IRWIN TAPE DRIVE SOFTWARE COOKBOOK

Application Note #1

May 24, 1984

Table of Contents

1.00 Introduction

The purpose of this cookbook is to provide reference material for designing, writing, and testing software for the Irwin tape drive models 110, 210, and 310.

This application note assumes that the programmer is familiar with the floppy disk interface and other hardware available. There will be little effort to describe the theory of hardware operation other than what is applicable to the Irwin tape drives. It is also assumed that the programmer has read the Irwin Tape Drive OEM Manual. Use the OEM Manual as a reference for terminology contained in this document. Also, throughout this application note there will be references made to the applications software for the Irwin tape drives. Familiarity with the Irwin software is beneficial, but not necessary.

2.00 Tape Operation and Format

This section discusses the physical tape and its operation. This section also discusses the format of the tape, including track format, block format, sector format, and the format of block 0 (where the format parameters are stored).

2.10 Tape Operation Modes

The Irwin tape drive has three modes of operation: streaming mode, start-stop mode, and in-place-update mode.

The tape is operating in the streaming mode when in constant motion while reading or writing. The streaming mode provides the most efficient operation of the tape because there is no time dedicated to starting and stopping the tape. With the Irwin tape drive streaming, writing 10.35 megabytes to the tape takes only 8 minutes (allowing 1 minute per track, for 8 tracks, with tape streaming from end to end). Programming is most difficult in the streaming mode due to timing requirements during data transfer.

The start-stop mode enables the user to start and stop the drive anywhere on the tape without loss of storage efficiency. Most tape drives operating in the start-stop mode lose storage capacity as the number of starts and stops during writing to tape increases. As these drives write to tape, the time between a stop and start is an inter-record gap. The Irwin tape is pre-formatted and consequently there is no loss in tape storage efficiency. Starting and stopping is useful when processor overhead does not allow the tape to stream. Use the start-stop mode to position the tape to a tape block for selective reading, selective writing, error retries, and in-place-updating.

The Irwin tape drive also provides an in-place-update mode allowing selective writing to any pseudo-sector in any block on the tape. This type of tape operation is a particularly beneficial feature of the Irwin tape drive family. Use the in-place-update mode to maintain and update directories, bad block maps, tape ID blocks, and other files on the tape.

The in-place-update mode gives the Irwin tape drive random access capability. This capability makes the Irwin tape drive an inexpensive, random access, large capacity memory device. The in-place-update mode is not found on most other tape drives.

2.20 Tape Servo Writing

All Irwin tape drives require a servo written tape, before any data can be written to the tape. The Irwin drive has servo write capability. It is important to note that before a tape can be servo written it must be bulk erased. Failure to bulk erase will result in unsuccessful tape servo writing.

2.30 Tape Format and Organization

A servo written tape has been physically and logically divided into tape blocks. Each block contains 11.95 inches of tape providing the user with a capacity of 9,562 bytes unformatted and accessible. The servo information is located at the beginning of each block and is not user accessible. This unformatted capacity is analogous to the unformatted floppy track.

Hence, a servo written tape appears to be a floppy diskette with many more tracks. To be compatible with the floppy disk controller, each tape block must be formatted and organized like a floppy disk track. Thus, format fields such as an index gap, header ID's, header gaps, data fields, and CRC's must be written to the tape.

The type of software that will write the format data onto the tape is similar to the software for a floppy disk. The format data written to both the tape and floppy disk is identical. When formatted, each tape block will contain sectors just like the floppy track.

Depending on the application, the user determines the sector information, including the number of sectors and the sector length. All Irwin software uses eight, 1,024 byte sectors per block. For a floppy disk, a greater number of sectors per track results in more formatting overhead and less useful storage per block. Since a floppy track is equivalent to a tape block, formatting overhead and useful storage considerations also apply to tapes. The eight sector format provides the lowest formatting overhead and the most user available space per block that many floppy controllers and controller chips will allow. (Refer to the NEC 765 and Western Digital 179x specifications and application notes for information on formatting floppy diskette tracks. This information is also applicable to tape blocks).

When formatting the data field, it is best to use the pattern 6Dh as a sector "fill character". The 6Dh data constant is the most difficult pattern to read in the MFM data encoding format. When verified, it provides a good test of tape media and data recovery electronics "quality".

To format the tape with the 5 1/4" MFM data format standard, Irwin recommends the use of write track commands. Use the tape block number in place of the cylinger number in each sector header ID.

2.31 Tape Format

Irwin has set recommended format specifications for the tape cartridges used in the Irwin tape drives. Irwin uses this format extensively. This format allows the most user available space with the least amount of ID header overhead.

The tape is first divided into 8 tracks, numbered 0 through 7. These tracks are organized on the tape in a serpentine fashion; the even numbered tracks are recorded in the forward direction (beginning-of-tape to end-of-tape, or BOT to EOT), while the odd numbered tracks are recorded in the reverse direction (EOT to BOT). Each track is then divided into 158 blocks, or cylinders. While blocks and cylinders are the same, blocks are the logical divisions and cylinders are the physical divisions of the tape. The cylinders are numbered 0 through 157 on each track, while the blocks are numbered 0 through 1,263 across the entire tape. (See Figure I - Tape Layout and Figure II - Track Layout) Each block is further divided into 8 sectors, numbered 0 through 7.

The format of each sector within a tape block is mini-floppy compatible. Each sector has a data area of 1 kilobyte in length, which makes each block a total of 8 kilobytes long. The total capacity of a tape track is therefore 1.294 megabytes and the total capacity of the entire tape is 10.355 megabytes. (See section 2.32, "Block Format" for block format details.)

The format parameters are stored in block 0, which is cylinder 0 of track 0. These parameters include the version number of the formatting program, the format date, the number of tracks per tape, the number of blocks per track, the number of sectors per block, the number of bytes per sector, the application program used on the tape, the volume number and limit, the time and date of the last tape change, and the track and cylinder of the first free block. These parameters are stored in duplicate. (See section 2.33 "Format Parameters - Block 0")

