# INSTRUCTION MANUAL

## PETDISK II

## 6502PDS

CGRS MICROTECH

P.O. Box 102 • LANGHORNE, PA 19047
(215) 757-0284

# PDOS II DISK OPERATING SYSTEM PROGRAMMER'S MANUAL

## I. INTRODUCTION

The PDOS II software package operates using both a core of program routines contained on ROM and a system of overlay programs loaded into the top 2K of system RAM from disk. These various software routines can be divided into five program blocks:

1. ROM Routines ($EA00  $ means hexadecimal)
   Disk primitive I/O drivers
   System initialization and boot
   Main Interface Routine

2. Logical Interface Routines (RAM at $7800, 32K Pet)
   File conversion routines
   Math routines- multiply, divide

3. BASIC Interface Routines (RAM at $7A00, 32K Pet)
   SAVE, OPEN, CLOSE, INPUT, PRINT, RUN, LIST
   Basic file handling routines

4. DOS-mode Routines (RAM at $7A00, 32K Pet)
   Command Interpreter
   LOAD, SAVE, EXECUTE, KILL
   File handling routines

5. DOS-mode Transient Routines (RAM at $7C00, 32K Pet)
   PRINT, HELP, RENAME, DUMP
   Utilities- Copy, Format, Patch, Compress

PDOS II has two modes of operation, the BASIC mode and the DOS-mode. The PDOS programmer must choose the mode of operation in which his program will run. The PDOS programmer must choose either the DOS-mode or BASIC mode. Most programs will operate in the BASIC mode as this is the mode that is first initialized. Utility functions such as assemblers, editors, and disk maintenance programs would be appropriate to the DOS-mode.

## II. BASIC MODE OPERATION

The BASIC mode is used when the PEDISK II System is being used from the resident BASIC Interpreter. User programs written in BASIC will utilize the PDOS BASIC mode. All PDOS BASIC commands (!OPEN, !PRINT, !LIST, etc.) as well as the Main Interface routine are resident in the system memory during the BASIC mode operation. These programs are included in program blocks two and three.

The Commodore BASIC interprets the lines of a program through a subroutine in lower memory that gets single characters from each program line. This subroutine is called CHARACTER GET or "CHRGET". Many utility programs including PDOS attach themselves to the BASIC by modifying the subroutine and intercepting the characters. Figure 1 shows how PDOS II interprets its commands.

If a valid PDOS command is interpreted, PDOS will save registers and stack. It will then get the file name (if any) from the program line and go to the proper routine via a jump table at the beginning of program block two ($7800). All PDOS BASIC commands are ended by a common exit routine which restores registers and stack before exiting thourgh the "CHRGET" routine back to BASIC.

