DECUS NO.	8-77
TITLE	PDP-8 DUAL PROCESS SYSTEM
AUTHOR	Richard M. Merrill
COMPANY	Digital Equipment Corporation Maynard, Massachusetts
DATE	May, 1967
SOURCE LANGUAGE	

PDP-8 DUAL PROCESS SYSTEM

Program Library Writeup | DECUS No. 8/8S-77

PURPOSE

The purpose of this system is to expedite the programming of multiprocessing problems on the PDP-8 and PDP-8/S. It maximizes both the input speed and the portion of real time actually used for calculations by allowing the program to run during the intervals between issuing I/O commands and the raising of the device flag to signal completion of the command. The technique also allows queuing of input data or commands so that the user need not wait while his last line is being processed, and each line of input may be processed as fast as possible regardless of its length.

This method is especially useful for a slower machine where the problem may easily be calculation limited but would, without such a system, become I/O bound.

The program may also be easily extended to handle input from an A/D converter. Here the input would be buffered by groups of readings terminated either arbitrarily in groups of N or by zero crossings.

REQUIREMENTS

600 octal registers for the system for two TTY's plus buffer space.

Several device configurations are possible.

USAGE

To start the system, the user program should first clear all device flags that have not been provided for in the interrupt routine ("NTRPT"). All such flags that are raised during run time will cause the system to loop.

The user program should then load KADR and RADR with the appropriate addresses in the user program to process the inputs. KADR and RADR may be changed by the user program at any time. When the user program has finished, it should jump to IDLE.

The registers GETCKØ and GETCRØ contain the addresses of the subroutines to get characters from the keyboard and the reader respectively. Similarly OUTCTØ and OUTCPØ contain the addresses of the subroutines to output characters on the TTY and on the high-speed punch.

The instructions for the high-speed paper tape equipment may be replaced with the I/O instructions for an additional TTY, if desired, as indicated in the listing. The user may want to tailor this program to his own particular application and will use this version only as a guide.

METHODS

The method used is simple: a circular character buffer is maintained for each input or output device being used.

Buffers

These buffers are assigned by the user program and should probably be at least twice as long as the largest sentence expected. The buffers are unloaded character by character via requests from the user program and hence might fill up if not called upon frequently enough.

Programmed I/O

Output proceeds in the same fashion by user calls (e.g. JMS I OUTCTØ) to send single characters. The overflow problem also exists here, but we have chosen to cause the output calls simply to wait until enough of the buffer has been unloaded to accept the next character. This, however, will detract from the overall efficiency of operation.

Real-Time Editing

In order to allow the user to correct his typing and to help provide the program with meaningful groups of input data, each line is stored in the buffer until the user has typed a carriage return indicating his approval of the data typed. Before doing this, however, he may delete preceeding characters by typing "rubout." A "#" will print for each character removed. The entire line may be killed by typing "   ."

Thus, a line is terminated by a CR and control, thereupon, passes to the user program. The address of the user process is stored in a register assigned to each input device. For example, KADR contains the address of the program that will process the next line of keyboard input.

Automatic Echo of Keyboard

This feature is provided as an option obtained when KECHOSW is non-zero.

Output Subroutine Calls

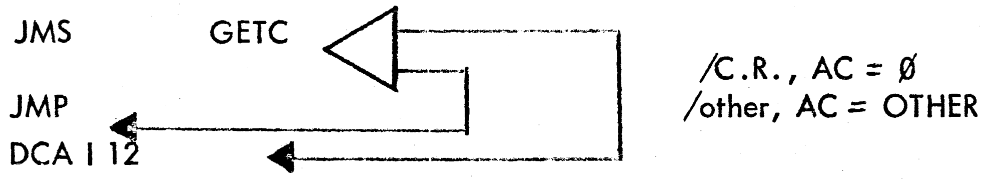
In order to have characters printed as they are typed, an echo mode is provided by a program switch for each TTY. These may be charged under program control so that the echo mode may be turned on or off at will.

Naturally, garbage may be produced if the program prints at the same time as the keyboard is struck, if the latter is in the echo mode. This is also the only circumstance which may cause a system hangup. Namely, if the output buffer is full when a character is to be echoed, the program will loop indefinitely since echoes are made while still in the interrupt process with the interrupt off.