FIGURE I – TAPE LAYOUT

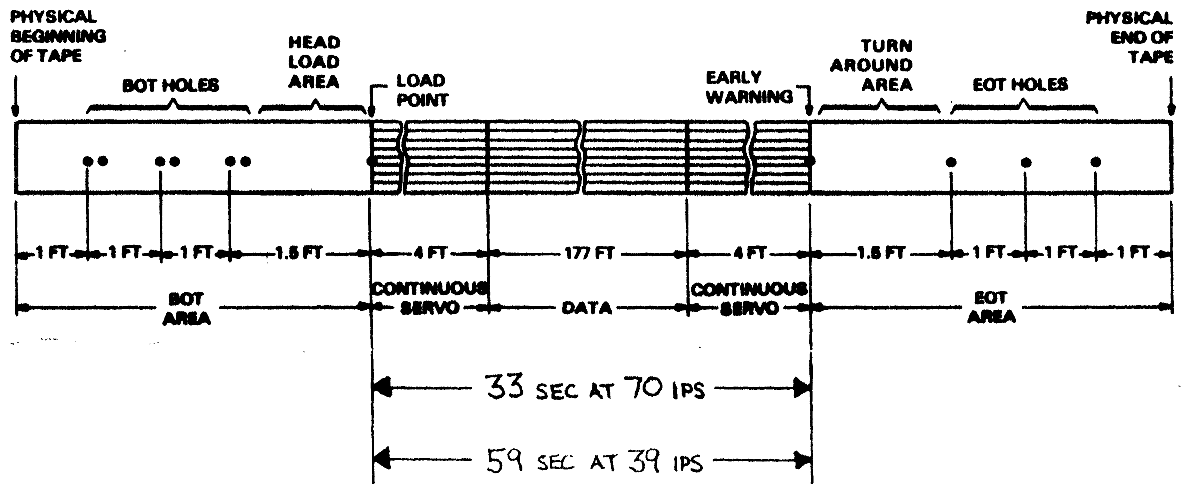
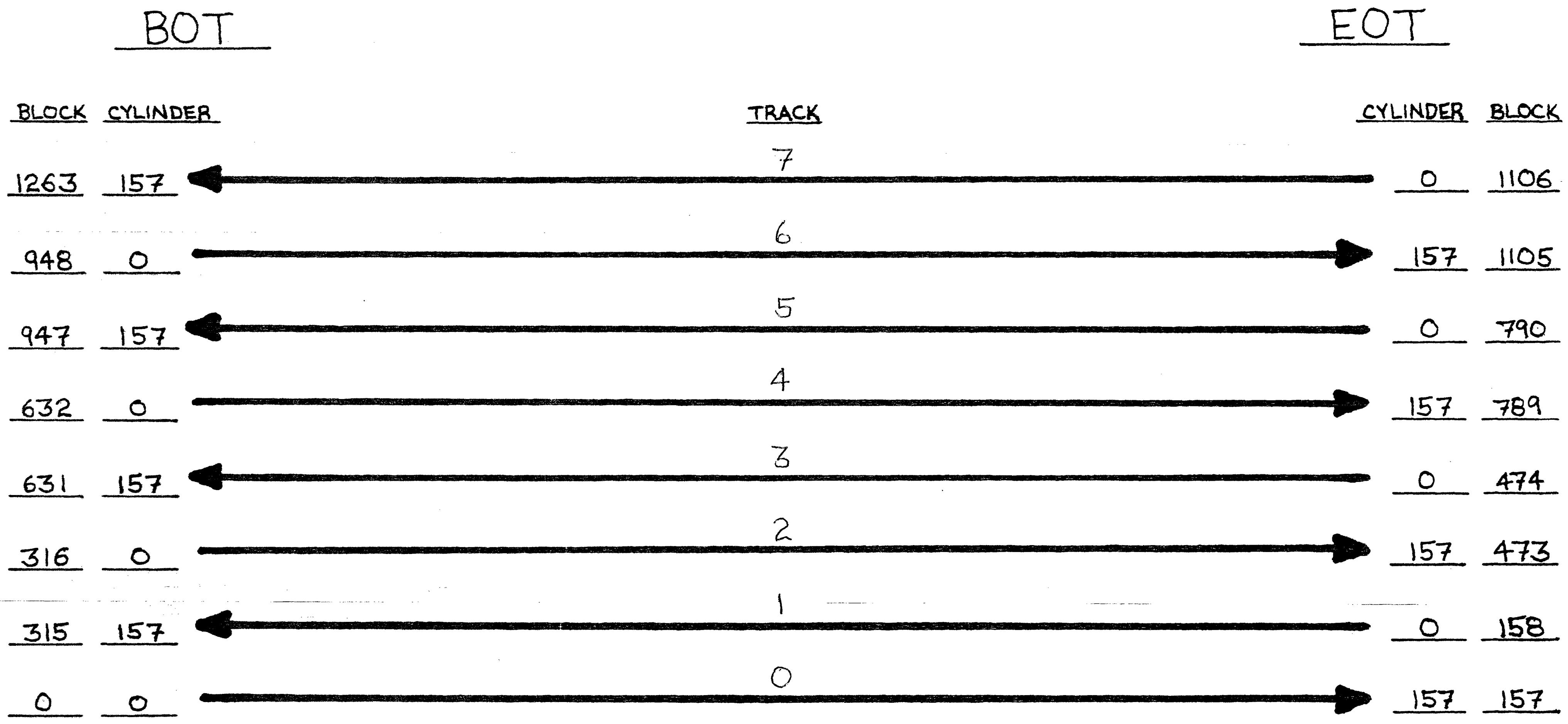


FIGURE II – TRACK LAYOUT



BOT			EOT	
BLOCK	CYLINDER	TRACK	CYLINDER	BLOCK
1263	157	7 ←	0	1106
948	0	6 →	157	1105
947	157	5 ←	0	790
632	0	4 →	157	789
631	157	3 ←	0	474
316	0	2 →	157	473
315	157	1 ←	0	158
0	0	0 →	157	157

2.32 Block Format

Each block of the tape is formatted as follows:

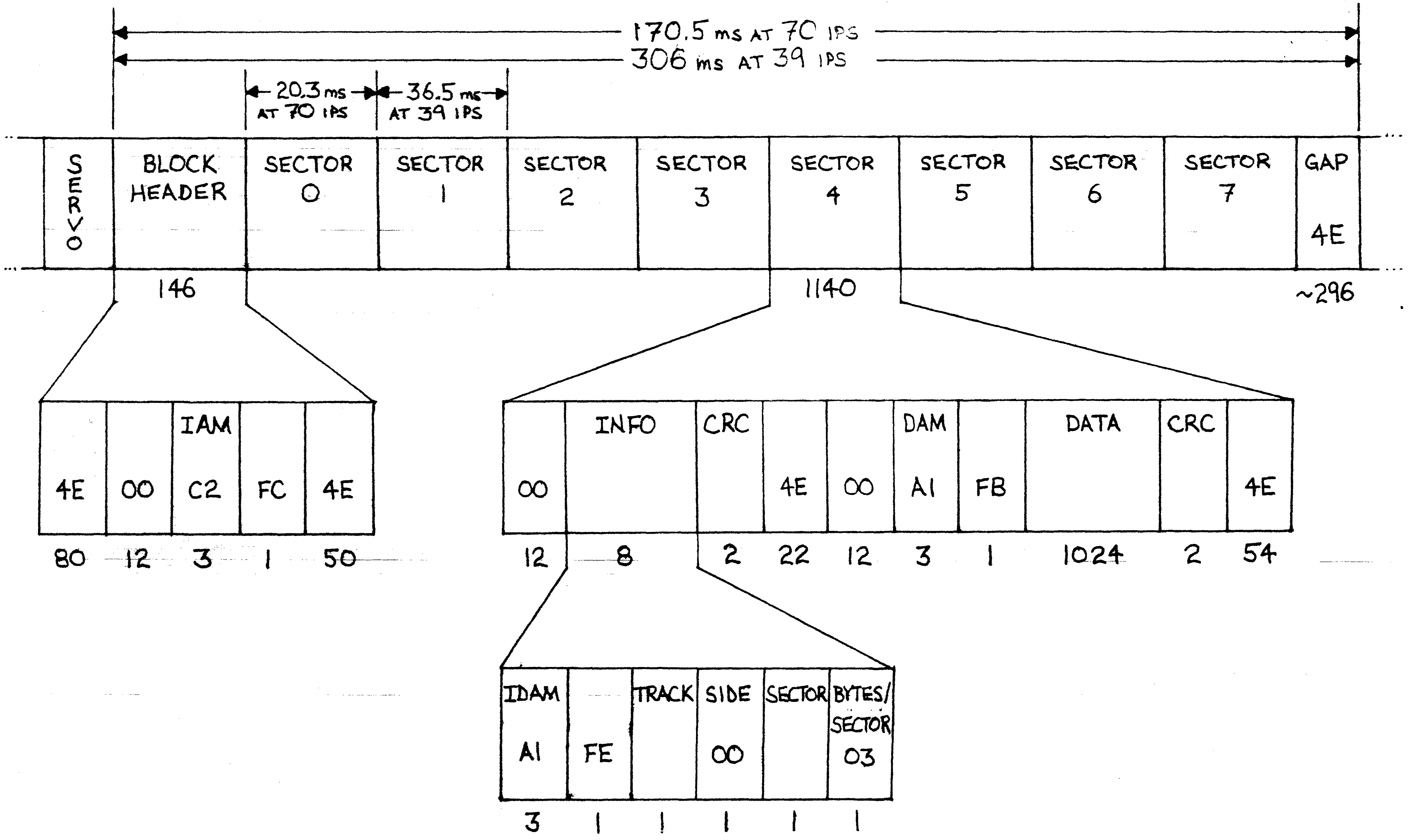
Number of Bytes	Hex Value	
80	4E	
12	00	
3	C2	(IAM - Index Address Mark)
1	FC	
50	4E	

(The following section is repeated 8 times, once for each sector.)

12	00	
3	A1	(IDAM - ID Address Mark)
1	FE	
1		(Track)
1	00	(Side)
1		(Sector)
1	03	(Bytes per Sector Flag)
2	CRC	
22	4E	
12	00	
3	A1	(DAM - Data Address Mark)
1	FB	
1024		(Data)
2	CRC	
54	4E	

On the following page is Figure III - Block Layout. This figure graphically shows the block layout. The top section shows a single block divided into the block header and 8 sectors. The next two sections show the contents of the header and a single sector.

FIGURE III — BLOCK LAYOUT



2.33 Format Parameters - Block 0

Irwin's application software uses block 0, which is cylinder 0 of track 0, to store format parameters and a bad block map. All of this information is stored in duplicate immediately after the format process. Sector 0 contains the format parameters of the tape. Sectors 1 and 2 contain the bad block map. Sector 3 contains nulls. Sectors 4 through 7 are duplicates of sectors 0 through 3.

Although this block 0 format is optional, Irwin recommends adherence to the format to insure interchangeability. The suggested format parameters are listed below. The bad block map is actually 1,264 bytes long (one byte for each block on the tape). Initially, every byte is set to 00h. When a block is found to be bad, the corresponding byte in the bad block map is changed to FFh. Also, since each block has 8 sectors and each byte has 8 bits, this bad block map can potentially be used as a bad sector map.

Description	Bytes	Data Type
Format program name and version number	0-36	ASCII
Date of format (from DOS)	37-44	ASCII
Tracks per tape	45-46	decimal
Blocks per track	47-48	decimal
Sectors per block	49-50	decimal
Bytes per sector	51-52	decimal
Application program version number (major)	53-54	decimal
Application program version number (minor)	55	decimal
Tape use flag (0-unused, 1-FIP, 2-IMAGE, ...)	56	decimal
Volume name	57-69	ASCII
Volume number	70	decimal
Volume limit	71	decimal
Date of last tape change (MM/DD/YYYY)	72-82	ASCII
Time of last tape change (HH:MM)	83-88	ASCII
Track of first free block	89	decimal
Cylinder of first free block	90	decimal
Reserved for application program use	959-1023	

2.40 Tape Positioning

Irwin tape drives have two speeds: 39ips for read/write operations, and 70ips for tape positioning. Data transfer is done at the 39ips speed while tape positioning is done at the 70ips speed. Positioning the tape is accomplished by counting index pulses from the floppy controller. Therefore if the current and desired tape positions are known (position being tape block number), the user can issue a fast motion command and a count of index pulses (each index pulse representing a passing block) to position the tape a few blocks before the desired block. To start data transfer after the tape has been positioned, the user must use the read/write speed.

For example, with a tape block length of 13.47 inches, a tape speed of 70ips, and a stop time of 400ms, it will take about 3 blocks to stop the tape. The user can assume the same 3 blocks for restarting the tape, (at the read/write speed), for a total of 6 blocks overhead. When the tape is going backwards, however, the 3 blocks starting overhead cancels some of the seek length. As an example, to seek block 100 when positioned at block 4, initiate a Move Physically Forward, count 90 index pulses (to get to block 94), issue a Stop Tape, and then initiate a Read Logical Forward. The tape would be positioned within 3 blocks of block 100 where a data transfer could begin.

Some floppy disk controllers and/or controller chips will not allow the host to access the index pulse line of the interface. The NEC 765 is an example of this. The index line is accessed by the NEC 765, but the host can not access the index line information. Therefore, the program can not simply count index pulses. (On the other hand, the Western Digital 179x shows index as a bit "S1" in the Status Register for Type One commands.) The NEC 765 can count index pulses and, if it does not find a disk sector address within two disk revolutions (or index pulses), it will time-out. The user can then issue read commands of invalid sectors to the NEC 765 and wait for a time-out, where every time-out indicates the passage of two tape blocks. If the invalid read commands are chained together, they become an effective index pulse counter. Using the previously mentioned figures, one can calculate the time it takes one block to pass the tape head at the Move Physical speed to be 192.4ms per tape block. In the above example (getting to block 100 from block 4), instead of counting 90 index pulses, a timer routine would time-out 17.316 seconds (90 x 192.4ms). This would be the equivalent time to pass 90 tape blocks under the head at the Move Physical speed of 70ips.