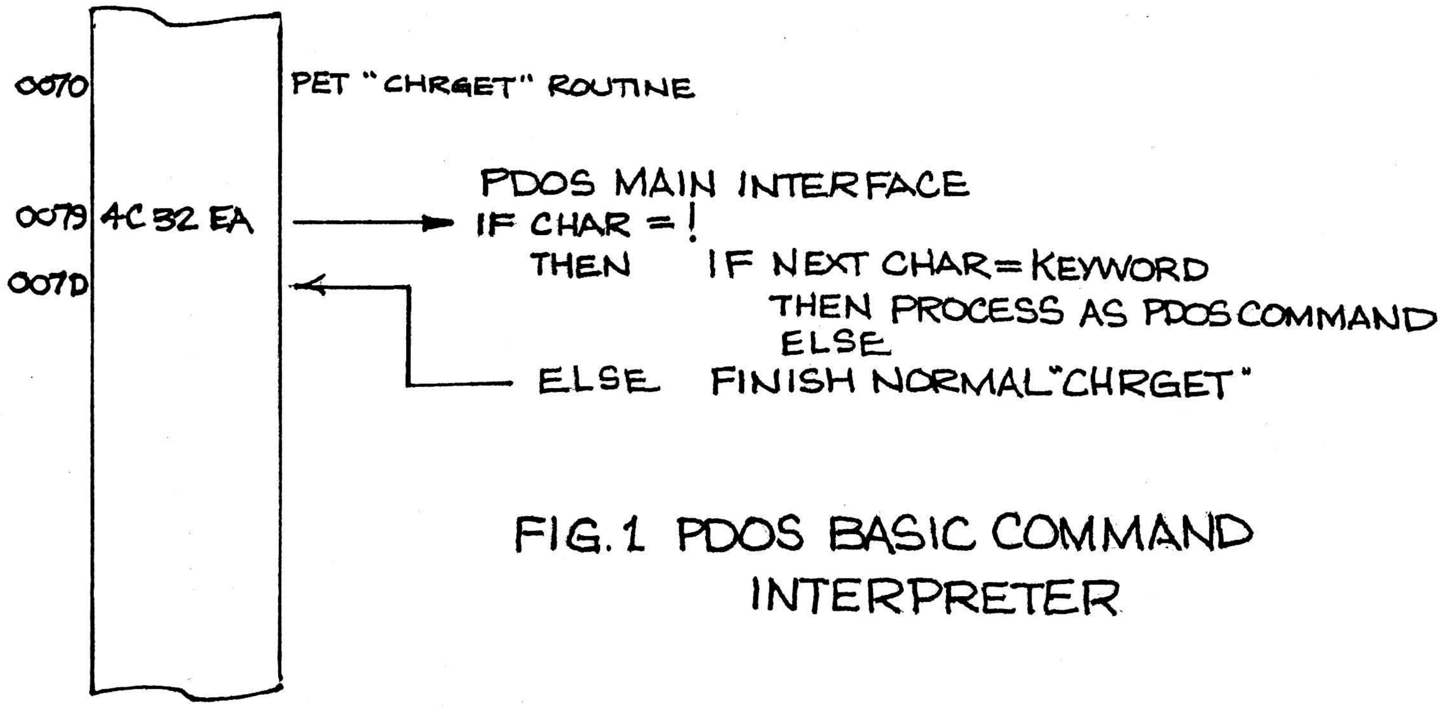

0070
PET "CHRGET" ROUTINE
0079
4C 32 EA
007D
PDOS MAIN INTERFACE
IF CHAR = !
THEN IF NEXT CHAR=KEYWORD
THEN PROCESS AS PDOS COMMAND
ELSE
ELSE FINISH NORMAL "CHRGET"

FIG. 1 PDOS BASIC COMMAND INTERPRETER

## III. DOS-mode OPERATION

In the DOS-mode, single letter commands make the disk utilities and monitor operations conveniently available. The DOS-mode is entered via the !SYS command from the BASIC mode. The !SYS command will read program block four (DOS-mode routines) from disk and overwrite program block three (BASIC Interface routines). Program blocks three and four cannot be resident in memory simultaneously. Once the DOS-mode routines are loaded, the program jumps to the DOS-mode cold start at $7A00.

The DOS-mode has a series of resident disk utility programs Each is called with a single letter command. Some of these routines are resident in memory. Other routines (program block 5) are contained on disk and are loaded into a special area of memory, the transient overlay area, as required. The DOS-mode command interpreter goes to the proper routine by scanning a list of commands contained in memory. If the command is not found, PDOS then looks for a command file on the disk. Command files are always named "*****_". The command letter is the last letter of the file name.

An example... After the !SYS command loads program block four (DOS-mode routines) into memory at $7A00 thru $7BFF, the computer jumps to $7A00 where it will clear the screen, display the ">" prompt and a cursor. It then waits for a command. If the "M" key is hit, the program will jump to a routine in ROM. When the operator is finished looking at memory, the "STOP" key is hit and the program will return to the ">" prompt. The routines required for the "M" command are contained in memory. If the "P" key is hit, the PDOS command interpreter will find that the "P" command is not contained in memory and it will go to disk drive 0 and search for a program called "*****P". If found, it will load the "*****P" file into memory at the transient overlay area - $7C00. If the load was successful, the program will jump to $7C00 and execute the command. When the command is finished the program will go back to the DOS-mode command interpreter warm start at $7C05.

