL DATA PROCESSING


B
HAVE ALL THREE CARDS BEEN DONE ?
N
C
Y
STORE AN EXTRA 7777
PROBK6
PROCESS INFORMATION FROM BANK 6 TO SCRATCH FILE
HAVE 150 CARDS BEEN PROCESSED ?
N
A
Y
D
READING FINISHED

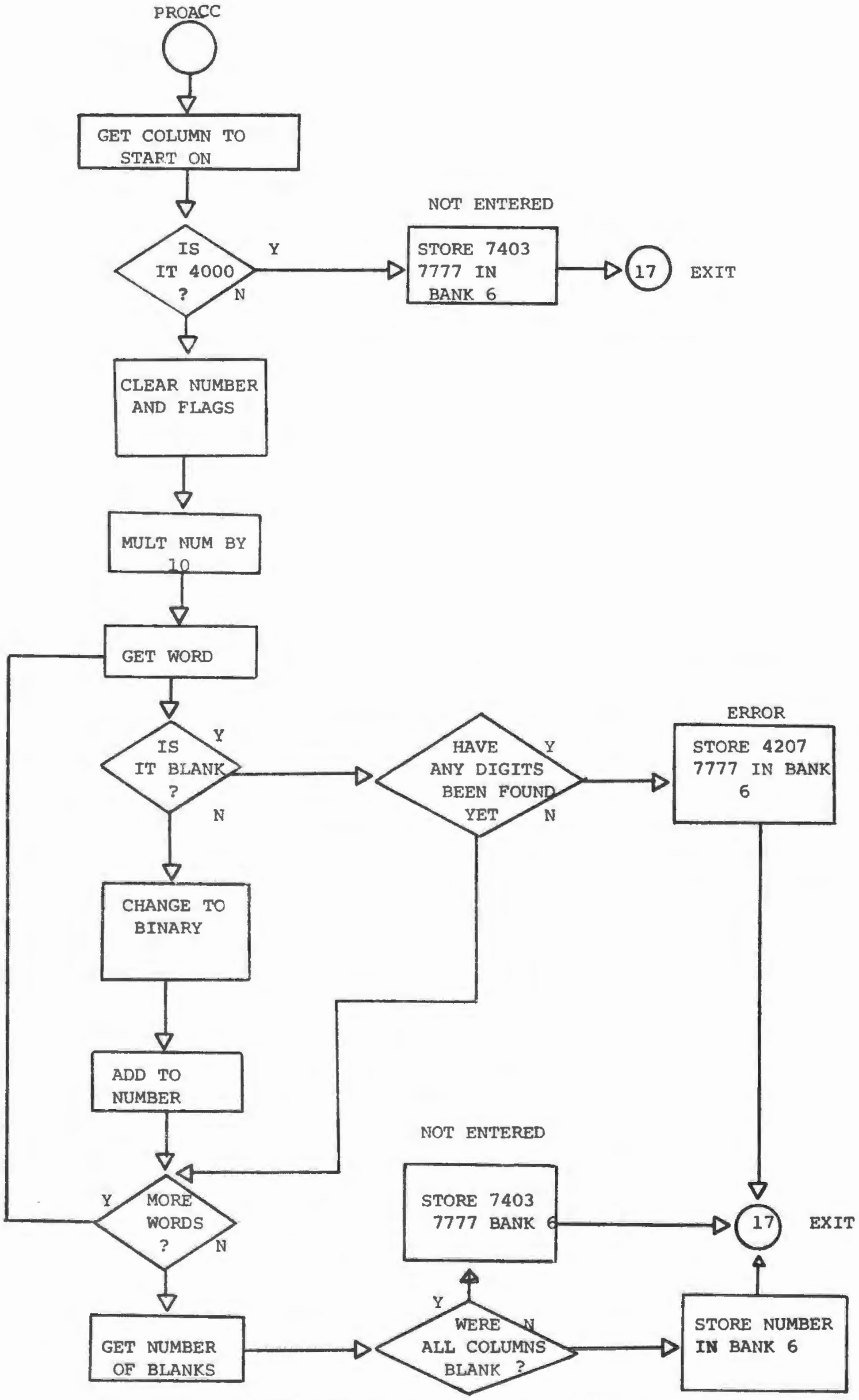
CARD READER TERMINATION LOOP



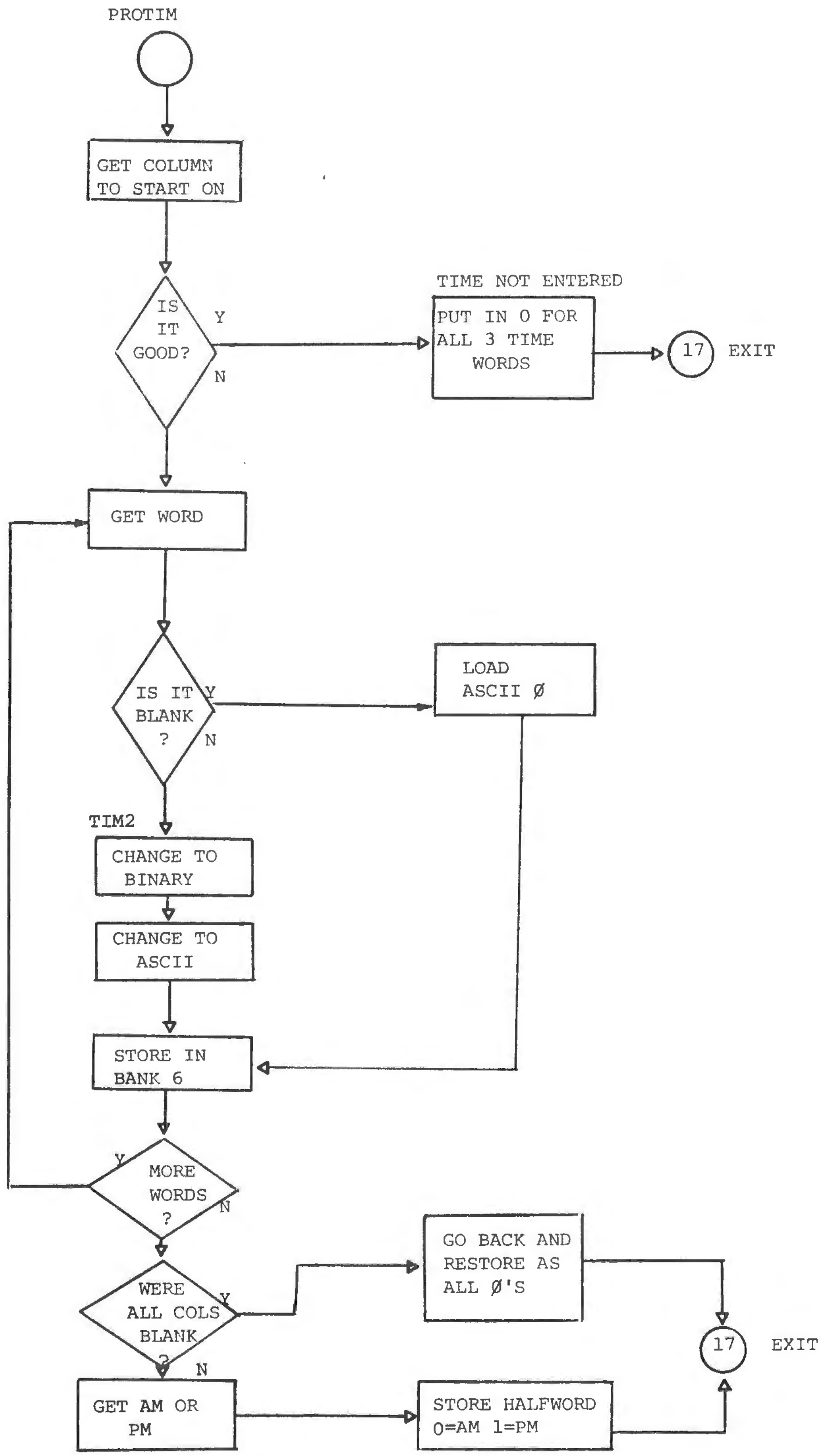
ROUTINE TO PROCESS CARD TYPE CODE



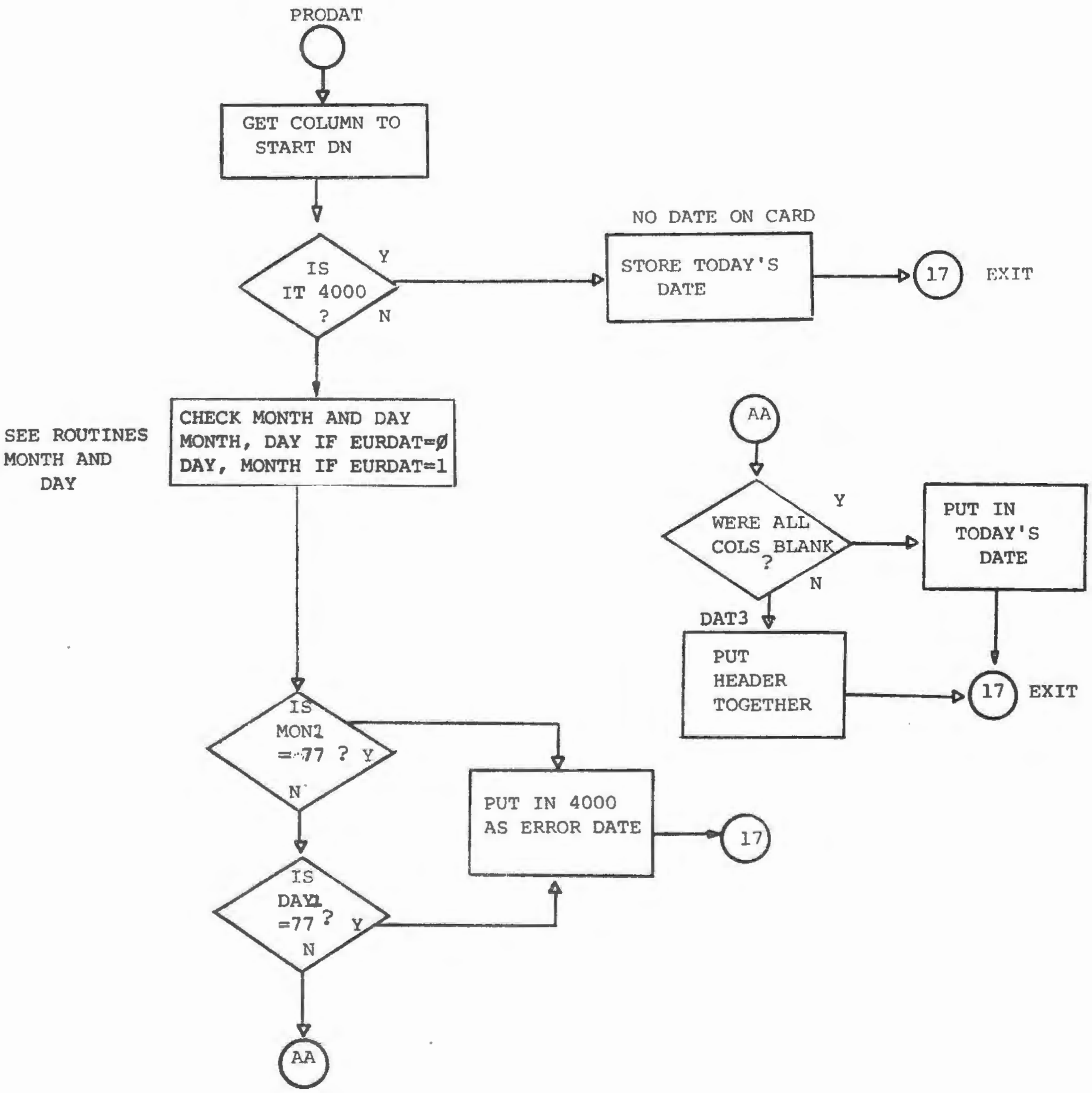
ROUTINE TO PROCESS
PATIENT NUMBER FROM CARD



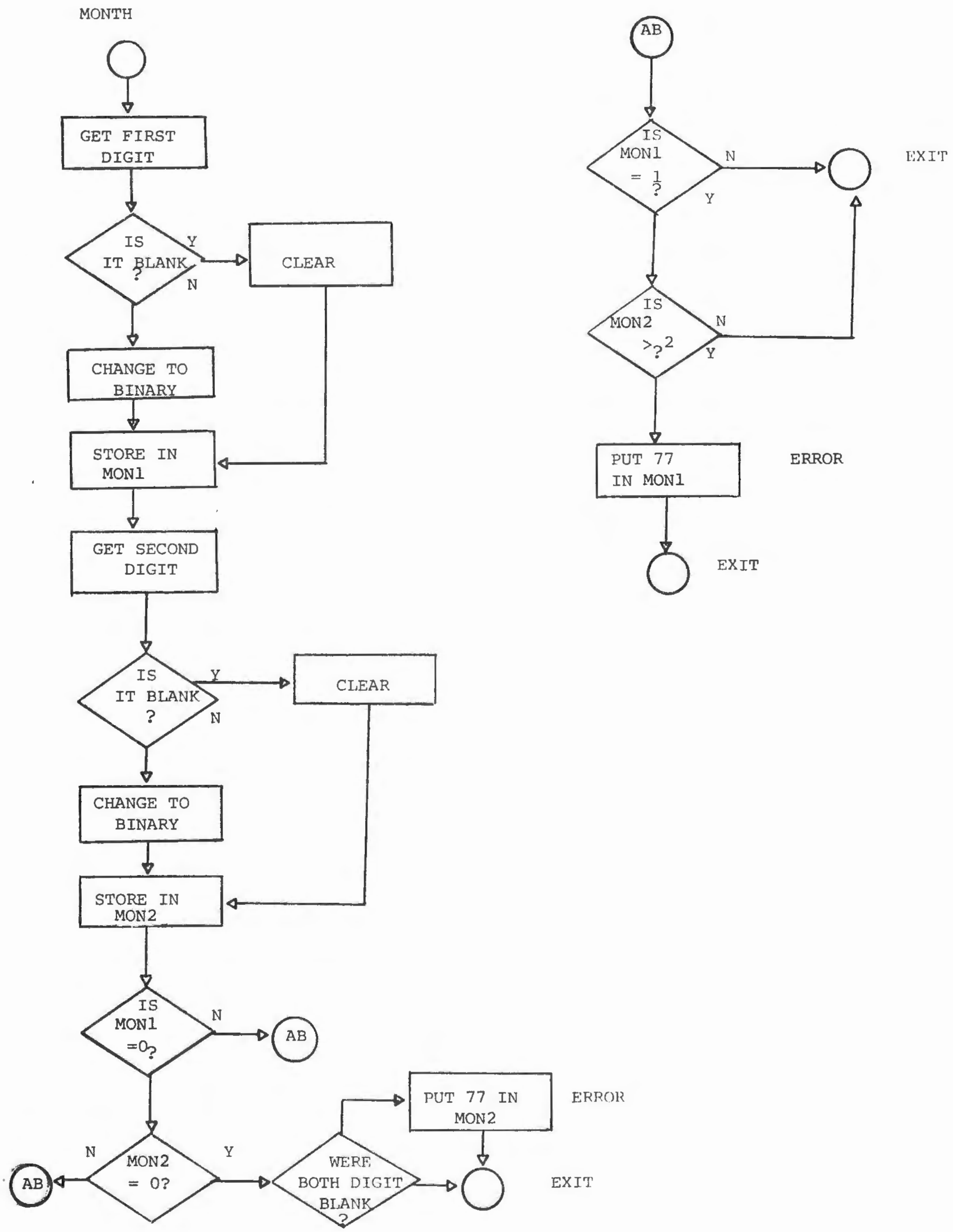
ROUTINE TO PROCESS ACCESSION NUMBER



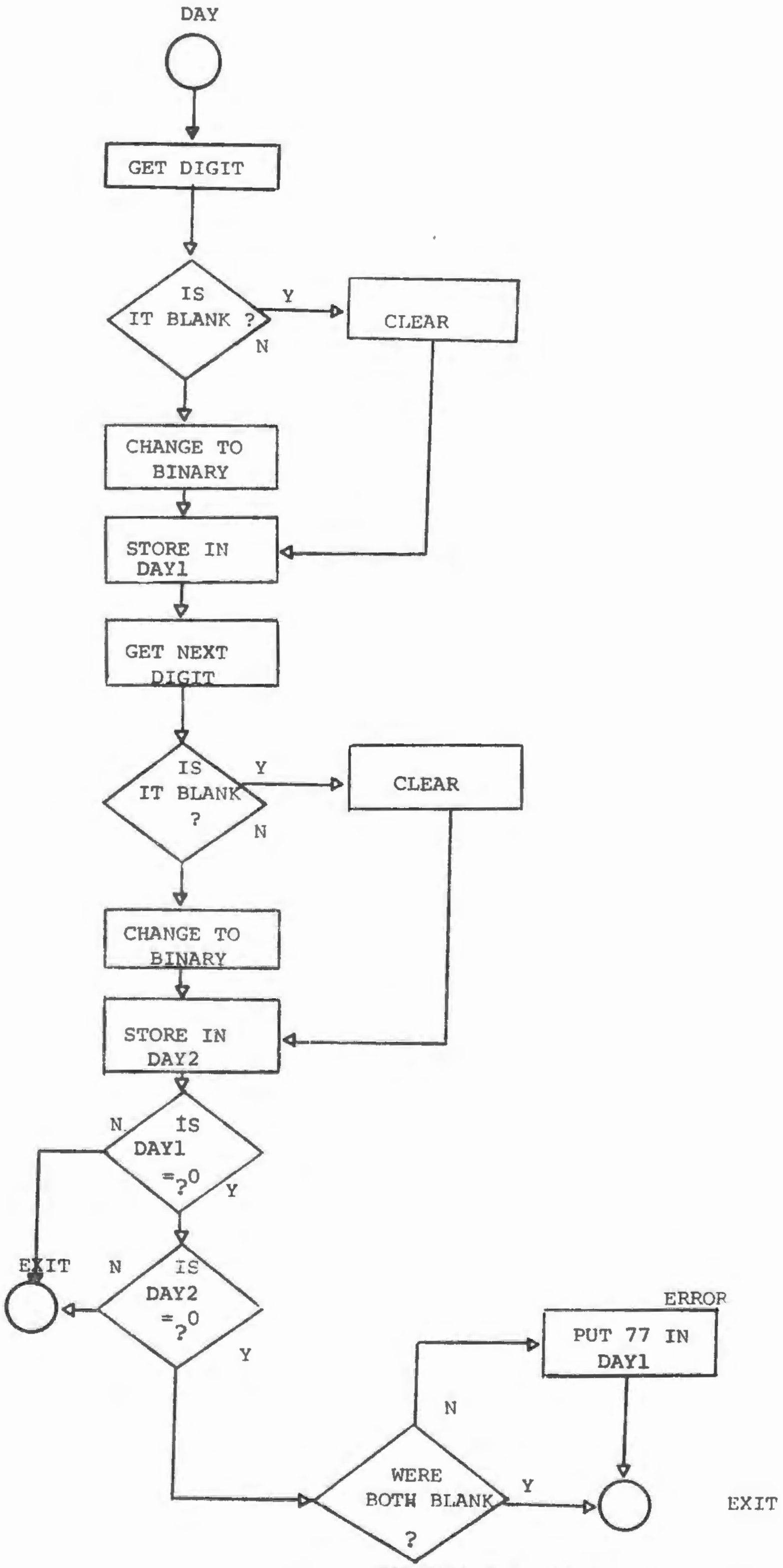
ROUTINE TO PROCESS TIME



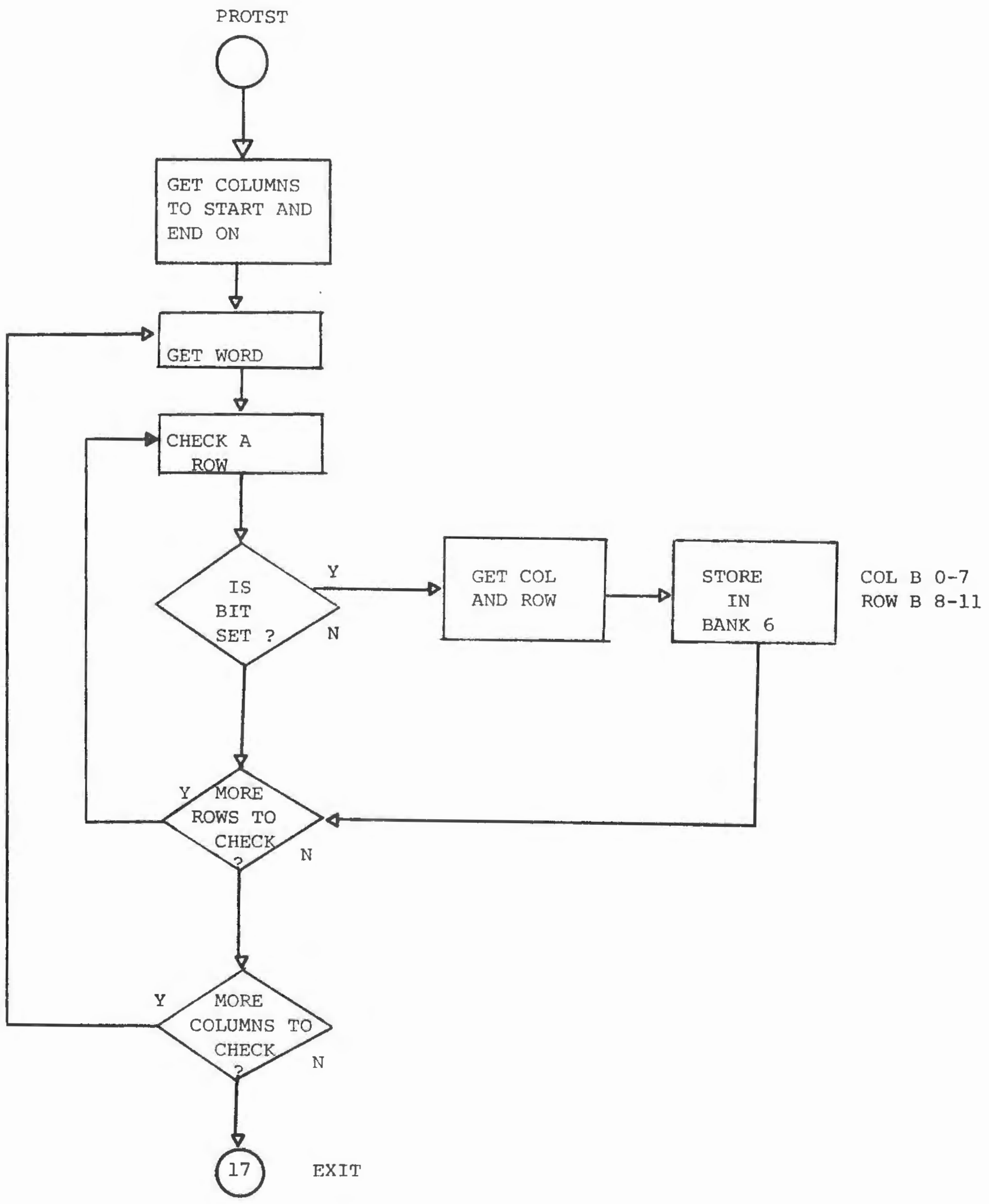
ROUTINE TO PROCESS DATE



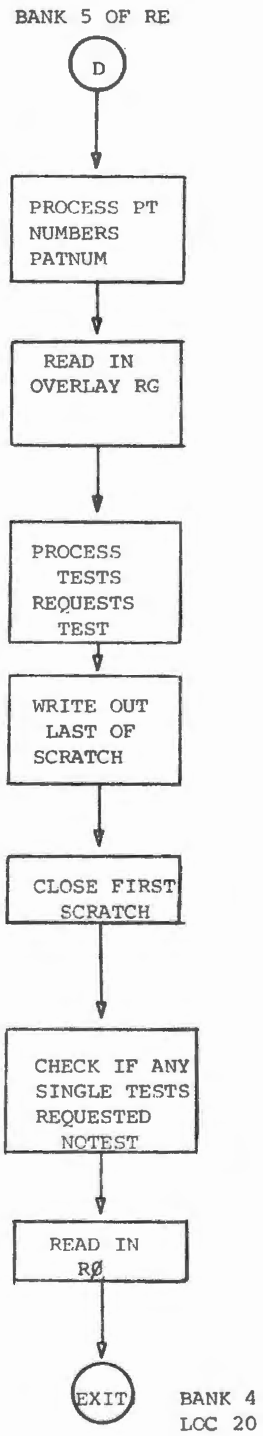
ROUTINE TO PROCESS MONTH
PART OF PRODAT ROUTINE



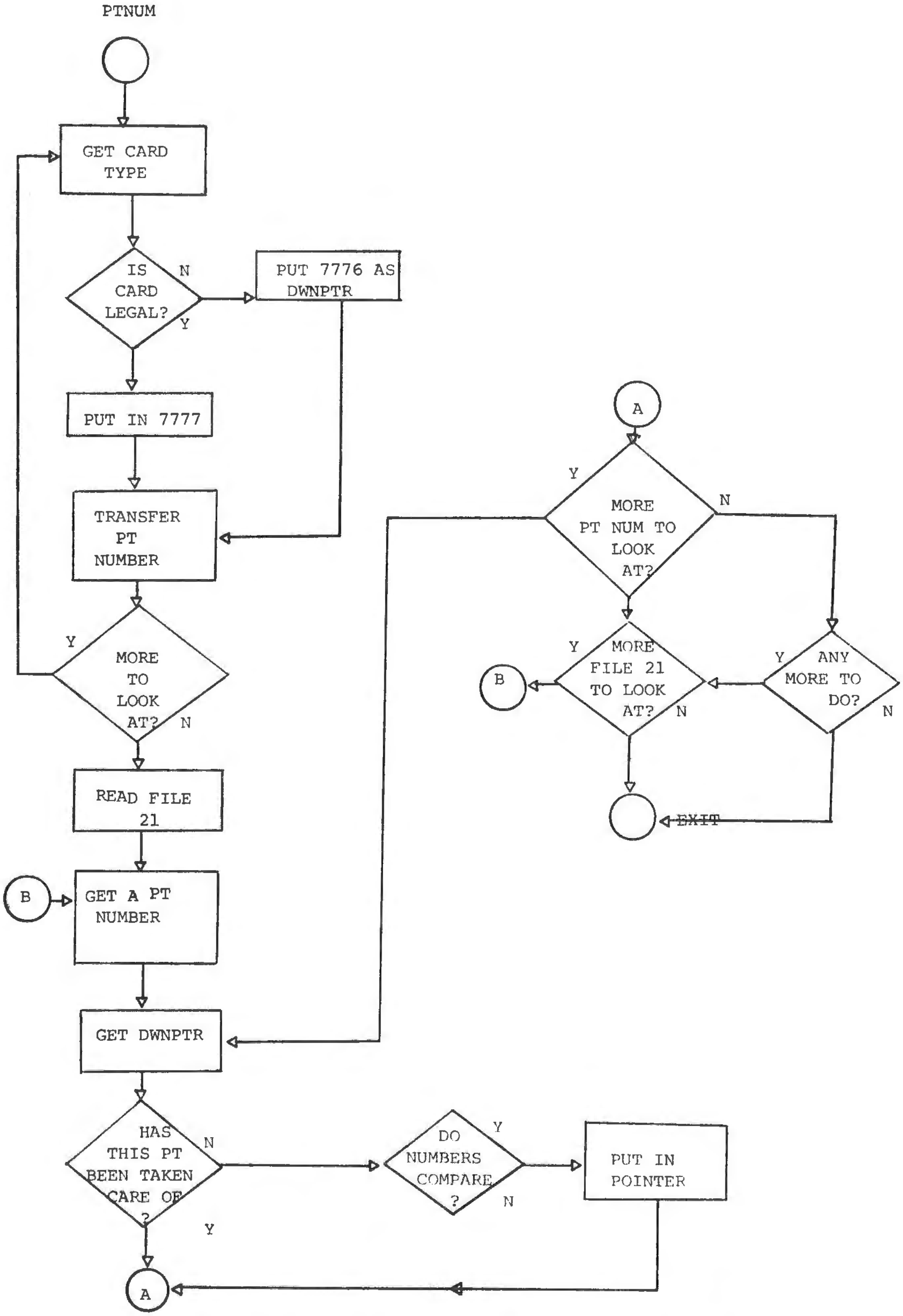
ROUTINE TO PROCESS DAY PART OF PRODAT ROUTINE



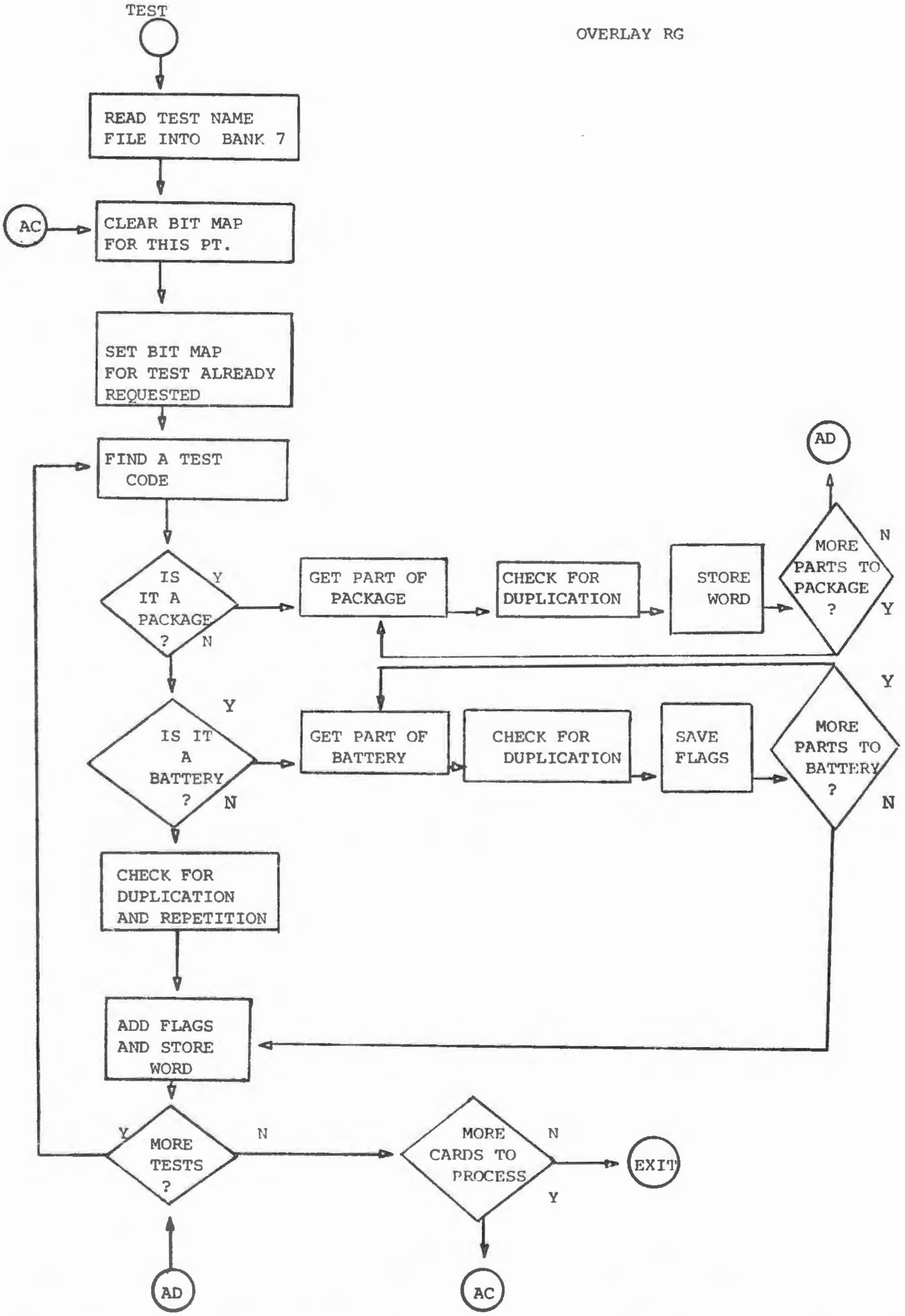
ROUTINE TO INTERPRET NON-COLUMN MARKINGS AS TEST REQUESTS


BANK 5 OF RE
D
PROCESS PT NUMBERS PATNUM
READ IN OVERLAY RG
PROCESS TESTS REQUESTS TEST
WRITE OUT LAST OF SCRATCH
CLOSE FIRST SCRATCH
CHECK IF ANY SINGLE TESTS REQUESTED NOTEST
READ IN RØ
EXIT
BANK 4 LOC 20