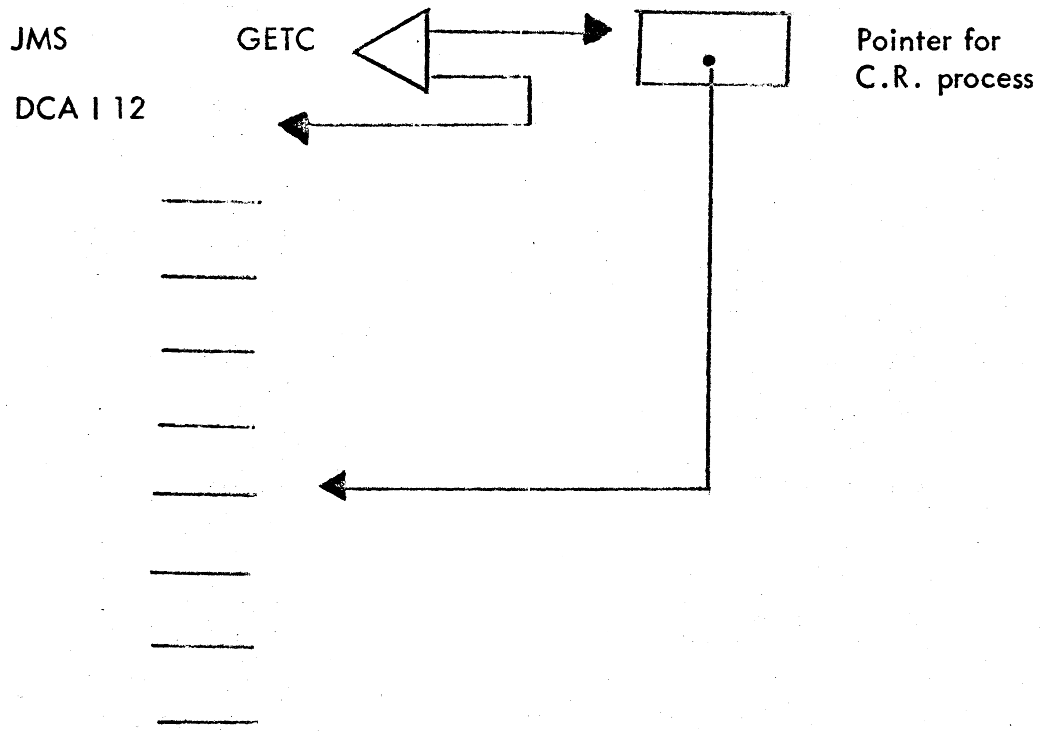
Input Subroutine Calls

The system will halt if the input buffers overflow or if the program asks for characters from the buffer that have not yet been read. The recovery to try in both cases is to type either " " or CR and then hit "CONTINUE." This circumstance might arise from the user's program asking for another character before going to IDLE, after receiving a carriage return.

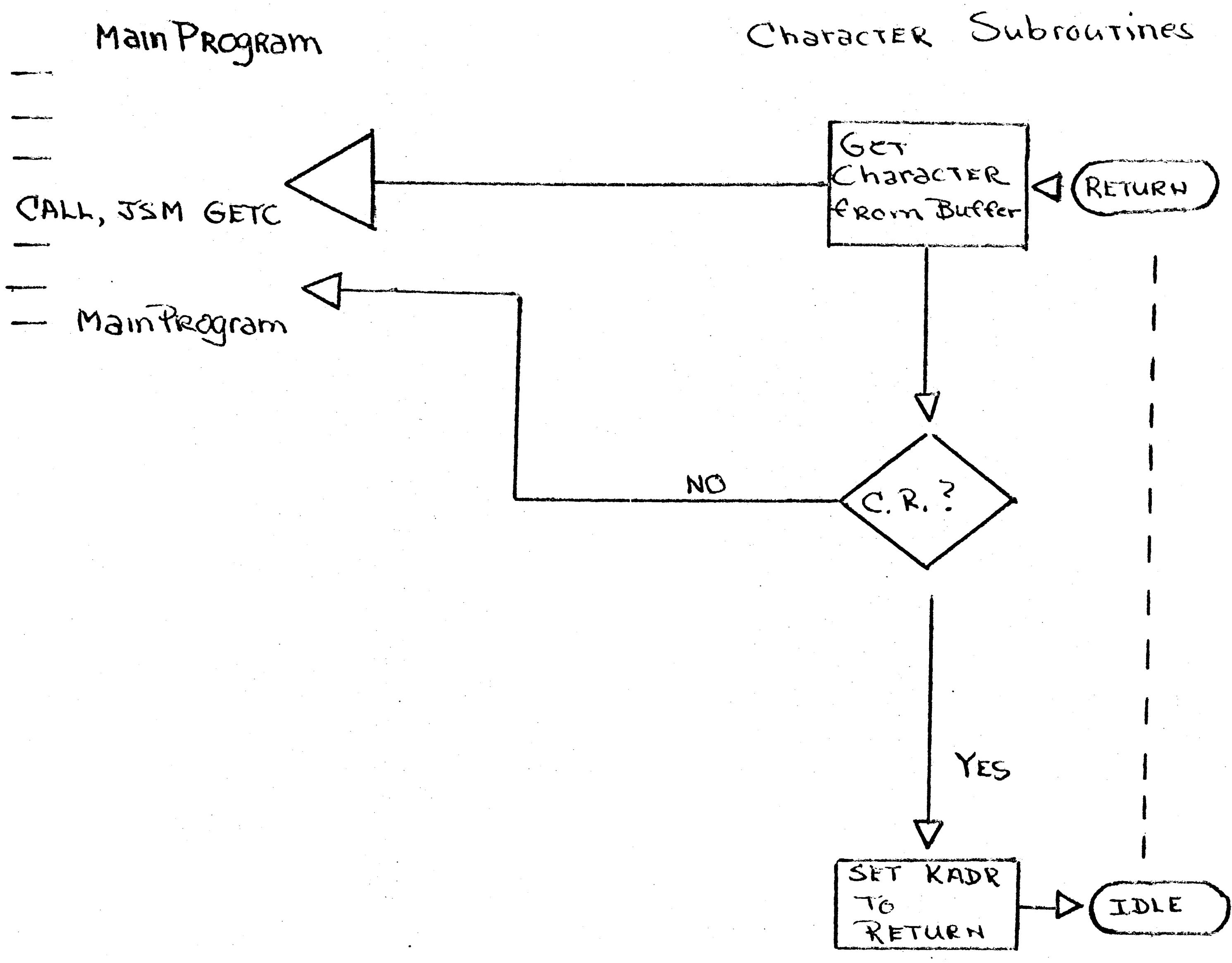
To avoid this, the routine has two returns: the first if the character is a CR and the second if it is some other character, as illustrated below.