User installed commands can easily be added to PDOS II. It is simply a matter of writing a program to occupy memory between $7C00 and $7DFF to impliment the command. The program must be saved on disk as "*****_", where _ is the letter used to call the program. The PDOS command interpreter will always scan the disk for transient command files and does not have a fixed set of commands. In cases where the memory space between $7C00 and $7DFF is not sufficient, the programmer can write a file loader routine that does fit between $7C00 and $7DFF which will load a much larger program into memory below $7800.

User programs operated from the DOS-mode should never return directly to BASIC. They must return to the DOS-mode command interpreter at $7C05. The DOS-mode "R" command will properly return the system to BASIC by reloading program block three (BASIC Interface Routines) before returning.

## IV. ORGANIZATION OF THE DISK

As previously stated, the PDOS program is divided into five program blocks. Program blocks two thru five are located on disk. Also contained on the disk are the "header" and the "directory", both part of the disk VTOC (Volume Table Of Contents). The PDOS system disk is organized as follows:

1. VTOC : Track 00 Sector 01 thru Track 00 Sector 08
   Header - First 16 bytes,

   | | |
   |---|---|
   | 1-8 | Disk Name |
   | 9 | Number of active files |
   | 10 | Next available track |
   | 11 | Next available sector |
   | 12-16 | Reserved, I.D. number |

   Directory Entry - 16 bytes per file,
   File Name, load data, etc.

2. SYSTEM FILE "******" : Track 00 Sector 09 thru Track 00 Sector $19
   The System File contains PDOS program blocks two, three, and four. It must be contained on any diskette used for system startup or mode switching. It is not necessary on diskettes used purely for storage. The System File is organized as follows:

   Program block 2 : Track 00 Sector 09 thru Track 00 Sector $15
   Program block 3 : included above

   Program block 4 : Track 00 Sector $16 thru Track 00 Sector $19

3. Transient Routines "*****_" Track 00 Sector $1A up-as required.
   Transient routines (Program block five) are contained on the diskette like any other file. These routines are used to implement various DOS-mode commands. They are provided on the master system diskette but are only necessary when the DOS-mode utilities provided are desired.

## V. FILE STRUCTURE

Each file in the PEDISK II system has a "directory entry". The directory entries are contained in the VTOC of the diskette. Each directory entry contains a lot of valuable information about each file as follows:

1.Bytes 0 thru 5 - File Name
All files are named with a six character name. When the operator enters a name with less than six characters, PDOS fills the remaining positions with ASCII space ($20).

2.Bytes 6 and 7 - Record length (data files) or
Entry address (program files)
File length (assembler files)
PDOS reserves these bytes to record either the length of each data record or the address to begin program execution for program/load files. These bytes are also used to record number of bytes in the file for assembly source files.

3.Bytes 8 and 9 - Block size (data files) or
Load address (program files)
PDOS reserves these bytes to record either the block size for data files or the address to start loading the program for program/load files.

4.Byte 10 - File type
PDOS recognizes several different types of files. Two classifications are possible:
a. Data files - containing data records
b. Program/load files - containing BASIC programs, assembly programs or source file memory images.

The full list of file types are:

0 - Sequential data file
1 - Relative data file
2 - Relative Sequential data file (future)
3 - BASIC program file
4 - Assembly source file
5 - Program load file
6 - Misc. text file
7 - Relocatable object file

5.Byte 11 - Not used at this time. PDOS records a 00.

6.Byte 12 - Starting Track of the file on disk.

7.Byte 13 - Starting Sector of the file on disk.

8.Bytes 14 and 15 - Number of sectors in the file.
NOTE:Program/Load files have a maximum of 255 sectors

The previous byte table described the various bytes in each directory entry. Each entry of the file contains all the information required to load and manipulate the file. For example, to load a program file PDOS does the following:

1. Get the file name from the program line or the operator.
2. Load VTOC sectors and check file names until a match is found.
3. Get the load addr, starting track and sector, and number of sectors. These are inputs to the "READ SECTORS" subroutine.
4. Jump to subroutine "READ SECTORS". Information is put into memory.
5. Check error flags, if none, finished.
6. If the program is to be executed, jump to the "entry address".