When using these tape positioning techniques, institute some kind of "read tape address" routine when exiting a higher tape speed and/or entering the read/write speed. This will indicate absolute tape position. In most cases the initial tape position will be a within 3 blocks of the desired block so the program will have to perform multiple "read tape addresses" before finally getting to the desired block. In some cases, especially when using the time-out positioning mode, tape speed error, and motor starting and stopping, time can accumulate and cause the tape to overshoot the desired block. If this should happen, it will be necessary to reposition the tape before the desired block. Since this over-shoot error would entail only a few blocks, one or more Pause commands can be used to back the tape up.

For random reading or writing, position the tape a few blocks before the desired block. Continuously read ID's until an ID is found in any sector in the block preceeding the desired block. Next, issue a read or write command. Succeeding sectors or blocks in the same track may then be read from, or written to, in the streaming mode.

2.41 Access Time

The BOT to EOT read/write time is 59 seconds. At positioning speed this is 33 seconds. Use these numbers when calculating access time. The time necessary to read or write the entire tape will be 8 tracks x 59 seconds, or about 8 minutes.

2.50 Verification

Like a floppy drive, the Irwin tape drive has no "on-the-fly" read-after-write capability. Therefore, to insure data recording integrity, it is important to reposition the tape after a write operation and reread the data written. In order to keep the repositioning time to a minimum, read-after-write verification should be done after all of the data has been written.

3.00 Hardware Considerations

This section discusses hardware needs, options, and operations. As these considerations will vary from system to system, use what is applicable to your system.

3.10 Floppy Disk Controller

There are two basic designs of floppy disk controllers. One type of controller (the type we will discuss below) uses a commonly available floppy disk controller chip. The second type of controller is based on discrete logic, a microprocessor (maybe bit-slice), or both.

Most of the floppy controller chips operate alike. The two most popular chips are the Western Digital and NEC chips. The discussion will be confined to these. Basically, these chips have a processor interface on one side and a floppy disk interface on the other.

The most important feature that the floppy disk controller requires is a mechanical and electrical 5 1/4" floppy drive interface. This type of interface is called the Shurgart SA450. Most 5 1/4" floppy disk drives (both full- and half-height, single-sided or double-sided) support this interface.

Areas of floppy disk controller design where incompatibilities arise are in the number of drives supported by the controller and the use of the motor on and side select signals. The Irwin tape drive can be set to 1 of 4 device selects, and does not use the motor on, side select, or direction signals.

Electrically, the Irwin drive has a 5 1/4" half-high footprint and uses the same connector as the SA450 type floppy drive. It fits on an existing daisy-chain and has the same addressability as a floppy drive. If the Irwin tape drive is the last device on the daisy-chain, it must have an Irwin terminator resistor pack (SIP style) installed. Otherwise, the terminator pack must be removed.

Another area of floppy disk controller concern is write precompensation. Irwin recommends 250ns of write precompensation on all blocks. Most controllers have write precompensation as an adjustment because this value varies from drive to drive. Having a write precompensation of less than 250ns may cause an increase in soft errors. Also the type of Data Recovery and/or Data Separator circuit will affect the number of soft errors. In some instances, a write precompensation as low as 125ns worked with no errors on read back. Errors relating to an inadequete write precompensation will appear in the form of CRC errors.

The last area of floppy disk controller concern is the Data Separator or Data Recovery circuit. A Data Separator recovers a serial data stream and the appropriate clock. This type of electronics is commonly found in a discrete logic or microprocessor based controller. Most of the chip-based controllers, such as those based on the NEC 765 and the Western Digital 179x, use a Data Recovery circuit to generate a synchronized data-clock window which directly drives the controller chip. In either case, it is important that both circuits are carefully designed to operate over a wide range of read-data-bit jitter that can be generated by the tape or floppy drive. This can be accomplished using a phase-lock loop design. Reading data from the Irwin tape drive is much like reading data from the inside tracks of a floppy disk. (The data recovery electronics have an easier time on the floppy's outside tracks than the tracks on the inside. This is due to the fact that the bit density rises on the inside tracks.)

3.11 8" Floppy Disk Compatibility Considerations

Eight inch floppy disk controllers are incompatible with the Irwin tape drives for two reasons: the MFM data bit rate is 500KHz as opposed to the required 250KHz, and the data connector and electrical connections are different for the 8" technology.

Both of these incompatibilities can be resolved with hardware modifications in most controllers. In most cases the data rate can be slowed down by halving the clock frequency to the controller chip. With the addition of some hardware logic, the 8" controller can be designed to respond to both data rates, switchable through software. Also, since the 8" controller specifies the same I/O lines used by the Irwin tape drive, a cable adapter can be made to interface the Irwin tape drive to the standard 8", 50-pin ribbon cable standard.

3.20 DMA - Direct Memory Access

In most microcomputers, a single processor is used to perform data transfers between all peripherals. In disk and tape controllers, the processor is a slave of that controller during the time of data transfer because of strict timing requirements. These timing requirements are needed to insure no loss of data due to a busy processor.

In a backup situation with a single processor, the processor initially does a disk access to retrieve data, and secondly, a tape access to store it. When the processor is reading information from the disk, the tape is motionless because the processor can not write to the tape at the same time it is reading from the disk. The flow of data is from disk-to-memory-to-tape as the microprocessor sequentially moves the data.

DMA is the hardware ability (under software control) to perform peripheral-to-memory (or vice versa) operations. Microcomputers have one or more DMA channels that can be used simultaneously. In a two channel DMA situation, one DMA channel would move data from disk-to-memory, and the other from memory-to-tape. In some instances, where both DMA channels cannot use the same memory buffer simultaneously, it is necessary to use two memory buffers and alternate them between the tape and disk DMA channels. In a multiple channel DMA system, the processor has the job of controlling the DMA device and managing memory. The size of the memory buffer is dependent on system timing between the tape and disk. In a single channel DMA controller, one of the peripherals (tape or disk) would have DMA capability while the other would depend on the host processor to move data from memory to peripheral.

3.30 Interrupts

Enabled interrupts are a peripheral's ability to direct processor control to the interrupt routine. Interrupts that are disabled, or turned off, are ignored by the processor. During a tape-to-disk data transfer, when host timing is very critical (even with DMA), it is important that no other peripherals interrupt the host processor. An interrupt may cause a stop of data transfer and possibly an untimely tape reposition. Therefore, make sure the software disables all of the hardware interrupts. For more information on software interrupts, see section 4.32, "Software Interrupts".