An alternative would be to assign a specific location to go to when a CR is received. Such a subroutine could be schematized this way:

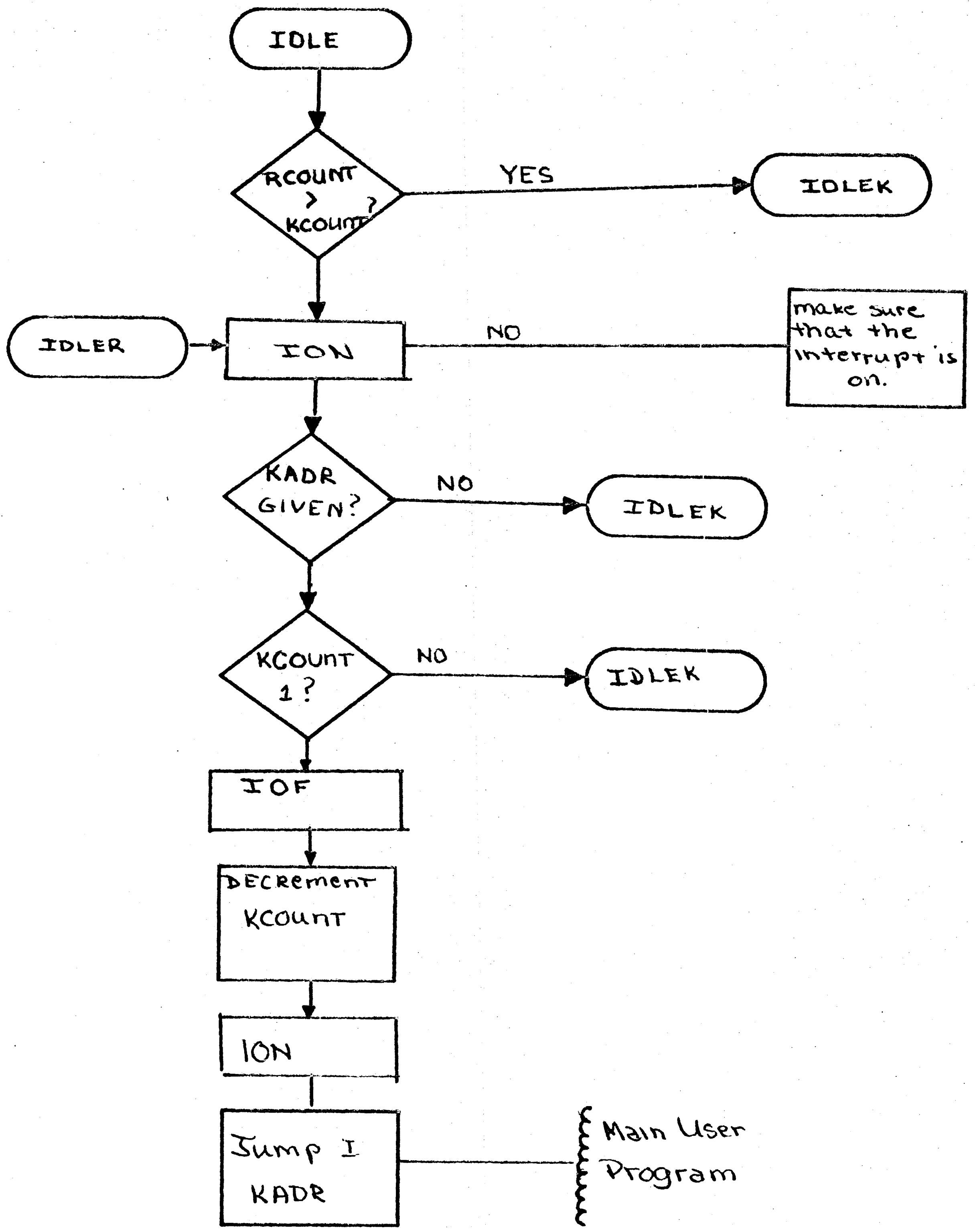


Another approach would be to write the user program completely without worrying about IDLE, CR's and KADR by writing the get character routine so that it always comes back with a character and waits in IDLE if no characters are available:



EXECUTION TIME

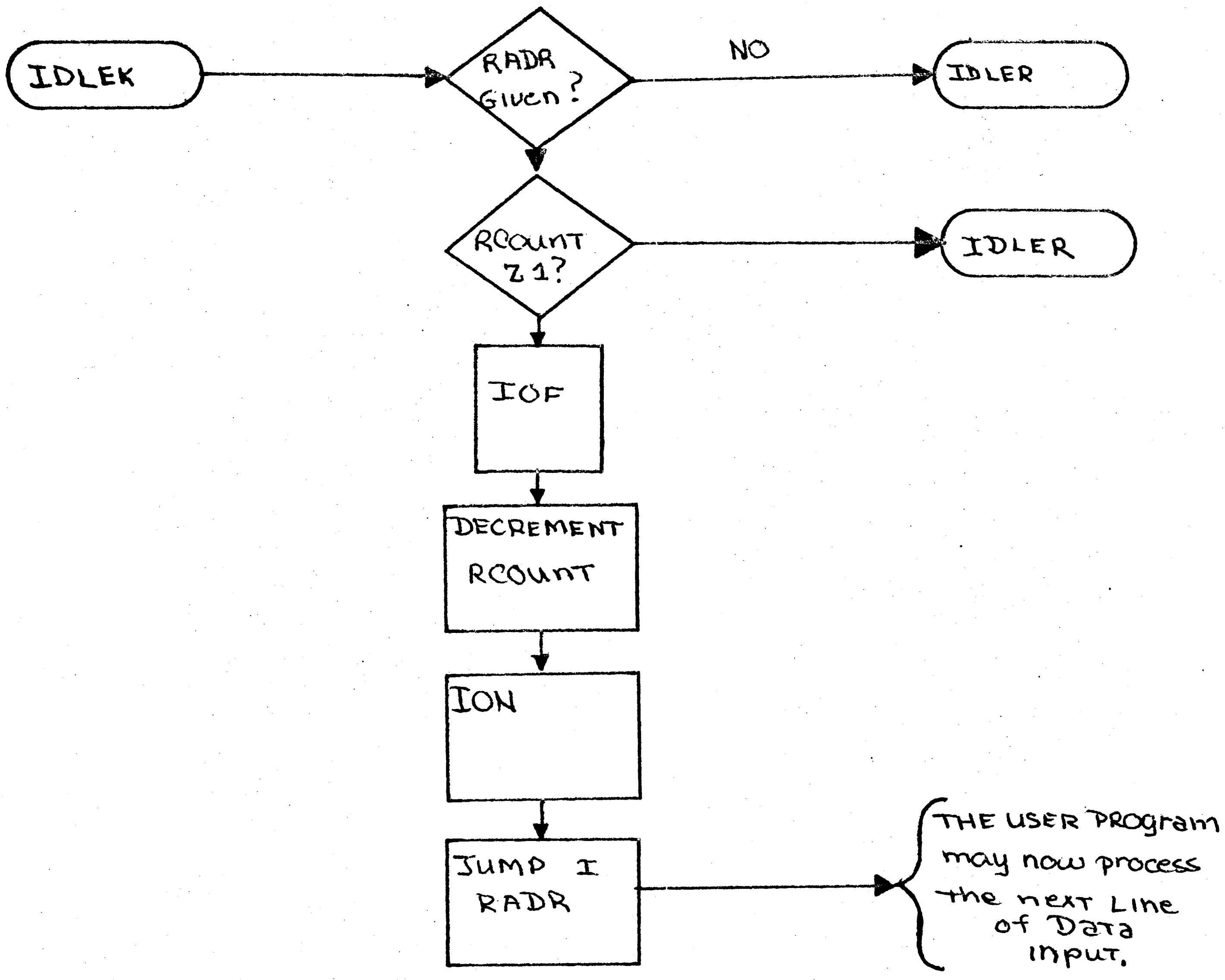
On the average, it takes about 60 instructions to process one character. For a keyboard or reader with a top speed of 10 cps, this represents only a 2.4% overhead.



The user program always jumps to IDLE when it has finished processing a line of data.