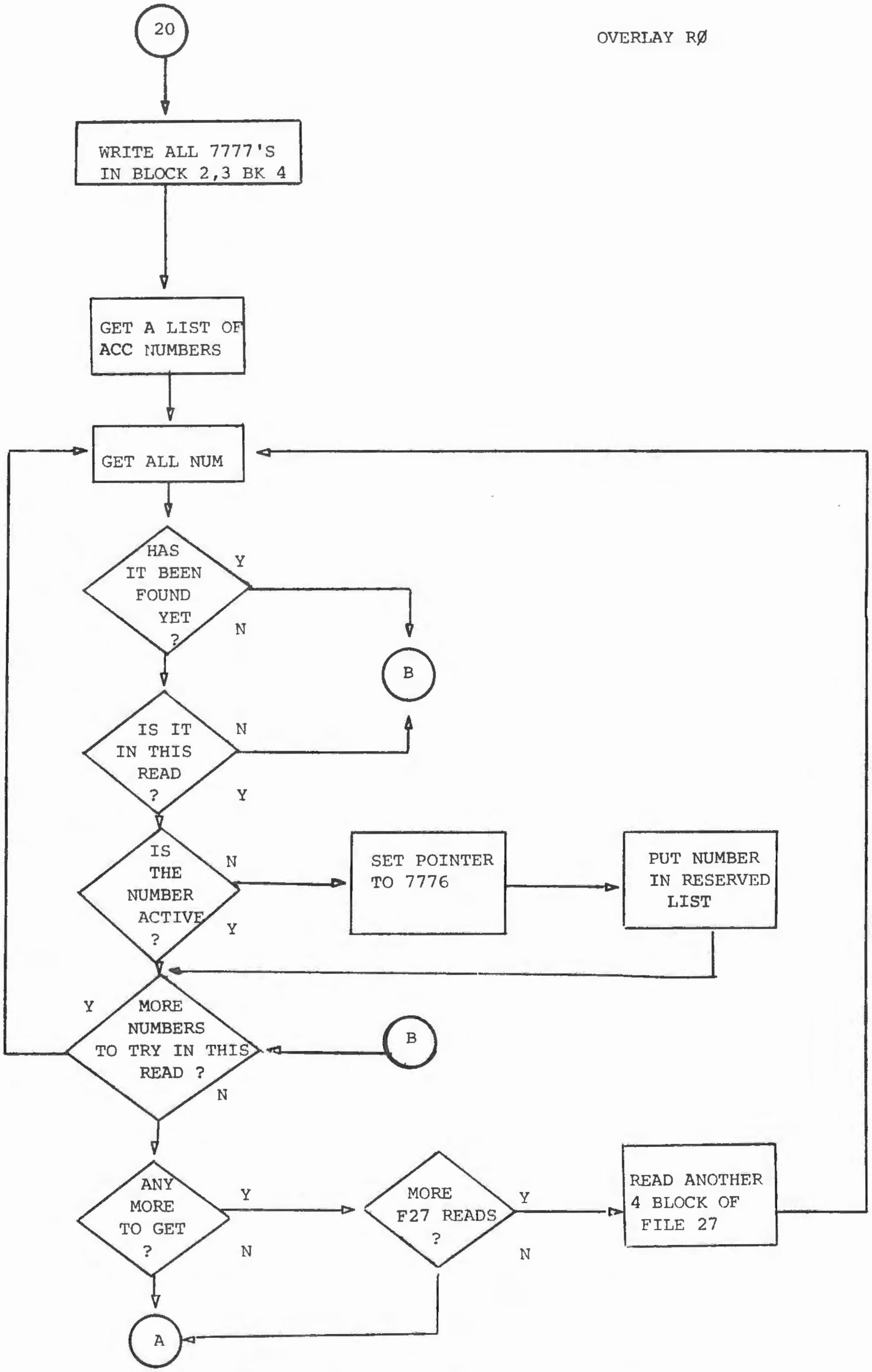
LOOP FOR ADDITIONAL ERROR CHECKING



ROUTINE TO ENTER FILE 26 POSITION FROM PATIENT NUMBER

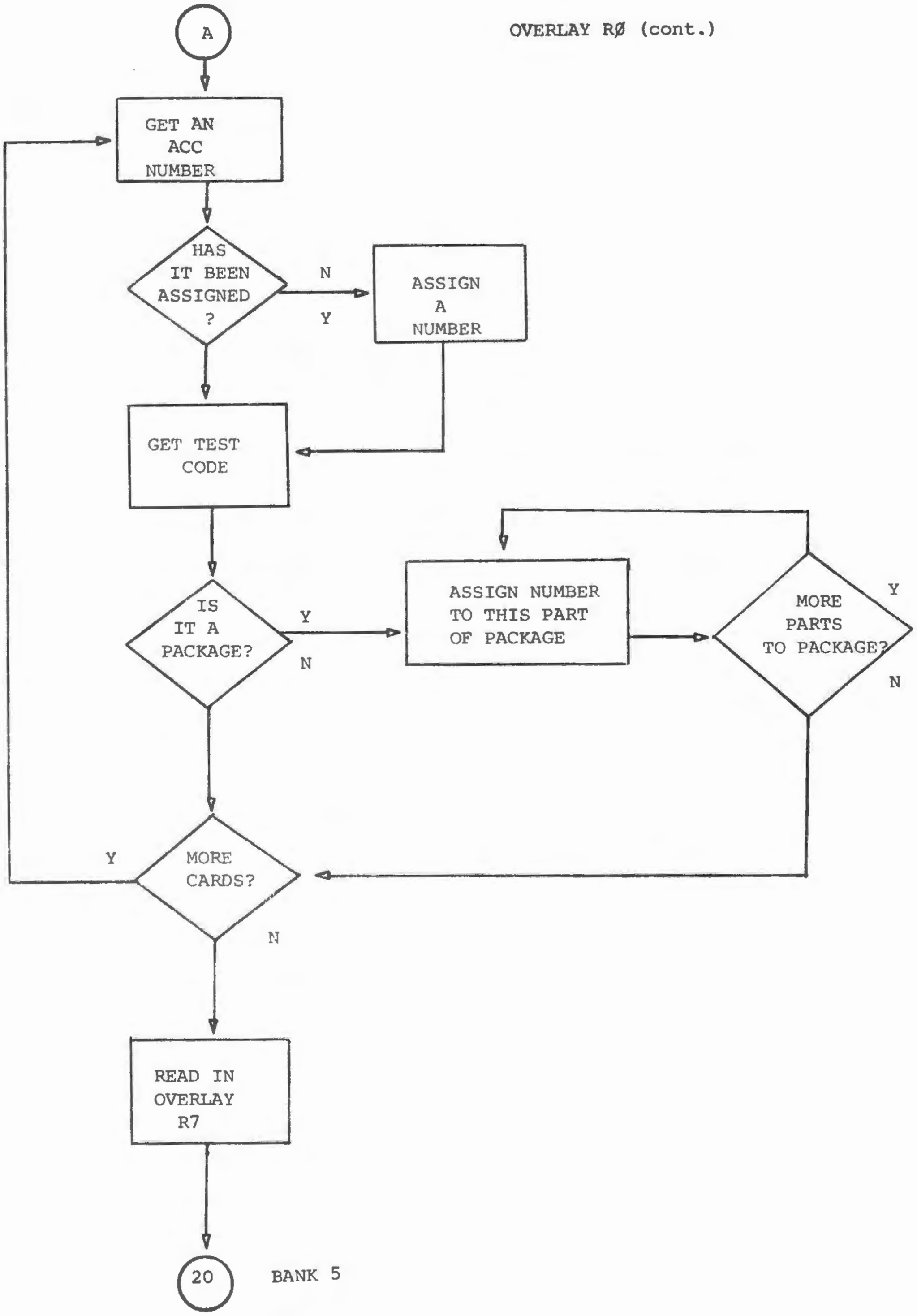


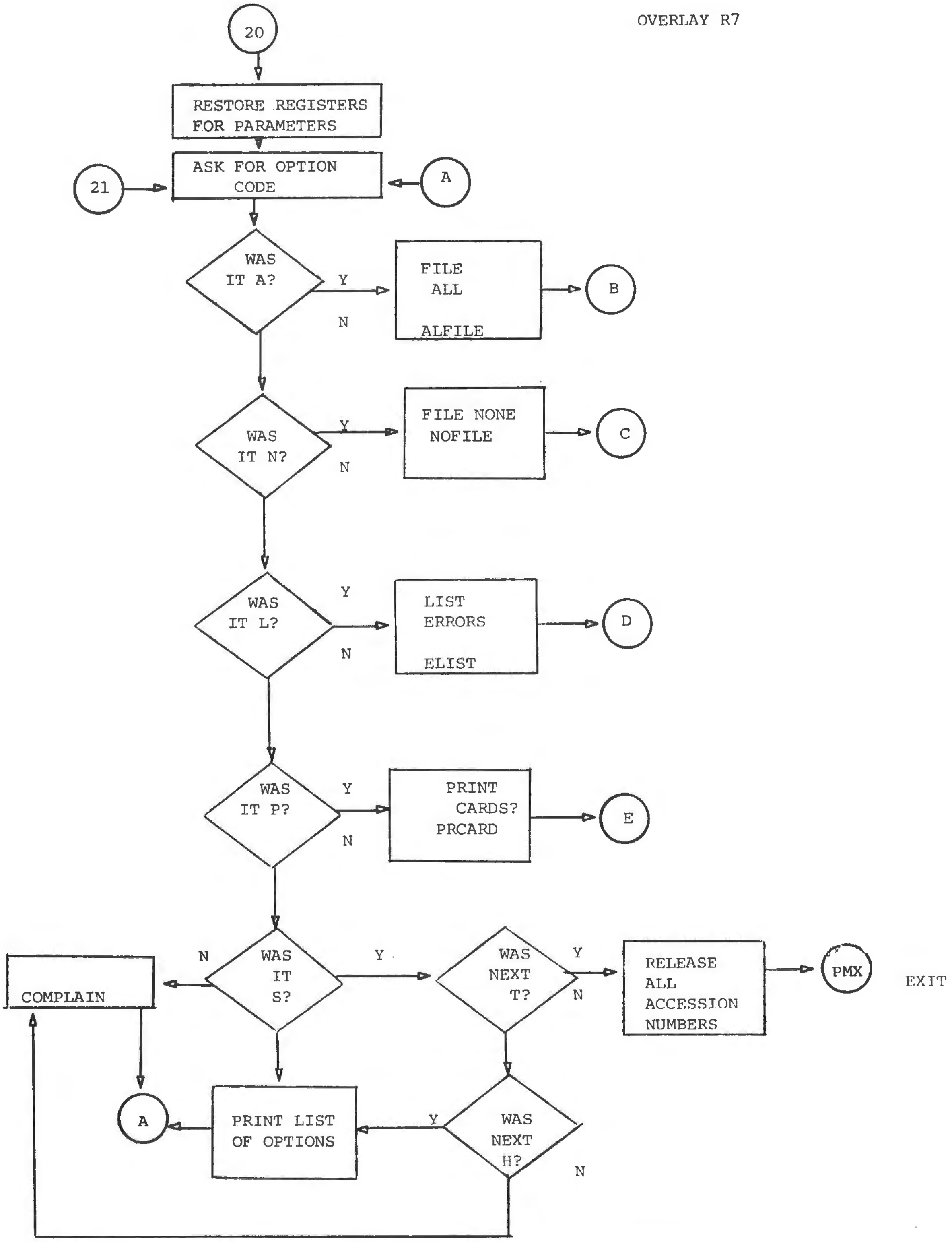
OVERLAY TO CHECK TESTS FOR DUPLICATION AND REPETITION AND SET APPROPRIATE FLAGS



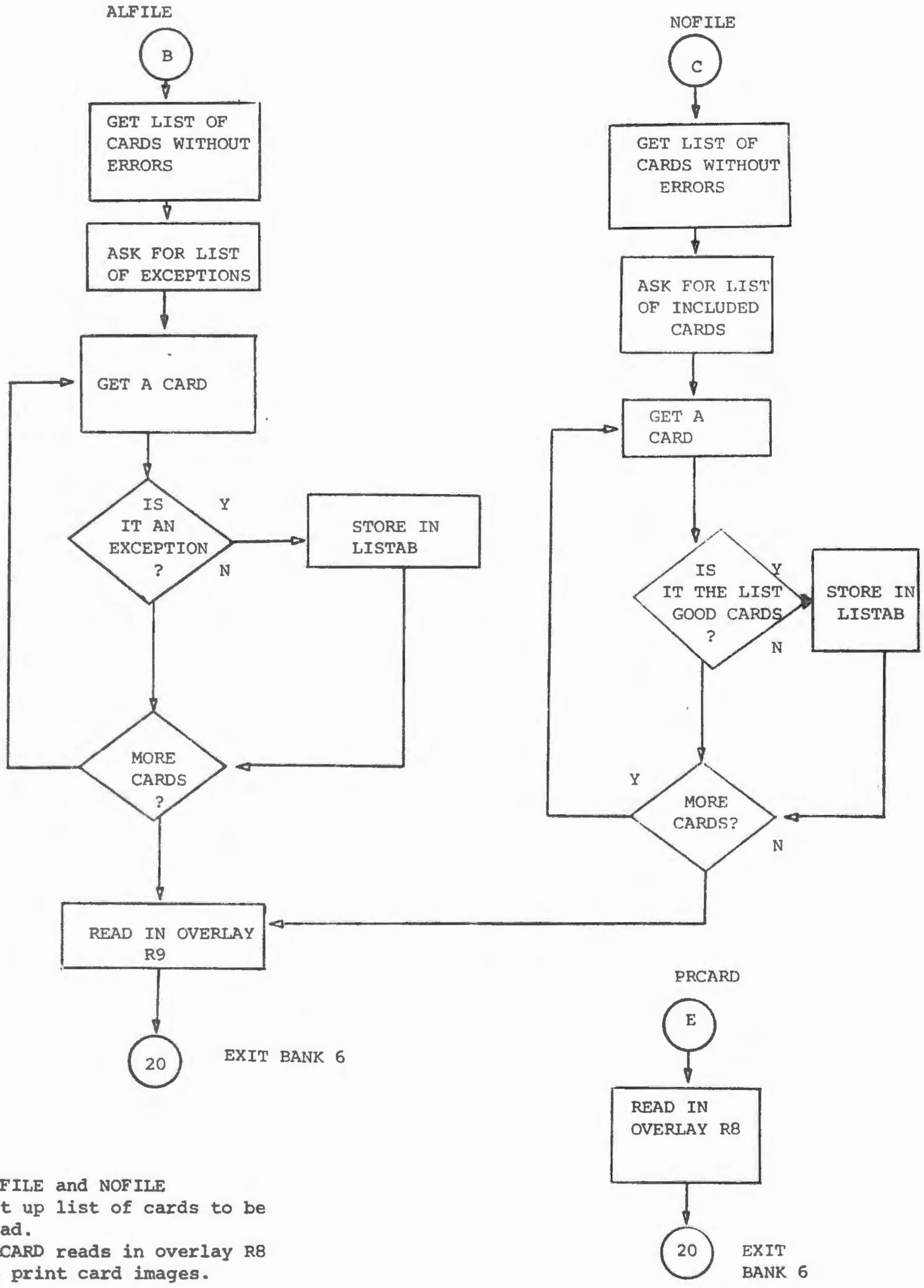
OVERLAY TO RESERVE ACCESSION NUMBERS

OVERLAY RØ (cont.)

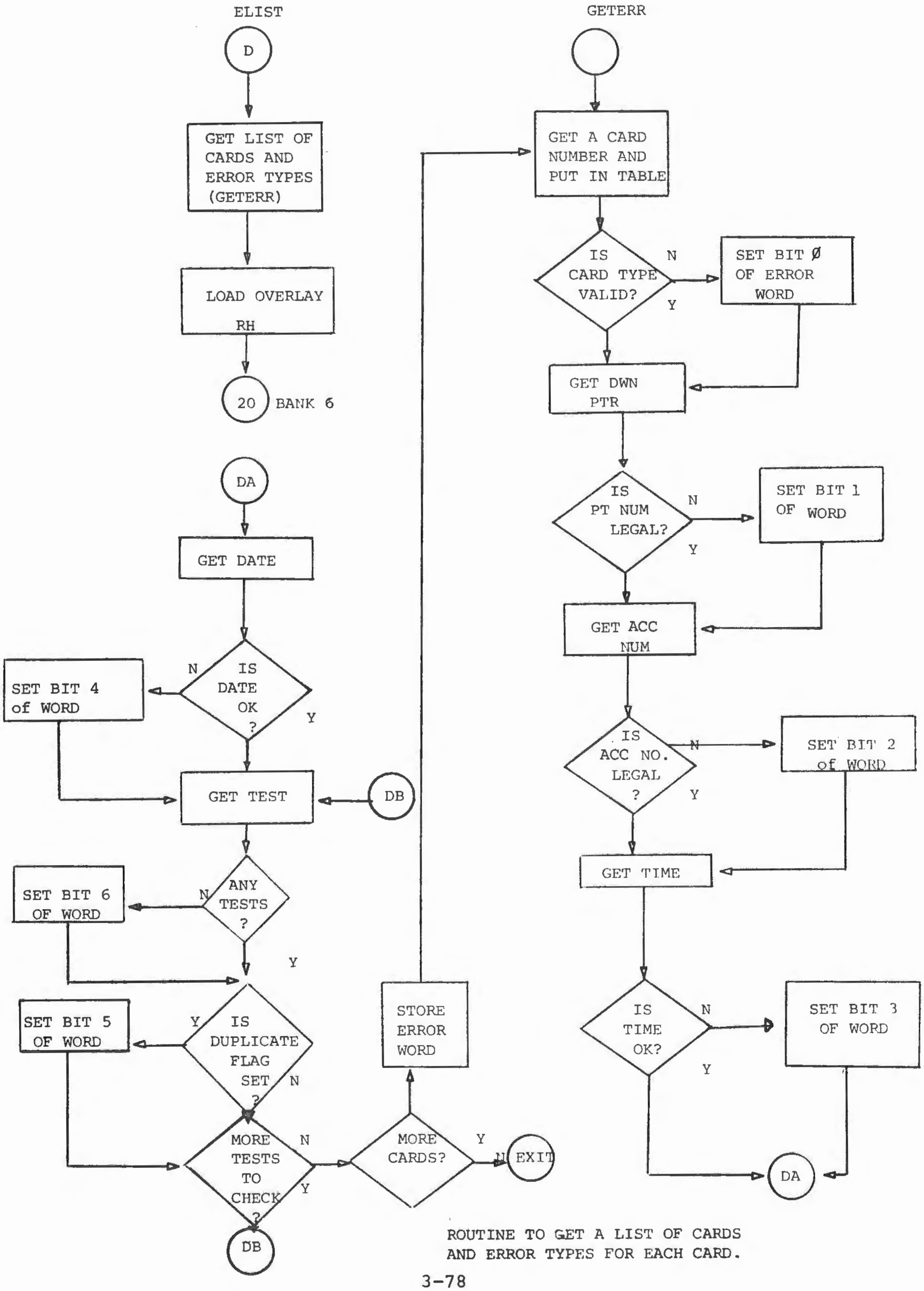




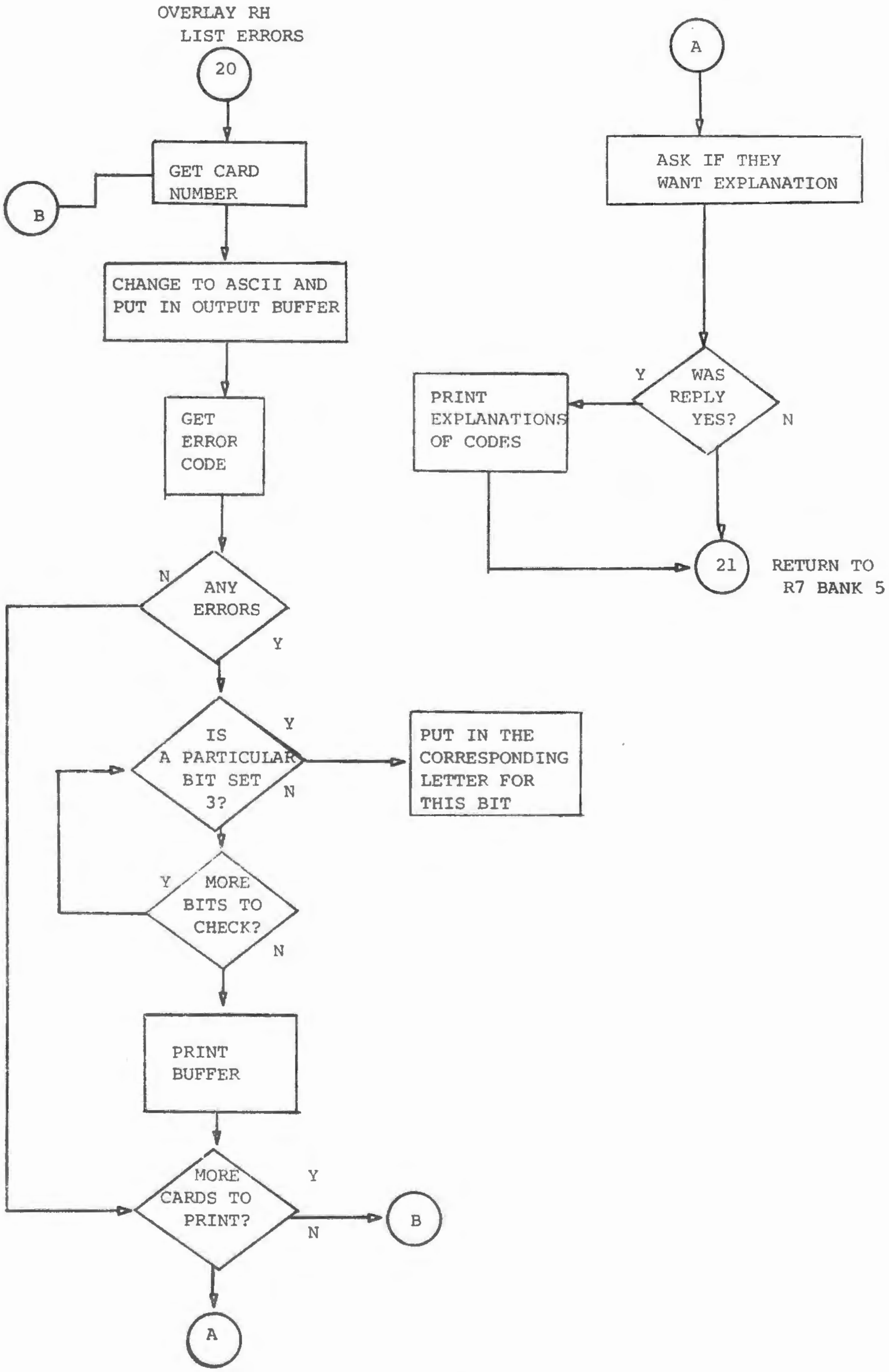
MAIN COMMAND LOOP FOR DATA VERIFICATION AND STORAGE

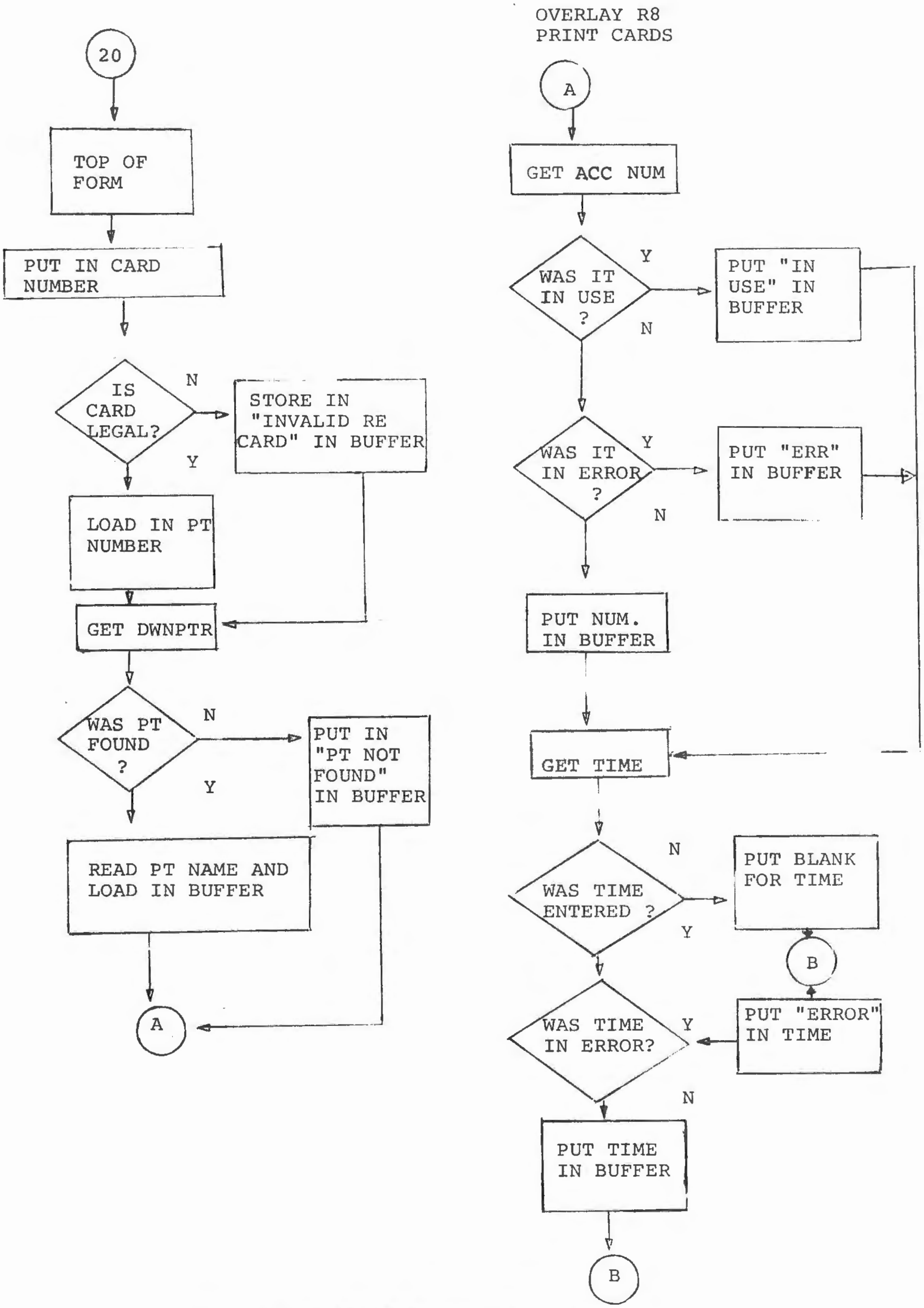


ALFILE and NOFILE
Set up list of cards to be read.
PRCARD reads in overlay R8 to print card images.



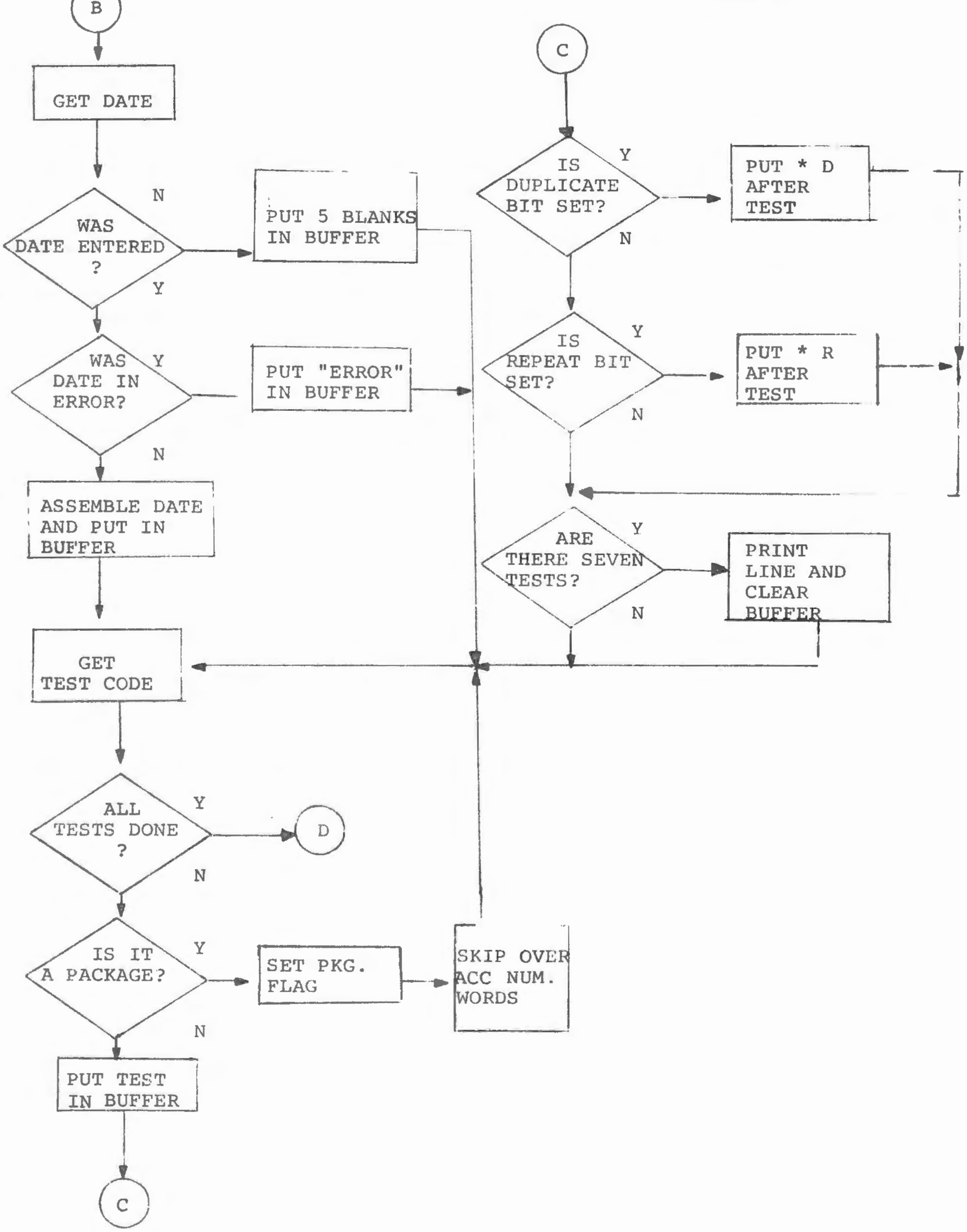
ROUTINE TO GET A LIST OF CARDS AND ERROR TYPES FOR EACH CARD.


OVERLAY RH
LIST ERRORS
20
GET CARD NUMBER
B
CHANGE TO ASCII AND PUT IN OUTPUT BUFFER
GET ERROR CODE
N
ANY ERRORS
Y
IS A PARTICULAR BIT SET 3?
Y
N
PUT IN THE CORRESPONDING LETTER FOR THIS BIT
Y
MORE BITS TO CHECK?
N
PRINT BUFFER
MORE CARDS TO PRINT?
Y
N
B
A
A
ASK IF THEY WANT EXPLANATION
Y
WAS REPLY YES?
N
PRINT EXPLANATIONS OF CODES
21
RETURN TO R7 BANK 5

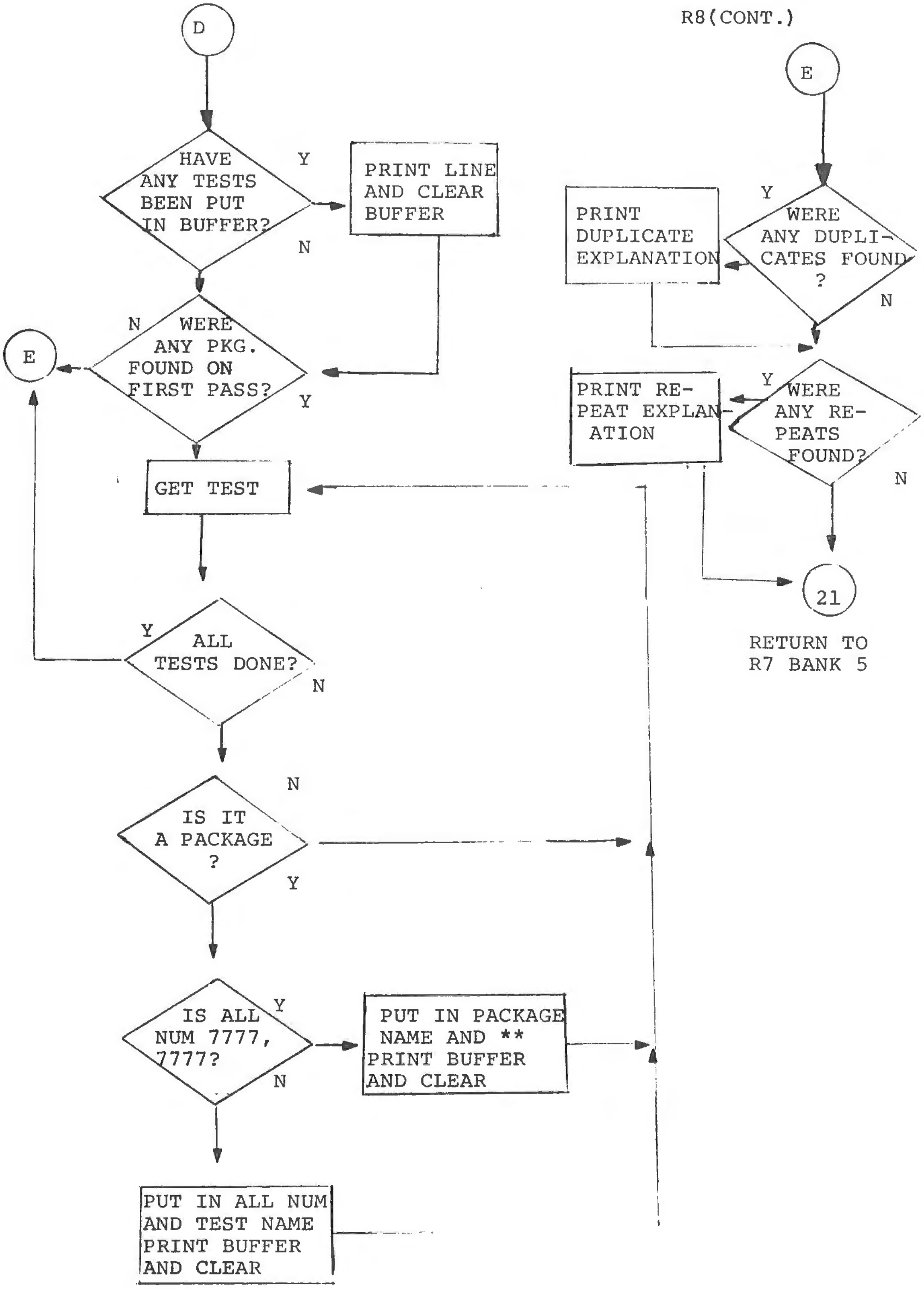


OVERLAY TO PRINT CARDS INFORMATION

R8 (CONT.)

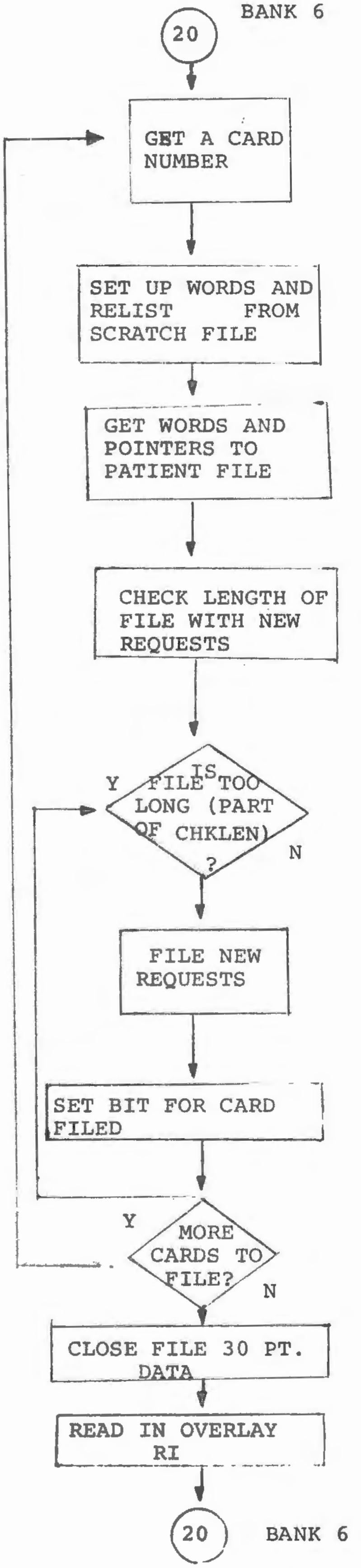


PRINT CARD ROUTINE (CONT.)


D
HAVE ANY TESTS BEEN PUT IN BUFFER?
Y
N
PRINT LINE AND CLEAR BUFFER
N
WERE ANY PKG. FOUND ON FIRST PASS?
Y
E
GET TEST
Y
ALL TESTS DONE?
N
N
IS IT A PACKAGE ?
Y
Y
IS ALL NUM 7777, 7777?
N
PUT IN PACKAGE NAME AND ** PRINT BUFFER AND CLEAR
PUT IN ALL NUM AND TEST NAME PRINT BUFFER AND CLEAR
R8(CONT.)
E
Y
WERE ANY DUPLI- CATES FOUND ?
N
PRINT DUPLICATE EXPLANATION
Y
WERE ANY RE- PEATS FOUND?
N
PRINT RE- PEAT EXPLAN- ATION
21
RETURN TO R7 BANK 5

PRINT CARD ROUTINE (CONT.)