## VI. PDOS PROGRAM DESCRIPTION

### BLOCK 1 - ROM

The PDOS ROM contains the disk "primitive" routines, system initialization, Main Interface Routine, VTOC search routine, load routine, and memory examine/change routine. Several of these routines are valuable for user programs. A jump table at the beginning of the ROM routes user software to the proper spot regardless of the PDOS version (V5 and up). User programs taking advantage of these subroutines should always use the jump table. As the PDOS software is improved, various absolute starting locations will change. The jump table will always be at the same spot.

JUMP TABLE FROM ROM

```
$EA00 JMP INITIAL - System initialization
$EA03 JMP DOMEM   - Memory examine/change
$EA06 JMP PRBLK   - READ SECTORS
$EA09 JMP PWBLK   - WRITE SECTORS
$EA0C JMP SEARCH  - Search VTOC for directory entry
$EA0F JMP LOADX   - Load file subroutine
```

Four subroutines, PRBLK, PWBLK, SEARCH, and LOADX will be useful for user written programs. In the following pages these subroutines will be described. No attempt is made at saving the X or Y registers within the subroutines, so bo careful! The complete source listing is available for those who need more details. See the SOURCE LISTING INFORMATION at the beginning of the Getting Started Manual.

## PRBLK - READ SECTORS

This subroutine will read a specified quantity of phsyical sectors from the specified disk drive. Inputs are:

```
Disk Drive # : $01 - Drive 0
               $02 - Drive 1
               $04 - Drive 2 stored in $7F91 (PDEVIC)
```

NOTE: A disk drive is selected by setting the corresponding bit as shown above.

```
Starting Track stored in $7F92 (PTRACK)
    Valid track numbers are:
    00 thru 39 ($27) for Model 540
    00 thru 79 ($4F) for Model 580
    00 thru 76 ($4C) for Model 877

Starting Sector             stored in $7F93 (PSECTR)
    sector range 01 thru 28 for 5 1/4" drive
    sector range 01 thru 26 for 8" drive

Number of sectors          stored in $7F96 (SCTCNT)

Load Address                stored in $00B7, lo byte (BLCKAD)
                                      $00B8, hi byte (BLCKAD+1)
```

If the data read was successful, the subroutine will return with the Z bit = 1. An error will cause the error message to be displayed and the subroutine will return with Z = 0.

FOR EXAMPLE, the following code reads four sectors from the disk drive 0 starting at Track 00, Sector 01. The data is put into memory at address $4000.

```
PDEVIC  .EQU $7F91
PTRACK  .EQU $7F92
PSECTR  .EQU $7F93
SCTCNT  .EQU $7F96
BLCKAD  .EQU $00B7
PDOSRM  .EQU $EA00
READD   .EQU $EA06
WRITED  .EQU $EA09

;SET UP INPUTS TO "READ SECTORS"

LDA #01  DRIVE 0,SET BIT 0
STA PDEVIC
LDA #00  TRACK 0
STA PTRACK
LDA #01  SECTOR 1
STA PSECTR
LDA #04
STA SCTCNT  FOUR SECTORS
LDA #00
STA BLCKAD
LDA #$40   ADDRESS=$4000
STA BLCKAD+1

;READ THE DATA

JSR READD

;CHECK STATUS DONE BY CALLING ROUTINE
;ZERO MEANS OK

RTS  FINISHED!
```

## PWBLK - WRITE SECTORS

This subroutine will write a specified number of physical sectors to the specified disk drive. Inputs are the same as the READ SECTORS subroutine. In this case the "load address" will specify where the information is to be recorded from.

If the write was successful, the subroutine will return with Z = 1. An error will cause the error massage to be displayed and the subroutine will return with Z = 0.