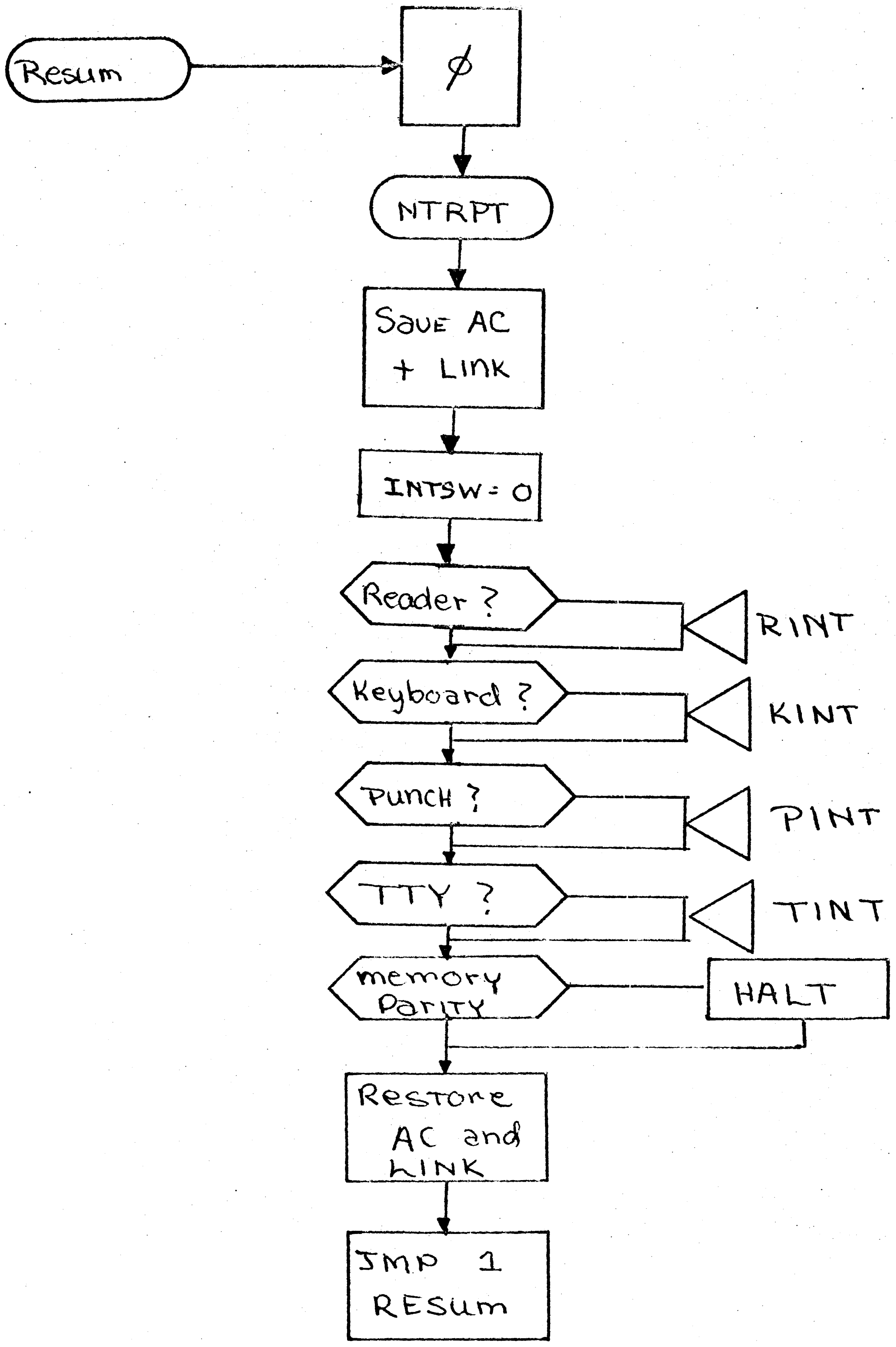
IDLE Loop and Switches

Figure 1



IDLE Loop For Keyboard

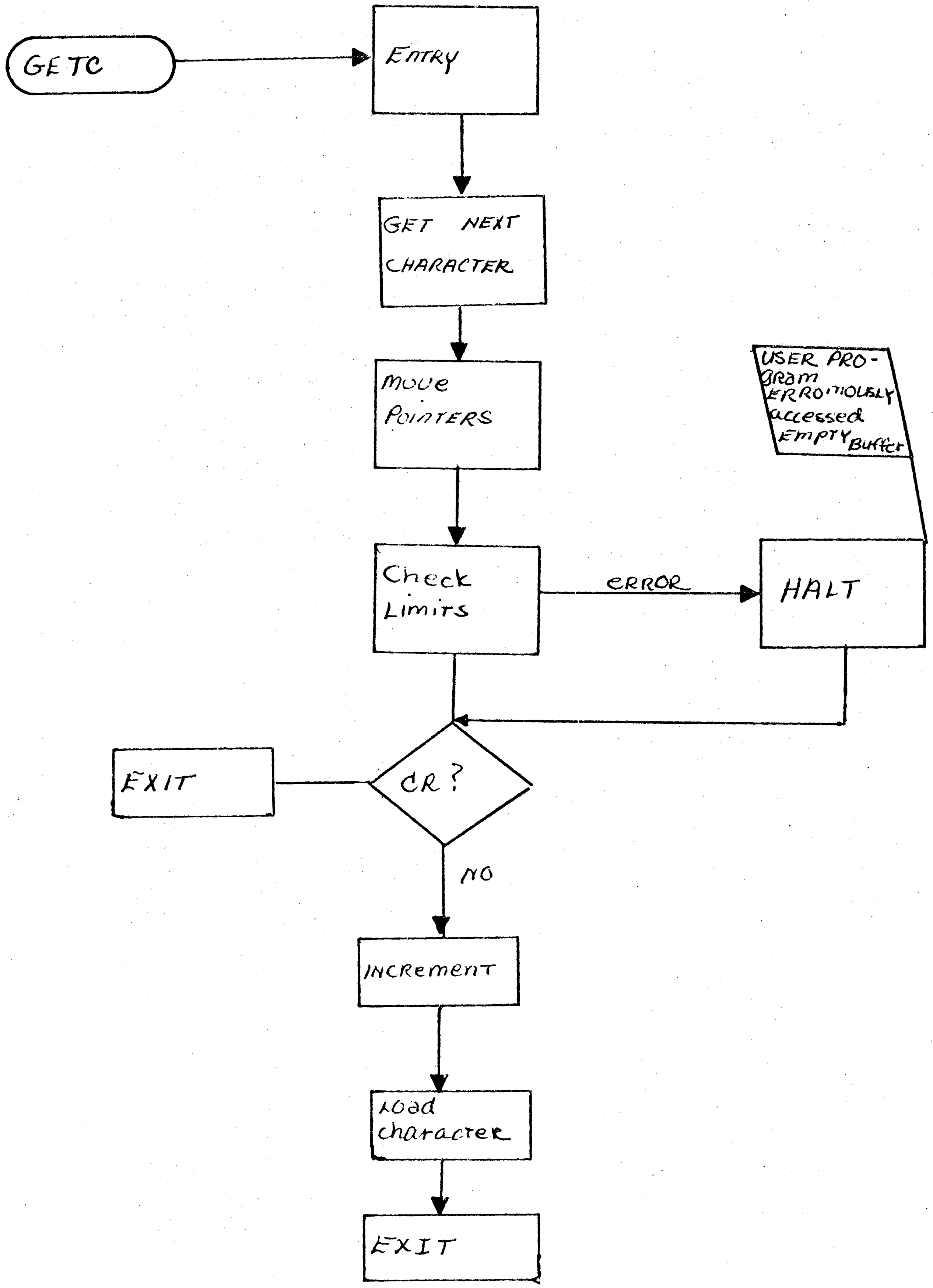
Figure 2



The computer interrupt hardware saves the program counter in location zero and jumps to location 1. Ther program then jumps to "NTRPT".