OVERLAY R9


20
BANK 6
GET A CARD NUMBER
SET UP WORDS AND RELIST FROM SCRATCH FILE
GET WORDS AND POINTERS TO PATIENT FILE
CHECK LENGTH OF FILE WITH NEW REQUESTS
Y
IS FILE TOO LONG (PART OF CHKLEN) ?
N
FILE NEW REQUESTS
SET BIT FOR CARD FILED
Y
MORE CARDS TO FILE?
N
CLOSE FILE 30 PT. DATA
READ IN OVERLAY RI
20
BANK 6

SEE NUMGET

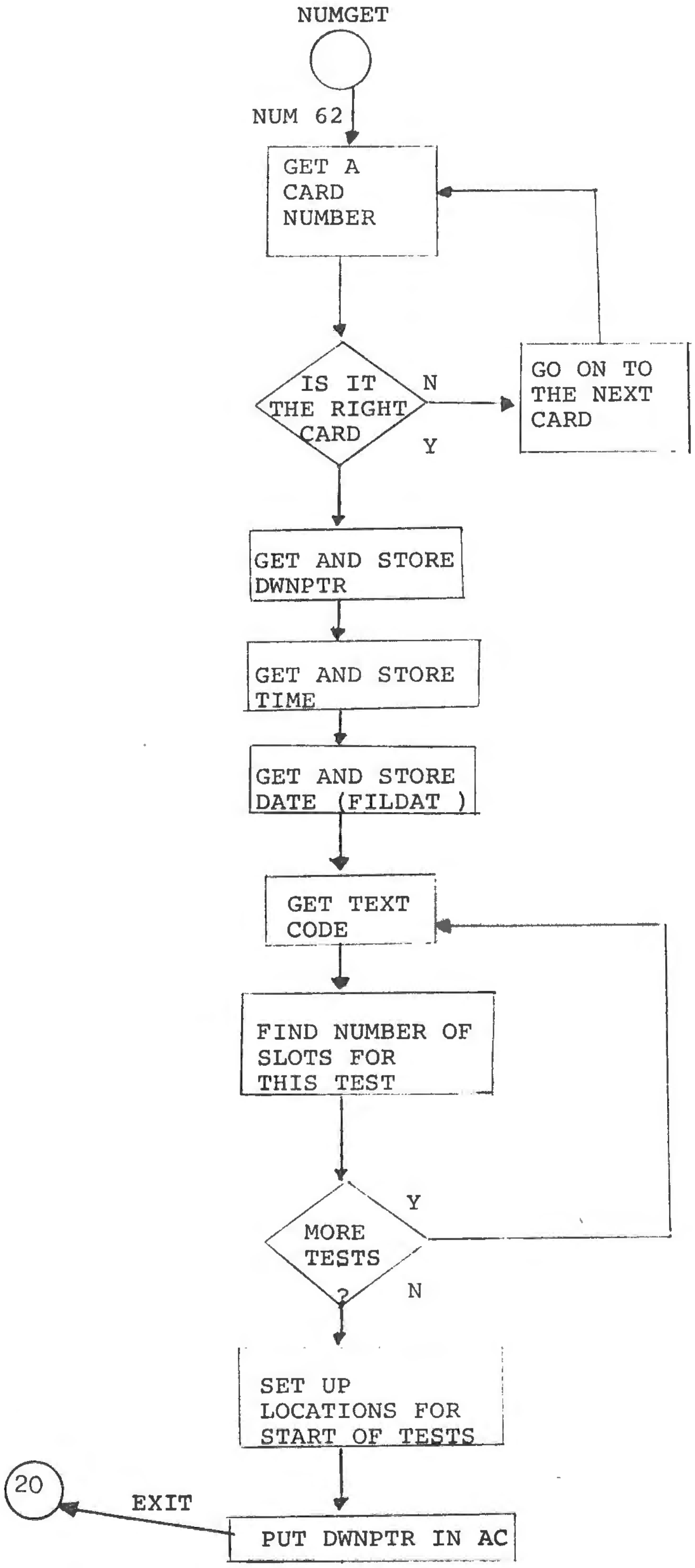
SEE PRELIM

SEE CHKLEN

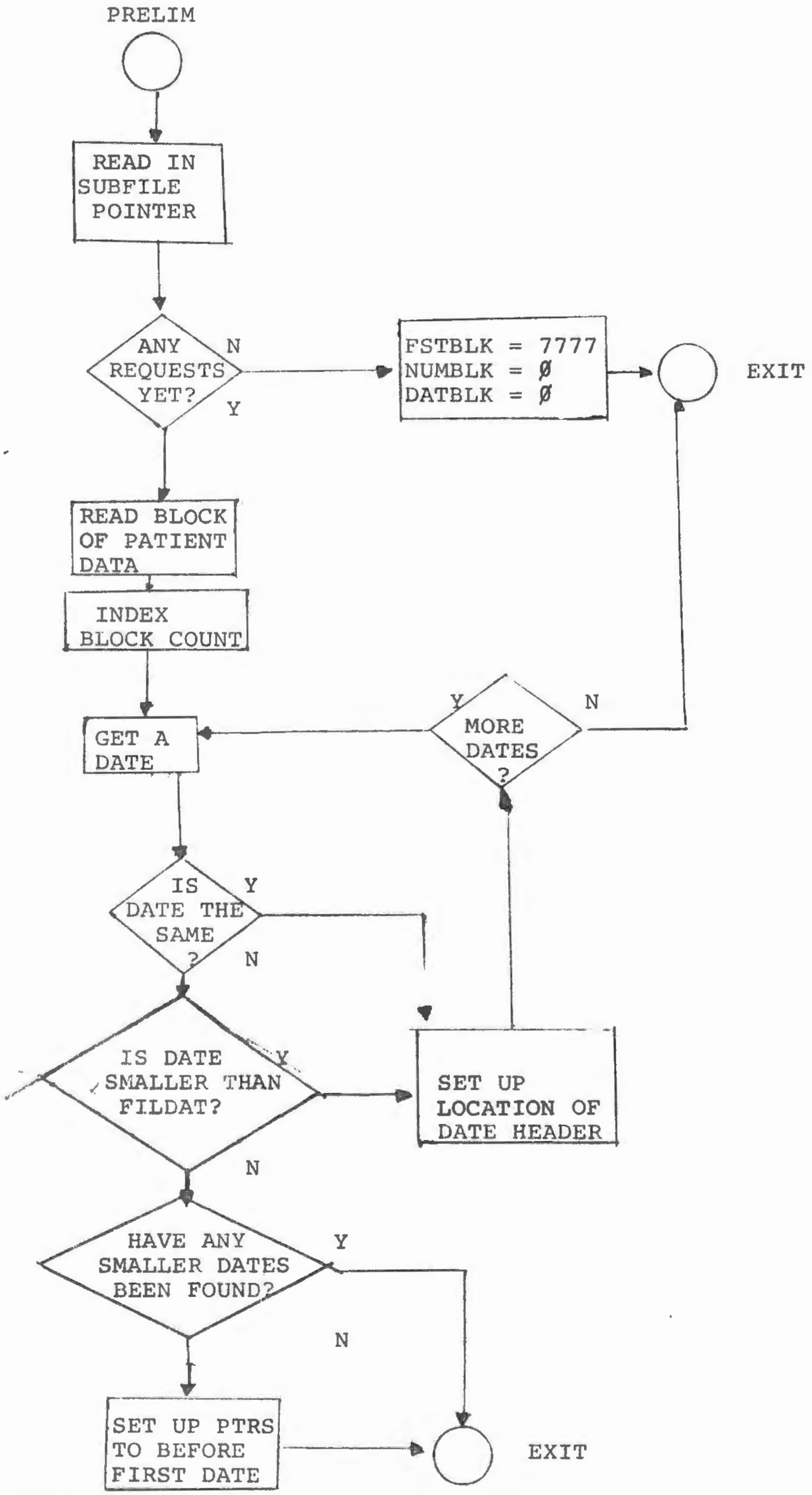
SEE PROCESS

OVERLAY TO ENTER DATA INTO PATIENT'S FILE

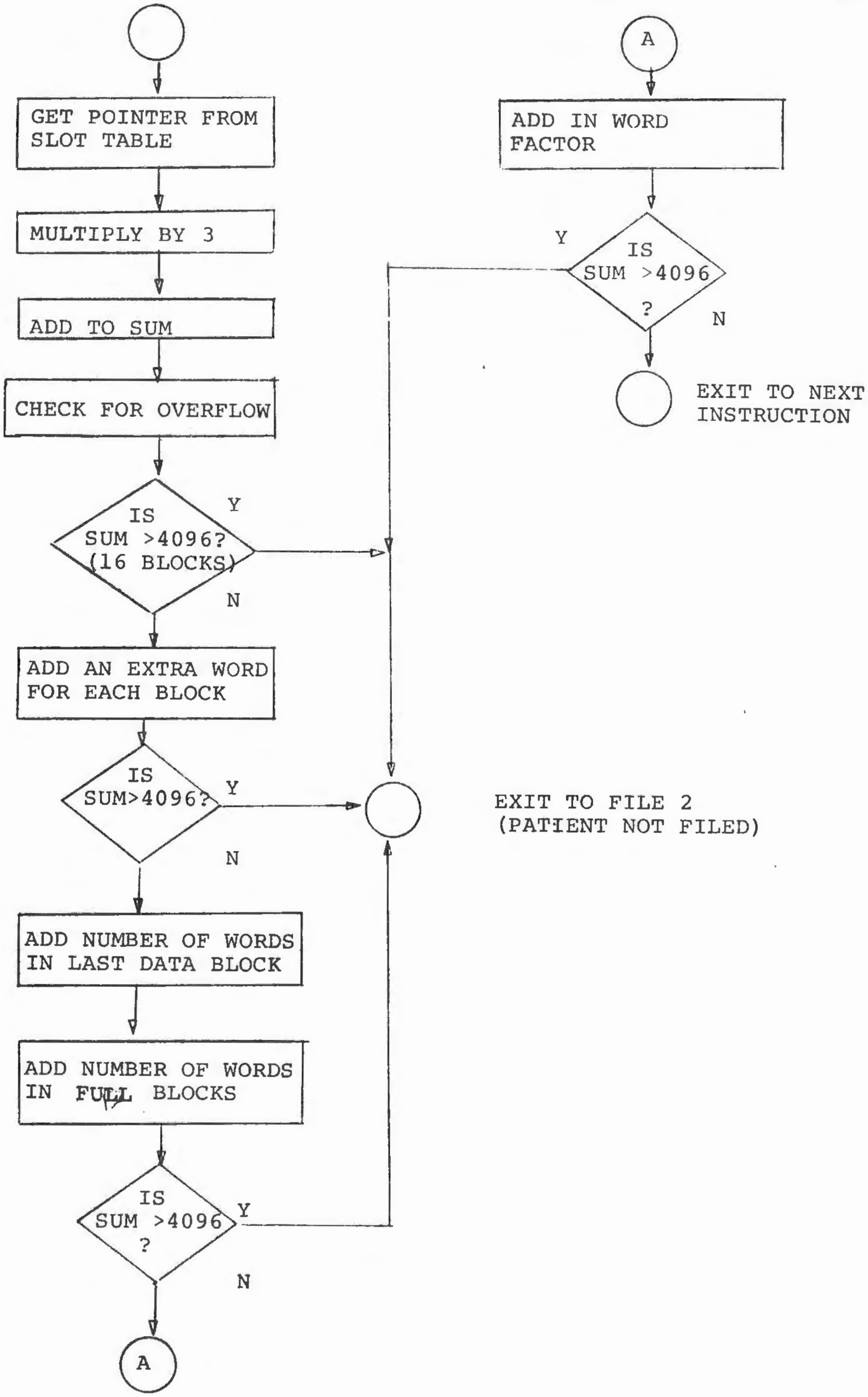
R9



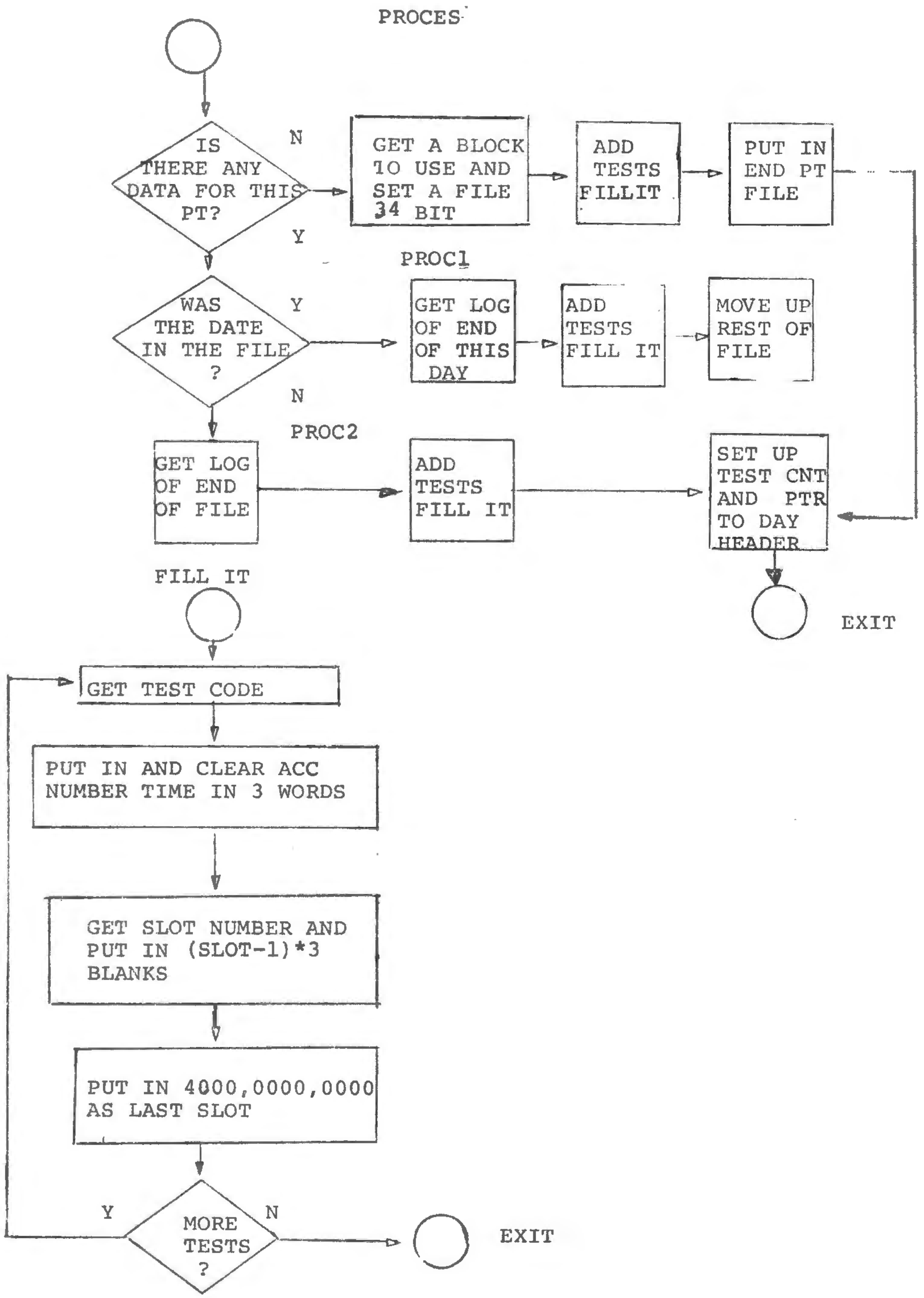
ROUTINE TO INITIALIZE PATIENT DATA FOR A CARD



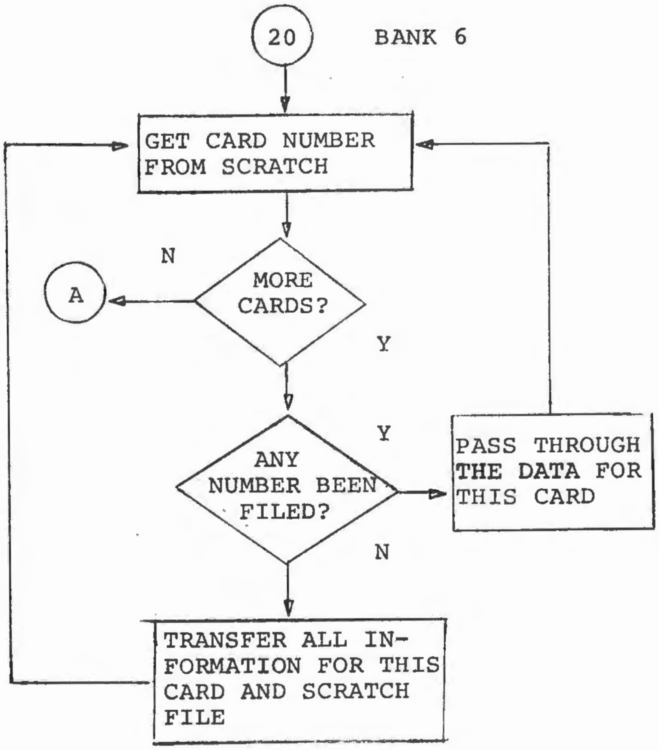
ROUTINE TO ANALYZE PATIENT'S FILE BEFORE STORING REQUESTS


GET POINTER FROM SLOT TABLE
MULTIPLY BY 3
ADD TO SUM
CHECK FOR OVERFLOW
IS SUM >4096? (16 BLOCKS)
Y
N
ADD AN EXTRA WORD FOR EACH BLOCK
IS SUM>4096?
Y
N
ADD NUMBER OF WORDS IN LAST DATA BLOCK
ADD NUMBER OF WORDS IN FULL BLOCKS
IS SUM >4096 ?
Y
N
A
A
ADD IN WORD FACTOR
Y
IS SUM >4096 ?
N
EXIT TO NEXT INSTRUCTION
EXIT TO FILE 2 (PATIENT NOT FILED)

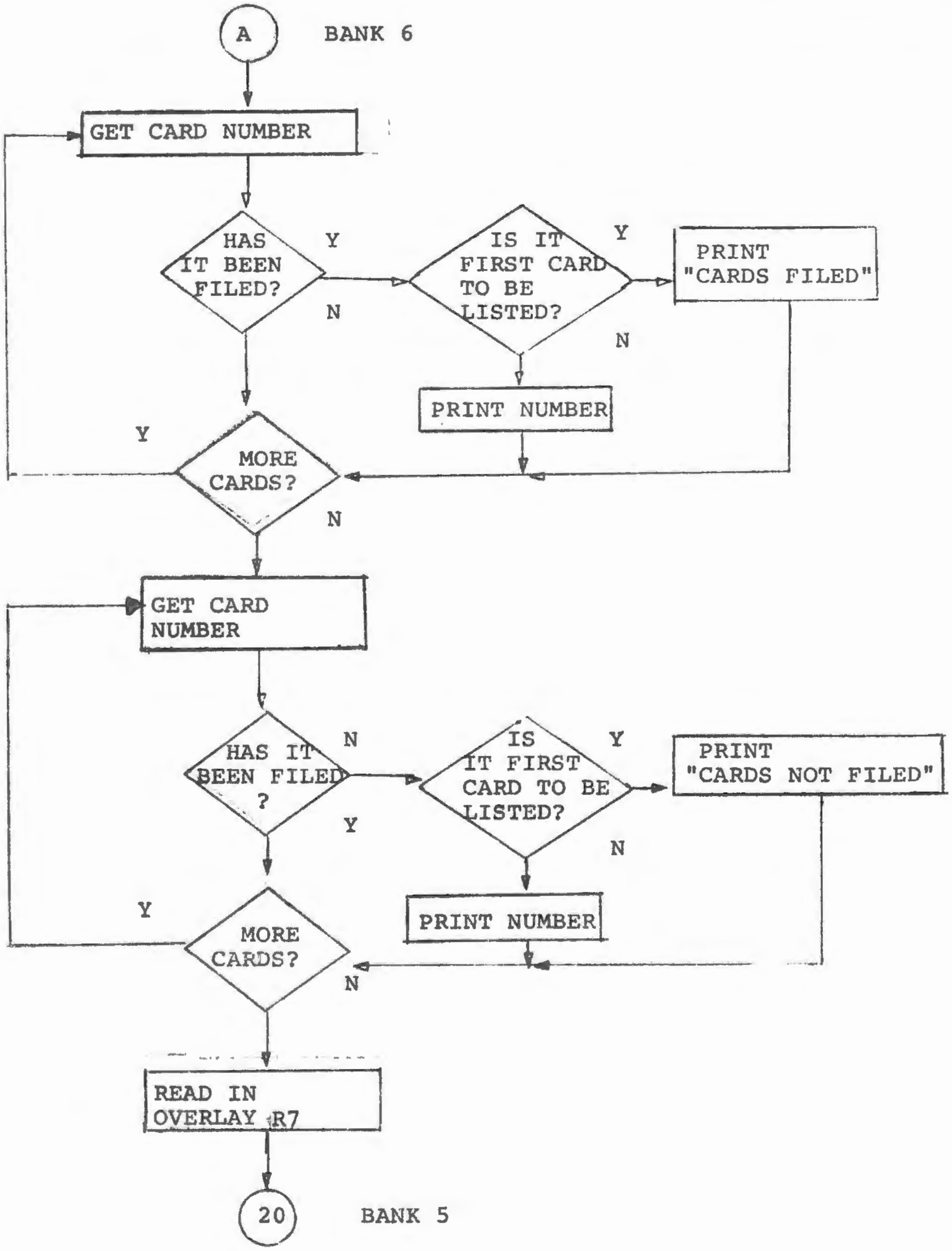
ROUTINE TO CHECK IF NEW REQUISITIONS WILL OVERFLOW
MAXIMUM ALLOTTED SPACE FOR THIS PATIENT


PROCES
IS THERE ANY DATA FOR THIS PT?
N
Y
GET A BLOCK TO USE AND SET A FILE 34 BIT
ADD TESTS FILLIT
PUT IN END PT FILE
PROC1
WAS THE DATE IN THE FILE ?
Y
N
GET LOG OF END OF THIS DAY
ADD TESTS FILL IT
MOVE UP REST OF FILE
PROC2
GET LOG OF END OF FILE
ADD TESTS FILL IT
SET UP TEST CNT AND PTR TO DAY HEADER
EXIT
FILL IT
GET TEST CODE
PUT IN AND CLEAR ACC NUMBER TIME IN 3 WORDS
GET SLOT NUMBER AND PUT IN (SLOT-1)*3 BLANKS
PUT IN 4000,0000,0000 AS LAST SLOT
MORE TESTS ?
Y
N
EXIT

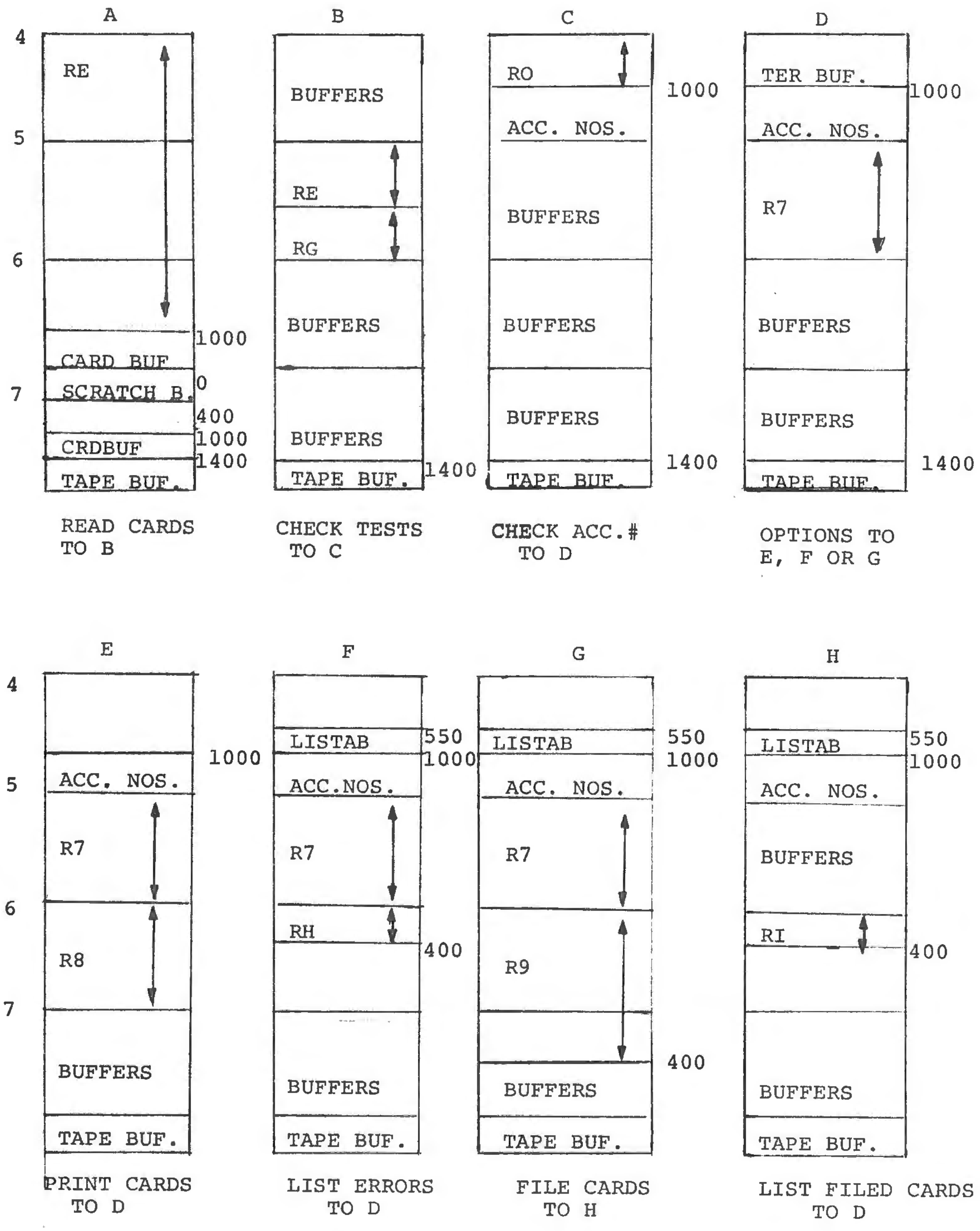
ROUTINE TO ENTER DATA IN A PATIENT'S FILE


20
BANK 6
GET CARD NUMBER FROM SCRATCH
N
A
MORE CARDS?
Y
Y
ANY NUMBER BEEN FILED?
PASS THROUGH THE DATA FOR THIS CARD
N
TRANSFER ALL IN-FORMATION FOR THIS CARD AND SCRATCH FILE

OVERLAY TO PRINT A LIST OF CARDS FILED AND NOT FILED


A
BANK 6
GET CARD NUMBER
HAS IT BEEN FILED?
Y
N
IS IT FIRST CARD TO BE LISTED?
Y
N
PRINT "CARDS FILED"
PRINT NUMBER
Y
MORE CARDS?
N
GET CARD NUMBER
HAS IT BEEN FILED ?
N
Y
IS IT FIRST CARD TO BE LISTED?
Y
N
PRINT "CARDS NOT FILED"
PRINT NUMBER
Y
MORE CARDS?
N
READ IN OVERLAY R7
20
BANK 5

CQRE MAPS FOR CARD REQUISITION ENTRY



A: READ CARDS TO B

B: CHECK TESTS TO C

C: CHECK ACC.# TO D

D: OPTIONS TO E, F OR G

E: PRINT CARDS TO D

F: LIST ERRORS TO D

G: FILE CARDS TO H

H: LIST FILED CARDS TO D

CHAPTER 4

DELETE DATA (DE)

The DELETE DATA program offers a choice of four options to delete patient data:

1. delete all test data for a specific patient,
2. remove all of the patient's test and administrative data,
3. remove all of the patient files for a specific patient type,
4. delete a test, battery, or package from a patient's file (refer to section 4.5).

Refer to the Sample DE dialogue illustration in Figure 4-1. The program operates in three phases for the first three items above:

> In Phase 1 you select the patient files whose data is to be deleted (either by individual or by type). The computer will then put all the patient files to be deleted in a "delete" status but will not actually delete them yet. If desired, you may request a list of patients' files in the "delete status" and then edit this list by removing files placed on it in error and then adding other files to be deleted.
>
> During Phase 2, any tasks, e.g., final summaries, can be run on the files in the "delete status". When Phase 2 is finished, the system automatically moves into the third phase which is the actual deletion of the patient data selected above. Once the program is in phase 3, there is no way to stop the deletion process (not even CTRL/S); however, in phases 1 and 2 you may restore the patient files by simply typing STOP↵ .

4.1 PRELIMINARY OPERATING PROCEDURE

1. Type CTRL/C to alert the system for a program call. Note that the DE program can only be run on one terminal.
2. Type DE↵ .
3. Observe the terminal for this message:

```
ENTER FUNCTION, STATUS, ACTIVITY (,DATE-MONTH/DAY)
ENTER CODE*
```

```
DE
ENTER FUNCTION, STATUS, ACTIVITY (,DATE-MONTH/DAY)

ENTER CODES * P,C,A,12/1
*DELETE MODE*

 * T,IN

 * P,724887

 *
*EDIT MODE*  E L OR C * L
OUTPUT DEVICE (1-TTY, 2-LPT, 3-SCOPE)
TYPE 1, 2 OR 3 * 1

           DELETION LIST  7/12/1972  PAGE 1

PAT. NAME              TYPE WARD ROOM  DR.       PAT. NUMBER

ZIMMERMAN HAROLD         IN 4W    123   RDF       721345

HONGISTO FRED            OP MEM                   724887

E L OR C * E
PATIENT # :  721345
PATIENT # :
E L OR C *
*DELETE MODE*

 * P,7213Ø2

 *
*EDIT MODE*  E L OR C * L
OUTPUT DEVICE (1-TTY, 2-LPT, 3-SCOPE)
TYPE 1, 2 OR 3 * 1

           DELETION LIST  7/12/1972  PAGE 1

PAT. NAME              TYPE WARD ROOM  DR.      PAT. NUMBER

HONGISTO FRED            OP MEM                 724887

JOHNSON MARY             SS MAT   1Ø    KLP     7213Ø2

E L OR C * C
BEGINNING DELETION
TTY IS FREE
```

Figure 4-1 Sample DE Dialogue

4. Type SHOW ME↲ to ask the computer to explain the optionals available. The following message is printed:

```
"FUNCTION"
P-DELETE PT.
D-DELETE ALL TEST DATA FOR A PT.
T-DELETE A SINGLE TEST/BATTERY/PKG.
"STATUS"
C-DELETE ONLY IF ALL TESTS COMPLETE
I-IGNORE STATUS
"ACTIVITY"
A-DELETE ONLY IF NO ACTIVITY SINCE LAST SUMMARY
I-IGNORE ACTIVITY
"DATE" (OPTIONAL)
DELETE ONLY IF NO REQUISITIONS SINCE SPECIFIED DATA
ENTER CODE *
```

5. Choose the "FUNCTION" option:

"P" if you wish to delete entirely all data for an individual patient or all data for patients of a certain type, e.g., all outpatients. That is to say, this option removes all the administrative and test data so that the selected patient data will no longer exist on the lab files.

"D" if you wish to delete just the test result data from the files for a particular patient or for all patients of a certain type. This option leaves the patient administrative data intact so that you may add new test results.

NOTE

P and D functions cannot be "mixed". You must finish one function first and then run the program again for the second function.

"T" if you wish to delete only a single test, battery, or package, e.g., if a test was accidentally requested in error. Note that this option does not require a STATUS or ACTIVITY as do options P and D above. See section 4.5 for further instructions.

6. Choose a "STATUS" option for only the P and D functions above:

"C" status will delete data only for those patients selected whose tests are all complete.

"I" status will delete data for all the selected patients, whether or not the tests are complete.

7. Choose an "ACTIVITY" for options P and D above:

   "A" activity will delete the patient data only if there has been no activity (requisitions) since the last SUMMARY.

   "I" activity allows the data to be deleted whether or not new activity has occurred since the last SUMMARY.

8. If you wish to delete those files which have had no new requisitions since a certain date, enter that date. The date may be any month and day in the current year. Note that if a new year has just begun, the computer will accept a December date from the previous year but not a November or ealier date. The date may also appear as a day/month if this format was chosen during installation. Type RETURN to omit the date.

9. Type the choices made in Steps 5-8 above, using the format:

   ```
   FUNCTION, STATUS, ACTIVITY (,DATE-MONTH/DAY)↵
   ```

   Example: If on December 16 you wish to delete all the data for out-patients whose doctor has not requested any new tests since December 1, whether or not all their tests are complete, type the following line:

   ```
   P,I,I,1/1↵
   ```

4.2 PHASE 1

4.2.1 DELETE Mode Function P or D

1. If function P or D was chosen above, the computer prints this message:

   ```
   *DELETE MODE*
   *
   ```

   and waits for your reply.

2. Type SHOW ME↵ to obtain the following list of valid replies for the P option:

   ```
   P,PT. #-DELETE SPECIFIED PT.
   T,TYPE-DELETE PTS. OF SPECIFIED TYPE
   *
   ```

   or this message for the D option:

   ```
   P,PT. #-DELETE DATA FOR SPECIFIED PT.
   T,TYPE-DELETE DATA FOR PTS. OF SPECIFIED TYPE.
   *
   ```

3. a. Type in P, and the patient number↵ if you wish to delete the data for a specific individual
   b. and go on to Step 4; or, type T and the 2-character patient type code↵ if you want to delete the data for all the patients of that type; e.g., T, OP↵ . Proceed to Step 5.

   You may also press the RETURN key which causes the program to enter EDIT MODE (see Section 4.2.2).

4. The computer responds by checking that the patient number you specified meets the criteria of STATUS and ACTIVITY. If the patient specified does not fulfill the criteria, the following message is typed:

```
PT. DOES NOT MEET SPECIFIED CRITERIA.
```

   Or, if the patient does meet the STATUS and ACTIVITY criteria specified, the computer responds with an asterisk indicating that the patient's files have been flagged for deletion and that you should continue on with the next patient number to be deleted.

5. If T,TYPE was chosen in Step 3, there will be no response except for an asterisk, which means that the data for patients of that type is flagged for deletion. You may now continue entering type codes for deletion.

6. When you have finished Steps 4 and 5, press RETURN to enter EDIT mode.

NOTE

> If you accidentally try to delete a patient file both individually and then by type, no harm is done. That patient file will not be printed twice on the delete list. No error message is printed.

4.2.2 EDIT Mode

1. In response to the RETURN above, the computer prints the following:

```
*EDIT MODE*
E L OR C *
```

2. Type SHOW ME↵ to ask the computer for an explanation. The following message is printed:

```
E-EDIT:  REMOVE PATIENT FROM DEL. LIST
L-LIST ALL PTS. IN DELETE STATUS
C-CONTINUE
```

3. Type L↲ to generate a list of patient files on the delete status. The delete listing will be in the same chronological order as you requested deletion.

4. The computer reacts by printing out this message:

```
OUTPUT DEVICE (1-TTY,2-LPT,3-SCOPE)
TYPE 1,2 OR 3 *
```

5. Choose the device desired and press RETURN. If the line printer is chosen and is in use, the computer will print: LPT IN USE. In this case, the message E L OR C * is again displayed. You may again select L and then choose another device.

6. A listing is now printed on the device specified. The first two printed lines have the following format:

```
DELETION LIST  mm/dd/yyyy   PAGE n
PAT. NAME      TYPE  WARD   ROOM  DR.  PAT. NUMBER
```

7. When the listing is done, the message E L OR C* is again displayed. You may choose the E option now by typing E .

8. The computer will respond with an asterisk indicating that you may type in any patient number and a RETURN to remove that patient number from the delete status list.

9. When you are finished removing any patient numbers which should not be deleted at this time, you may wish to type RETURN to return to the E L OR C * message and then generate another list to double-check the numbers to be deleted. If you wish to delete more patient data, type E in response to the E L OR C * message to put the computer again in "delete" mode and identify more patients to be put in "Delete Status". You may alternate between EDIT Mode and DELETE Mode as many times as necessary.

10. When the list of deletes is completely satisfactory, type C↲ to continue to Phase 2.

4.3 PHASE 2

During this phase, you may run any program (such as final summary) necessary on the patient files to be deleted.

1. The computer reacts to step 10, section 4.2.2, by printing an asterisk. If you wish to go directly into the deletion process (Phase 3), type NONE↲ . However, if you wish to request final programs on the "delete status" files, type the 1- to 4-character alphanumeric program code name (defined at installation) and RETURN. This code name instructs the system to automatically run the list of final programs which your lab has selected. There may be more than one code if your lab handles different type of discharges.

2. When the programs have finished running, the system proceeds automatically into Phase 3.

4.4 PHASE 3 (DELETION)

1. When phase 3 starts, the system notifies you of that fact by printing:

```
BEGINNING DELETION
```

2. There is a fairly lengthy pause while the system completes the deletion process. When the process is complete, the program prints TTY IS FREE and exits.

4.5 FUNCTION T OPERATING INSTRUCTIONS

1. If function T was typed in Step 9 of the preliminary operation (Section 4.1), the terminal will print the following question:

```
PAT. #:
```

2. Enter the patient number associated with the test which you plan to delete and press RETURN.

3. The terminal then prints the question:

```
TEST NAME*
```

4. Type the test, battery, or package code you wish to delete and press RETURN.

5. The following message is printed:

```
REQ. DATE*
```

6. Type the requisition date in the fomat mm/dd or dd/mm (depending on which one was selected during installation in your lab) and press RETURN.

7. If a test or battery name was used in step 4, above, the computer will ask for the accession number by printing the message:

```
ACC. NUM*
```

8. Type in the accession number for the test or battery to be deleted and press RETURN.

9. The system proceeds to check the information; and, if it finds no errors, deletes the test and prints:

```
DELETED
```

If there is more than one occurrence of the same test with the same accession number on the same day on file, the following message is printed:

```
x OCCURRENCES ON FILE (x = number found)
RUN SUMMARY FOR THIS PT. ON THIS DAY
CHOOSE OCCURRENCE BY ORDER ON SUMMARY
```

This means that when a summary is run for the patient's file, the duplicate tests will be printed out in a certain order. If you wish to delete the second occurrence of a duplicate test, for example, use the number 2. You may need to run a ward report also for this patient on that day if you are not certain which occurrence to delete because the patient has so many occurrences of that test.