FOR EXAMPLE, the following routine will record two sectors (128 bytes each) to disk drive 1 starting at Track 01, Sector 01 from memory at $0400.

```
;SET UP INPUTS TO "WRITE SECTORS"

LDA #02   DRIVE 1, SET BIT 1
STA PDEVIC
LDA #01   TRACK 1
STA PTRACK
STA PSECTR SECTOR 1
LDA #02   TWO SECTORS
STA SCTCNT
LDA #00
STA BLCKAD
LDA #$04   ADDRESS = $0400
STA BLCKAD+1

;WRITE THE DATA TO DISK

JSR WRITED

;STATUS CHECK DONE BY CALLING ROUTINE
;ZERO MEANS OK

RTS  FINISHED!
```

## SEARCH

This subroutine will search the specified disk drive VTOC and attempt to find the specified file name. If the file name is found, the "directory entry" will be in memory and a zero page memory location will be set to point to it. Inputs are:

```
Disk drive #  stored in $7FB1 (MAINFCB+17)

File Name : 1st character stored in $7FA0 (MAINFCB   )
            2nd character stored in $7FA1 (MAINFCB+1)
            3rd character stored in $7FA2 (MAINFCB+2)
            4th character stored in $7FA3 (MAINFCB+3)
            5th character stored in $7FA4 (MAINFCB+4)
            6th character stored in $7FA5 (MAINFCB+5)
```

for file names with less than six characters, remaining character locations must be filled with $20, ASCII space.

The outputs are:

```
Next TRACK and SECTOR - next available TRACK and SECTOR on the
specified disk are stored in  NEXT TRACK - $0056 (HNXTTR)
                              NEXT SECTOR - $0057 (HNXTSC)

Pointer to the "directory entry" is stored in two memory locations:
                              LO BYTE    - $0022 (TEMPAD)
                              HI BYTE    - $0023 (TEMPAD+1)
```

If a successful name match is made, SEARCH returns with the accumulator = 00, Z = 1. If a complete search is made and no match found, the subroutine returns with the accumulator = $7F, Z = 0. If a disk read error occurs, SEARCH returns with the accumulator = $FF, Z = 0.

FOR EXAMPLE, the following code will find a program file named "TEST" on disk drive 0 and load it into memory.

```
MAINFCB .EQU $7FA0    MAIN FILE CONTROL BLOCK
SEARCH  .EQU $EA0C    JUMP VECTOR TO SEARCH SUBROUTINE
TEMPAD  .EQU $22
SELECT  .EQU $E900   Drive select register

;SET UP INPUTS FOR SEARCH

LDA #01   DRIVE 0,SET BIT 0
STA MAINFCB+17
LDA #'T   ASCII T
STA MAINFCB
LDA #'E   ASCII E
STA MAINFCB+1
LDA #'S   ASCII S
STA MAINFCB+2
LDA #'T
STA MAINFCB+3
LDA #$20  ASCII SPACE
STA MAINFCB+4
STA MAINFCB+5 FILL REMAINING NAME CHARS WITH SPACE
```