3.40 Addressability

To operate the Irwin tape drive on an SA450 interface, low level device software must be capable of allowing the hardware to perform primitive floppy disk operations. These operations include drive selection, head stepping, track data transfers, and controller operation interruption. For these reasons the software needs addressability to the controller hardware.

This controller accessibility and the amount of access is dependent on the design of the controller. A very smart controller that executes high level commands such as read/write sector, data block, or file, from the host may not be usable with the Irwin tape drive because the controller is incapable of performing low-level operations. Most controllers of this type have their own microprocessor and program ROM. They interface to the host through these high-level commands and perform the necessary low-level operations as they are needed in order to accomplish the high-level commands. If enough information is available about the design of such a controller, its ROM may be reprogrammed to directly provide the low-level commands needed by the Irwin tape drive.

3.50 Memory

A tape drive application program typically takes 20K to 60K of memory, not including buffers. Depending on the hardware configuration, buffer requirements may be small or quite large. In a system with DMA, buffers should be about 8K to 16K. In a non-DMA system, the buffers should be as large as possible to keep tape repositioning and start/stopping to a minimum.

3.60 Power Supply

It is important that the power supply provides for the power requirements of the Irwin tape drive. An inadequate supply of power to the tape drive may cause a variety of problems. Software problems, failure to stop a tape at the end of the cartridge, and other hardware problems may be due to the inadequate supply of power. Power supply problems are the most frequent reasons for tape drive failure.

4.00 Software Considerations

This section considers application program design from a conceptual standpoint by discussing low-level device drivers, drive needs, and application considerations. Considerations will be from a systems standpoint. An attempt will be made to lead the programmer through tape drive program design.

Irwin has designed a tape drive that uses a floppy disk controller to interface the tape drive to a host system. To minimize the cost of the tape drive, Irwin uses an existing floppy disk controller to interface the tape drive rather than require a separate tape controller. It is not Irwin's intention to provide a tape mechanism that would use existing floppy disk software to control it, but rather to provide a tape drive with its own software identity that would operate through a floppy disk controller. This method will keep cost down by utilizing existing hardware, but will recognize that the device is a tape drive, not a floppy drive, and that new software can and should take advantage of that fact.

4.10 General Software Requirements

While the Irwin tape drive is different from a floppy disk drive, many aspects of the software are the same. This results from the fact that both use the same controller. Since the controller was designed for floppy disks, it is necessary to program it to "think" like a tape drive controller.

The software is divided into two areas of operation: data transfer and tape positioning. Data transfer operations will use software identical to floppy software, while tape positioning will require a different set of software.

In general, software for implementing the Irwin tape drive needs to convert a floppy disk controller into a tape controller. This involves careful programming to convert the floppy controller or floppy controller chip to do tape controller functions. The popular floppy disk controller chips, the Western Digital 179x and NEC 765, can present conversion problems if careful programming techniques are not applied. A good understanding of the operation of these controller chips is a prerequisite for doing the low-level device programming.

4.20 Software/Hardware Interaction Concerns

When programming a tape or disk peripheral, the peripheral requires that data be transferred at a specified time, rate, and amount. Speed performance is dependent on the interaction between hardware and software and should be optimized.

Tape peripheral performance is optimized when data tranfer is performed continuously. This means that the program should keep the tape moving. Stopping and pausing are lengthly operations due to the relatively long starting and stopping times of the tape.

There are three items of software/hardware interaction that the programmer needs to address: interrupts, DMA, and memory buffer usage.

It is necessary to know which interrupts are used by the system and when they will happen. If an untimely interrupt occurs during tape data transfer, the tape may not be serviced in time, causing a reposition. To prevent an unnecessary tape reposition the interrupt should be disabled.

The programmer will have to determine which interrupts to disable. Since the controller will always interrupt to the floppy disk handler, it will be necessary to "patch in" the address of a tape interrupt handler. Other interrupt routines, depending on operating system and usage, may have to be "patched" in order to trap interrupts that may reset or change the status of the floppy disk controller. An example of this is IBM PC-DOS ROM BIOS interrupt 13. It is also important to restore all interrupt routine pointers to their original states.

DMA is necessary for streaming tape operation. Knowledge of your system's particular DMA scheme and its operation is a prerequisite to writing data transfer routines. Considerations here include the speed of the DMA channel(s), setup time, and buffer transfer design. Factors in the buffer transfer design include the number of buffers to use, the buffer size, the buffer address, and buffer speed.

Memory buffer usage is a function of the DMA hardware architecture and the amount of memory available for buffer usage. If the system has no DMA, then use as large a buffer as memory will permit. A large buffer will transfer as much data as possible to-or-from the tape and as a result will keep the number of starts and stops to a minimum. If the system has DMA, the buffer size will depend on the DMA architecture and speed.

With the aid of DMA and interrupts, a buffer management scheme using overlapping I/O is a good method to keep tape repositioning to a minimum. In this scheme there are one or more memory buffers used in the data transfer. One DMA channel continuously reads data into the buffer(s) from the tape or disk peripheral while another DMA channel writes data from the one or more buffers. The two DMA channels operate independently, one filling memory, one dumping memory, and at completion both interrupting the processor.

A buffer management program controls the DMA and memory. To optimize overlapping I/O, use as much memory as possible for the buffer(s). Also, remember that hard disk data transfer, in most cases, will be faster than transfer to-or-from the tape.

Because hard disk data transfer is faster than tape data transfer, an attempt should be made to keep the buffer(s) full of data for a disk-to-tape operation, and keep the buffer(s) empty in the case of a tape-to-disk operation. In some cases, usually involving processor overhead (directory work, file searches, interrupts, and hard disk errors (retries)), the average tape data transfer rate becomes greater than the hard disk data transfer rate. If this happens, the tape will have to stop and reposition while the disk catches up. In order to keep this occurence to a minimum, it is good practice to perform hard disk operations until all the buffer(s) are full or empty (depending if you are transferring data to or from the tape). These hard disk operations should be performed when the tape stops or repositions due to a faster tape data transfer. This will enable the hard disk to get a head start on data transfer when the tape is put back into motion.

4.30 Software Design

Most application program design is done with the "top-down" approach. In a top-down approach the user's needs are determined first. Implementation is then performed by specifying software modules through hierarchical levels progressing down to the primitive operations known as the low-level device drivers. Tape application programs are not significantly different, however floppy controller hardware influences program design. It is therefore necessary to do a "bottom-up" program specification while also doing a top-down specification. In this bottom-up approach the low-level routines are considered first. Implementation is then performed by specifying software modules through hierarchical levels progressing up to the user interface. Where the two designs meet is the optimal trade off between programming goals, user's needs, hardware considerations, and operation speed.