10. Press RETURN and then continue adding tests to be deleted.

4.6 ERROR MESSAGES

MESSAGE	MEANING	ACTION
DELETE RUNNING AT ANOTHER TERMINAL	DE may be run only on one terminal at a time.	Consolidate the processing.
FILE 3 OVERFLOW	Too many patient files in delete status.	Notify your supervisor.
INVALID INPUT	Some invalid format has been used.	Try again using the correct format.
NOT FOUND	Test name or accession number is not in the file.	If the name or number was typed incorrectly, try again.
PAT. NOT FOUND	This patient number was not in the files.	If you mistyped the number, try again.
PLD ERROR	The programs needed cannot be found on the disk.	Notify your supervisor.
WAITING FOR LPT	Line printer is malfunctioning or turned off.	Check the line printer for the problem.

4.7 INTERNAL DESCRIPTION

When started, DE performs an initialization procedure to determine the size of file 30 and file 26. The program reads the first 5 words of file 21 to establish constants for buffer sizes corresponding to the size of the patient number. The code is located in the disk buffer and is destroyed when the program starts. The program builds a mask of 77 or 00 bytes to mask out the reference field of a patient number when DE is concerned only with the patient number reference field. This mask code is located in the Teletype buffer and is destroyed when DE starts.

4.7.1 P and D Options

The code at PASS1 is executed to build a dayheader with todays date and control is transferred to PASS1M. File 3 is opened for write and left open for the rest of the program. File 3 is "cleared" by writing 7777 in the first few words. The user is then asked to pick the criteria for deletion and the delete function at PASS1A. If the T option is chosen, DT is loaded and control transfers to PASS1L/

When called, DE will always take the PASS1 path. When DE is reloaded by some other program (e.g., after the user has run programs using a file 3 built by DE) and the necessary flags have been restored, DE will find STAGE2 flag set and proceed directly to PASS2. PASS2 is a subroutine that reads successive words from file 3 and uses them as file 26 positions just as if the user had directly entered a file 26 position instead of a patient number. The deletion is actually done.

The code at PASS1L asks for individual patients or types of patients to delete. Each time a patient file is entered, control passes to DEPAT and then back when done. Each time a type is entered, control passes to BATCH and back when done. The code in PASS1L allows the user to build a deletion list. Typing carriage return passes control to PASS1F. This allows the user to get a hardcopy listing of the deletion list and to edit it (remove patient files from it). Typing carriage return again transfers control back to PASS1L, to allow the user to enter more patient files for the deletion list. Transfers back and forth between PASS1L and PASS1F can occur as many times as is necessary.

When the user finally types a C as input to PASS1F, the STAGE 2 flag is set and a PLD loads a chaining program which will generate reports for the records in file 3. An address of a list of items to be saved and restored when DE is reloaded is passed to the chaining program.

The edit routine (to remove patient data in file 3) uses the DELSRT subroutine. It can do this because in the startup of PASS1 changes are made to DELSRT to make it look at file 3 instead of file 2. The BATCH subroutine does just the same thing that DEPAT does, except that it does it repeatedly on file 25 instead of asking for a single patient number.

The code for generating listings of the delete status file consists of three sections:

1) The INQURY subroutine that checks a list of addresses of GET subroutines to get the administrative data needed and then checks a list of addresses of buffers to put together to make a text line to send to a logical output device for successive file 3 entries.
2) GET subroutines that are copied directly from the AD program.
3) A set of subroutines that allow the program to print information through a logical output device.

Once the user has identified a patient file or type to delete, DESUB determines if the patient file should be deleted and completes the actual deletion process.

4.7.1.1 DESUB

This routine examines the file 30 data of a given patient to find out if the patient file should be deleted. It uses the criteria that have been established from the preliminary dialogue to decide whether this patient is a candidate for deletion. While examining the file, DESUB builds a list of the file 30 blocks in use and another list of accession numbers that were assigned to tests still incomplete. Only the first 128 accession numbers are kept. Any more are ignored, and would not be cleared in file 27 later. Only the first 19 blocks of a file will be examined and saved. The 20th block is treated as a logical end of file. When the end of a patient's data file occurs, control passes to Q0BATCH if, according to the deletion criteria established in the preliminary dialogue, the file is to be deleted. If the program is in PASS1 (building a delete status file) the subroutine WRITF3 will be called. The new file 26 position will be appended to the end of file 3. If the program is in PASS2, the patient file will now actually be deleted. Note that DESUB has to go through just as much work in PASS1 or PASS2, building the same kinds of core lists and rechecking this patient's criteria.

In PASS2, accession numbers in the list are cleared one at a time from file 27. If the patient had some file 30 data, all of file 34 is read into core. The bits are cleared for blocks in the block list, and the whole file is written back out. This is all that is necessary to satisfy the "D" option. The subroutine now checks to see if the "D" option was chosen. If so, it returns. If not, DELSRT is called to delete the patient file from the sort files. Then DELAD is called to delete patients' administrative data. The DELAD subroutine is simple and generalized so that if non-standard pieces of data are added for a particular site, it is merely necessary to add four words of data to a parameter list to delete the information in that file. The process described in this routine is then sufficient to handle the "P" and "D" functions. The calling subroutines for the different options do all of the preparatory work to get a file 26 position of a patient file to delete. DESUB does the work of deciding whether the file ought to be deleted and then may or may not do the actual work of deleting it. If the criteria imply the file is not to be deleted, control is transferred to Q2BATCH. If the program is in PASS1 and not in batch mode, the user will be notified that this patient file will not be deleted.

4.7.1.2 BMP30

This subroutine allows calling subroutines to treat the patient's file 30 data as if it were one contiguous file. If called with AC equal to 0, BMP30 initializes itself and reads the first block from the file. If called with AC not 0, this number is used to move through the file. The blocks found are stored in a block list in case the calling routine needs to use them later. Before the next block in a string is read, the W30FLG flag on page 0 is checked to see if the current block should be written back out first. If an invalid forward pointer, or too many blocks, or a 7777 occurs, there is an indirect jump through the end of file transfer address END30 on page 0. Otherwise, when the routine returns, the AC contains the value of the word to which it was specified to move. It is not necessary to save this word since the buffer in core still contains the block that it was found in, and APOS30 on page 0 contains the absolute address of this word in the buffer. Note that the P and D options always use the same end of file jump address, but that this is modified several times on each pass with the T option.

4.7.2 Single Test Deletion Option of DE

The code and text for this option comprise 4 blocks and are in overlay DT. A good deal of this code exists to take into consideration the possibilities that multiple occurrences of the same day header can occur in file 30 and that multiple occurrences of tests with the same accession number can occur in each of these day headers.

For each new item to delete, it is necessary to enter the patient number, the test name, the day on which it was requisitioned, and if the test is not a package, the accession number that was assigned to it. Two lists of interest are kept. One list contains up to eight occurrences of dayheaders (the same value) of the one specified. More than eight is treated as a logical end of file. The entries in the list contain three words. The first word is the "sequence number" of the occurrence of a candidate for deletion. The next two words contain the file 30 block and word in which the dayheader occurred. The other list contains two word entries containing the block and word in which the test or package occurred. The nth entry in this list is "sequence number" n. After these lists are built, they are examined to see how many entries occurred. Normally, there will only be one dayheader and one entry. If there are no entries, the program types "NOT FOUND" and prepares for the next input. If there is more than one entry, it is up to the user to now specify which occurrence to delete. The program types a message telling the user to run a summary for this patient on this day, and to choose the sequence number of the proper entry to delete.

A summary should reflect the order within the actual data file that multiple occurrences of a given test under disjoint occurrences of the same dayheader are present. The user must determine under which accession numbers the occurrences were entered. It may take some time to determine which occurrence to pick, so if the user does not want to make a choice, typing a carriage return will escape back to the beginning of the "T" option. Once the proper sequence number has been typed, the entry will be deleted, picking the proper dayheader and entry from the two lists.

The program now reexamines the entry to be deleted, If the entry to be deleted is a test or battery, it is examined to see how many incomplete results are present. This number will be used to adjust the test count later. The accession number then becomes the only entry of an accession number list. If the entry is a package, this process must be repeated for every test and battery in the package. Up to 128 entries can be put into the accession number list. The test count is then adjusted. Then the file 30 entry is changed to reflect its new delete status. After this, the data in the dayheader is reexamined for incomplete results. If a test with an incomplete result is found, the program checks for its accession number in the accession number list and removes it from the list. When this process is done, the remaining accession numbers in the list will be deleted from file 27. The program then returns to the beginning of the "T" option.

4.7.3 Utility Routines

There are a number of utility subroutines at the beginning of the programs. The code developed by source modules AD+DE.1 and AD+DE.2 is common to both programs. Generally these routines have their own unique constants associated with them, even though there may be some duplication of values, program modularity is more important. The routines for terminal I/O use registers R5, R6, and R7 destructively and do not require their contents to be maintained between calls of the routines.

TYPIN

This subroutine performs some initial processing of a terminal buffer already input. The 48 word input buffer is unpacked into a 72 word buffer. A carriage return is put in the 73rd word. The left most 4 bits of all 73 words are cleared. If the first word of the buffer is a carriage return, CRFLAG flag is set. If not, leading blanks are eliminated. If the first non-blank character is a carriage return, both CRFLAG and BLANKF flags are set. If a non-blank character is found before carriage return, the next five characters are checked for "STOP" followed by a carriage return. If found, the program exits. If not, control is returned to calling program and R7 points to the word before the first word containing non-blank input.

TYPER1

Word following call contains address of a packed ASCII string. UNPACK is called to unpack it, and a carriage return is put at end of the buffer. TYPMSG is called to type it out, with AC equals two for TMX.

TYPER2

Works exactly like TYPER1 except that an altmode instead of carriage return is put at end of buffer and AC equals three for TMX.

TYPER3

Works like TYPER1 except that nothing is put at end of buffer and control returns immediately without typing anything out.

LFEED

Appends a carriage return to the output buffer and calls TYPMSG to type out buffer, with AC equal to two for TMX.

UNPACK

AC contains address of packed text string. Text is unpacked and appended to TTY buffer and followed by a blank. Unless TYPER3 has been called since the last time TYPER1 or TYPER2 was called, this will amount to unpacking the text into the beginning of the TTY buffer.

PACKBF

This subroutine packs a string of 8-bit ASCII characters into a buffer of 6-bit characters. Upon entry, the AC contains the address of a core buffer whose first word is the two's complement negative of the number of words in the buffer. Register R7 (when incremented) points to the address of the string of text to be packed. The characters will be stripped and packed until either 1) a carriage return is encountered or 2) the destination buffer is full, whichever comes first. If carriage return occurs first, the buffer will be filled to the end with blank bytes (octal 40).

BINSUB

This subroutine converts a binary double precision number into a packed 6-bit ASCII text buffer. The calling JMS+1 contains the address of the first word of the two word source buffer; calling JMS+2 contains the destination buffer address, the first word of which is the negative of the maximum size of the buffer in words.

TYPMSG

Contents of the AC is used for the TMX after the TTW. KRUNCH is called first to pack the 72-word buffer into a 48-word buffer in monitor format.

SUDEV

Asks user to specify an output device. Linecount constnat and logical device indicator (OUTDEV) are established. Page count is set to 1.

ODOPEN

The device indicated by OUTDEV is opened. If open is unsuccessful, control returns at call+1, else it returns at call+2. A form feed is done on the logical output device.

ODCLOS

The device indicated by OUTDEV is closed.

DVOUT

Whatever is in the output buffer is sent out via the device specified by OUTDEV. If the output buffer is empty, this will result in a linefeed on that device. The line count is incremented and compared to the report length. If they are not equal, control returns at call+1.
If they are equal, a form feed is done on the logical output device, and control returns to call+2.

ODFD

A logical formfeed is done on the device specified by OUTDEV. The current line count is reset to 1.

DMULT

The contents of LHMUL and RHMUL considered a double precision integer are to be multiplied by the contents of the AC, the results being put in LHRES and RHRES by successive additions.

CMDPTR

This subroutine calculates absolute addresses in the sort file using the formula (HOSIZE + 1) *DIRNUM.DIRNUM is the current directory being worked on and DSPTR is the relative address with that directory.

OFILE

This subroutine opens a list of files. The word following calling JMS contains the address of the list of files to open. The next word contains the trap command, either DKR or DKW. If all the files in the list cannot be opened, it closes all files that it has opened so far, does an unconditional TMX, and tries again to open the list.

CFILE

This subroutine closes all files currently open.

DECSUB

This is a double precision decimal to binary conversion subroutine. On entry, AC equals 0 and contents of R7, when incremented, point to string of ASCII digits followed by a carriage return. Calling JMS + 1 contains address of first word of a two word buffer. If there are seven or less digits followed by a carriage return, conversion is made and control is returned to calling location +3. Otherwise, return is made to calling location 2.

DISK1

This subroutine allows disk access in block mode. The three parameters following the calling JMS are:

1) 2000 plus 2 times the file to be accessed,
2) the block to access,
3) the quarter and number of blocks to access.

DISK2

This subroutine allows disk access in word mode. The three parameters following the calling JMS are:

1) 3001 plus 2 times the file number,
2) the core buffer address to use,
3) the size of the transfer minus 1.

The disk address within the specified file will always be taken from the double precision buffer on page 0 (LHRES and RHRES).

Both disk routines use the switch WRSWCH to determine whether reading or writing is to be done. If monitor returns at the error return, the program will exit immediately.

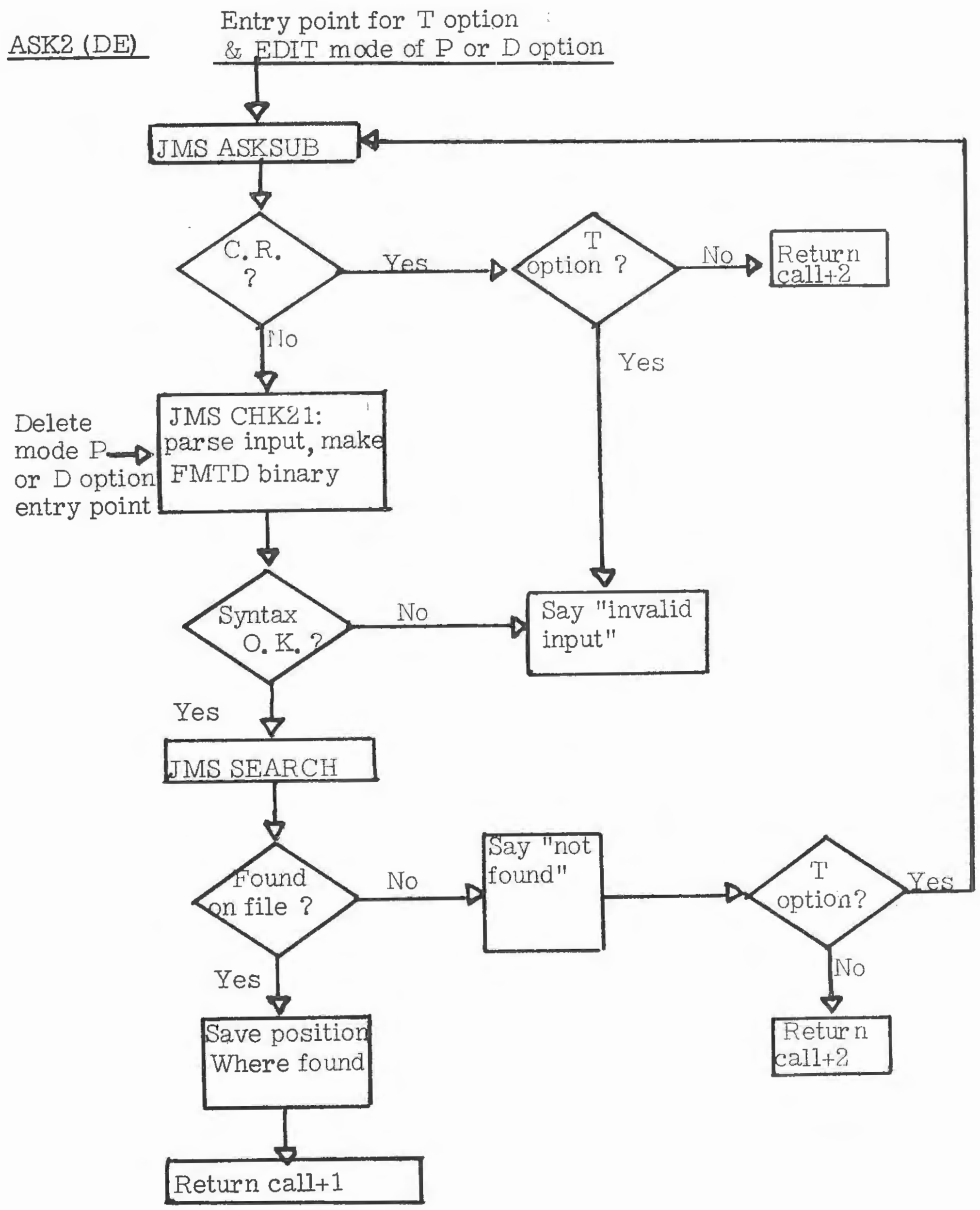
LPOUT

The output buffer is packed into the lineprinter buffer. The constant at the beginning of the buffer (which is necessary for the PACK subroutine) is converted temporarily to the form that monitor expects. LPSEND is called to actually do the trap. Control will not return to LPOUT until the line is sent.

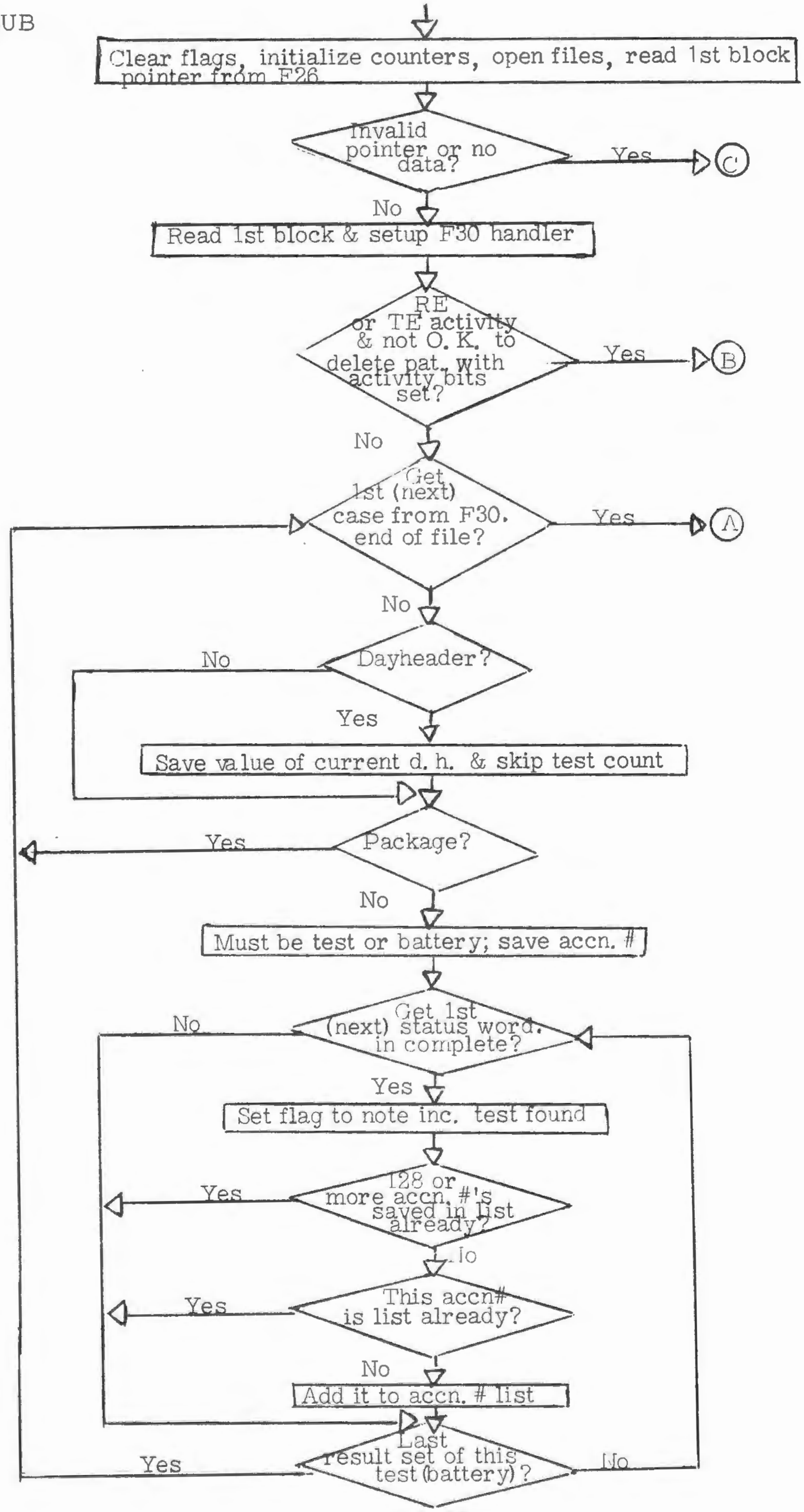
SEARCH

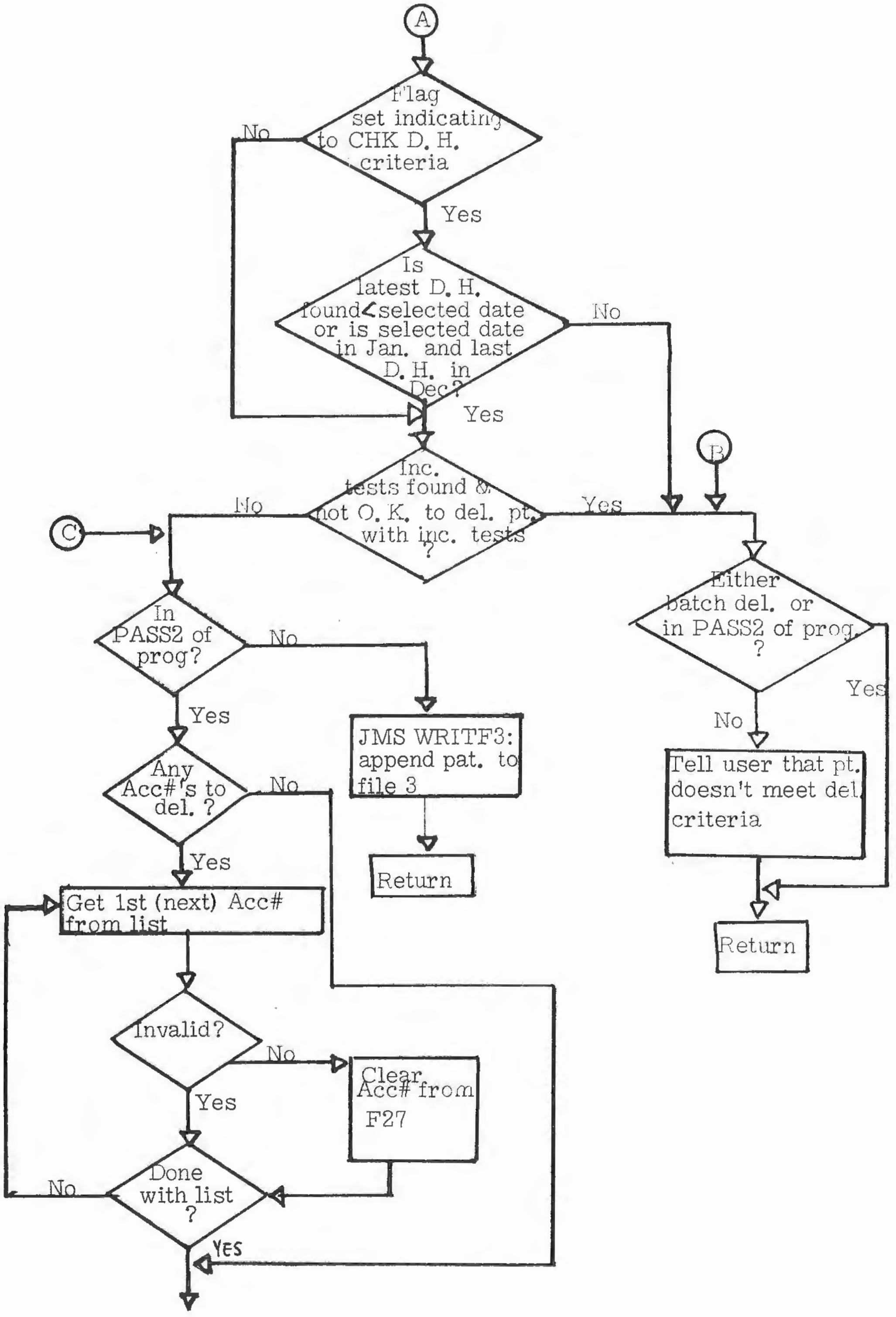
A specified file is searched for a specified bit pattern in each successive logical record. (The logical record length is also specifies). If such an entry is found, control returns a call+1 and the relative position of the record is contained in F26POS. If not, control returns at call+2.

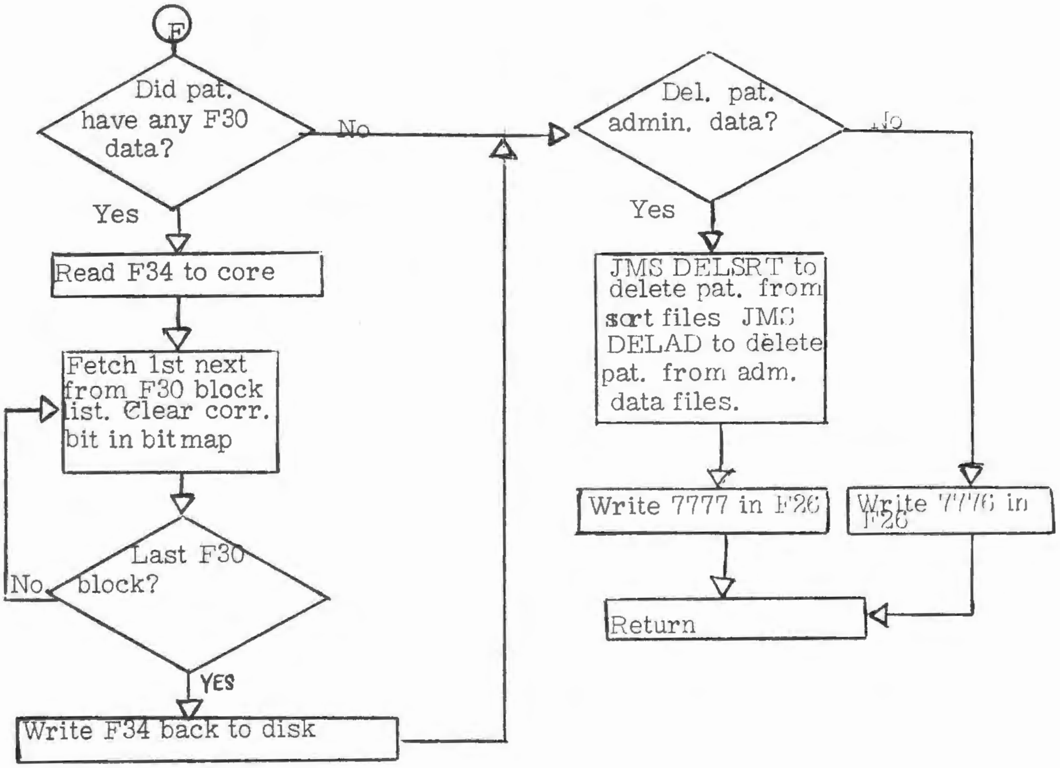
4.8 DELETE DATA Flow Charts

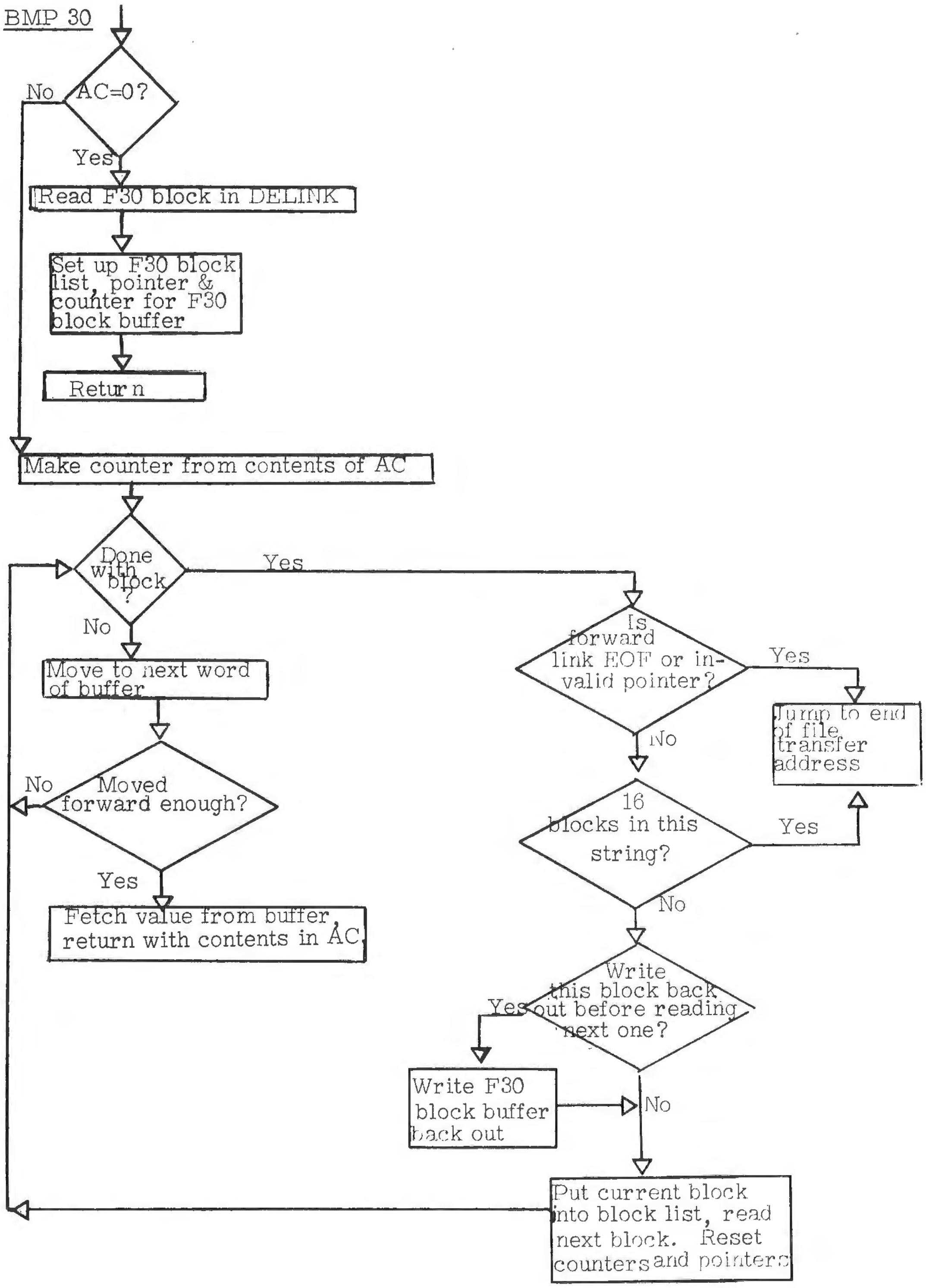


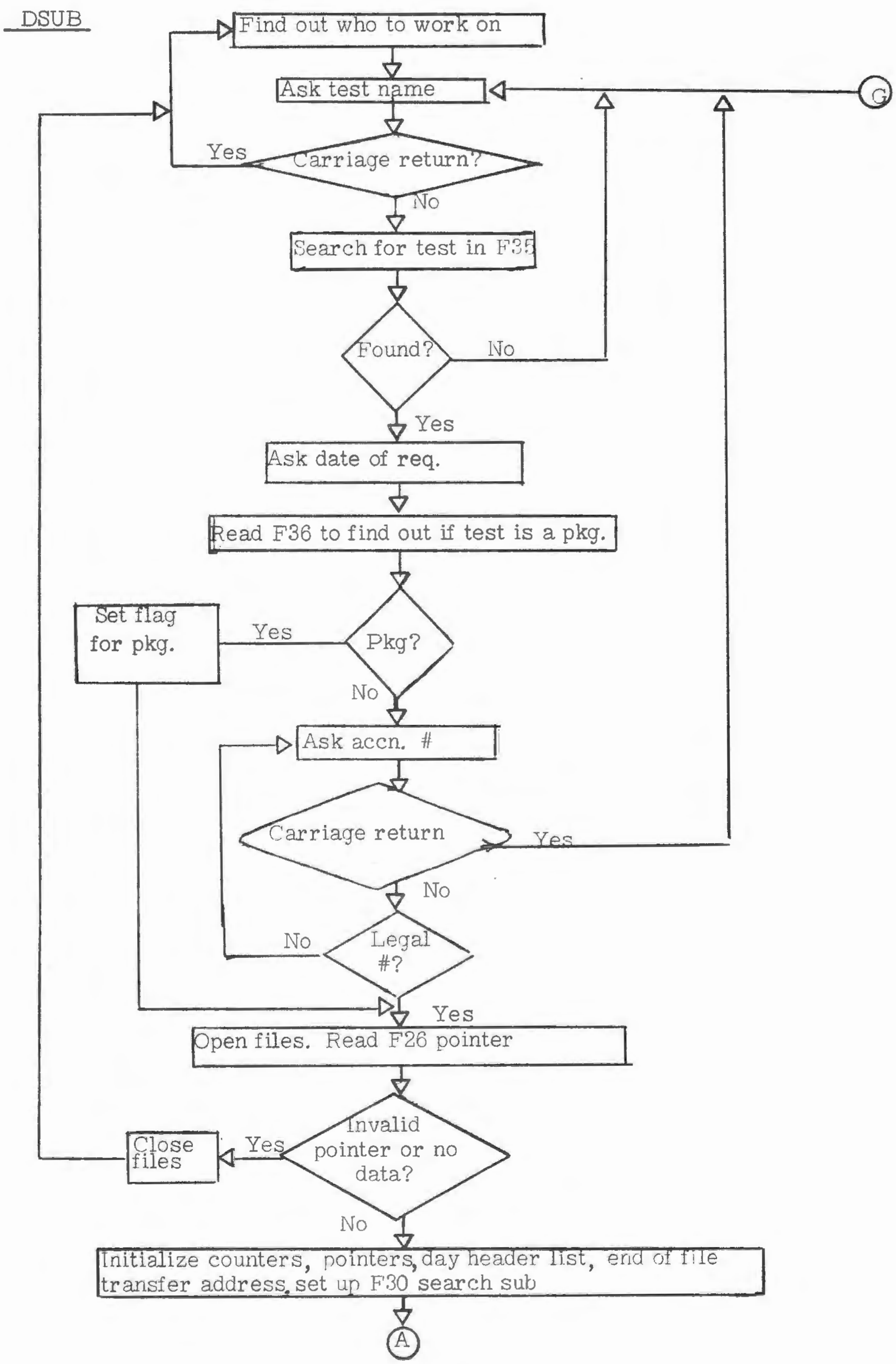
DESUB

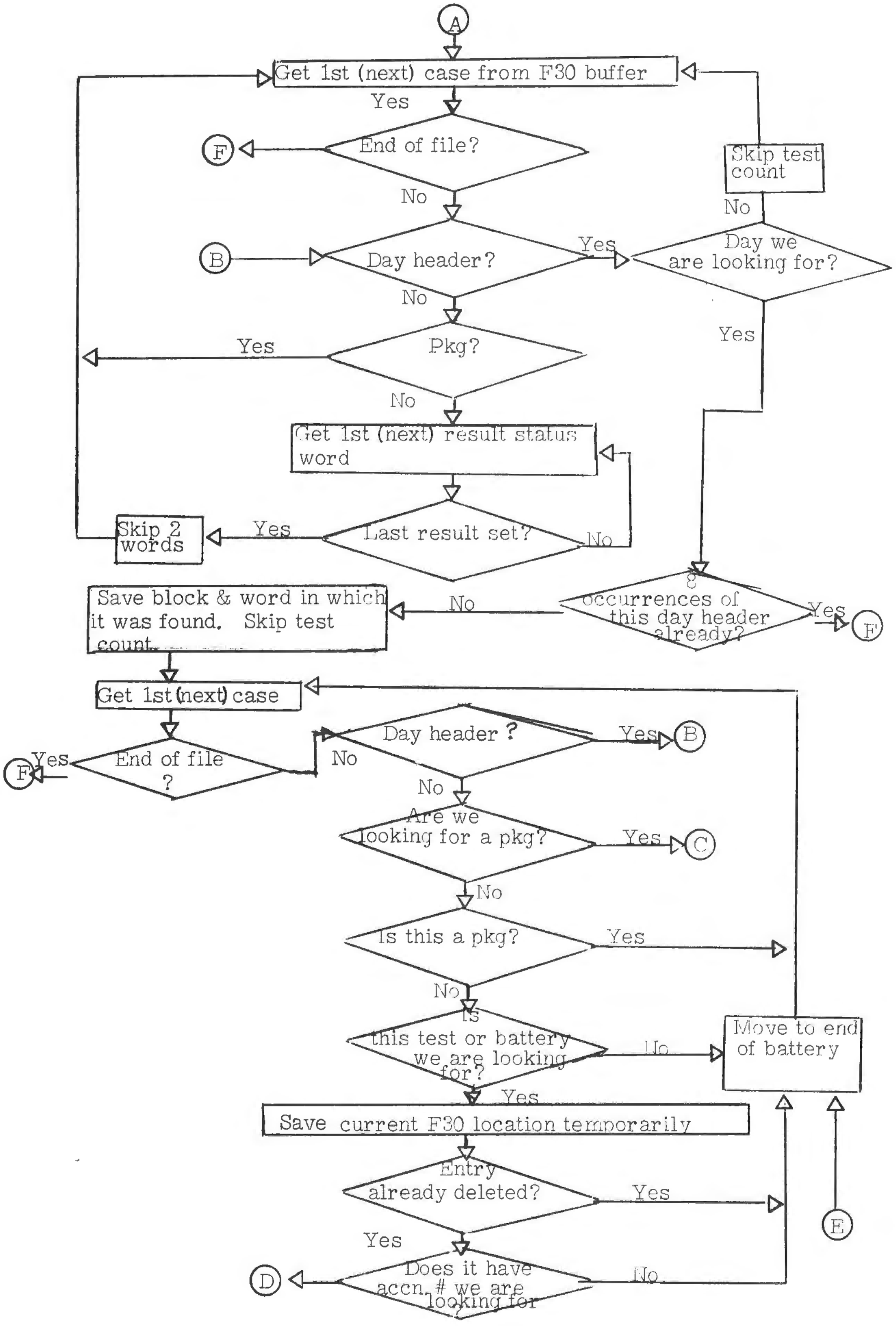

Clear flags, initialize counters, open files, read 1st block pointer from F26
Invalid pointer or no data?
Yes
C
No
Read 1st block & setup F30 handler
RE or TE activity & not O. K. to delete pat. with activity bits set?
Yes
B
No
Get 1st (next) case from F30. end of file?
Yes
A
No
Dayheader?
No
Yes
Save value of current d. h. & skip test count
Package?
Yes
No
Must be test or battery; save accn. #
Get 1st (next) status word. in complete?
No
Yes
Set flag to note inc. test found
128 or more accn. #'s saved in list already?
Yes
No
This accn# is list already?
Yes
No
Add it to accn. # list
Last result set of this test (battery)?
Yes
No