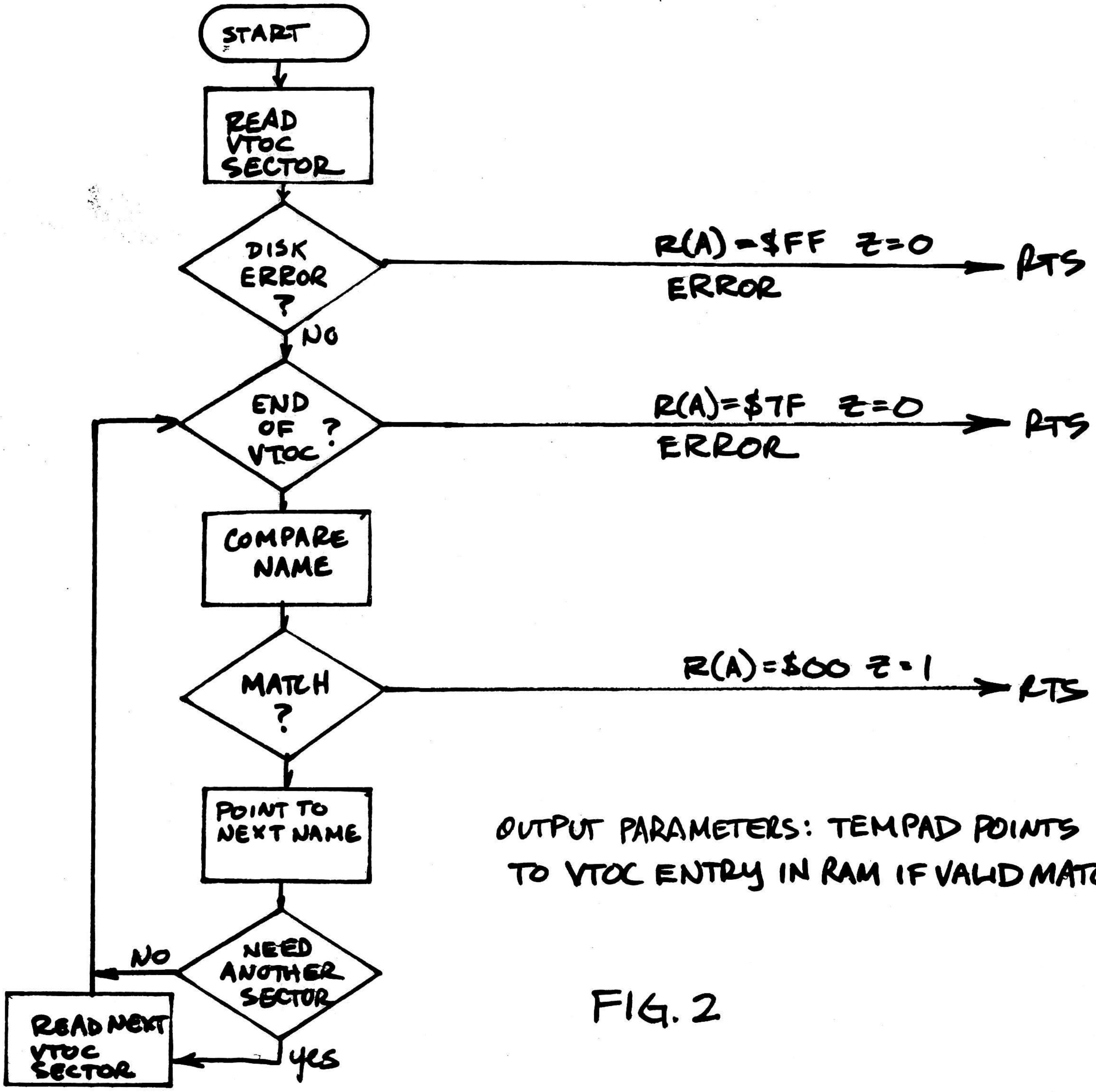
```
 ;SEARCH FOR DIRECTORY ENTRY

 JSR SEARCH
 ;CHECK STATUS

 BMI QUIT  DISK ERROR,STOP NOW
 BNE NOFIND CAN'T FIND FILE

 ;LOAD THE FILES:GET INPUTS TO "READ SECTORS"

 LDA MAINFCB+17  GET DISK DRIVE NUMBER
 STA PDEVIC
 LDY #8
 LDA (TEMPAD),Y  GET BYTE #8 FROM DIRECTORY
 STA BLCKAD      STORE LOAD ADDRESS LO
 INY
 LDA (TEMPAD),Y  GET BYTE #9 FROM DIRECTORY
 STA BLCKAD+1    STORE LOAD ADDRESS HI
 LDY #12
 LDA (TEMPAD),Y  GET BYTE #12 FROM DIRECTORY
 STA PTRACK      STORE IN TRACK NUMBER
 INY
 LDA (TEMPAD),Y  GET BYTE #13 FROM DIRECTORY
 STA PSECTR      STORE STARTING SECTOR OF THE FILE
 LDY #15
 LDA (TEMPAD),Y  GET BYTE #15 FROM DIRECTORY
 STA SCTCNT      STORE # OF SECTORS IN THE FILE

 ;LOAD THE FILE: READ SECTORS

 JSR READD
QUIT RTS

NOFIND  LDA #0
        STA SELECT   DESELECT THE DISK DRIVE
        JMP FINDMSG  OUTPUT "CANT FIND MESSAGE" AND RETURN
```