An example of the merged approach can be illustrated in the design of a backup program. A backup program is designed to allow the user a fast and easy hard-disk-to-tape backup which will be done on a frequent basis. To design a backup program, first you must know how the controller hardware works and its capabilities. In addition, you must know how the operating system and file system interfaces work. Having optimized the program for backup, the restore function may take longer, which is acceptable since it is used less frequently. In the process of examining both the top-down and bottom-up designs, you will make decisions regarding the user interface, information to be backed up (files, directories, or the entire disk), how tape movement and repositioning is to be done, and how the DMA and memory are to be used.

4.31 Software Transportability

Software transportability is another consideration when writing programs of any kind. Transportability means the ability to export of software across hardware, operating system, and file system boundaries. Hardware boundaries are crossed by low-level device routines, usually written in assembler, with other programming done in a high-level language. Operating system and file system boundaries are crossed through careful program design which uses a modular appoach and keeps all system dependencies in a minimum of program modules.

4.32 Software Interrupts

One particular concern with operating systems is software interrupts. Some of these interrupts interact directly with the floppy disk controller, resetting the controller or its parameters, causing loss of controller initialization with respect to the tape application program. An example of this is the IBM PC-DOS ROM BIOS Interrupt 13 that occurs after a hard disk read error. This interrupt routine recalibrates both the hard disk and floppy disk drives. The recalibration causes the tape program to "forget" where the NEC 765's track register contents are. This in turn will affect sending commands to the drive.

Another consideration dealing with software interrupts is a general policy of trapping unwanted interrupts and redirecting them to a new handler. The ideas and philosophies will differ with hardware, operating systems, and file systems.

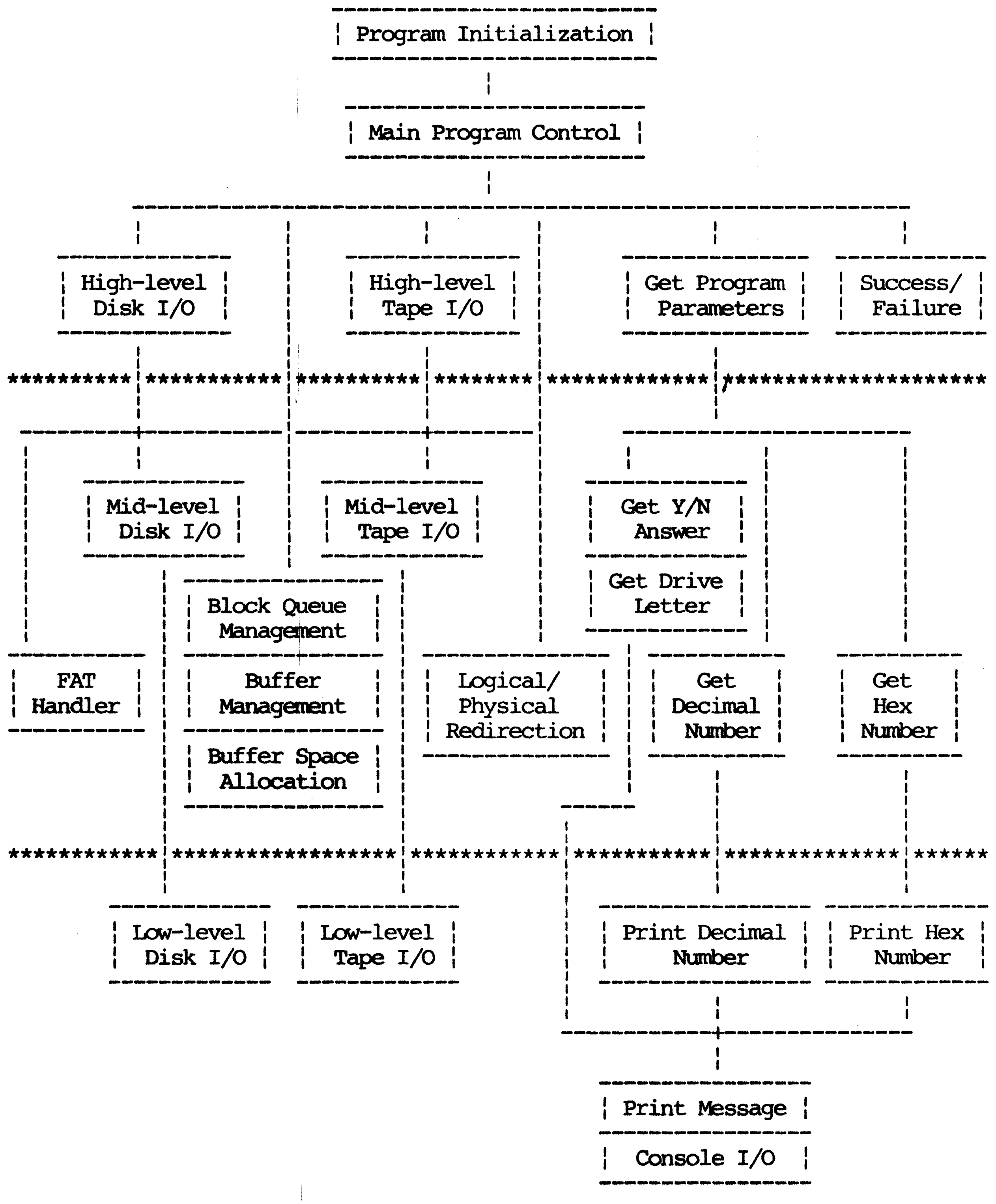
4.40 Software Design Example

We begin the software design process with a top-down design procedure to determine the general flow of the program. A flow-chart showing the program outline can be very helpful. (See Figure IV - IMAGE Flow-Chart. The organization of the Irwin IMAGE program which performs an image backup from disk-to-tape and an image restore from tape-to-disk is shown in this figure.)

Using a top-down approach, at the top level is the user interface, if any, followed by the main structure of the program. Further down the flowchart are the data handling and manipulation routines. This middle level of the program will deal with the operating system being used, the desired organization on tape, and considerations about the specific hardware used. At this middle level the designer should probably move away from the top-down approach towards the bottom-up approach. At the bottom level of the program are the low-level I/O and hardware interfacing routines.

In most tape application programs there will be similar main program tasks. Following Figure IV is a list of some of these programming tasks. This list is not meant to be complete, nor does any program depend on the existence of the listed routines.

FIGURE IV - IMAGE FLOWCHART



User Interface
These routines "talk" to the user, get any information needed to operate the program, and output any information back to the user. All error messages should be handled through the user interface. Ergonomics and human factors should be taken into consideration when designing the user interface routines.

Command Parser
This routine deciphers the user's input and passes the needed parameters and program control to the proper routine.