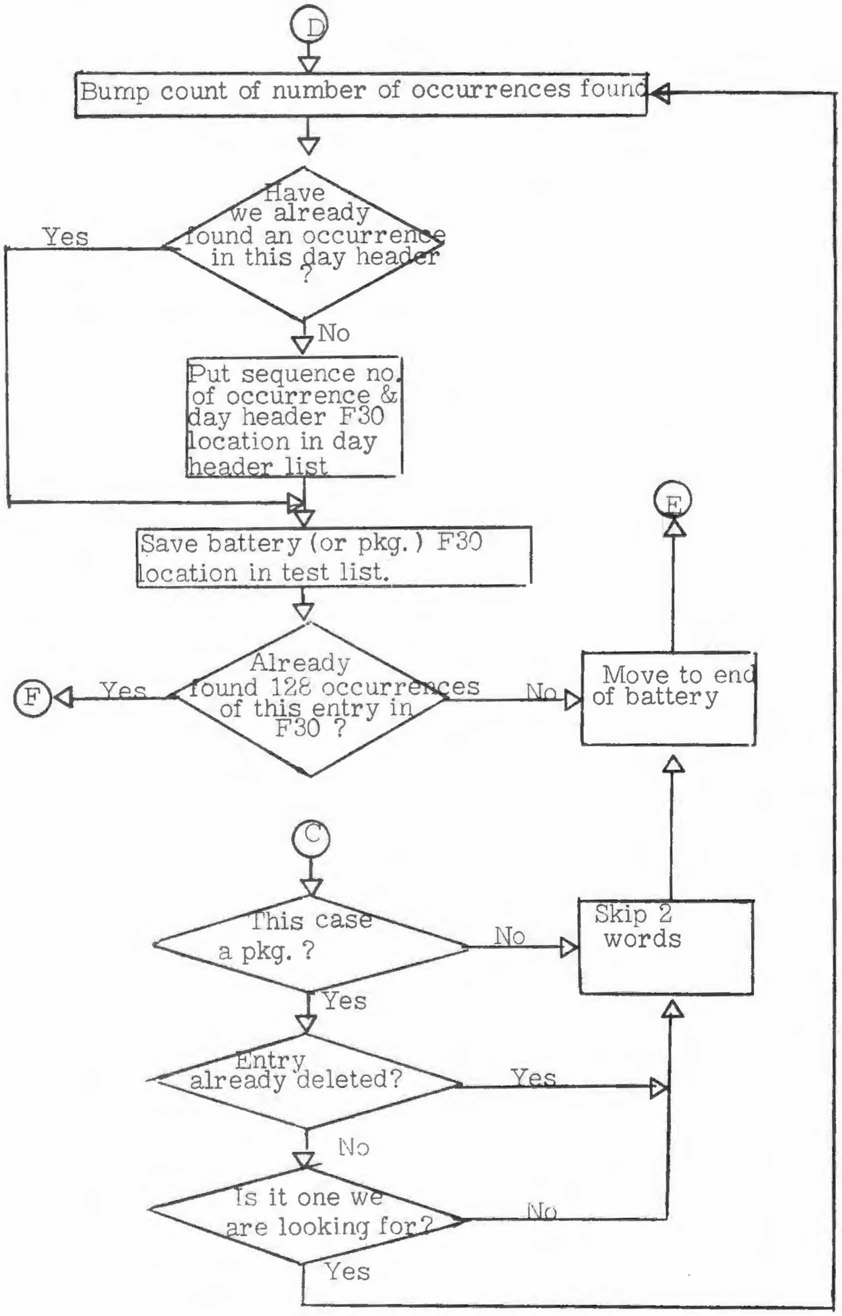

A
Flag set indicating to CHK D. H. criteria
No
Yes
Is latest D. H. found<selected date or is selected date in Jan. and last D. H. in Dec?
No
Yes
B
Inc. tests found & not O. K. to del. pt. with inc. tests ?
No
Yes
C
In PASS2 of prog?
No
Yes
JMS WRITF3: append pat. to file 3
Return
Either batch del. or in PASS2 of prog. ?
Yes
No
Tell user that pt. doesn't meet del. criteria
Return
Any Acc#'s to del. ?
No
Yes
Get 1st (next) Acc# from list
Invalid?
No
Yes
Clear Acc# from F27
Done with list ?
No
YES

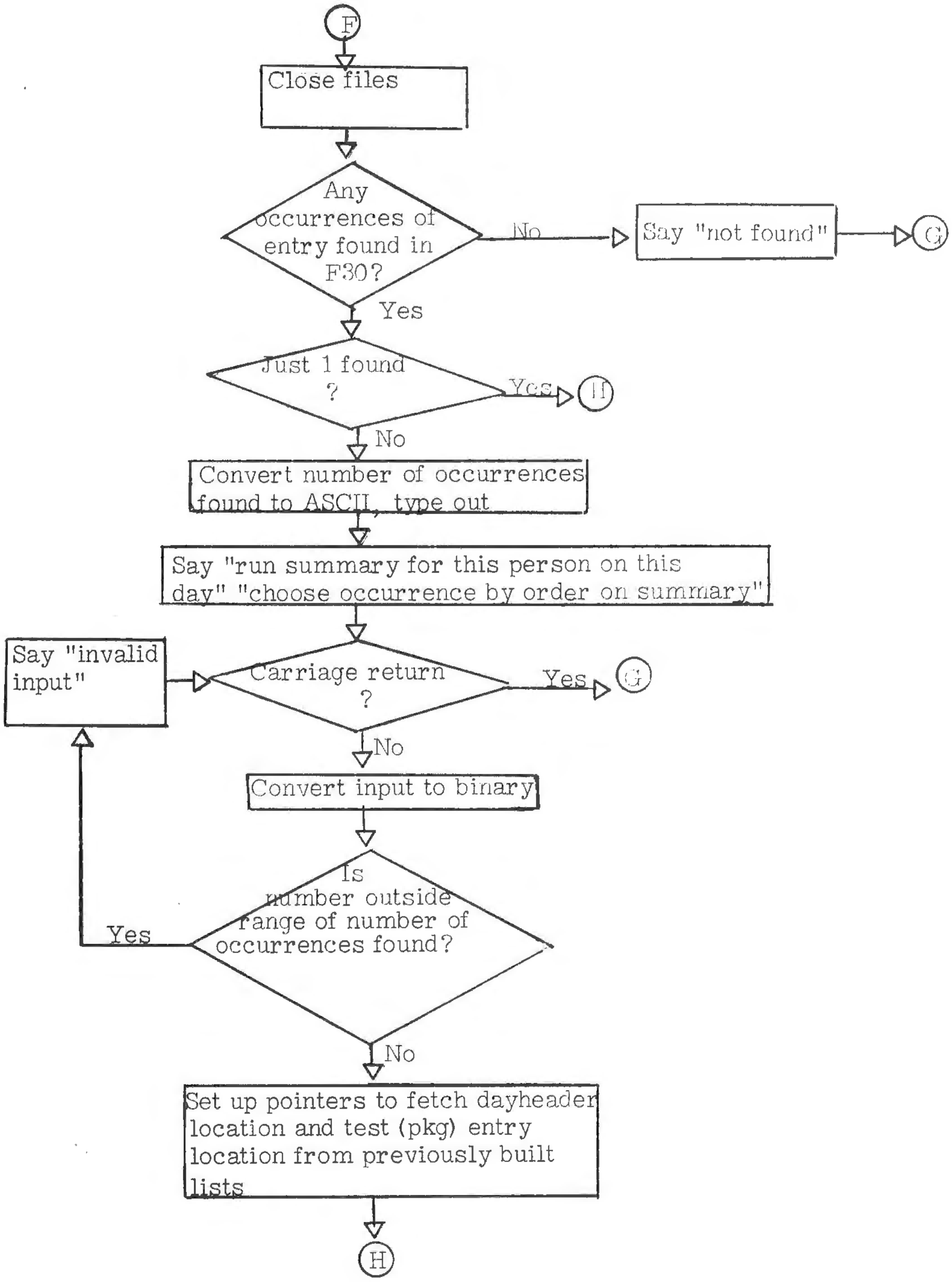

F
Did pat. have any F30 data?
No
Yes
Read F34 to core
Fetch 1st next from F30 block list. Clear corr. bit in bit map
Last F30 block?
No
YES
Write F34 back to disk
Del. pat. admin. data?
No
Yes
JMS DELSRT to delete pat. from sort files JMS DELAD to delete pat. from adm. data files.
Write 7777 in F26
Write 7776 in F26
Return

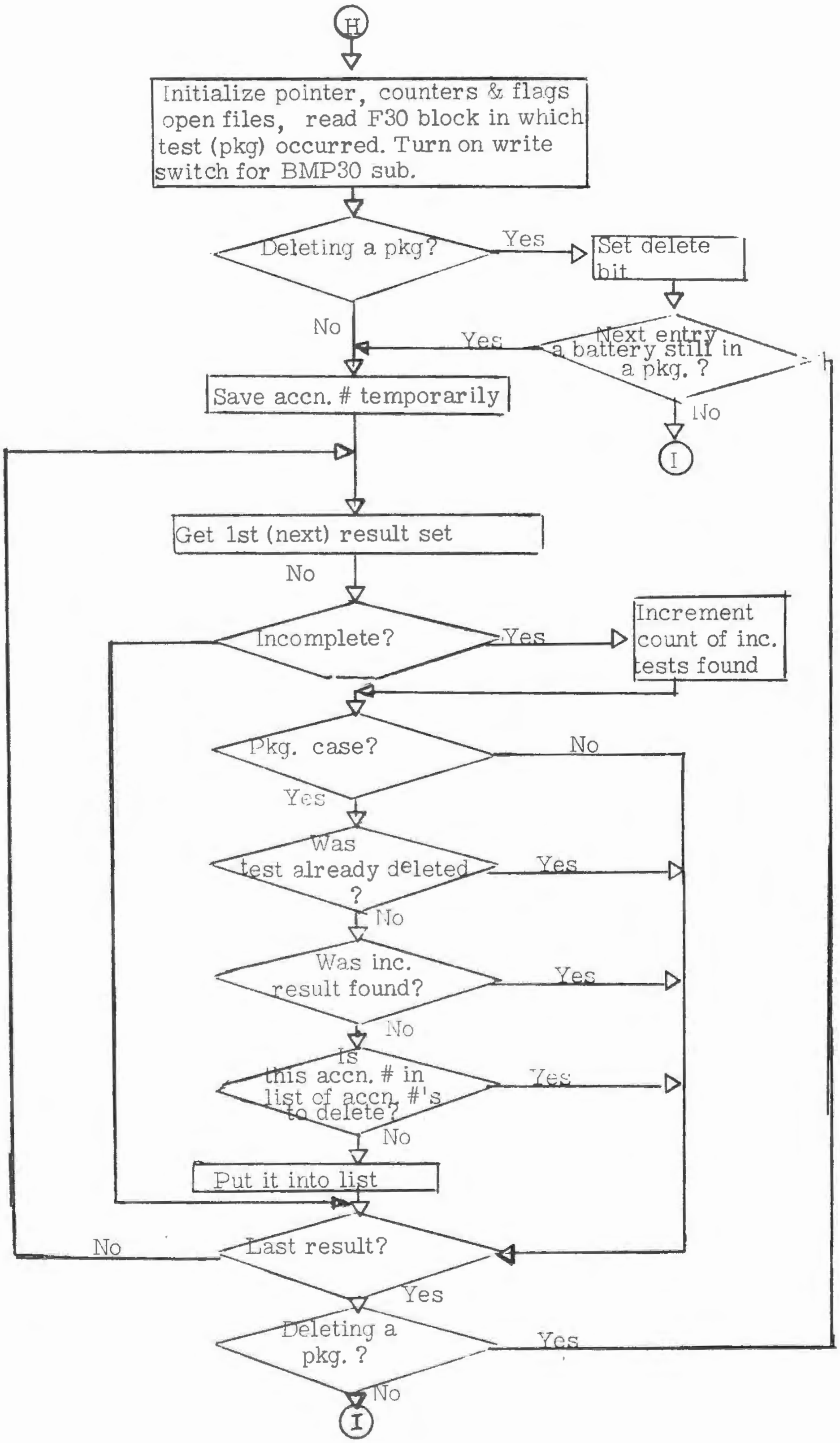
BMP 30
AC=0?
No
Yes
Read F30 block in DELINK
Set up F30 block list, pointer & counter for F30 block buffer
Return
Make counter from contents of AC
Done with block ?
Yes
No
Move to next word of buffer
Moved forward enough?
No
Yes
Fetch value from buffer, return with contents in AC
Is forward link EOF or invalid pointer?
Yes
No
Jump to end of file transfer address
16 blocks in this string?
Yes
No
Write this block back out before reading next one?
Yes
No
Write F30 block buffer back out
Put current block into block list, read next block. Reset counters and pointers

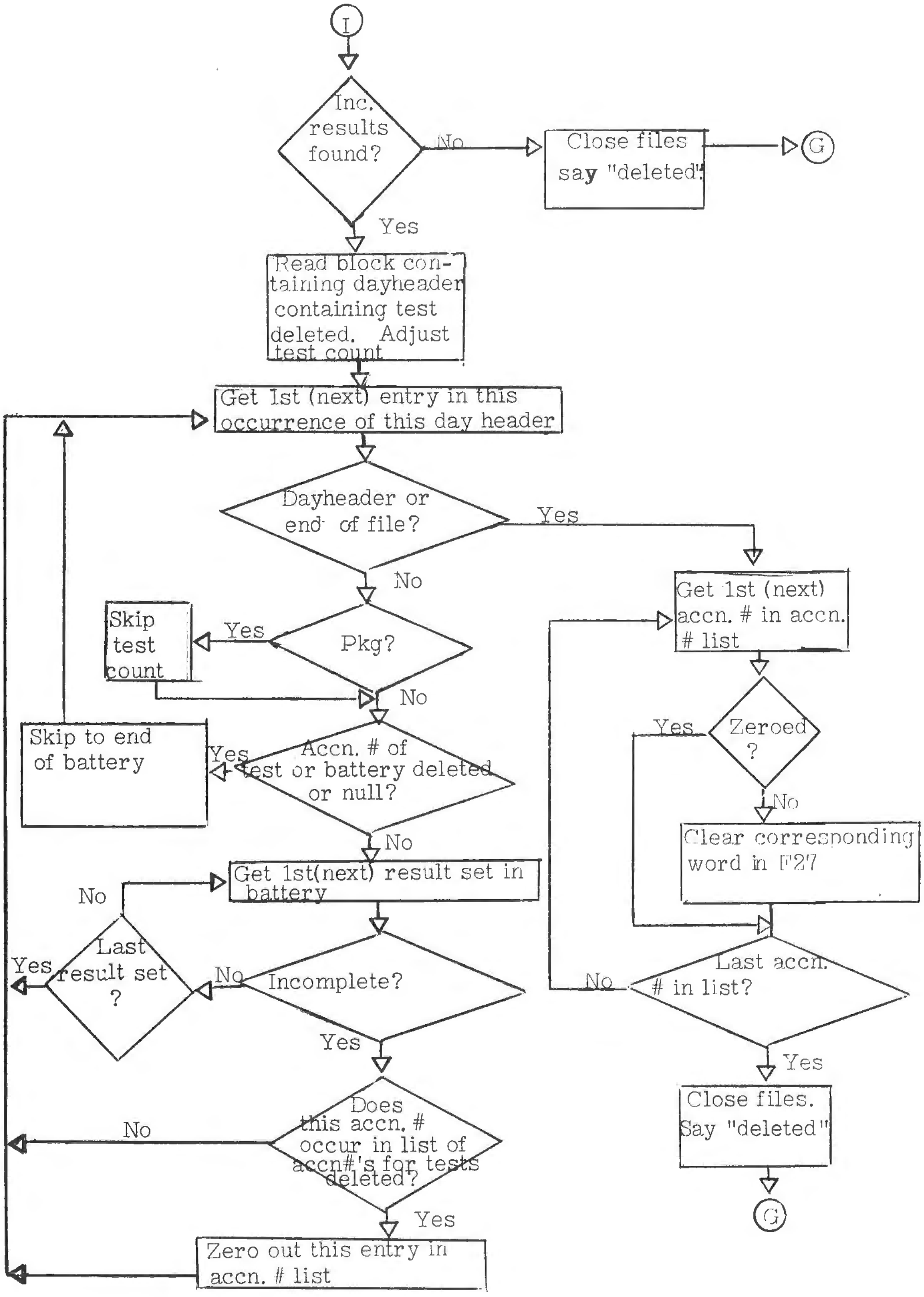

DSUB
Find out who to work on
Ask test name
G
Yes
Carriage return?
No
Search for test in F35
Found?
No
Yes
Ask date of req.
Read F36 to find out if test is a pkg.
Set flag for pkg.
Yes
Pkg?
No
Ask accn. #
Carriage return
Yes
No
No
Legal #?
Yes
Open files. Read F26 pointer
Close files
Yes
Invalid pointer or no data?
No
Initialize counters, pointers, day header list, end of file transfer address, set up F30 search sub
A


A
Get 1st (next) case from F30 buffer
Yes
End of file?
F
No
Skip test count
No
Day header?
B
Yes
Day we are looking for?
No
Pkg?
Yes
Yes
No
Get 1st (next) result status word
Skip 2 words
Yes
Last result set?
No
Save block & word in which it was found. Skip test count.
No
8 occurrences of this day header already?
Yes
F
Get 1st (next) case
Day header ?
Yes
B
F
Yes
End of file ?
No
No
Are we looking for a pkg?
Yes
C
No
Is this a pkg?
Yes
No
Is this test or battery we are looking for?
No
Move to end of battery
Yes
Save current F30 location temporarily
Entry already deleted?
Yes
E
Yes
D
Does it have accn. # we are looking for ?
No


D
Bump count of number of occurrences found
Have we already found an occurrence in this day header ?
Yes
No
Put sequence no. of occurrence & day header F30 location in day header list
Save battery (or pkg.) F30 location in test list.
Already found 128 occurrences of this entry in F30 ?
Yes
F
No
Move to end of battery
E
C
This case a pkg. ?
No
Skip 2 words
Yes
Entry already deleted?
Yes
No
Is it one we are looking for?
No
Yes

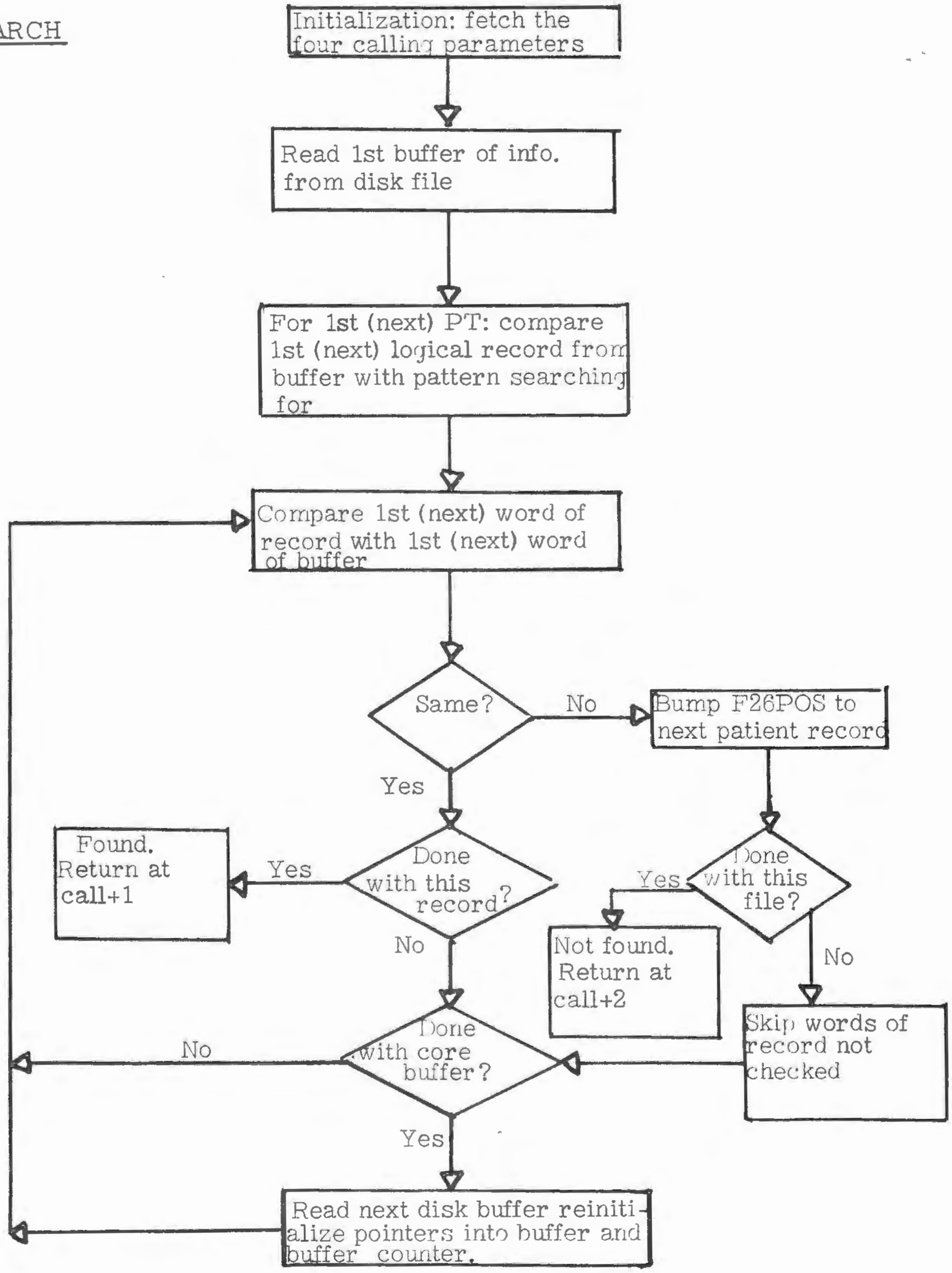

F
Close files
Any occurrences of entry found in F30?
No
Say "not found"
G
Yes
Just 1 found ?
Yes
H
No
Convert number of occurrences found to ASCII, type out
Say "run summary for this person on this day" "choose occurrence by order on summary"
Say "invalid input"
Carriage return ?
Yes
G
No
Convert input to binary
Is number outside range of number of occurrences found?
Yes
No
Set up pointers to fetch dayheader location and test (pkg) entry location from previously built lists
H


H
Initialize pointer, counters & flags open files, read F30 block in which test (pkg) occurred. Turn on write switch for BMP30 sub.
Deleting a pkg?
Yes
Set delete bit
No
Yes
Next entry a battery still in a pkg. ?
No
I
Save accn. # temporarily
Get 1st (next) result set
No
Incomplete?
Yes
Increment count of inc. tests found
Pkg. case?
No
Yes
Was test already deleted ?
Yes
No
Was inc. result found?
Yes
No
Is this accn. # in list of accn. #'s to delete?
Yes
No
Put it into list
No
Last result?
Yes
Deleting a pkg. ?
Yes
No
I



I
Inc. results found?
No
Close files say "deleted".
G
Yes
Read block containing dayheader containing test deleted. Adjust test count
Get 1st (next) entry in this occurrence of this day header
Dayheader or end of file?
Yes
No
Pkg?
Yes
Skip test count
No
Accn. # of test or battery deleted or null?
Yes
Skip to end of battery
No
Get 1st(next) result set in battery
Incomplete?
No
Last result set ?
No
Yes
Yes
Does this accn. # occur in list of accn#'s for tests deleted?
No
Yes
Zero out this entry in accn. # list
Get 1st (next) accn. # in accn. # list
Zeroed ?
Yes
No
Clear corresponding word in F27
Last accn. # in list?
No
Yes
Close files. Say "deleted"
G

SEARCH


Initialization: fetch the four calling parameters
Read 1st buffer of info. from disk file
For 1st (next) PT: compare 1st (next) logical record from buffer with pattern searching for
Compare 1st (next) word of record with 1st (next) word of buffer
Same?
No
Bump F26POS to next patient record
Yes
Done with this record?
Yes
Found. Return at call+1
Done with this file?
Yes
Not found. Return at call+2
No
No
Skip words of record not checked
Done with core buffer?
No
Yes
Read next disk buffer reinitialize pointers into buffer and buffer counter.

CHAPTER 5

TEST UPDATE

The Test Update (TE) program allows the technician to enter or edit results in the patient test data file (file 30). New test results may be entered through either terminal or card reader input; previously entered results may be edited using the terminal.

The program is divided into two major sections, one handling terminal input, the other handling card reader input. Program TE is initially loaded and immediately asks for the input device. If input is through the card reader, T3 is loaded and jumped to. TE handles terminal input.

5.1 TERMINAL INPUT

5.1.1 Initial Input

TE asks three initial questions: mode of operation, tech code and test/workstation name. Once these have been specified, they cannot be changed without leaving the program and re-entering.

TE first requests the mode of operation, ENTER, MODIFY, or STOP. STOP terminates the program immediately. ENTER and MODIFY differ in the way in which a pointer to the first block of patient data is obtained. For ENTER, the accession number must be active so file 27 provides an immediate pointer to the first block of data. For MODIFY, the accession number may no longer be assigned to the patient for whom data is being filed, so the patient number is used to obtain a pointer to the first block of data.

Once mode is established, TE requests the tech code, a number from Ø to 63, for the results being entered. The six-bit binary number thus obtained is stored in bits 6-11 of the status word of each result which is updated.

TE then asks for the test/workstation for which test results are being entered. Input may be a test name, a battery name, or, when operating in the ENTER Mode, a workstation package name.

a. If a test is entered, instances of that test requested as a single test within the patient data will be found.

b. If a battery is entered, instances of that battery requested within the patient data will be found. Also, any instance of a request for a single test which is part of the battery will be found.

c. If a workstation is entered, every request for a single test which is part of the workstation will be found. In addition, for every battery in the workstation, each result within the battery for a test which is explicitly part of the workstation will be found. For tests within the battery which are not themselves part of the workstation, results will not be found.

5.1.2 Maps Constructed

TE constructs two core resident tables on the basis of the test/workstation name entered. The first is a test map in bank 6/quarter 2. It contains a 6-bit slot for every test/battery/package in the system, in the order in which they were defined in TABDATA.

a. If a test is requested, its slot is given a 1 and all others are set to 77[NULL].

b. If a battery is requested, its slot is given a 4Ø, each slot corresponding to a test in the battery is given a number equal to its position in the battery, starting with 1 and all others are set to 77[NULL]. The battery may have no more than 37_8 tests in it.

c. If a workstation is requested, all slots corresponding to tests in the workstation are given sequential values starting with 1, all slots corresponding to batteries in the workstation are given sequential values starting with 40, and all other slots are set to 77[NULL]. The workstation may have no more than 37_8 tests mentioned in it. It may also have no more than 37_8 batteries mentioned in it. And the sum total of all the tests mentioned within all the batteries, including duplications, may be no more than 340_8.

The second map is an image of file 42 for each battery which has a non-null value in the test map, residing in bank 6/quarter 1. The map is divided into two sections. The first section is 40_8 locations long and contains in each location a pointer to the second section of the map. For each battery in the test map, identified by an entry between 40 and 76, the entry points to a location in this index to the battery map: 40 points to location 1, 41 to location 2, etc. The entry in the battery map index points to the initial location in the second section of the map, where an exact copy of the file 42 pointers for the battery associated with that index location may be found. The second section of the battery map is 340_8 locations long.

5.1.3 Patient Identification

After the test and battery map have been constructed, TE is ready to start filing data. The technician must identify the patient for whom the data is to be stored.

In the ENTER mode, the accession number associated with the test results to be filed must be active and must still be assigned to the patient for whom those tests were requested. TE saves the accession number for test identification and gets a pointer to the first block of data for the patient in question from file 27. The first block of file 30 is then read and location 2 of that block provides a pointer to file 26 which is used to get the patient name.

In the MODIFY mode, the accession number associated with the results to be edited may be free or may have been reassigned, so the technician must provide the patient number. The patient number provides a pointer into file 26 which in turn provides a pointer to the first block of patient data in file 30. The accession number is also requested to provide identification of the test result in question.

Once the accession # or patient #/accession number combination has been provided, TE types out the patient name for the technician to verify. If the technician rejects the name, TE goes back and asks for the next patient identification. If the name is accepted, TE is ready to file results. It loads overlay T2 into bank 5 and jumps to it.

5.1.4 Finding Tests in F30 Data

T2 starts at the beginning of the patient's file 30 data and searches for a match on accession number and test type. A match on test type for an individual test is a non-null entry in the corresponding slot of the test map. For a match on test type for a battery, the slot corresponding to the battery in the test map must be non-null. T2 then goes to the battery map locations for the battery. Each location contains a pointer to a test. For each test pointer in the battery map, T2 looks at the slot in the test map corresponding to that test. A non-null value in the test slot constitutes a match on test type.

5.1.5 Filing Results

As T2 moves through the file 30 data, it saves the most recently encountered day header and the location - block number and location within the block - of the outstanding test counts. When a match on test type/accession number is encountered in the file, T2 types out the date of the test and the test name from file 35. If the test result is already complete, T2 decodes the result and prints that out also. On numerical results, T2 checks the abnormal bit (status word bit 3) and, if it is set, prints out a message that the result is abnormal. T2 then waits for the technician to input the new test result.

If the technician types just Carriage RETURN, T2 simply skips over this test and goes on to look for the next test type/accession number match in the file. Typing just a dash (-) deletes the test by storing an English result pointer of Ø as the test result and marking the test as complete. A test deleted in this manner can be restored at any time through TE.

If the technician types a numerical or English result and possibly a dash followed by a modifier, TE decodes the result, checks for legality of the entire result, and stores it.

A numerical result may be in the range 0-2047000 and may have up to 7 decimal places. Ignoring decimal places, for numbers in the range 0-2047, four significant digits plus a scale factor are saved. For results above this range only three significant digits plus a scale factor are saved. T2 checks to see if the result is within the normal range, and if it is not, T2 prints a message and sets the abnormal bit (status word bit 3) for that result.

An English result or a modifier must have been defined in TABDATA. The four character code for the result is typed in and T2 searches file 44 for an exact match. A pointer to the result position in file 44 is stored as the test result. A modifier to a numerical result must be one of the first 511_{10} English results defined in TABDATA (since a pointer to a modifier is only 9 bits long).

Any English result code may be used as a modifier to an English result. Bit Ø of the second word of the two-word result identifies the result type:

```
          bit Ø=Ø      numerical result
               =1      English result
```

Any error in the result or the modifier causes the entire input to be rejected. T2 then retypes the date/test name/previous result and waits for new input.

When a result has been accepted and properly formatted, T2 stores the two result words in the proper two words of the file 30 data in core. The new tech code and abnormal bit (if required) are added to the old status word, the result complete bit is set, and T2 stores the status word too. Then the block of F30 data in core is written back onto the disk. If the result for this test was incomplete when T2 began, T2 then reads the block of F30 data containing the outstanding test count into core, subtracts 1 from the count and writes the block back onto the disk. T2 then reads in the first block of patient data, sets the new results activity bit, bit 1 of word 1, and writes the block back onto the disk. T2 then returns to TE to ask for the next patient identification.

5.1.6 Examples

Below is a sample conversation with TEST UPDATE in the ENTER and the MODIFY modes. In the first example, input is in the ENTER mode and is for workstation WSAD which contains individual tests NA, K, CL, CO2, and PH. In the second example, input is in the MODIFY mode and is for the test GLUC. In the examples, terminal output from TE is underlined.

```
TE

INPUT DEVICE 1-TELETYPE[1]  2-CARD READER
TYPE 1 OR 2*  1

E M OR S * E
TECH CODE * 12
TEST/WORKSTATION NAME * WSAD

ACC # * 31   FAWCETT MICHAEL     Y

  Ø8/24  NA      :  137.

  Ø8/24  K       :   -

  Ø8/24  CL      :  1Ø2.

  Ø8/24  CO2     :  NREQ

  Ø8/24  PH      :  6.8

  OUTSIDE NORMAL RANGE

ACC # * STOP

TTY IS FREE


TE

INPUT DEVICE  1-TELETYPE  2-CARD READER
TYPE 1 OR 2 *  1
E M OR S * M
TECH CODE * 12
TEST/WORKSTATION NAME * GLUC


PAT # * 826ØØ

ACC # * 5864   CANN ALLISON     Y

  Ø8/24  GLUC   168.  (ABNORMAL):    256. - SEE

PAT # *  395721Ø

ACC # *  65  WINCHELL SUSAN     Y

  Ø8/24  GLUC  :

PAT # * STOP

TTY IS FREE
```

[1] Teletype is a registered trademark of the Teletype Corporation.