MODULE: SEARCH

PURPOSE: READ VOLUME TABLE OF CONTENTS (VTOC) ON SPECIFIED DRIVE # AND SEARCH FOR SPECIFIED FILE NAME.

INPUT PARAMETERS: SPECIFIED FILE NAME - SIX ASCII CHARACTERS LOCATED IN FIRST SIX POSITIONS OF MAINFCB ($7FA0)

SPECIFIED DRIVE NUMBER - 1, 2 or 4 IN MAINFCB+17.



OUTPUT PARAMETERS: TEMPAD POINTS TO VTOC ENTRY IN RAM IF VALID MATCH.

FIG. 2

## LOADX

This subroutine will load a Program/Load type file into memory. It operates in a manner similiar to the previous example. It will not load data files (types 0, 1, or 2) however.
Inputs are:

Disk drive # stored in $7FB1 (MAINFCB+17)

```
File Name : 1st character stored in $7FA0 (MAINFCB   )
            2nd character stored in $7FA1 (MAINFCB+1)
            3rd character stored in $7FA2 (MAINFCB+2)
            4th character stored in $7FA3 (MAINFCB+3)
            5th character stored in $7FA4 (MAINFCB+4)
            6th character stored in $7FA5 (MAINFCB+5)
```

for file names with less than six characters, remaining character locations must be filled with $20, ASCII space.

Outputs of the subroutine are:
The specified file is loaded into memory at the address specified in the directory entry for the file. If the file cannot be found or if it is a data file, PDOS will prompt with six question marks "??????". PDOS will then deselect the specified disk drive and return with register X = $FF. If a disk read error occurs, the subroutine will set register X = $FF. A successful load will be indicated with register X = $00.

FOR EXAMPLE, the following code will load a program called "TEST" into memory at the address specified by the directory entry:

```
LOADD   .EQU $EA0F
MAINFCB .EQU $~~7AF0~~ 7FA0
NAME    .BYTE 'TEST  '
;NOTE THAT TWO SPACES EXIST AFTER THE NAME

        LDY #5  SIX CHARACTERS
MORE    LDA NAME,Y  GET CHARACTER FROM NAME
        STA MAINFCB,Y  STORE CHARACTER
        DEY
        BPL MORE       FINISHED?
        LDA #01        DISK DRIVE 0, SET BIT 0
        STA MAINFCB+17

;LOAD THE PROGRAM!

        JSR LOADD
;CALLING ROUTINE CHECKS FOR ERRORS
;IF X=00, OK

        RTS
```

## BLOCK 2 - LOGICAL INTERFACE ROUTINES

The Logical Interface Routines begin with a jump table which provides a jump to each of the BASIC commands. The other routines in the Logical Interface block include:

| | |
|---|---|
| SAVEFILE | -save a file subroutine |
| CVTSCT | -convert # bytes to # sectors |
| SETINX | -create new directory entry, write new start point |
| FNDEND | -find ending address of file |
| CVTTRK | -convert # sectors to # tracks/sectors |
| MULT | -multiply CALC1*CALC2=CALC3 (4 bytes) |
| DIVIDE | -divide CALC1 by CALC2 = CALC3 remainder in CALC1 |
| SYSB | -BASIC !SYS command, loads block four into memory. |

## BLOCK 3 - BASIC INTERFACE ROUTINES

The BASIC Interface Routines include routines to impliment the PDOS BASIC commands. These include !SAVE, !OPEN, !CLOSE, !INPUT, !RUN, !PRINT, and !LIST. (PDOS V5) Other subroutines will set the index counter, manage file control blocks, and end BASIC commands.