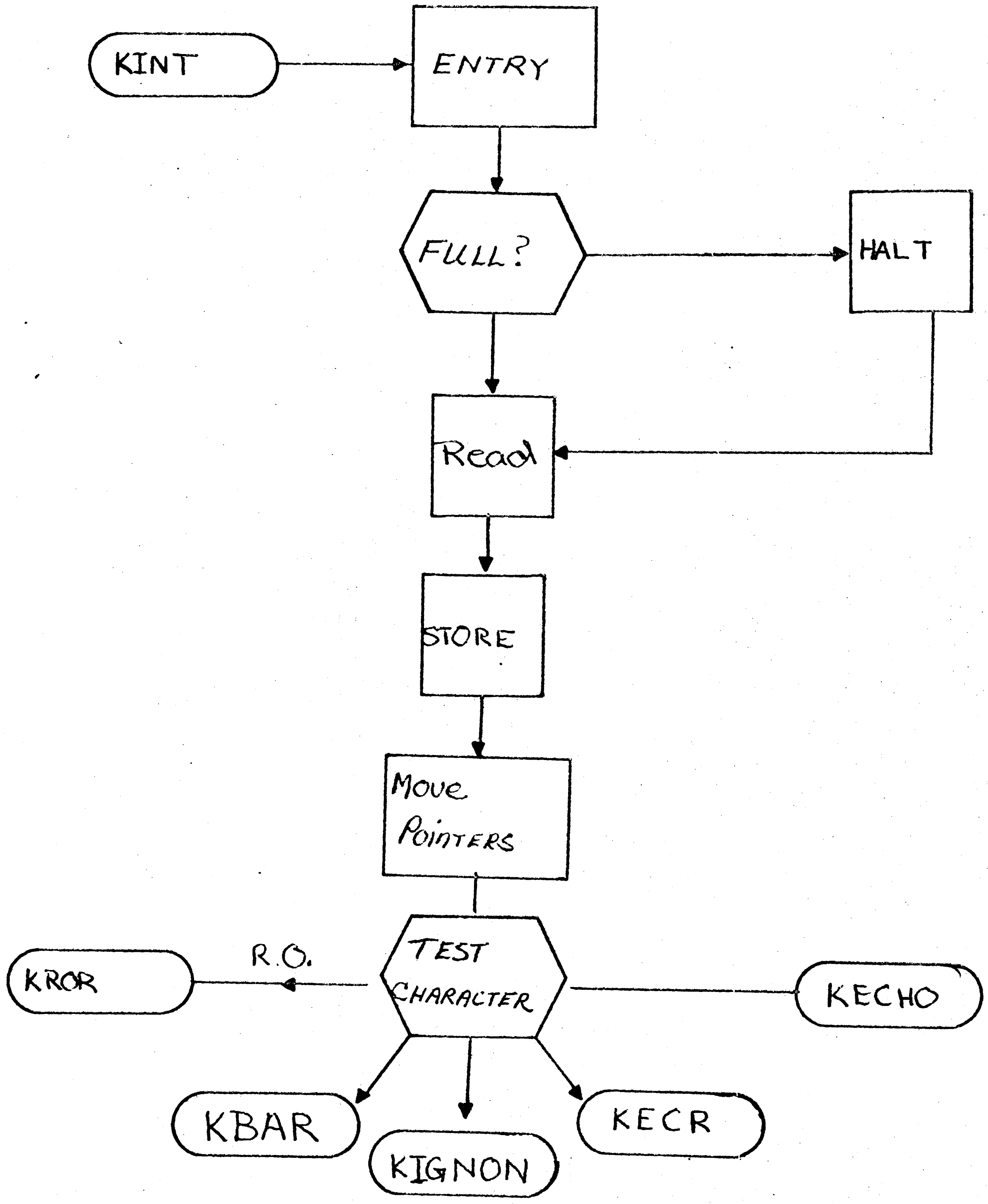
INTERRUPT PROCESSING

Figure 3



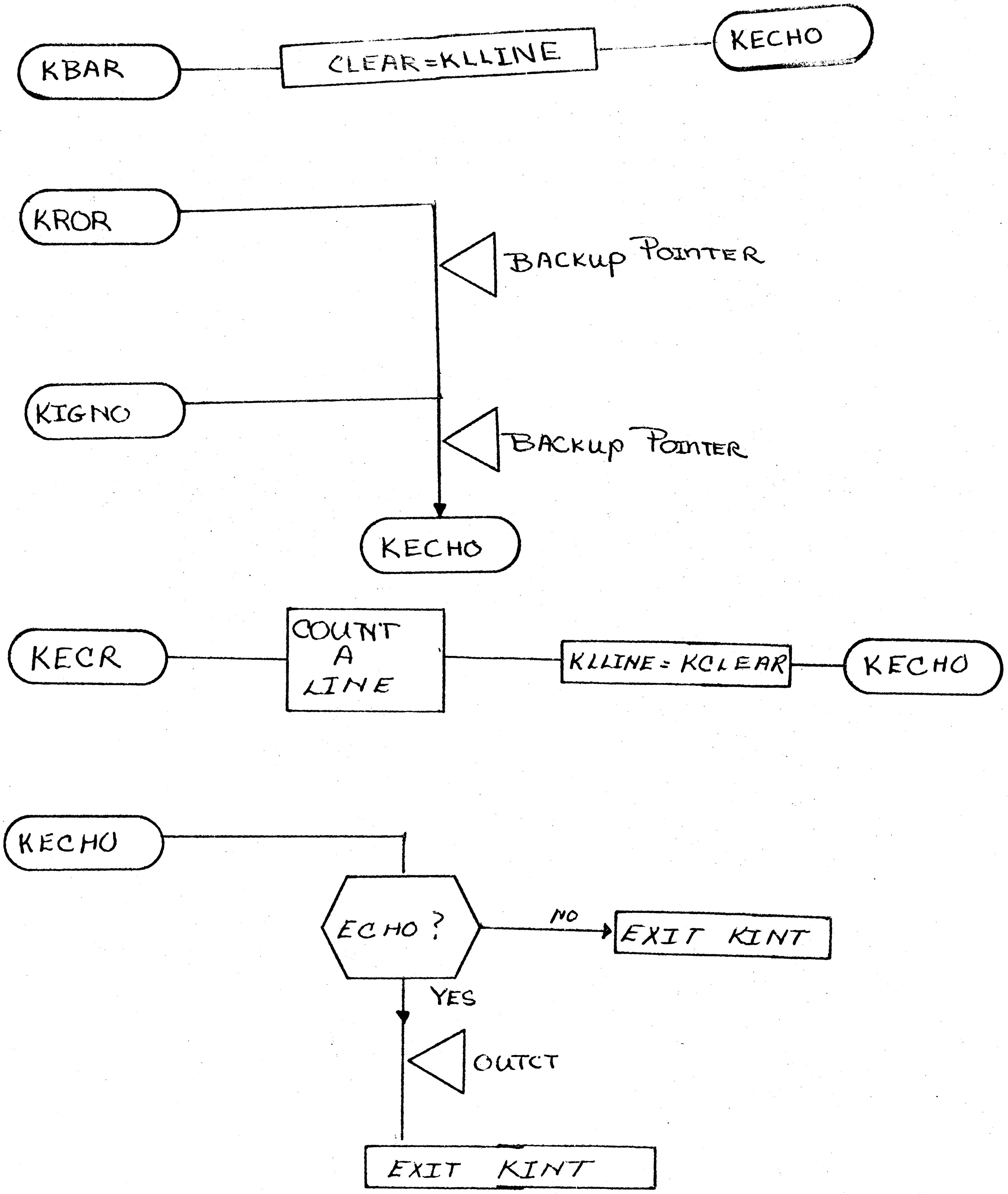
If a HALT occurs in the keyboard program, the user may hit "←" or C. R. on the keyboard and then press continue on the console. This procedure should help recover somewhat from all system halts.