5.2 CARD READER INPUT

The card reader portion of TEST UPDATE is divided into two parts, the reading in and verification of the cards and the actual filing of the data contained on the cards. Input is assumed to be in the ENTER mode, and no result is filed unless there is an empty slot available for it.

5.2.1 Initial Conversation

The initial conversation is handled by T5. T5 first asks for the output device for listing the cards, terminal or line printer. It then instructs the technician to load the cards into the card reader and make sure the card reader is ready before going on with the program. T5 then selects the first scratch file it finds available - out of files 10-15 - on which to store the images of the cards read in. Once this conversation has been completed, it is never repeated.

5.2.2 Verification

5.2.2.1 Card Identification - T3 initiates a read of three cards. If for some reason three cards cannot be read, either because the cards ran out or because of some hardware problem with the card reader, T3 records the fact, to be dealt with later, and processes all those cards which were read in correctly.

To process a card, T3 checks the first three fields of the card to be sure it is a valid card. Field 1, column 1, contains the card reader package identification. Rows 12 and 11 must be blank, identifying the card as a TEST UPDATE card. Rows 0-5 contain the six-bit ASCII code of the third character of the card reader package name, CD*T. T3 searches file 46 for a match on the package name. When a match is found, T3 gets the pointer to the file 50 definition of the package and stores it on top of column 1 in the card image. If any portion of field 1 is not valid, the card is rejected and its number in the pack of cards being read is printed out on the appropriate device, along with a message about why the card was rejected.

Field 2, columns 2-5, contains the accession number for the data on the card. T3 removes the four columns, converting them to six-bit ASCII and storing the characters packed in columns 2 and 3 of the card image. The ASCII is then converted to a binary accession number which is stored in the card image columns 4 and 5 in the order low accession number/high accession number. T3 gets a pointer to the first block of data for the patient using this accession number from file 27, and from word 2 of the first block of data gets a pointer to file 26 for retrieving the patient name. The pointer to the first block of data in file 30 is stored on top of column 6 of the card image. The ASCII accession number and the patient name are stored in an output buffer for the card. If any error occurs, either because the accession number contains a nonnumeric character (a numeric character is a single punch in a column/rows 0-9) or is completely blank (leading and trailing blanks are ignored), or because the accession number is not in use, the card is rejected and the appropriate message is output.

Field 3, columns 6-7, contains the technician code for the date on the card. T3 takes the two columns, converts them first to ASCII and then to binary, and stores the binary on top of column 7 of the card image. T3 checks the tech code to be sure it is between 0 and 63 and adds the ASCII to the output buffer. If any error occurs in the tech code format, either because of a nonnumeric character or an all blank field, or because the tech code is not in the range 0-63, the card is rejected and the appropriate message is output.

When the three fields have been decoded as much as possible (when an error is encountered, processing is terminated), the number of the card in the pile plus as much information as was processed plus an error message, if necessary, is output on the chosen device.

If the card is rejected, T3 does not decode any information, but simply goes on to decide what to do for the next card. If the card is not rejected, TE goes on to decode the information on the remainder of the card.

5.2.2.2 Card Result Decoding - T3 looks at each result field of the card as defined by file 50. A blank result field, one in which there are no marks/punches in any columns, is ignored. For a non-blank field, T3 gets the test/battery name for the result and then decodes the result field. No result error checking is done; T3 simply interprets the card image and prints what it sees.

For a numerical field, each column is converted to an ASCII character. The ASCII characters are stored, packed, on top of the first half of the numerical field in the card image. For each column, no mark is a blank, a mark in rows 0-9 is a digit, a mark in row 11 or 12 is a decimal point, more than one mark in a single column is a question mark. The test/battery name and the result are added to the output buffer.

For an English field, there can be more than one result in the column. T3 checks each row that could contain a result, as defined by file 50, and for each result it finds filled in, adds the test/battery name and the English result code from file 44 to the output buffer. Marks in rows which do not correspond to English results are ignored. For an English result field, T3 makes no changes to the binary card image.

T3 prints out one test result per line. When the end of this card is reached, T3 sets a bit in a card bit map corresponding to the number of this card in the stack of cards to indicate that so far the data on this card is to be filed.

(If a card was rejected, T3 resumes here.) T3 then writes the image of the card in core onto the next 80 words of the scratch file. T3 looks to see if all the cards read properly on the current read - word 361_8 of the card reader buffer contains a count - have been processed and if they have not, simply returns to process the next card. If all the good cards have been filed, T3 looks at the status word, word 360_8 of the card reader buffer, to determine if any errors occurred on the read. If no error bits are set, three cards were correctly read and processed and T3 returns to initiate the next read. An error bit is set if T3 runs out of cards in the card reader. In this case, T3 sets a flag to indicate that there are no more cards to process and goes directly to the card verification. If any other error occurs, T3 informs the user of the problem (feed error, motion error, light/dark error, word count overflow) and requests that the user either terminate now or fix the problem and continue reading cards. If the user elects to terminate, T3 sets the flag to process no more cards and goes to the card verification for those cards which were read. If the user fixes the card reader problem, T3 returns to initiate the next card read.

T3 has room to store 48 card images on the scratch file. It reads 46-48 cards, depending upon whether there are any errors during reading, before going to the card verification.

5.2.2.3 Card Verification - When all the cards (up to 48) have been read and printed out on the appropriate device, the technician can verify the cards and decide whether or not to file all the cards which were listed. If all cards are to be filed, T3 loads T4 into bank 4 and jumps to it. If not all cards are to be filed, T3 asks whether any cards are to be filed or whether only selected cards are to be filed. If no cards are to be filed, T3 looks to see if any cards remain to be read and, if cards remain, initiates a new read.

If the technician wants to selectively file some cards, he can type in the numbers of the cards which are not to be filed (a card number accompanies the output of each card) and as each number is entered, T3 clears the bit for that card in the card bit map. When all cards which are not to be filed have been entered, T3 loads T4 and jumps to it to file the remaining cards.

T4 attempts to file all the data on every card which has not been rejected either by T3 or by the technician. The technician can either file or not file a particular card but cannot select information within a single card to be filed or not filed.

5.2.3 Data Filing

T4 attempts to file all information on each card which has its corresponding bit set in the card bit map. To file a card, T4 first checks file 27 to be sure that the accession number of the card has not been freed or reassigned, in which case the card is rejected. T4 then begins to scan the patient data looking for the last group of tests within the data which are using the current accession number. Within this group of tests/batteries are found all the open result slots for this accession number. And when another accession number, day header, or end of file is encountered, this accession number does not appear anywhere farther down in the file. (This is known because an accession number can only be assigned to a group of tests and batteries if it is currently free, and when it is assigned, all the tests result slots are set up in consecutive locations of file 30 data.) Once this group of tests is located in the file 30 data, T4 is ready to file the results on the card. T4 first locates the next nonblank result field on the card and then attempts to file the result. Card results are filed differently depending upon whether the card result field is defined for a test or a battery.

5.2.3.1 Test - T4 searches the appropriate portion of file 30 data looking for a match on accession number/test type/incomplete result slot. A test which was ordered as part of a battery will be found. If no result slot is found, T4 rejects the entire result field, printing out the test name and the result decoded in the same way as T3 decoded it on the terminal. If the result slot is found, T4 decodes the result and saves it for storing in file 30.

If the result field is numeric it must be in legal format, i.e., no internal blanks, no nondigit characters, at most one decimal point, and it must be within the acceptable range for a TE result (as defined under TE terminal input). If no decimal point appears within the number, it is assumed to be to the right of the last digit. If the result is legal, it is converted to a binary mantissa plus scale factor and saved for later storing. T4 then checks to see if the result is inside the normal range and if it is not,T4 prints an appropriate message and sets the abnormal flag (status word bit 3).

If the result field is English, T4 looks at the result field from top (bit 0) to bottom (bit 11) and saves as the English result the file 44 pointer corresponding to the first legitimate mark it encounters. It is possible to have two legitimate marks in a single column. This is in the case where the second English result, that is, the one farther down in the column, is a modifier (one of the first 511 English results defined in TABDATA for a numerical result; any English result for an English result.) In this case, the file 44 pointer corresponding to that row is saved as a modifier to the English result. If there are more than two results in a column or if the second result is not a modifier, the entire result field is rejected.

If the result field is legal, and if no modifier has been entered as above, T4 looks for a modifier to the English or numeric result in the next full result field on the card. (If the result field happens to be numerical and illegal, T4 searches for a modifier and if it finds one, rejects that along with the numerical result.

This is to prevent storing the modifier as the legal result for the test in question. The same problem does not arise in the case of an English field.) In order to be interpreted as a modifier the result field must a) be in the next full result field following the current test result field (that field must naturally be defined for the same test type as the current test), b) be an English result field, c) have only one mark/punch in it, and d) have a mark/punch which corresponds to a modifier. If there is a modifier to the result, according to the above conditions, the result is interpreted as a file 44 pointer and is saved. If there is no modifier, the new result field is processed later in the same manner as the current result field is being processed.

If a legal result, with or without modifier, has been saved, T4 stores the two word result in the empty result slot which was found. The result complete bit and the new tech code are added to the result status word. If the abnormal result bit applies, it is also set. The block of file 30 data is then written onto the disk. T4 also increments the count of test results which have been saved since the last time the current test count was updated.

If the result field is illegal for some reason, T4 prints out the accession number, test name, and result(s) in the field, along with the appropriate error messages on the terminal.

After the result field(s) has been processed, T4 returns to process the next full result field on the card.

5.2.3.2 Battery - If the result field is for a battery, T4 searches the file 30 data for a match on accession number/battery type. Once the battery is found, T4 looks for the first empty result slot within the battery. The result is filed in this result slot.

If the field is numeric, it must be legal as described for a test. The result is decoded and stored in the appropriate slot in the file 30 data in core. A check is made for abnormality, as in a test.

If the result field is English, there may be as many results in the field as desired. The first result in the field is stored in the first result slot in the battery. T4 then looks to see if there if there is another empty result slot within the battery. If there is, T4 stores the next English result in the field in that slot.

T4 continues in this manner until either the end of the battery or the end of the result field is reached. If the end of the battery is reached and there are still results in the result field, the remaining results are rejected. Modifiers are never stored in a battery. Every English result is stored as a separate result for a test within the battery.

If the end of the result field is reached and there are more empty slots in the battery, T4 looks at the next non-blank result field on the card. If it is defined for the same battery as the present one, T4 proceeds to file the results in the next free slots, as above.

When the battery is full, or when there are no more consecutive nonblank result fields for the battery or when an illegal result field (numeric) is encountered, T4 writes all the battery results in core onto the disk. For each result that is updated, the count of results filed is incremented by 1.

Results which are illegal or results for which there are no open result slots are rejected.

5.2.3.3 Updating the Test Count and Activity Bit - The outstanding test count is updated when the end of a card is reached. T4 reads in the block containing the test count for the test results which were updated (they all appear under one day header), subtracts the number of results updated, and writes the block back onto file 30.

T4 also updates the test count as above if an illegal result field is encountered. Before exiting for terminal output, T4 updates the test count for all legal results which have been filed so far.

When a card is completed, T4 checks to see if any results from that card were filed. If any were, T4 reads in the first block of patient data, sets the new results activity bit (bit 1 of word 1), and writes the block back out on the disk.

5.2.3.4 Clearing the Accession Number - When the end of a card has been reached and the test count updated, T4 makes a final scan of the file 30 data (beginning where the last instance of the accession number was found initially and going to the end of the data) looking for incomplete result slots with the current accession number. If none are encountered, the test update for the current accession number is finished. T4 checks file 27 to be sure the accession number still belongs to the same patient and if it does, T4 clears the accession number in file 27. T4 then returns to process the next card to be filed.

5.2.4 Continuing

When T4 has finished filing all the cards which are legal, it checks the flag set by T3 to see if there any any more cards waiting to be read and filed. If there are, T4 loads T3 and jumps to it. If there are no more cards to be read, T4 terminates.

5.3 SPECIAL PACKAGES USED BY TE

TE uses two types of special packages during its operation, one during terminal input - workstations, the other during card reader input - TE card reader packages.

5.3.1 Workstations

Workstation packages, of the form WSab, can appear anywhere in the second segment of TABDATA. "WS" identifies the package as a workstation and a is a letter identifying the workstation. b is not used by TEST UPDATE.

For example:

```
///WSAD - AUTOMATED ELCT - ØØ.ØØ
   NA
   K
   CL
   CO2
   ELCT
   NAK
```

Assume battery ELCT consists of the tests

```
   NA
   K
   CL
   CO2
   PH
   OSMO
   PCO2
```

If a technician chose workstation WSAD to update, he would be able to update all individual tests NA, K, CL, and CO2, all NA's and K's ordered as part of the NAK battery, and all NA's, K's, CL's and CO2's ordered as part of an ELCT battery. TE would not allow the user to update the PH, OSMO, or PCO2 of the ELCT. It would also not find any instances of an NA, K, CL, or CO2 ordered as part of a battery other than NAK or ELCT.

The technician would not need to know how the tests were actually requested in the data file, and TE would not indicate.

5.3.2 Card Reader Packages

TE card reader packages, of the form CD*T, can appear anywhere in the second segment of TABDATA. "CD" identifies the package as a card reader package, "T" identifies it as a TE package and * is an alphabetic character identifying the package.

The first line of the package definition is of the form

```
///CDAT - CARD 1 - ØØ.ØØ
```

Following that is a definition of each field on the card, one field per line.

For a numeric field, the format is:

```
test/battery code-beginning column of field-# of columns in field
                                  (8-80)                  (1-9)
```

For an English result the format is:

test/ battery name	column of field (8-8Ø)	number of results in column (1-12)	row of result 1	code of result 1	row of result 2	code of result 2

For example:

```
///CDAT-CARD 1 - ØØ.ØØ
   NA - 30 - 7
   K - 40 - 7
   K - 50 - 1 - SEE - 3 - NR - 5 - WR - 7 - R
   CL - 60 - 7
   CO2 - 70 - 7
```

The numerical result for NA is in columns 30-36. The numeral result for K is in columns 40-46. K also could have an English result (if the numerical field of the card is blank) or a modifier (if the numerical field of the card is full). The column containing the result is 50. A mark in row 1 corresponds to English result code SEE, a mark in row 3 corresponds to English result code NR, and so forth. CL and CO2 are numerical result fields in columns 60-66 and 70-76 respectively.

The TABDATA input is translated into a package definition in file 50. For each result field on the card, the package definition contains a group of words where:

word 1: bit 0 = 1 for last test/battery in card reader package
0 otherwise

bits 1-11 = pointer to test/battery type in file 36

for a numerical result:
word 2: bit 0 = 0 for numerical

bits 1-7 column where result begins

bits 8-11 = width in columns of result

for an English result:
word 2: bit 0 = 1 for English

bits 1-7 column where result lies

bits 8-11 = number of different English results in this column (1-12)

word 3: bits 0 - 3 = row of this result

bits 4 - 11 = pointer to this result in file 44

.
.
.

word n+2 bits 0 - 3 = row of n^{th} result

bits 4 - 11 = pointer to this result in file 44

5.4 CARD FORMAT

The TE card format is as follows:

column 1: rows 12-11 = blank

rows 0-5 = 6 bit ASCII character which is third character of TE package name

rows 6-9 = ignored

columns 2-5: four digit accession number, one digit per column

columns 6-7: two digit technician code, one digit per column

columns 8-80: results to be filed, as defined in the file 50 card reader package definition

for a numeric result

rows 12-11 = decimal point

rows 0-9 = digits 0-9 respectively

for an English result

rows 0-11, numbered from top to bottom of card, as defined in file 50

5.5 ASSEMBLY INSTRUCTIONS

The TE source is broken up into four parts:

TE-T2 which chains to T1, and T4 which chains to T3-T5 which chains to T6. The binaries are stored on the start up tape according to the following scheme.

Source	Overlay Name on Startup tape	first block number	number of blocks
TE-T2, T1	TE	1,	4
	T1	11,	2
	T2	5,	4
T4, T3-T5, T6	T3	5,	4
	T4	1,	4
	T5	11,	3
	T6	15,	1

CHAPTER 6

MANUAL CALCULATIONS

The function of the MANUAL CALCULATION (CA) program is to accept raw data from the terminal, perform the calculation of final test results (answers), and automatically transfer the calculated results along with the appropriate raw data to the patient file when the accession number(s) is entered. The program can be thought of as a desk calculator interfaced directly to the patient files.

6.1 INPUT/OUTPUT

Only terminal and disk are used. The terminal is used to obtain input from the user, type replies for the user, and type reports on request. The disk files are used as follows:

FILE	NAME	USAGE
00	PROGRAM FILE	read only
35	TEST TYPE CODES	read only
44	ENGLISH RESULT CODES	read only

Files 20, 27, 30, 36, and 42 are used through DATA-PF as described in Chapter 7 , Patient Data Filer.

6.2 FUNCTIONAL DESCRIPTION

The user calls for manual calculations by typing ↑C and entering CA). The terminal response is:

```
CALCULATION NAME IS *
```

A 4-character name for the calculation is entered. If an incorrect name is entered, the terminal prints:

```
NO SUCH NAME
1.  TRY AGAIN
2.  SHOW LIST
3.  STOP

SELECT *
```

The number of the option selected is entered. Entering 3↵ causes the CA program to terminate and the terminal prints:

```
CA PROGRAM DONE
```

The CA program may also be stopped at any time by entering STOP↵. The above message is printed and the program terminates.

Selecting number 2 causes a list of available calculations to be printed as follows:

```
CALC NAMES ARE
ELCT
CCLR

etc.
 .
 .
```

The same list is printed when SHOW ME is entered for a calculation name. After printing the list or after selection of option 1, a calculation name is again requested:

```
CALCULATION NAME IS *
```

When a correct code name is entered for a calculation code the terminal responds with the full calculation name, date and time and asks for the TECH CODE. For example, entry of CCLR↵ results in the following printout:

```
CREATININE CLEARANCE
DATE  XX/XX/XX
XXXX HRS
TECH CODE IS *
```

The tech code can be 1-63. If an incorrect number is entered, the message:

```
ERROR - TECH CODE 0-63
```

is printed out and the code is requested again.

After a legal tech code is entered, the terminal prints the data entry format for the calculation. For Creatinine Clearance, for example, the format is:

```
ENTRY FORMAT
U-VOL, PERIOD, U-CRET, P-CRET
```

The data for urine volume (U-VOL), period in hours (PERIOD), urine creatinine (U-CRET), and plasma creatinine (P-CRET) are entered after the * on one line separated by commas. For example, a set of data might be:

```
*2173, 24, 65, 2.3
```

The data are entered in the ORDER SPECIFIED IN THE ENTRY FORMAT. Data may be any number up to five digits with or without a decimal point. Thus, 65 and 65.0 are the same number. If a data line is entered in incorrect format any one of several error messages may be printed, as follows:

MESSAGE	MEANING
ERROR-DATA OVER 5 DIGITS	a number is too large
ERROR-TOO MANY	may be too many entries on a data line
ERROR-NOT ENOUGH	may be not enough entries on a data line
ILLEGAL CHARACTER-ONLY ., & 0-9	probably an alpha character entered
ERROR-MULTIPLE DECIMAL POINTS	a number has more than one decimal point
ERROR-BLANK DATA	an entry was made without digits

After the data are entered, the calculated result(s) are printed:

```
CCLR - XX.XX
```

Next, an English result will be requested if it is required. For example:

```
          SPECIMEN TYPE IS *
```

A nonexistent English result causes the message

```
          NO SUCH CODE
```

and the result is requested again. Any legal English result (from File 44) may be entered. (SPFL for spinal fluid, SER for serum, etc.). If the particular calculation type does not require an English result, the question is not asked.

Finally, an accession number(s) is requested for the calculated results:

```
          PLASMA ACC # = *
```

Five types of entries can be made for the accession number.

ENTRY	ACTION
XXXX↵	When a valid accession number (1-9999) is entered, the terminal responds with the patient's name: (PT NAME) OK? * Enter Y↵ or N↵ to accept or reject the name. Y↵ causes the data to be filed. N↵ causes the ACC# to be requested again. ↵ is treated as an N↵. After filing the results, the terminal prints: FILED NEXT CALC * and waits for new data to be entered.

XXXX, R↵ — An accession number followed by a comma and R treats the accession the same as above, but after the results are filed a formatted final report with the patient's name and nursing station is typed for the calculation before going to NEXT CALC as above.

R↵ — When R alone is entered, the terminal prints:

```
DATE  XX/XX/XX
NAME *
N.S. *
```

waiting after each * for name and nursing station to be typed on the terminal. When the second ↵ is pushed the formatted calculation report is printed and the program proceeds to NEXT CALC. No data is filed.

N↵ — Entering N↵ causes the program to proceed directly to NEXT CALC. No results are filed.

↵ — A ↵ alone skips the current accession number and proceeds to the next accession number or, if there are no more, to the NEXT CALC. Results for any valid accession number are filed.

Accession number entries may result in the following error messages

MESSAGE	MEANING
NO REQUISITION FOR THIS ACC#	No requisition was entered or all results have been filed for it already.
(name) NOT SAME PATIENT	if more than one valid accession number is entered, subsequent number(s) did not belong to same patient as first entry.
TTTT ALREADY FILED	results already in patient's file for test (TTTT) indicated.
TTTT NOT REQUESTED	accession number was OK but test indicated (TTTT) was not requested.

As before, whenever STOP↵ is entered, the CA program terminates.

When more than one accession number is required, for example, urine and plasma for a CCLR package, each is requested. Any parts of a package can be filed by skipping accession numbers with ↵ as described above. Whenever a disk error is detected, the teleprinter prints:

```
DISK ERROR, CA PROG TERMINATED
```

The CA program terminates automatically. The program may be called into the computer again and the operation retried. If file 27 or 30 is not available, the computer prints:

```
WAITING FOR FILE 27 OR 3Ø
```

until the file becomes available.

6.3 PROGRAM NAME AND TABLES

The basic Manual Calculation program consists of five routines in the program file:

ROUTINE	DIAL	
FILE	MSC	
NAME	NAME	COMMENTS
CA	CAPROG	Only routine called from TTY. CA is basic control program for the system.
FP	LBCM5-FP	Floating point routine for LABCOM 5.
CT	CATEXT	The CT program first tries to load the specified calculation from the disk program file. If the disk load is unsuccessful, the Tape Loader (TA) program is read into bank 6. The CT program tape loader modify subroutine temporarily modifies the Tape Loader (TA) program to allow the Tape Loader (TA) program to run in bank 6 and to load the specified calculation from the UJ2E tape into bank 6. The Manual Calculation program then proceeds. If the specified calculation isn't found on the UJ2E (unit 5) tape, CANNOT LOAD TTY IS FREE is printed on the calling terminal as the CA program exits. The CT program was assembled with the tape loader modify subroutine and with a seventeen program calculation name table (maximum 21). A calculation to be loaded by the tape loader is restricted to a maximum of three blocks. The CT program was also assembled without the tape loader modify subroutine and with the seventeen program calculation name table (maximum 34). This table starts at location 1060.

ROUTINE FILE NAME	DIAL MSC NAME	COMMENTS
CF	CALCFILE	Asks for English results, accession numbers, and files results in patient file through DATA-PF.
PF	PF	Files 1 result in patient data file. Also used by ACcession number entry.

In addition to the above five routines, each calculation has a CALCULATION SUB PROGRAM which specifies the formats, LINC and floating point code, and procedures to be used. Each SUB PROGRAM is filed as follows:

SUBPROGRAM FILE NAME	DIAL MSC NAME	FULL NAME
C1	CRCL	Creatinine Clearance
C2	ELPR	Electrophoresis
C3	STC1	Urinary Hydroxy Steroids
C4	STC2	Urinary Keto Steroids
C9	UREC	Urea Clearance
K5	PSPE	PSP Excretion Test
K4	OEST	Urinary Oestriol Test
K2	BRSC	Bromo Sulphthalein Clearance
C5	UCAL	Urinary Calcium
C6	UUNA	Urinary Sodium
C7	UUCL	Urinary Chlorine
C8	UUKK	Urinary Potassium
K3	ALGR	A/G Ratio
K6	ESTR	Estriol
K7	PRPH	Porphyrin
K8	HEME	Hematin
K9	CPKS	Creatinine Phosphokinase
K1	MECA	Manual Entry Calculation

A table of the available calculations is at the end of the **CATEXT** manuscript (Location 1Ø6Ø). It has the format:

#1L

C		
C		4 char name in ASCII
L		
R		
215		EOL in ASCII
C	1	Prog file name, stripped ASCII
E		
L		4 char name in ASCII
C		
T		
215		EOL
C	2	Prog file name, stripped ASCII
etc.		
7777		End of List

(6 words per calculation code)

When a calculation name is entered by a user, this table is searched for a name match. The CT program first tries to load the specified calculations sub program C1, C2, etc. from the disk program file into bank 6. If the disk load is unsuccessful, the Tape Loader (TA) program is read into bank 6 and modified to load the calculation subprogram from the UJ2 tape which resides on unit five. If the specified calculation isn't found on the UJ2 tape,

```
CANNOT LOAD

TTY IS FREE
```

is printed on the calling terminal and the CA program exits. A calculation sub program which is designed to be loaded from the UJ2 tape must be less than 4 blocks long. If it is stored on the UJ1 tape it may be 4 or less blocks long.

Two procedures are necessary to implement a new calculation program.

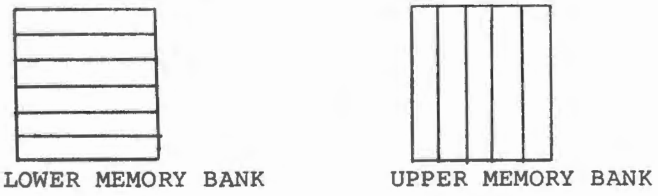
1. Prepare a SUB PROGRAM manuscript for the calculation and put the new SUB PROGRAM in the program file.
2. Put the new calculation SUB PROGRAM name (e.g. CCLR) and program file name (e.g. C4) in the table of available calculations in the CATEXT manuscript.

Putting a new name in the CATEXT table will be obvious from looking at the CATEXT manuscript. The preparation of a new CALCULATIONS SUB PROGRAM is explained in detail in section 6.5.

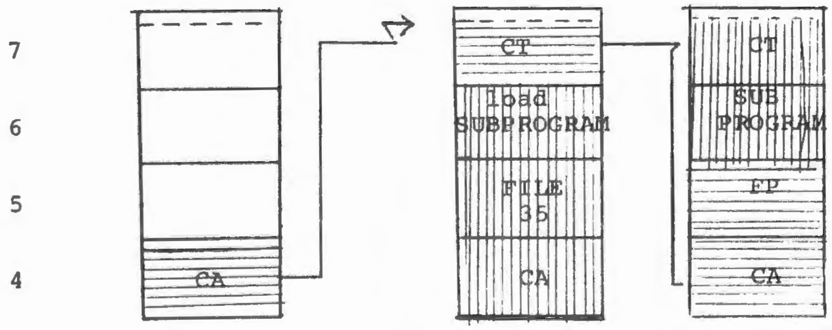
6.4 MEMORY BANK DIAGRAMS

The following gives the memory bank snapshots during various stages of the calculation procedure.

NOTE



SETUP TO CALCULATE

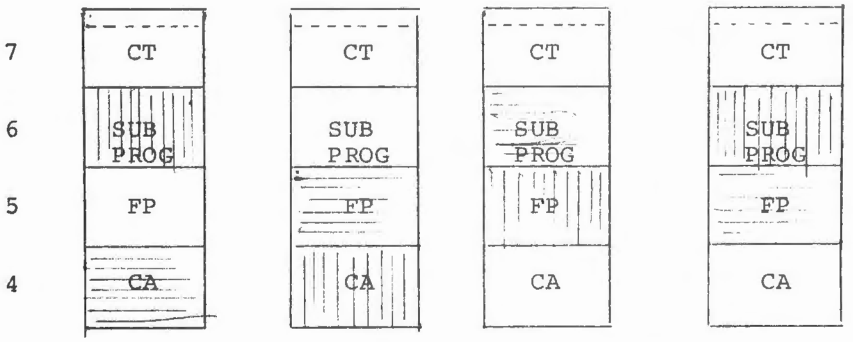


CA called from terminal. CA loads CT. Transfers control to CT.

CT asks for CALC name, loads the CALC SUB PROGRAM and looks up all test types in File 35. Returns control to CA.

CA prints CALC name, date, time, requests TECH CODE. Prints ENTRY FORMAT. FP may be LMB during ASCII to FLT PT conversions.

ENTER DATA AND CALCULATE



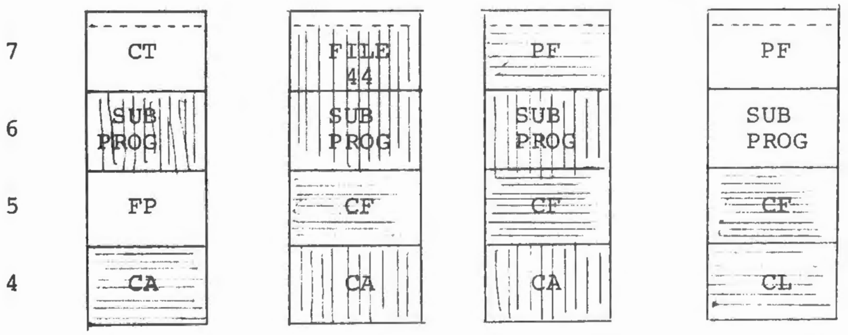
Enter data line on terminal

Convert ASCII to FLT PT or FLT PT to ASCII

Execute LINC code in SUB PROG

Execute floating point code in SUB PROG

PRINT AND FILE RESULTS



Print results (also see FLT PT to ASCII conversion)

Enter English results. CF use TTY buffer in CA

Enter ACC# and file results

Exit. If summary reports, CL loaded in Bank 4 and started at loc.2

6.5 PREPARATION OF CALCULATION SUBPROGRAM

The basic calculation programs (CA, CT, CF, and FP) operate from information in the SUB PROGRAM. To understand the use of information in the SUB PROGRAM is to understand the calculation system. A flow diagram is provided at the end of this chapter. The order in which information is used in the SUB PROGRAM implies the order of operation of the basic calculation programs, although it is not exact in some cases.

Implementing a "new" calculation requires the development of a new SUB PROGRAM. The following sections provide descriptions of the information requirements in the SUB PROGRAM. When reading this description it is assumed the reader is referring to examples in manuscripts (e.g., CCLR, ELCT). All pointers referred to can be found at the beginning of the SUB PROGRAM manuscript.

6.5.1 Calculation Name (pointer A8+2ØØØ, loc. 2Ø)

The full calculation name to be printed when the SUB PROGRAM is called is put in as an ASCII string at tag A8. 72 characters maximum.

6.5.2 Data Field (pointer I8+2ØØØ, loc. 3Ø)

The data field contains all data (constants, raw data, answers, English results, etc.) in six word formats:

WORD 1 ABC XXX XXX XXX

* A	Precalculation input data if = 1
* B	Main calculation input data if = 1
C	not used
X	test type code number

WORD 2 ENM MMM SSS PPP

* E	result is English if = 1
N	not used
M	English modifier
S	scale factor for result
* P	no. decimal places in result

(same as channel storage word 4)

WORD 3   RRR   RRR   RRR   RRR

R   numeric result or English pointer

(same as channel storage word 5)

WORD 4-6 floating point word

For each datum in the calculation there must be a corresponding 6 word data format in the data field. Only the bits indicated by * need to be set in each data format. Other information is provided automatically by the basic programs.

6.5.3 Entry Formats (pointer B8+2ØØØ, loc. 21; E8+2ØØØ, loc. 24)

The entry format is entered as an ASCII string at tags B8 and E8. Up to 72 characters are permitted. The line "ENTRY FORMAT" is always printed by the basic programs before printing the ASCII string.

6.5.4 Number of Entries

The number of data inputs required for the precalculation is put in SUB PROGRAM location 34, the main calculation number is put at location 35. Both are in octal. These numbers are used by the basic program to check for the correct number of inputs.

6.5.5 Floating Point Code (pointer B8+2ØØØ, loc. 21; E8+2ØØØ, loc. 24)

The floating point operations are entered by executing the JMP C8 at location 22 for the precalculation and the JMP F8 at location 25 for the main calculation. If there is no code for a calculation the tags C8 or F8 must have a return to bank 4 at exit+1. For example:

```
C8,  LDA   LOAD RETURN
     Ø
     STC .+2

     LMB4
     JMP  /RETURN .+1
```

Actually, any code, whether LINC or floating point, can be executed at C8 or F8 as long as the JMP return is saved. Upon entry at C8 or F8, bank 5 contains the floating point subroutine (LBCM-5FP) for

LABCOM 5 set to return to the SUB PROGRAM in bank 6 after each use.

6.5.6 Program Patches

Program patch 1 (JMP K8, loc. 32) and patch 2 (JMP L8, loc 33) are provided for additional insertion of program operations. (Refer to the flow chart Main Calculation at the end of this chapter.) As for the floating point codes (C8, F8), the tags I8 and L8 must contain at least a program to return to bank 4 at exit+1. Upon entry to patch 1 or 2, the floating point program is in bank 5 set to operate from bank 6.

6.5.7 Print Result Formats (pointers D8+2ØØØ, loc. 23; G8+2ØØØ, loc. 26; H8+2ØØØ, loc. 27)

Free formatted text reports for data can be generated at three times -1) after precalculation, 2) after main calculation, 3) at end of procedure. These reports are specified freely through the use of ASCII characters, tags, and appropriate delimiters. The general format is:

```
TAG,    ASCII TEXT
             .
             .
             .
             .
           7777        (a 7777 means result tag follows)
        data tag 1     (tag for data in data field)
        ASCII TEXT
             .
             .
             .
           7777
        data tag 2
            215        (end of line)
        ASCII TEXT
             .
             .
             .
           7777
        data tag 3
        ASCII TEXT
           etc.
             .
             .
             .
           7777        (two 7777's means end of report)
           7777
```

The basic program transfers the text string to the terminal buffer until a 215 (EOL) is encountered. Data formatted in ASCII is inserted automatically whenever a tag following a 7777 is encountered. When constructing a text line one should allow for the length of the result so that more than 72 characters are not generated on a single line. If a line exceeds 72 characters it is truncated. A data tag can not refer to an English result. Only numeric results are automatically formatted for calculation output.

6.5.8 List of Result Types (pointer T8+2ØØØ, loc. 41)

All results in the data field which are to go to the patient file are listed at T8 in the following format:

```
T8,    AA          (four character stripped ASCII name
       AA          as in file 35)
       data tag    (associated tag in data field I8)
       BB          (second name)
       BB          etc.
       data tag
       7777        (end of list)
```

The result types in list 8T are looked up in file 35 and the numeric test types inserted in each data word 1 in field I8 each time the SUB PROGRAM is called. Thus, the data names must correspond to the correct names in file 35.

If a name is changed in file 35 it must also be changed in the calculation SUB PROGRAM. The list should also include any result in file 35 which has an English answer (e.g., SPEC, specimen type). The English result is looked up automatically in file 44 when it is entered.

6.5.9 Accession Number Entry (pointer J8+2ØØØ, loc. 31)

Text for accession number entry is similar to the report formats with the addition of two locations for the accession number.

Example:

```
          J8,    ASCII TEXT
                      .
                      .
                      .
                     375       (ALT MODE)
                      Ø
                      Ø        locations for ACC number
                 data tag 1    list of data tags to be
                 data tag 2    filed with ACC #
                    7777       (end 1st ACC #)
                      .
                      .        text, etc. for another number
                      .
                      .
                      .
                    etc.
                    7777       two 7777's end ACC #
                    7777       entry procedure
```

The basic programs type the text up to ALT MODE, accept the accession number, and file the associated data for the tags listed with each number.