## BLOCK 4 - DOS-mode ROUTINES

Block four is loaded into memory at $7A00 thru $7BFF and overwrites the BASIC Interface Routines, block three. Program execution from the cold start location $7A00 will clear the screen, deselect all drives and display the prompt ">". Program execution from the warm start location $7A05 will be identical except that the screen will not be cleared.

After the prompt is displayed, PDOS will await a single letter command. When this is received, the command interpreter will check for the following; L, S, M, R, G, X, or K. These routines are contained in memory. If the command was not one of the above, PDOS will search for a "command file" on disk drive 0. Each command file is named "*****_" where _ is the letter of the command. If the PDOS command interpreter does not find a command file with the proper letter in the name, it will print six question marks ?????? and go to the DOS-mode warm start. If the correct command file is found, it is loaded into the transient overlay area at $7C00 and PDOS jumps to $7C00 to begin execution of the command.

The DOS-mode command interpreter occupies memory to $7A4F. Starting at $7A50 a jump table provides a route to the DOS-mode commands contained in memory. The jump table is as follows:

```
     $7A50 JMP KILL      "K"
     $7A53 JMP LOADC     "L"
     $7A56 JMP SAVEC     "S"
     $7A59 JMP DOMEM     "M"
     $7A5C JMP GO        "G"
     $7A5F JMP EXECUTE   "X"
```

Other routines contained in block four include:

| | |
|---|---|
| LDTRAN | - Load transient command file |
| GETNAME | - Get file name from user and put in MAINFCB |
| GETDEVC | - Get disk drive # from user, put in register A |
| LOADFIL | - Get file name and load that file |
| EXECUTE | - Execute Program/Load file after loading from disk |
| GO | - Get memory address from user and jump to it |
| SAVEC | - Save a Program/Load file |
| KILL | - Delete a directory entry from the VTOC of the disk |

Several useful machine language subroutines are presented here for reference:

STRING - will print a string of ASCII characters.
Format is the same as BASIC, last character must be a $00.

STRING

```
EFED 856C      STRING STA STGAD
EFEF 846D             STY STGAD+1
EFF1 A0FF             LDY #$FF
EFF3 C8        STRL1  INY
EFF4 B16C             LDA (STGAD),Y
EFF6 F006             BEQ STGEND
EFF8 20D2FF           JSR PRINT
EFFB 18               CLC
EFFC 90F5             BCC STRL1
EFFE 60        STGEND RTS
```

••••••

GETNAME - will ask the operator for a file name and store it in MAINFCB as proper input for several PDOS subroutines.

GETNAME

```
3AA3                ;      GET FILE NAME FROM USER

3AA3 A962     GETNAME  LDA #KNAME(L)

3AA5 A03A              LDY #KNAME(H)
3AA7 20E7EF            JSR STRING
3AAA A000              LDY #0
3AAC 20CFFF   GETNLP1  JSR INPUT     GET KEYS FROM PET
3AAF C93A              CMP #':
3AB1 F008              BEQ GETNELP1
3AB3 99A03F            STA MAINFCB,Y
3AB6 C8                INY
3AB7 C007              CPY #7
3AB9 30F1              BMI GETNLP1
3ABB A920     GETNELP1 LDA #$20
3ABD C006     GETNLP2  CPY #6
3ABF 1006              BPL GETNELP2
3AC1 99A03F            STA MAINFCB,Y
3AC4 C8                INY
3AC5 D0F6              BNE GETNLP2
3AC7 20CFFF   GETNELP2 JSR INPUT   GET NEXT KEY,DEVICE #
3ACA 20DB3A            JSR GETDEV3
3ACD 8DB13F            STA MAINFCB+17
3AD0 60                RTS


3A62 0D       KNAME    .BYTE $0D
3A63 46494C45          .BYTE 'FILE? '
3A69 00                .BYTE 0
```