Front End Calculations
These routines perform calculations needed and pass the information to the calling program. The information can be the number of tapes needed, which bytes to transfer, which flags to set, and which data pointers and buffers to set up.

Operating System Interface
These routines read and write data to and from the disk through the operating system and the file system. This reading and writing can be done on a file, logical allocation unit, or disk sector basis. Basically, these routines will be an interface to the system and file services provided by the operating system.

Tape File/Format Manager
These routines interface to intermediate level tape routines and operating system routines. Actual calls to read and write to-or-from the tape are done here. These routines manage tape data flow and tape format. Most of the application program code will be in these routines.

Buffer Manager
These routines organize and manage the buffers. These routines will interface with the DMA (if available). Timing will be a major consideration in this routine.

4.41 Mid-Level Tape I/O Routines

The mid-level tape I/O routines facilitate the transportability of the application software through operating system and hardware boundaries. These routines will be called by the tape file manager and buffer manager. These routines make the tape look like one continuous stream of 1,264 tape blocks (158 blocks/track x 8 tracks). All tape and tape head positioning is automatically done. On the following pages is a list of the Irwin TP1 routines that are suggested for use. (TP1 denotes mid-level routine.)

TP1ONL

Get Drive Number

Calling Parameters

None

Return Parameters

None

Description

This routine gets the tape drive's physical unit number. The number is stored internally for future reference. This routine is called once per program to insure that the appropriate hardware (the controller and the tape drive) is on-line.

TP1OFL

Remove Drive Line

Calling Parameters

None

Return Parameters

None

Description

This routine removes the drive from the line. This routine is called once per program.

TP1REDMNT

Read Mount

Calling Parameters

None

Return Parameters

None

Description

This routine "mounts" or prepares a new tape for reading and assumes the tape drive is already on-line. The user is asked to insert a tape cartridge, if necessary. Then, a seek load point command and a seek track 0 command are issued. This routine is called once per tape.

TP1WRTMNT

Write Mount

Calling Parameters

None

Return Parameters

None

Description

This routine "mounts" or prepares a new tape for writing and assumes the tape drive is already on-line. The user is asked to insert a tape cartridge, if necessary. Then, a seek load point command and a seek track 0 command are issued. The tape cartridge is also checked to see if the write protect tab is set. This routine is called once per tape.

TP1RED

Read Block

Calling Parameters

buffer address	- Address of the buffer for the data to be read.
block	- Block number.

Return Parameters

None

Description

This routine reads a block from the tape. All of the necessary tape motion is handled internally. This routine fails if it is unable to initiate the read. A failure return means that a retry will also fail. Any less serious problems will be returned by TP1REDWT. When this routine returns, the tape will continue motion.

TP1REDWT

Read Wait

Calling Parameters

None

Return Parameters

None

Description

This routine waits for the completion or error return from a TP1RED call. Errors are returned to the calling program as return codes.

TP1WRT

Write Block

Calling Parameters

buffer address	- Address of the buffer for the data to be written.
block	- Block number.

Return Parameters

None

Description

This routine writes a block from the tape. All of the necessary tape motion is handled internally. This routine fails if it is unable to initiate the write. A failure return means that a retry will also fail. Any less serious problems will be returned by TP1WRTWT. When this routine returns, the tape will continue motion.

TP1WRTWT

Write Wait

Calling Parameters

None .

Return Parameters

None

Description

This routine waits for the completion or error return from a TP1WRT call. Errors are returned to the calling program as return codes.

TP1PAUSE

Pause

Calling Parameters

None

Return Parameters

None

Description

This routine backspaces the tape once. This routine is used when the streaming mode is ending to position the tape so it will be ready to start streaming where it left off.

TP1STOP

Stop

Calling Parameters

None

Return Parameters

None

Description

This routine stops the tape. This routine is used when the streaming mode is ending.

TP1CONT

Continue

Calling Parameters

None

Return Parameters

None

Description

This routine starts the tape with a forward motion. This routine is called after TP1PAUSE or TP1STOP to restart the tape motion.

TP1DMNT

Dismount

Calling Parameters

None

Return Parameters

None

Description

This routine "dismounts" or "unloads" the tape. A seek load point command is issued without waiting for completion. This routine is called once per tape.

4.42 Low-Level Device Routines

The low-level device routines provide tape operation on the hardware. These routines are custom written for the hardware involved. They are typically written in assembler and are called only by the mid-level routines. On the following pages is a list of the Irwin TPO routines that are suggested for use. (TPO denotes low-level routine.)

TPO Return Codes

These are the suggested return codes for the TPO low-level tape I/O routines. These are the return codes that have been implemented at Irwin. The codes that each routine will receive are dependent on the implementation of the TPO routines which is in turn, dependent on the system.

Description	Code Number
Still busy, waiting for not busy failed	-01
Command accepted	00
Command not accepted	01
Receive time-out, read controller error	02
Send time-out, write controller error	03
Controller error, invalid controller response	04
Record not found, no valid ID read	05
Sector CRC error, checksum error on record	06
DMA error, DMA processor missed DRQ, data lost	07
Tape is write protected	08
ID not found, no valid ID read	09
Interrupt time-out, I/O never properly completed	10
DMA boundary, internal boundary alignment problem	11
Error code out of range, internal problem with program	?? (other)

TPOINI

Controller Initialization

Calling Parameters

load time	- Head load time in ms (suggest 4).
unload time	- Head unload time in ms (suggest 480).
step rate	- Step speed in ms (suggest 6).
io gap	- Gap length to use for read/write (suggest 017h).

Return Parameters

None

Description

This routine initializes the floppy disk controller for tape usage. The floppy hardware interrupt vectors are saved and replaced with new interrupt vectors for the tape routines. Depending on the controller, the calling parameters are passed on to the controller, saved for reference use, or just ignored. Any type of software or hardware initialization that needs to be done once per program should be done in this routine.

TPOTRM

Controller Termination

Calling Parameters

None

Return Parameters

None

Description

This routine terminates the tape's usage of the floppy disk controller. All of initialization processes are reversed in this routine. Most notably, the floppy hardware interrupt vectors are replaced.

TPOONL

Drive Select

__Calling Parameters__

drive — Drive number.

__Return Parameters__

None

__Description__

This routine selects or initializes the chosen drive. This is performed with a controller reset and a recalibrate command. It must be called once prior to the first call to any other function with the same specified "drive" parameter. This routine may be called again after TPOOFL.

TPOOFL

Drive Unselect

__Calling Parameters__

drive — Drive number.

__Return Parameters__

None

__Description__

This routine unselects the specified drive. It must be called once after the last call to any other function with the same specified "drive" parameter. This routine must be called before TPOTRM. (In many systems this routine may do nothing or not even exist.)