6.5.10 English Results (pointer R8+2ØØØ, loc. 42)

English results are accepted by placing text at 8R as follows:

```
           R8,   ASCII TEXT
                      .
                      .
                      .
                     375       (ALT MODE)
                 data tag 1    (only one data tag)
                 ASCII TEXT
                      .
                      .
                      .
                     375
                 data tag 2
                    etc.
                    7777       (end of English results)
```

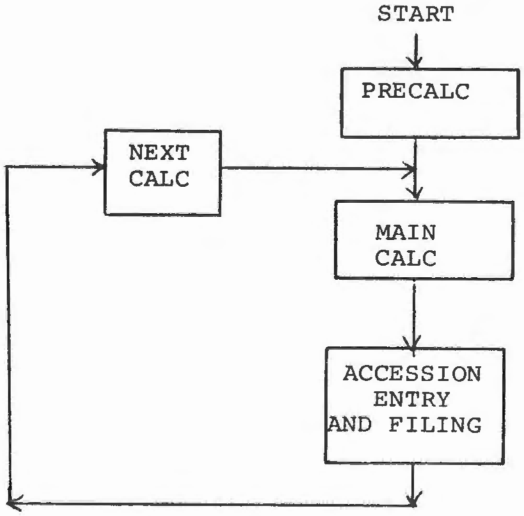
The English result response entered on the terminal is looked up in file 44 and the numeric code inserted in the data result word by the basic programs. The English result type (not answer) is assumed to be in the file 35 list at T8.

6.5.11 General Directions

Making all of the above entries in the SUB PROGRAM allows a new calculation procedure to be implemented. The system allows specification of the operational formats, the results for the patient file, numeric and English result types, and combinations of LINC and floating point code. It is assumed that new calculations will be implemented at the SUB PROGRAM level of programming.

6.6 FLOW DIAGRAM

A rough flow diagram for a typical calculation is shown below. The precalculation can be used to enter a set of standard test results which can be referred to by a series of calculations applying to different patients.



6.7 PROGRAMMING

Refer to a typical calculation listing to see the general layout.

The pointer table at the beginning of the program is standard for all calculations (except for the numbers of input data).

Some of the program sections need not appear in a particular program - e.g. precalc entry and output formats, final report format and English result text. When these are omitted, the corresponding symbol tags for these sections do not appear in the body of the program, and error messages are displayed during assembly. All the other standard program sections should be included (with dummy entries if necessary) to ensure correct operation of the calculation control programs.

"Program patches 1 and 2" are executed immediately before and after the main calculation, respectively. LINC coding can be inserted in these places if desired (after the STC instruction).

The precalculation and main calculation can include mixed floating-point and LINC coding - e.g.,

```
(LINC instructions)

    LIF 5
    JMP 2Ø
(floating point instructions)
(last instruction 2 digits only)
(LINC instructions)

    LIF 5
    JMP 2Ø
(floating point instructions)
(last instruction 2 digits only)
(LINC instructions)

    LIF 4
(return jump instruction)
```

Care needs to be taken when writing the floating point coding - so that, for example, integer-operand instructions are not used with floating point operands, and vice versa.

In addition to writing the calculation sub-programs, the programmer must: -

1. Write the programs on a startup tape using the System Build option (the usual directory code names for calculation programs are C1 to C9).
2. Alter, assemble and re-write the CATEXT (CT) program to include a directory of calculation codes (four letters and/or numbers - no blanks) and the corresponding two-character codes used on the startup tape.
3. Include in TABDATA the calculations which involve filing of results in patient files. The calculation four-character code name is used as the code name of a calculation battery, and items within the battery are results or data for filing. The four-character codes for the latter must also appear in the calculation sub-program under "results for filing".

There are at present some errors in the calculation control programs. Consequently: -

1. Accuracy in most calculations is limited to about three significant decimal figures.
2. The single-line formats (e.g. for data entry) are limited to about 48 characters.

6.7.1 Floating Point Subroutine

The floating point package is a combination of the 1966 version of the Stanford Floating Point package (DFPF66) and a special driver package by Arthur A. Eggert. The package occupies LINC Bank 5 and executes the code in the calling bank. The calling bank is assumed to be bank 4, but can be changed per instructions in the LBCM-5FP manuscript. (See also the subroutines B1 and B2 in CAPROG.)

6.7.2 Floating Point Instructions

Floating point is used by inserting a series of two-word instructions. The first word is the operand and the second the operation. The calling sequence is:

```
LIF 5                       / LMB TO BANK 5
JMP 2Ø                      / ENTER SUBROUTINE
operand 1
operation 1
operand 2
operation 2
operand 3
operation 3
    .
    .
    .
operand n
operation n
return from subroutine
```

6.7.3 Operand

The operand specifies an address in three forms:

1) Direct Address: 5ØØ, for example, is interpreted as the address of the number to be used in the operation. If the number is an integer, then only location 5ØØ is used. If the number is in floating point format, locations 5ØØ, 5Ø1 and 5Ø2 would be used.

2) Indirect Address: Setting the 4ØØØ bit refers to an address for the number required. For example, 45ØØ means the address of the required number is in location 5ØØ. Any location may be used as an indirect address (not restricted to the LINC index registers).

3) Zero: A zero operand refers to the floating point accumulator.

6.7.4 Operation Codes

A complete list of operation codes is given below. The codes below 4Ø_8 are the standard codes for the Stanford package. The operation codes 4Ø_8 and above are special transcendental functions. All operations, including transcendental functions, use less than 0.5 seconds. The Floating Point driver will do a TMX after any lengthy transcendental operation. Note that the transcendental operations do not use the address field (operand) since X is assumed to be in the Floating Point Accumulator (FAC).

The 4ØØØ bit set in an operation code means another code follows. If it is not set the subroutine returns to execute LINC code. For example:

```
LIF 5
JMP 2Ø
operand
4ØØØ+operation code
operand
4ØØØ+operation code
operand
operation code
return here from Floating Point
```

Op Code	Operation
0	compute square root of the value in operand. Leave result in FAC
1	clear FAC and add operand to FAC (LOAD operand.
2	add operand to FAC
3	complement operand and leave result in FAC
4	multiply operand by FAC; result in FAC
5	divide FAC by operand; result in FAC
6	divide the operand by FAC; result in FAC
7	add an integer operand to FAC; result in FAC
10	multiply an integer operand by FAC; result in FAC
11	divide the FAC by an integer operand
12	divide an integer operand by the FAC
13	convert FAC to an integer; leave in LINC accumulator
14	convert an integer to a floating point word; leave in FAC
15	clear FAC and operand
16	compare size of operand with FAC; larger left in FAC
17	compare size of operand with FAC; smaller left in FAC
20	check the sign of the operand; depending on whether it is positive, negative or zero, leave in the LINC accumulator+1, -1 or zero.
21	add FAC to operand and store in operand, i.e., add to memory
22	subtract operand from FAC; leave result in FAC
23	store FAC in operand; also, leave in FAC

24	set sign of operand positive; leave in FAC
25	set sign of operand minus; leave in FAC
26	subtract integer operand from FAC. Result in FAC
40	X^n, raise value X to integer power n; n is in operand field, result in FAC
41	X^y, raise value X to floating point power y. Operand is address of floating point number y. Result in FAC.
42	e^x, raise e to power of x; x is in FAC; result left in FAC; operand field not used.
43	$\text{Log}_{1\emptyset}X$, take log of X to base 10, X in FAC; result left in FAC; operand not used.
44	$\text{Log}_{y}X$, take log of X to base y. X in FAC; operand is address of floating point y; result left in FAC

(For all the following operations, X is put in FAC before executing the code. Result is left in the FAC. Operand field is not used, but location must be allocated. User may use operand location.)

45	ln X
46	sin X
47	cos X
50	Tan X
51	arcsin X
52	arccos X
53	arctan X
54	hypersin X
55	hypercos X
56	hypertan X

Certain limitations concerning the transcendental functions (codes 40 and over) should be taken into account. Except for the X^n routine, whose speed is proportional to [n], the routines are relatively slow. This is because they require summations of series, and although the summations have been modified to cut down on the number of terms, it should not be assumed that they are instant. The longest routines are tan X and all the arcfunctions. Moreover, due to the extensive modification necessary to get the arcfunctions to run, they are inaccurate to the extent of 1 part in 100,000 for especially bad cases, though they are usually better. Other functions are good to at least one part in a million, usually 1 part in 10 million or more.

The transcendental function package also contains useful floating point constants, which can be accessed by the user.

Constant	Address in Bank 5
$\pi/2$	751
π	767
ln2	746
e	772
10	744

All angles are expressed in radians. Remember that:

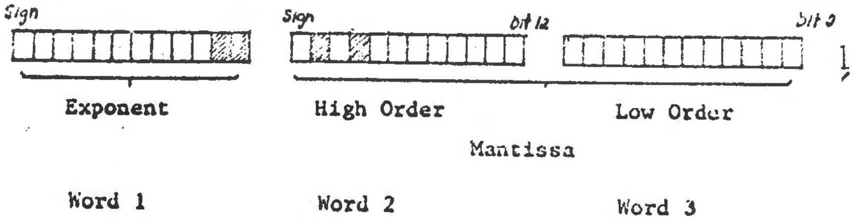
$$\text{degrees} = \frac{\pi \text{ x radians}}{18\emptyset}$$

6.7.5 Floating Point Format

The following explanation of the floating point word is reproduced from the Stanford description of floating point.

> The use of double precision floating point arithmetic seems essential if the LINC is to serve as a statistical processor. In using this type of arithmetic, the programmer trades speed and space for ease in dealing with large numbers. Programs which are extremely laborious to write and debug may become rather trivial using these routines.

A. A form of double precision floating point number:



B. Floating an integer involves shifting the number right across the binary points until it is a fraction, and then counting the number of shifts to make up the exponent. In the above case, 5 = 101.000 in binary. Three shifts right produce 000.101. Since the binary point is always located between bits 23 and 22, the floating point number contains 101 in bits 22, 21 and 20. The exponent equals 3. Another way of representing this binary number is .101 x 2^3.

C. Normalized floating point numbers always contain their most significant bit in bit 22. The above number could be represented in an unnormalized mode, such as .010 x 2^4. But it is never represented this way in the floating point routines, since this would waste precision out at the right end. In its normalized mode, .101 x 2^3, the number contains 23 bits of precision. This corresponds to more than 7 decimal digits.

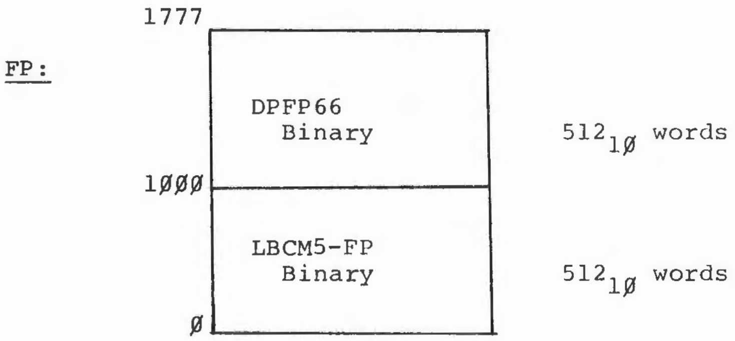
D. Fixing a floating point number is the reverse of the float. It is shifted left across the binary point until the exponent equals zero. The fractional part remaining, if any, is either discarded or used for rounding off the integer.

E. Negative numbers are represented as the one's complement of positive numbers as in standard LINC integers. The mantissa (high and low order words) is merely complemented. Note that there is no sign bit in the low order word.

F. Negative exponents indicate that the number is less than one and has been shifted left until it is normalized. The sign of the exponent should not be confused with the sign of the mantissa. .101 x 2^{-3} is no more a negative number than is 5 x 10^{-3}. In the former the minus exponent indicates that if the number were fixed it would be .000101. The floating point routines would give a zero if requested to fix this number.

6.8 SPECIAL ASSEMBLY INSTRUCTIONS

The driver package (LBCM5-FP)) manuscript is assembled under DIAL for the first 512_{10} words (two blocks) of Bank 5. The two blocks of the DPFP66 (Stanford Floating Point) are available only as binary at present. The package is stored on the start up tape as 4 blocks under the name FP. The first two blocks are LBCM5-FP and the second two are DPFP66.



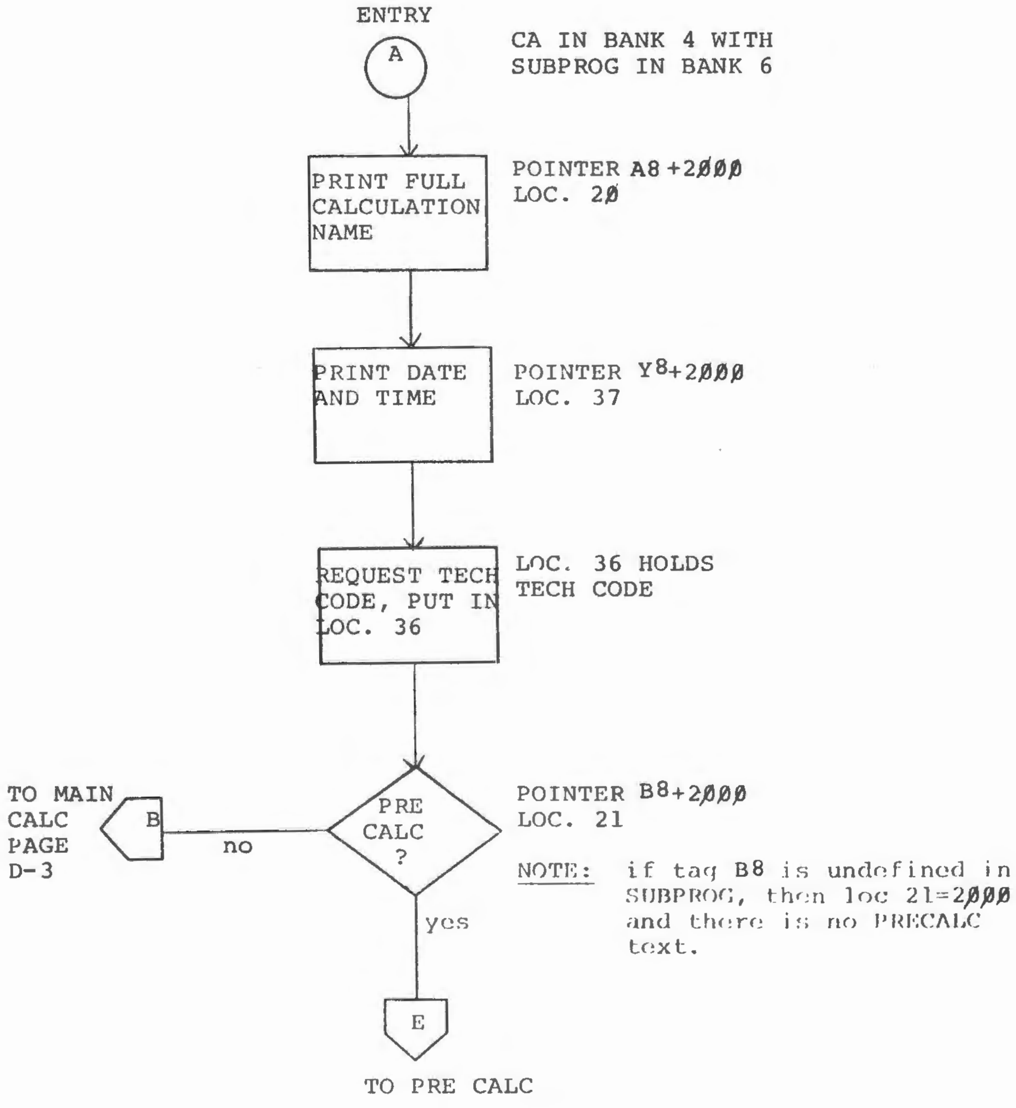
The rest of the overlays are assembled and stored in the program file as follows:

Startup Tape File Name	DIAL File Name	No. Binary Blocks
CA	CAPROG	5
CF	CALCFILE	5
CT	CATEXT	4

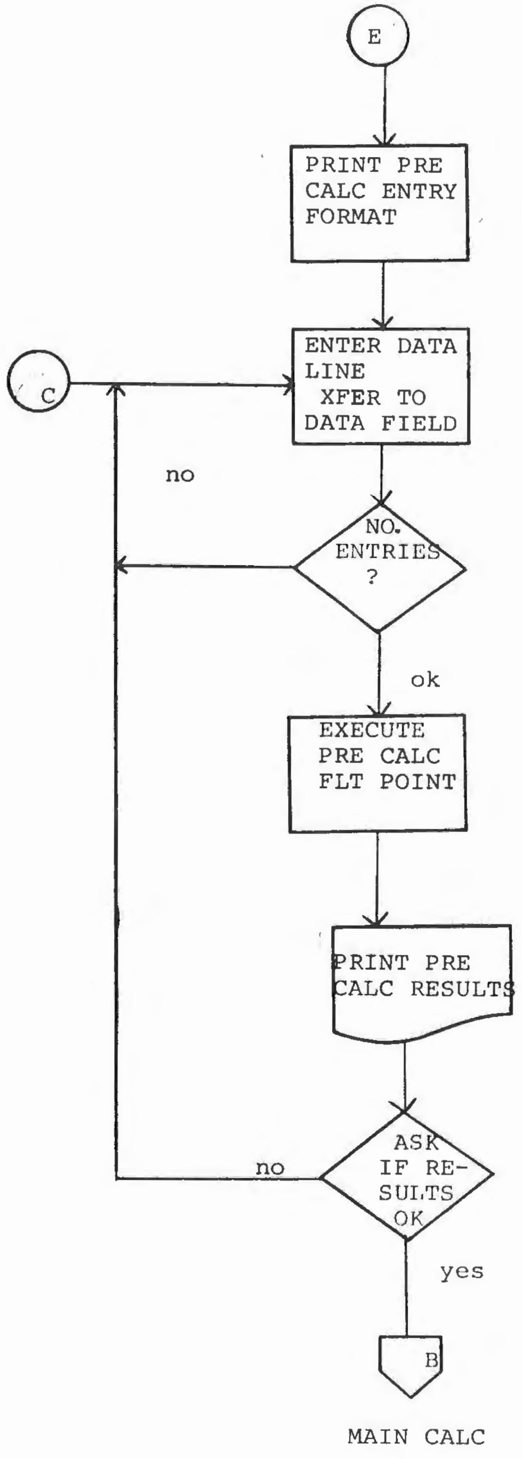
There are equate statements at the end of CAPROG which reference addresses in CATEXT.

6.9 FLOW CHARTS FOR MANUAL CALCULATIONS

CALCULATION SUBPROGRAM
FLOW DIAGRAM


ENTRY
A
CA IN BANK 4 WITH SUBPROG IN BANK 6
PRINT FULL CALCULATION NAME
POINTER A8 +2ØØØ
LOC. 2Ø
PRINT DATE AND TIME
POINTER Y8+2ØØØ
LOC. 37
REQUEST TECH CODE, PUT IN LOC. 36
LOC. 36 HOLDS TECH CODE
PRE CALC ?
POINTER B8+2ØØØ
LOC. 21
NOTE: if tag B8 is undefined in SUBPROG, then loc 21=2ØØØ and there is no PRECALC text.
no
B
TO MAIN CALC PAGE D-3
yes
E
TO PRE CALC

PRECALCULATION


E
PRINT PRE CALC ENTRY FORMAT
C
ENTER DATA LINE XFER TO DATA FIELD
no
NO. ENTRIES ?
ok
EXECUTE PRE CALC FLT POINT
PRINT PRE CALC RESULTS
no
ASK IF RE-SULTS OK
yes
B
MAIN CALC

POINTER B8 +2ØØØ
LOC. 21

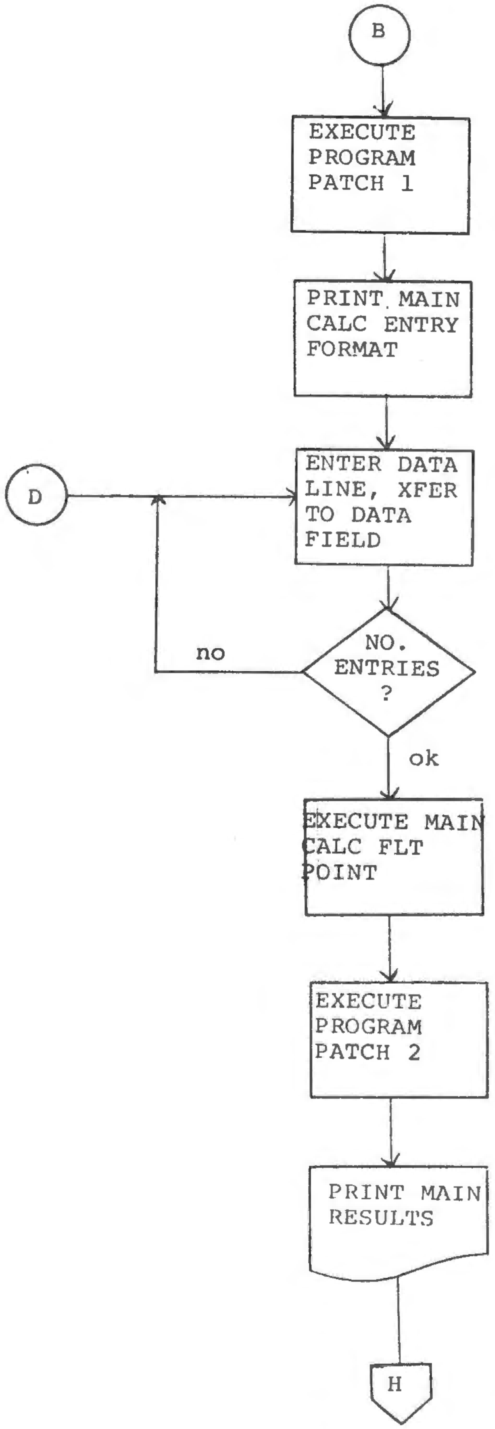
POINTER I8 +2ØØØ
LOC. 3Ø
(also see section on data format)

Check Loc. 34 for
CORRECT NO. ENTRIES

EXECUTE JMP BC
at Loc. 22

POINTER D8 +2ØØØ
Loc. 23

MAIN CALCULATION


B
EXECUTE PROGRAM PATCH 1
PRINT MAIN CALC ENTRY FORMAT
D
ENTER DATA LINE, XFER TO DATA FIELD
no
NO. ENTRIES ?
ok
EXECUTE MAIN CALC FLT POINT
EXECUTE PROGRAM PATCH 2
PRINT MAIN RESULTS
H

execute JMP K8
loc. 32

pointer E8+2ØØØ
loc. 24

pointer I8+2ØØØ
loc. 3Ø
(see section on data format)

check loc. 35 for
correct no. entries

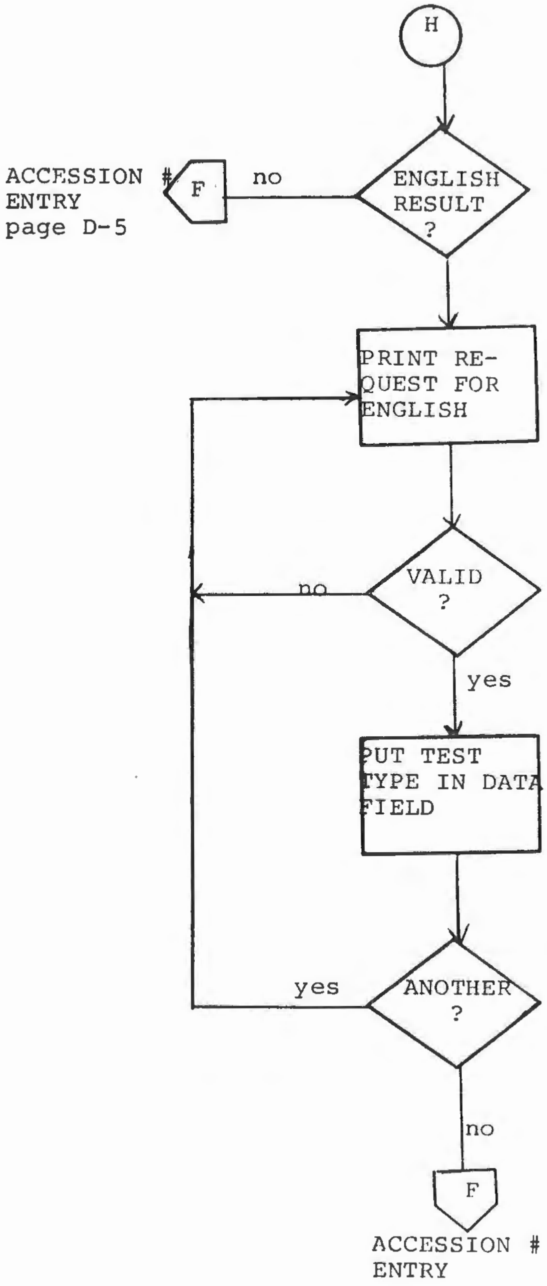
execute JMP F8
loc. 25

execute JMP L8
loc. 33

pointer C8 +2ØØØ
loc. 26

TO ENGLISH RESULT ENTRY

ENGLISH RESULT ENTRY


H
ENGLISH RESULT ?
no
F
ACCESSION # ENTRY page D-5
PRINT RE-QUEST FOR ENGLISH
VALID ?
no
yes
PUT TEST TYPE IN DATA FIELD
ANOTHER ?
yes
no
F
ACCESSION # ENTRY

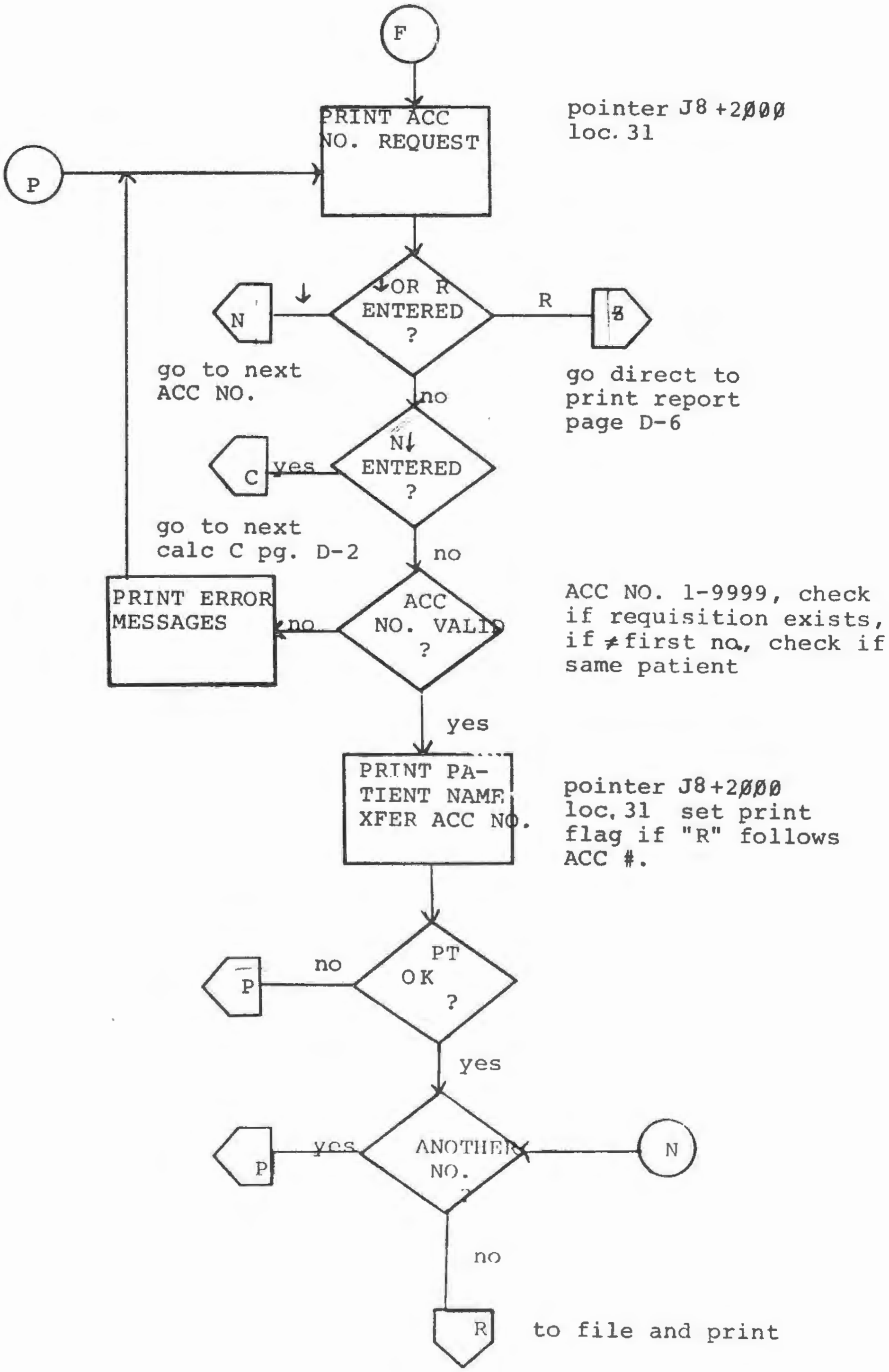
if no English, tag R8
will be undefined and
loc. 42 will =2ØØØ

pointer R8 +2ØØØ
loc. 42.

look up 1 to 4 char
entry in file 44.

pointer I8+2ØØØ
loc. 3Ø

ACCESSION NUMBER ENTRY



pointer J8+2000
loc. 31

go to next
ACC NO.

go direct to
print report
page D-6

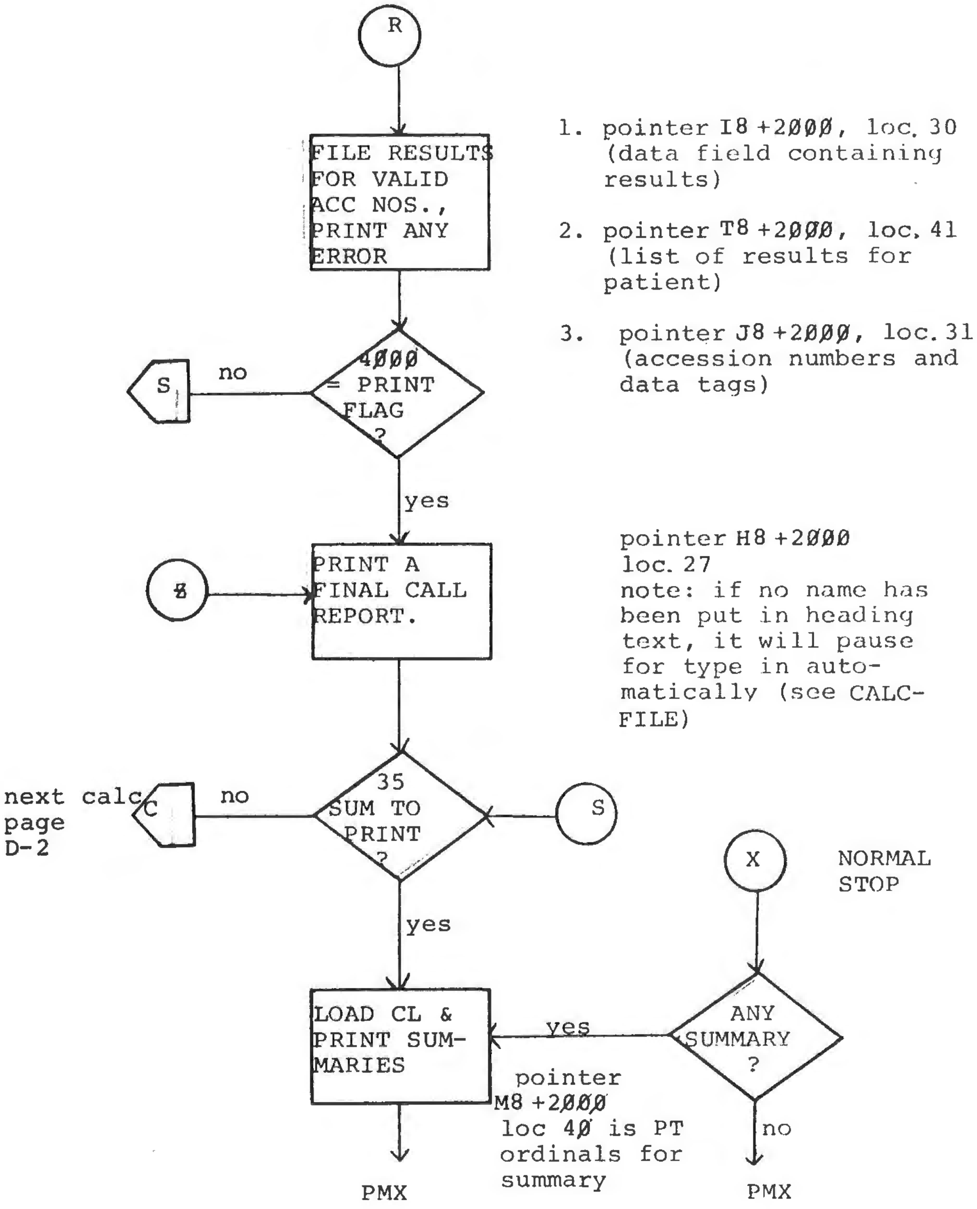
go to next
calc C pg. D-2

ACC NO. 1-9999, check
if requisition exists,
if ≠ first no., check if
same patient

pointer J8+2000
loc. 31 set print
flag if "R" follows
ACC #.

to file and print

FILE RESULTS AND PRINT REPORT



1. pointer I8 +2ØØØ, loc. 30 (data field containing results)

2. pointer T8 +2ØØØ, loc. 41 (list of results for patient)

3. pointer J8 +2ØØØ, loc. 31 (accession numbers and data tags)

pointer H8 +2ØØØ
loc. 27
note: if no name has been put in heading text, it will pause for type in automatically (see CALC-FILE)

pointer M8 +2ØØØ
loc 4Ø is PT ordinals for summary

CHAPTER 7

ACCESSION NUMBER ENTRY

The Accession Number Entry (AC) program enables the technician to enter accession numbers as well as to obtain the computer calculated results of a given test from the channel storage area, edit them, and store the final results in the appropriate slot in the patient's test data file. The program utilizes a conversational format with the technician which is designed to promote a complete exchange of information in the minimum amount of time.

7.1 INPUT/OUTPUT

Only terminal and disk are used. The terminal is used to obtain input from the user, type replies for the user, and to type reports on request. Disk files are used as follows:

File	Name	Usage
00	PROGRAM FILE	read only
20	PATIENT NAME FILE	read only
22	N.S./DR. FILE	read only
27	REQUISITION INDEX	read and write
30	TEST DATA	read and write
32	CONTROL/SCHEDULE BLOCKS	read only
33	CHANNEL STORAGE	read and write
35	TEST TYPE CODES	read only
36	TEST PARAMETERS	read only
40	TEST NAMES	read only
41	ENGLISH RESULT FILE	read only
44	ENGLISH RESULT CODES	read only

7.2 FUNCTIONAL DESCRIPTION

AC consists of three phases: (1) initial conversation, (2) edit conversation, (3) finalization.

7.3 INITIAL CONVERSATION

Call AC. The program responds:

```
ACC # ENTRY hhmm HOURS  mm/dd/yy
TECH CODE *
```

The user responds with a number 1-63. He may type STOP at any time. He is then asked:

```
AUTO. NAME(S) *
```

The user responds with the names of the automatic instruments which he wishes to edit. If more than one, the names must be separated by commas and the instruments must have been collated when set up. If he requests more than one instrument or if the requested instrument has more than one result, the user is asked:

```
CONSIST. DEL.  *
```

The user may type in one or more result codes (separated by commas) for those results which are not to be filed. The user is then informed and queried in one of two ways:

```
HIGHEST CUP EDITED IS NNN.  START AT CUP *
```

or

```
HIGHEST EDITED IS PLATE NNN, CUP NNN.  START AT
(PLATE, CUP) *
```

The user responds with the cup number or the plate number, comma, cup number. The program then enters the edit phase.

7.4 EDIT CONVERSATION

AC searches for the next filable result or control, automatically skipping standards and special cups. When one is found, it types either:

```
XX S *
```

for a sample, or:

```
XX C *
```

for a control. The user then may type any of the following:

/↵	Move to the next cup.
/XX↵	Move to cup XX.
AAAA↵	where AAAA is a test name. Move to that result.
↵	Move the next result.
-AAAA↵	where AAAA is an English modifier. Modify the whole cup.[1]
AAAA↵	where AAAA is an English result. Replace the whole cup.[1]
XX.X↵	where XX.X is a number. Replace the whole cup <u>only if</u> there is only one result per cup.
R↵	Print a formatted report.
#XXXX ↵	where Ø≤XXXX≤9999. Assign an accession number to this cup.[1] (replies "FILED"). FOR CONTROLS ONLY: accept the control (replies "ACCEPTED")

After moving to a result within the cup, the following options (listed above) work differently:

-AAAA↵	where AAAA is an English modifier. Modify this result only.[1]
AAAA↵	where AAAA is an English result. Replace this result only.[1]
XX.X↵	where XX.X is a number. Replace this result only.