INPUT SUBROUTINES
Figure 4

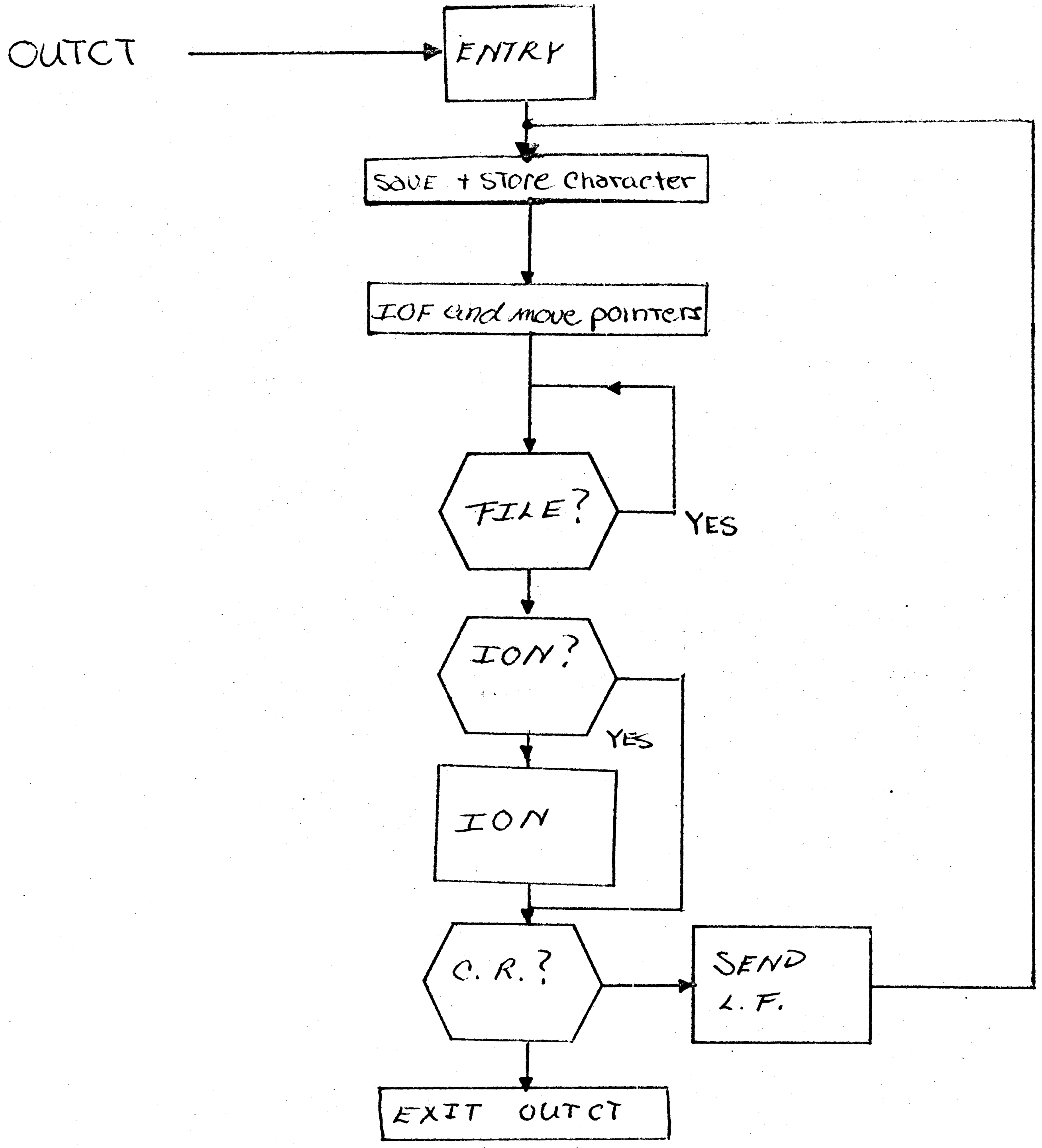


This is a typical subroutine to process an interrupt caused by the keyboard/reader. It is run with the interrupt mode turned off.

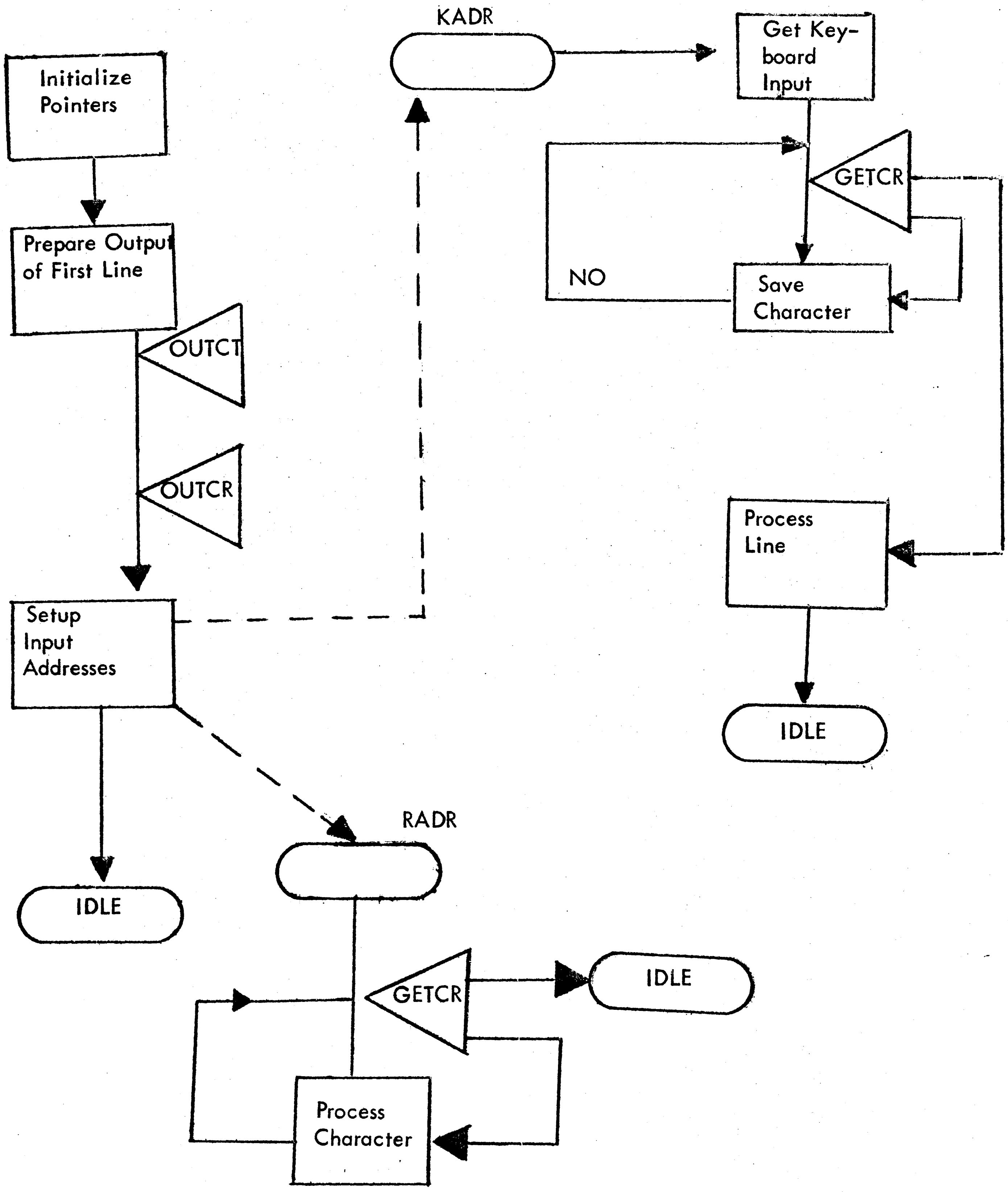
Figure 5



Character Processing
Figure 6



Character Output
Figure 7



USAGE ILLUSTRATION

Figure 8