In addition, after moving to a result within the cup, the following option is available

?	Delete a result from channel storage and do not file it in the patient file

The above operations may be combined on one line by separating the elements with a comma. The line is scanned from left to right. Anything to the right of:

```
    1)  an error
or: 2)  /
or: 3)  /XX
```

<u>is ignored.</u>

[1] Not applicable to controls.

7.5 FINALIZATION

When the user types STOP or a terminal error has occurred (see below) the program terminates by typing:

```
          AC DONE
```

7.6 ERROR MESSAGES

Error messages arise from two situations: user errors, and system or hardware errors. When an error occurs, one of the messages below is typed out.

Regardless of the cause, errors can be classified as:

a) Terminal - Terminal errors cause AC to exit as if STOP had been typed.

b) Non-terminal - Non-terminal errors do not cause AC to stop. The program continues in the usual way.

User Errors (all user errors are non-terminal)

USE DIGITS 0-9	A non-digit was found in a numeric field.
CODE 0-63 ONLY	A number greater than 63 was given for the TECH CODE.
XXXX NOT SETUP	The specified instrument, XXXX, has not been SETUP on-line.
XXXX BEING EDITED	The specified instrument, XXXX, is being edited from another terminal
NOT A COLLATED SET	The instrument names given were not SETUP together (COLLATED) and hence cannot be edited together.
ALL RESULTS MAY NOT BE DELETED	The user has requested to consistently delete all results.
NO CODE NAME (XXXX). RETYPE LINE.	The user has specified a non-existent code name. Only this item and ones to the right must be retyped.

CODE NAME (XXXX) TOO LONG. RETYPE LINE

The user specified a code name of more than 4 characters. The first 4 are shown in the message. The entire line must be retyped.

CODE NAME (XXXX) TOO LONG

The user specified a code name of more than 4 characters. The first 4 are shown in the message. Only this item and the ones to the right must be retyped.

TOO MANY CUPS

The user has specified a cup number greater than 2047.

REQUESTED CUP BEYOND LAST DATA

The cup specified by the user is not yet available. This message occurs when a specific request is too high and when the user has reached the end of the available data.

XXXX CANNOT BE A MODIFIER

The indicated code name can be used as an English replacement only, not as a modifier.

XXXX CANNOT BE CHANGED

The user has attempted to enter a new result for XXXX which either has already been filed or is premarked as "do not file." This error occurs when trying to give an English result to a control.

NUMBER CANNOT REPLACE WHOLE CUP

The user has typed a numeric replacement at the cup level for a cup with more than one result.

MULTIPLE DEC. POINTS FOR XXXX

Two decimal points were typed.

RESULT FOR XXXX TOO BIG

The numeric replacement given for result XXXX was greater than 2047000.

RESULT FOR XXXX TOO SMALL

The numeric replacement given for result XXXX was less than .0000001.

ENTIRE CUP MAY NOT BE DELETED

The user has typed the delete character (?) at the cup level for a cup with more than one result. If a cup is not desired, simply move to the next cup.

ACC. # TOO BIG

The user has typed an accession number larger than 9999.

TYPE ONLY # FOR ACCEPTING CONTROLS

The user typed more than just "#" to accept a control. This ofter happens when an accession number is typed by mistake.

NON-DIGIT IN ACC. #

The user typed a character other than 0-9 in the accession number.

NO SUCH ACC. #

The accession number given by the user is not active at this time. That is, there are no incomplete results identified by the specific accession number.

PREVIOUS ACC. # GIVEN

The user has already specified an accession number for this cup and verified the name. No other number may be given.

NO REQ. FOR: XXXX, XXXX, . . .

The specified accession number did not include requests for the listed result types.

PREV. DATA FOR: XXXX, XXXX, . . .

The specified accession number included a request for the listed result types, but the data has already been entered into the patient file.

<u>System Errors:</u> Non-Terminal

WAITING FOR FILE XX

The specified disk file is not immediately available, thus the program waits until it becomes available. An error in the system may be indicated if the program does not proceed beyond this point within a reasonable time.

SKIPPING CUP XXXX: DISK ERROR FILE 33

The indicated cup number (XXXX) is being skipped because of a disk error.

DISK ERROR FILE XX

A disk read error occurred while reading file XX during a non-critical operation. This error may be non-terminal in the files shown below. The following values of XX have effects as follows:

XX	may mean error in:
20	Patient Name printout
22	Nursing Station printout
36	Units Name printout
40	Test Name printout
41	English Result printout
44	Test Code Name printout

NO FILE NAME FOR XXXX

The code name XXXX is used in the on-line system as the name of a result but the patient filing system has no such file name. Hence, results for XXXX are not filed in the patient file when an accession number is given. This message usually indicates a logical error in the formation of the Control Blocks or the Test Type Code Table.

System Errors: Terminal

SUMMARY TABLE FULL

This message does not indicate a real error condition. However, the list of patients on whom all tests are complete has filled the available space. AC will quit as if the user had typed STOP. After summaries are printed, the user may again call AC and continue.

DISK ERROR FILE XX

A disk error occurred while reading or writing file XX during a critical operation. This error may be terminal in the files shown below. The following are the possible files XX:

XX	Name
20	Patient Name
27	Requisition Index
30	Patient Test Data
32	Control/Schedule Blocks
33	Channel Storage
35	Test Type Codes
36	Test Parameters
42	Battery Table
44	English Result Codes

NO PROGRAM A1 AND/OR PF

The program A1 and/or PF could not be loaded from the disk. AC will quit as if STOP had been typed. See Note 1.

NO PROGRAM AX

The program AX (where X is a digit 2 to 5) could not be loaded from the disk. AC quits as though STOP had been typed. See Note 1.

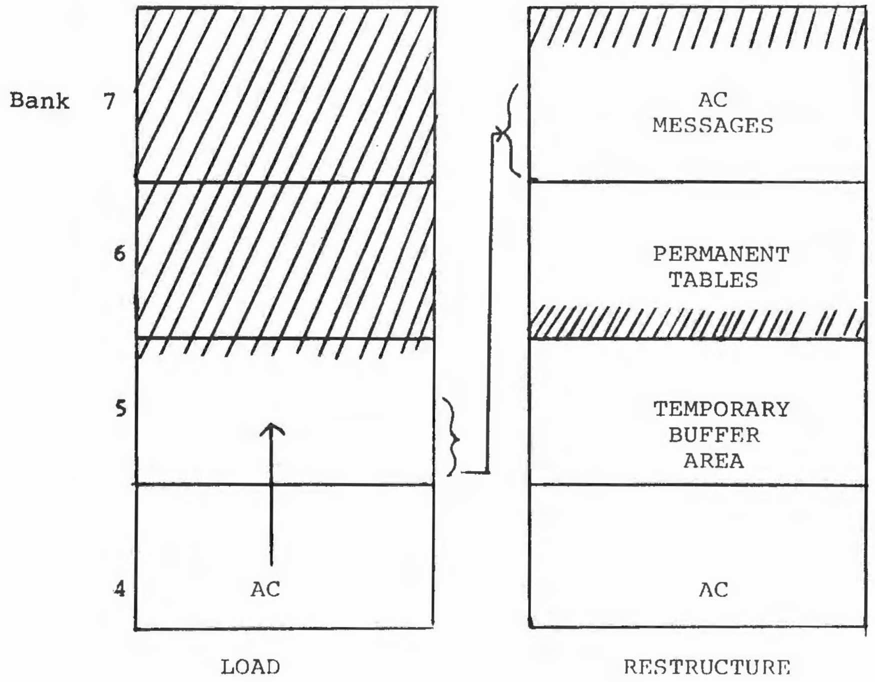
Note 1: A "NO PROGRAM" message may be caused by one of two conditions.

a) The indicated program may not be in the program file. If so, software personnel should be notified.

b) The indicated program may actually be in the program file, but it could not be read without error. If so, notify hardware maintenance personnel.

7.7 PROGRAM STRUCTURE

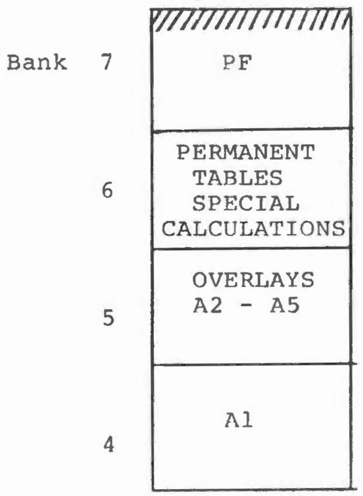
Calling AC causes Banks 4 and 5 to be loaded. Bank 5 is transferred to Bank 7.



INITIAL CONVERSATION

During the initial conversation quarters 2 and 3 of Bank 6 are laid out as permanent buffer and table space. A complete description of this area can be found in the attached diagram.

At the end of the initial conversation two programs are loaded: A1, the resident edit control, into Bank 4 and PF, the data filer, into QØ and 1 of Bank 7. All further requests from the terminal are handled by A1 and usually cause an overlay (A2-A5) into Bank 5. The special calculation routines are loaded into QØ and 1 of Bank 6 if needed.



EDIT PHASE

All input in the edit phase (except verification of patient name) uses the input buffer in Bank 6. A1 starts scanning the buffer to determine what the user requested. If an overlay is needed, it is loaded. After executing the overlay, A1 continues to scan the buffer until the last item has been processed. Listed below are the functions of each section A1 through A5.

A1 — Read cup from channel storage, load overlays, decode input buffer, handle requests /, /XX, and STOP.

A2 — Print formatted report requested by R. If no accession # has been given the user is asked for nursing station and name. The user's reply is simply ignored, not read.

A3 — Modify or replace a cup or result. All inputs of the type AAAA or -AAAA cause this routine to be loaded. If the input is AAAA but not an English result, the move to result routine (A4) is loaded on the assumption that AAAA may be a result name. Any input of the type XXX.X or ? also cause A3 to be loaded.

A4 — Move to a result. Any input of AAAA which is not an English result causes A4 to be loaded. Also an input of just ↵ (carriage return) causes A4 to be loaded. A4 sets up a line to type consisting of the test name, the result, any modification, etc.

A5 — File results and controls. Any input starting with # causes A5 to be loaded.

7.8 ASSEMBLY PROCEDURE

Assemble AC,A1,A2,A3,A4, and PF under DIAL and store on the start up tape in the following manner:

Source Name	Number of Assembled Blocks	Name on Start up Tape	Blocks to be Stored (1st block, number of blocks)
AC	11	AC	1,7
A1	5	A1	1,4
A2	5	A2	1,4
A3	4	A3	1,3
A4	3	A4	1,2
A5	5	A5	1,4
PF	3	PF	1,2

PERMANENT BUFFER $ TABLES FOR AC

BANK 6

LOCATION		
1000	BINARY TEST TYPE	A9, CONTROL BLOCK TRAILERS
	4 CHAR TEST CODE NAME	RESULT #1
	LO NORMAL	
	HI NORMAL	
	BINARY TEST TYPE	
	4 CHAR TEST CODE NAME	RESULT #2
	LO NORMAL	
	HI NORMAL	
	⋮	
	LO NORMAL	RESULT #15_{10}
	HI NORMAL	

1113	132_{10}LOCS.	B9, CHANNEL STATUS BLOCKS

.ALLOWANCE FOR 8 COLLATED CHANNEL

1317	0123	RESULT # 1
	0123	RESULT # 2
	⋮	⋮
	0123	RESULT # 15.

C9, RESULT TABLE

.BIT 0=1 IF FIRST RESULT ON CHANNEL
.BIT 1=1 IF THIS RESULT NOT REQUESTED
.BIT 2=1 IF THIS RESULT IS A CONSISTENT DELETION
.BIT 3=1 IF DELETE THIS RESULT THIS CHANNEL

1336	▨	CHAR. 1
	⋮	CHAR. 2
	▨	CHAR.73

E9, TTY INPUT BUFFER

.UNPACK ROUTINE PUTS 8 BITS PER WORD, ADDS A CR. JUST IN CASE, SQUEEZES OUT 0,100.200.240 AND 300, DESTROYS STATUS WORD, LOOKS FOR STOP).

LOCATION			
1447	CHAR 49 | CHAR 1 CHAR 2 72 | CHAR 48	F9,	TTY OUTPUT BUFFER
1527	RESULT=1 IN 5-WORD CHANNEL STORAGE FORMAT RESULT # 15	G9,	CHANNEL STORAGE BUFFER A 75_{10} LOCATIONS
1642	RESULT #1 IN 5-WORD CHANNEL STORAGE FORMAT RESULT #15	H9,	CHANNEL STORAGE BUFFER B 75_{10} LOCATIONS
1755	EOP-3 | FILE 33 | 1 9B DISK ADDRESS WORD COUNT-1	J9,	READ/WRITE CH. STATUS
1762	▨ | $Ø-63_{10}$	K9,	TECH CODE
	$1-2047_{10}$	L9,	NEXT CUP TO EDIT
	-1 to -15_{10}	M9,	-#RESULTS/COLLATED SET
	-1to-8	N9,	-#CHANNELS/COLLATED SET
	▨ | $Ø-255_{10}$	P9,	PLATE SIZE
	$Ø-2047_{10}$	Q9,	HIGHEST CUP EDITED
	Ø 0A1-15	R9,	CURRENT RESULT W/IN CUP
	EOP3 | FILE 33 | 1	U9,	READ/WRITE 0-1. STORAGE A OR B
	9G OR 9G		
	DISK ADDRESS		
	WORD COUNT -1		
1776	NULL=7777	V9,	PATIENT ORDINAL THIS CUP

CHAPTER 8

PATIENT DATA FILER

The Patient Data Filer (DATA-PF) program is a pair of LINC subroutines to store results into the patient file. The subroutines are written to occupy Quarters 0-2 of Bank 7. The data to be filed may be anywhere in banks 4-7. The subroutines are filed under PF in the program file. They are currently used by ACcessions number entry and manual CAlculations.

I/O Used:

Disk - as follows

File 20	Patient Names	Read only
File 27	Requisition Index	Read/Write
File 30	Patient Test Data	Read/Write
File 36	Test Type Parameters	Read only
File 42	Battery Table	Read only

8.1 RETRIEVE PATIENT GIVEN AN ACCESSION NUMBER

The first subroutine determines, given an accession number, the patient ordinal and name. The calling sequence is shown below.

```
LDA I
   LIF X   [where X = current bank]

LIF 7
JMP 2Ø
   ACCESSION NUMBER, HI (Ø-2$_8$)
   ACCESSION NUMBER, LO (Ø-7777$_8$)
   TECHNICIAN CODE      (Ø-77$_8$)
   DISK ERROR RETURN
   ACC. # ERROR RETURN
   NORMAL RETURN
```

If a disk error occurs, the subroutine returns at the indicated location. The accumulator contains the file number where the error occurred. The calling program prints an error message and exits.

If no requisition has been entered for the given accession number, the ACC. # ERROR RETURN will be taken.

If the requisition does exist, the NORMAL RETURN is executed. After a NORMAL RETURN the patient ordinal is found in the accumulator. The patient name (20_{10} characters in stripped ASCII) is stored starting at location 1000 in Bank 7, packed two to a word.

8.2 FILE DATA FOR PREVIOUS GIVEN ACCESSION NUMBER

To file data for a given accession number, use the following calling sequence. This call must be given after the call described in paragraph 8.1 and must originate in the same bank.

The address of the data as given in the calling sequence refers to the upper 4K of memory (field 1). The address is independent of the LMB and UMB indicators. All data to be filed must be within a bank, i.e., it can not lap over bank boundaries. The data to be filed has a three word format:

WORD N	Result type (0-776_8)
WORD N+1	Same as channel storage word 4
WORD N+2	Same as channel storage word 5

The calling sequence is:

```
  OPEN WRITE FILE 27
  OPEN WRITE FILE 3Ø

 LIF 7
 JMP 23
  ADDRESS OF FIRST DATA WORD (Ø-777$_8$)
  NUMBER OF RESULTS TO FILE (1-525$_8$)
  DISK ERROR RETURN
  NORMAL RETURN
```

The subroutine closes both files 27 and 3Ø before either return is executed, thus the calling program must re-open the disk files for each filing operation.

If any disk error occurs, the subroutine returns as indicated. The accumulator contains the file number where the error occurred. The calling program prints an error message and exits.

After a NORMAL RETURN, the accumulator <u>must</u> be inspected. If bit 0=1, then at least one result was not filed. If bit 1=1, then the summary printout table is full and "CL" must be loaded.

After a NORMAL RETURN, only those sets of data where bit 0=1 in word N have been filed. In those sets where bit 0=0, the calling program may inspect bit 1 of Word N to determine the cause of failure to file. Bit 1=0 means no result of this type was requested. Bit 1=1 means the result type was requested but the data had already been filed.

Restrictions: The largest battery is assumed to be 32_{10} results or less. Longer batteries cause serious malfunction. The 20_{10} character name read by the first subroutine is destroyed by the second. The summary printout table is big enough for only about 35_{10} patients.

8.3 DESIGN CONSIDERATIONS

The following assumptions have been made in the coding of these subroutines:

End-of-file can occur only at specific places:

1) as the 4th word of the first data block
2) just after the outstanding test count
3) just after a package header
4) just after the last result in a battery or
5) as the continuation pointer (last physical word in a block)

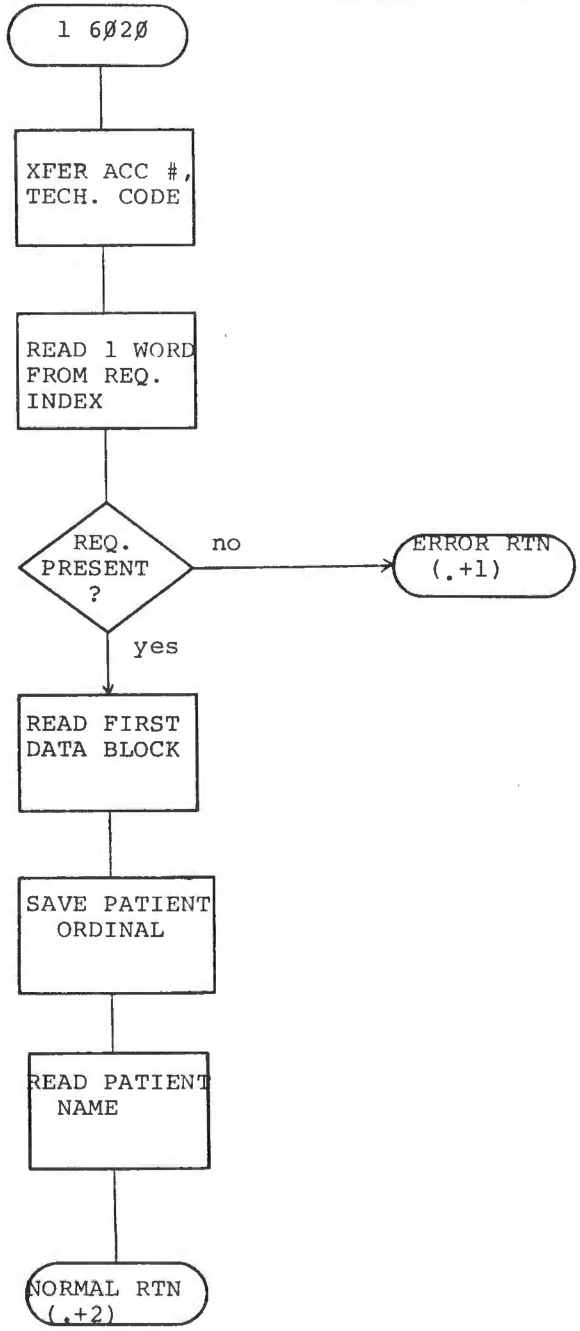
If the 4th word of the first data block is not an end-of-file, then it is automatically assumed to be a day header. A requisition cannot request results for two or more days with a single accession number. This is not to say that an accession number cannot be reused, but that a given active accession number can have results for one day only. If all results for the given accession number are filled, then the appropriate word in the requisition index is reset to 7777. Disk writes are not checked.

8.4 ASSEMBLY INSTRUCTIONS

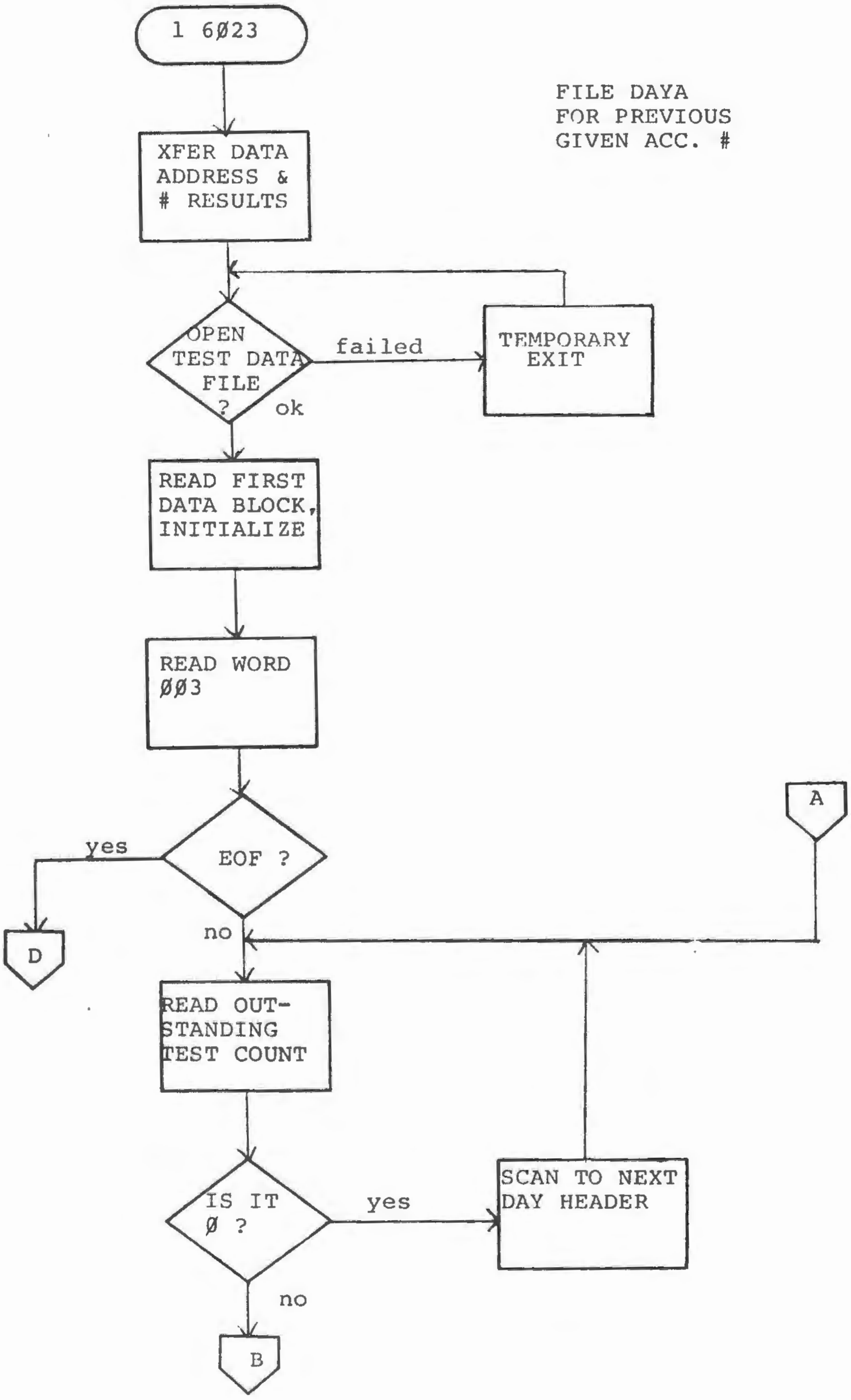
Set the conditional assembly parameter in PF to the size of core for the system being assembled. Patient Data Filer is assembled from a DIAL source named PF and blocks 1,2 of the binary are stored on the start up tape under the name PF.

8.5 FLOW CHARTS FOR PATIENT DATA FILER

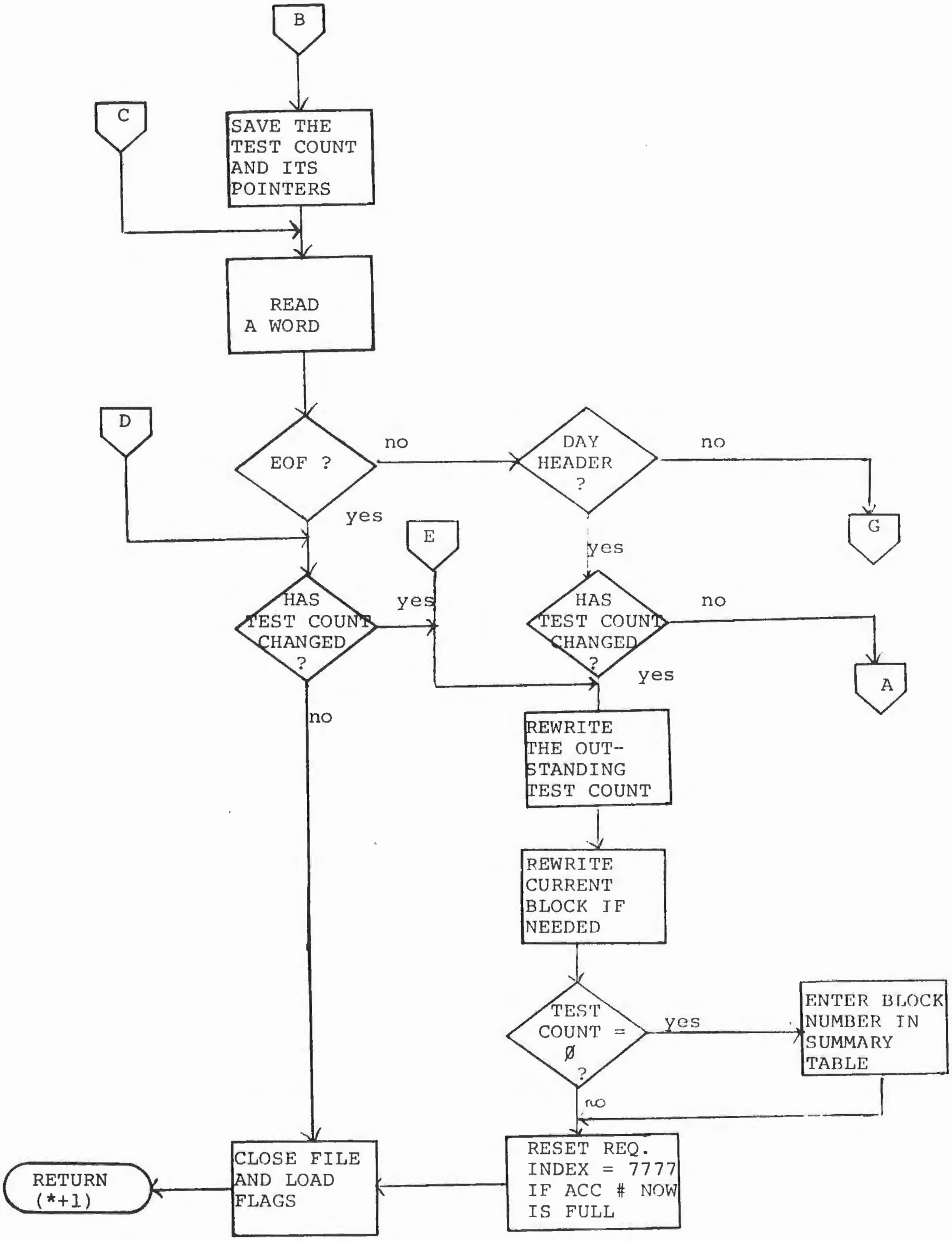
GIVEN ACC #:
RETRIEVE PATIENT ORDINAL



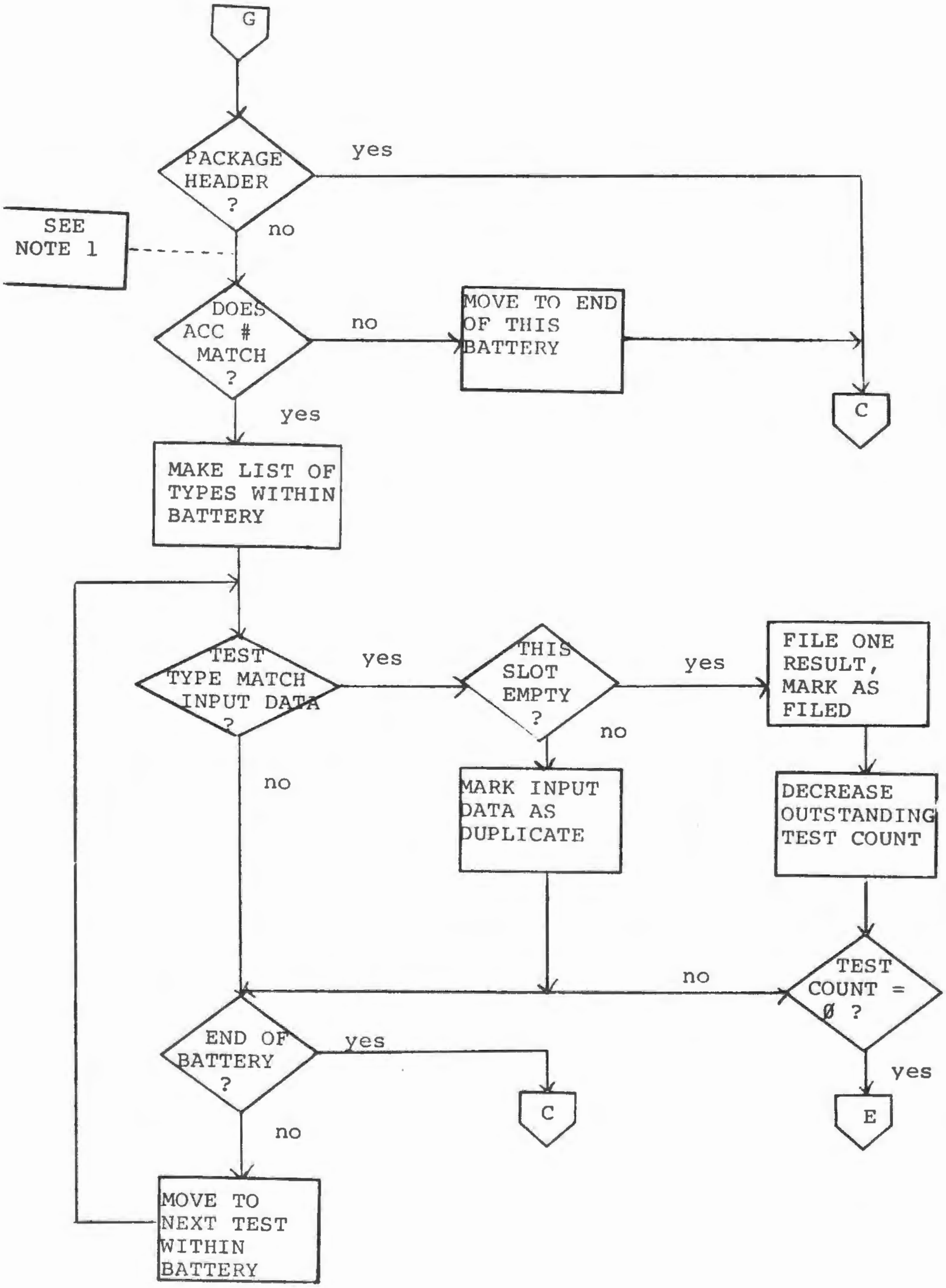
FLOW - 1


1 6Ø23
FILE DAYA
FOR PREVIOUS
GIVEN ACC. #
XFER DATA
ADDRESS &
# RESULTS
OPEN
TEST DATA
FILE
?
failed
ok
TEMPORARY
EXIT
READ FIRST
DATA BLOCK,
INITIALIZE
READ WORD
ØØ3
EOF ?
yes
no
D
A
READ OUT-
STANDING
TEST COUNT
IS IT
Ø ?
yes
no
SCAN TO NEXT
DAY HEADER
B

FLOW - 2


B
C
SAVE THE TEST COUNT AND ITS POINTERS
READ A WORD
D
EOF ?
no
yes
DAY HEADER ?
no
G
E
yes
HAS TEST COUNT CHANGED ?
yes
HAS TEST COUNT CHANGED ?
no
A
yes
no
REWRITE THE OUT-STANDING TEST COUNT
REWRITE CURRENT BLOCK IF NEEDED
TEST COUNT = Ø ?
yes
ENTER BLOCK NUMBER IN SUMMARY TABLE
no
RESET REQ. INDEX = 7777 IF ACC # NOW IS FULL
CLOSE FILE AND LOAD FLAGS
RETURN (*+1)

FLOW - 3


G
PACKAGE HEADER ?
yes
no
SEE NOTE 1
DOES ACC # MATCH ?
no
MOVE TO END OF THIS BATTERY
C
yes
MAKE LIST OF TYPES WITHIN BATTERY
TEST TYPE MATCH INPUT DATA ?
yes
THIS SLOT EMPTY ?
yes
FILE ONE RESULT, MARK AS FILED
no
no
MARK INPUT DATA AS DUPLICATE
DECREASE OUTSTANDING TEST COUNT
no
TEST COUNT = Ø ?
yes
E
END OF BATTERY ?
yes
C
no
MOVE TO NEXT TEST WITHIN BATTERY

NOTE 1: AT THIS POINT WE MUST HAVE, BY ELIMINATION, THE FIRST WORD OF A BATTERY.

FLOW - 4

HOW TO OBTAIN SOFTWARE INFORMATION

Announcements for new and revised software, as well as programming notes, software problems, and documentation corrections are published by Software Information Service in the following newsletters.

Digital Software News for the PDP-8 and PDP-12
Digital Software News for the PDP-11
Digital Software News for 18-bit Computers

These newsletters contain information applicable to software available from Digital's Software Distribution Center. Articles in Digital Software News update the cumulative Software Performance Summary which is contained in each basic kit of system software for new computers. To assure that the monthly Digital Software News is sent to the appropriate software contract at your installation, please check with the Software Specialist or Sales Engineer at your nearest Digital office.

Questions or problems concerning Digital's software should be reported to the Software Specialist. In cases where no Software Specialist is available, please send a Software Performance Report form with details of the problems to:

Digital Equipment Corporation
Software Information Service
Programming Department
Maynard, Massachusetts 01754

These forms, which are provided in the software kit, should be fully filled out and accompanied by Teletype output as well as listings or tapes of the user program to facilitate a complete investigation. An answer will be sent to the individual and appropriate topics of general interest will be printed in the newsletter.

Orders for new and revised software manuals, additional Software Performance Report forms, and software price lists should be directed to the nearest Digital Field office or representative. USA customers may order directly from the Software Distribution Center in Maynard. When ordering, include the code number and a brief description of the software requested.

Digital Equipment Computer Users Society (DECUS) maintains a user library and publishes a catalog of programs as well as the DECUSCOPE magazine for its members and non-members who request it. For further information, please write to:

Digital Equipment Corporation
DECUS
Programming Department
Maynard, Massachusetts 01754

CLINICAL LAB 12
INPUT PROGRAMS
DEC-12-MCL1A-A-D

READER'S COMMENTS

Digital Equipment Corporation maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback -- your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability.

Did you find errors in this manual? If so, specify by page.

How can this manual be improved?

Other comments?

Please state your position. ________________ Date: ________

Name: ________________ Organization: ________________

Street: ________________ Department: ________________

City: ____________ State: ____________ Zip or Country ________

--------------------- Fold Here ---------------------

--------------- Do Not Tear - Fold Here and Staple ---------------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Digital Equipment Corporation
Software Information Services
146 Main Street, Bldg. 3-5
Maynard, Massachusetts 01754