TPORECAL

Recalibrate

Calling Parameters

drive	- Drive number.

Return Parameters

None

Description

This routine attempts to "awaken" the drive with a controller reset and a recalibrate command. This routine is performed automatically if the drive needs it as a result of an error.

TPORESET

Controller Reset

Calling Parameters

None

Return Parameters

None

Description

This routine attempts to "awaken" the controller with a controller reset command. This routine is performed automatically if the controller needs it as a result of an error.

TPOBUSY

Check for Busy

Calling Parameters

drive	- Drive number.

Return Parameters

busy	- Drive busy flag.

Description

This routine checks whether the specified drive is busy or not.

TPOCOMM

Issue Command

Calling Parameters

drive	- Drive number.
steps	- Number of step pulses in the command. (See Step Pulse Command List on the following page.)
wait/status	- Flag meaning "wait till end and report status".

Return Parameters

None

Description

This routine issues the command which corresponds to the number of step pulses specified. (See Step Pulse Command List on the following page.) If the wait/status flag is set, the routine will wait until the command is executed and return with the status in the return code.

Step Pulse Command List

Command	Number of Pulses
Return busy status	0
Stop tape	2
Pause	3
Seek load point	4
Move physically forward	5
Move physically reverse	6
Report normal completion	7
Report drive presence	8
Report end-of-tape status	9
Report beginning-of-tape status	10
Report cartridge presence	11
Report track found	12
Report new cartridge	13
Move logically reverse	14
Move logically forward	15
Turn on second pulse	16
Turn off second pulse	17
Seek track n (0 <= n <= 7)	20 + n
Write servo	31

TPOREDI

Initiate Read

Calling Parameters

drive	- Drive number.
buffer address	- Address of the buffer for the data to be read.
cylinder	- Cylinder number.
sector	- Sector number.
sector count	- Number of sectors to be read.

Return Parameters

None

Description

This routine initiates a read to the controller. If the system has DMA, the routine returns immediately and reports any errors. If the system does not have DMA, the routine returns after the read and saves the error code to be reported later by TPOIOWT.

TPORED

Read

Calling Parameters

drive	- Drive number.
buffer address	- Address of the buffer for the data to be read.
cylinder	- Cylinder number.
sector	- Sector number.
sector count	- Number of sectors to be read.

Return Parameters

None

Description

This routine performs an entire read. This is accomplished through a call to TPOREDI and a call to TPOIOWT. Any errors are returned immediately.

TPOWRTI

Initiate Write

Calling Parameters

drive	- Drive number.
buffer address	- Address of the buffer for the data to be written.
cylinder	- Cylinder number.
sector	- Sector number.
sector count	- Number of sectors to write.

Return Parameters

None

Description

This routine initiates a write to the controller. If the system has DMA, the routine returns immediately and reports any errors. If the system does not have DMA, the routine returns after the write saves the error code to be reported later by TPOIOWT.

TPOWRT

Write

Calling Parameters

drive	- Drive number.
buffer address	- Address of the buffer for the data to be written.
cylinder	- Cylinder number.
sector	- Sector number.
sector count	- Number of sectors to write.

Return Parameters

None

Description

This routine performs an entire write. The write is accomplished through a call to TPOWRTI and a call to TPOIOWT. Any errors are reported immediately.

TPOIOWT

I/O Status (for Wait)

Calling Parameters

drive	- Drive number.

Return Parameters

None

Description

This routine waits for the completion of the TPOREDI/TPOWRTI I/O. This could signal the end of I/O activity or report any errors to the calling program.

TPOSNS

Write Protect Status

Calling Parameters

drive	- Drive number.

Return Parameters

protected	- Write protect status.

Description

This routine indicates the write protect status of the tape cartridge.

TPONDX

Count Index Pulses

Calling Parameters

drive	- Drive number.
pulse count	- Number of pulses to count.

Return Parameters

None

Description

This routine counts the specified number of index pulses and returns. In an implementation where only even numbers are counted, odd numbers are rounded down.

TPOFRMFL

Fill Format Buffer

Calling Parameters

buffer address	- Address of the buffer to be used in formatting.
sector length	- Length of the sectors.
sector count	- Number of sectors.
format gap	- Gap length actually written (suggest 034h).

Return Parameters

None

Description

This routine fills the buffer that is used to format the tape. The buffer length must be more than the track length. (Twice the track length is sufficient, as it depends on the gap length.)

TPOFRMT

Format

Calling Parameters

drive	- Drive number.
buffer address	- Address of the buffer to be used in formatting.
cylinder	- Cylinder number.
sector length	- Length of the sectors.
sector count	- Number of sectors.
format gap	- Gap length actually written (suggest 034h).

Return Parameters

None

Description

This routine formats the specified number of sectors in the specified block of the tape. The routine TPOFRMFL must be called before this routine can be executed. The calling parameters "count" and "format gap" must be the same in this routine call as they were in the call to TPOFRMFL. As in TPOFRMFL, the buffer length must be longer than the track length. (Twice the track length is sufficient, as it depends on the gap length.)

TPOID

Read ID

Calling Parameters

drive	- Drive number.

Return Parameters

cylinder	- Cylinder number.
sector	- Sector number.

Description

This routine reads the current ID.

5.00 Considerations for the Western Digital WD179x and WD279x Series

This section discusses the special considerations needed when using floppy disk controllers based on the Western Digital series of floppy disk controller chips. The following is a list of the Western Digital commands used in the TPO routines and the command parameters:

(Parameter options will vary between Western Digital models 1791, 1792, 1793, 1794 and models 1795 and 1797. The variations deal with the difference in the handling of side select options.)

Restore & Seek	- load head at beginning operation, no verify, 3ms stepping rate
Read & Write Sector	- number of sectors to transfer, select side zero or set sector length, set head delay to 0, set side select update to zero or disable side compare, set data address mark when writing
Write Track	- set head delay to zero, set side select update to zero
Force Interupt	- set as needed
Read ID	- set head delay to 0, set side select update to zero

5.10 Using the Western Digital Commands

The Western Digital commands are used in the following ways:

Restore
The tape drive executes a simulated recalibrate which is a good test to see if the drive is "awake". The Western Digital (WD) internal track register is set to zero. This command is used in the TPOONL and TPORECAL low-level device drivers.

Seek
The Seek command is used to issue the command pulse train to cause the Irwin tape drive to execute its set of commands. This command is issued by the TPOCOMM low-level device driver. Since the WD internal track register is accessible to the software, the Irwin seeking philosophy is to zero out the track register, load the data register with the desired tape command pulse number, and then execute the seek command.

Read Sector/Write Sector
These two commands perform the actual reading and writing of sectors in a tape block. They are used in low level device drivers TPOREDI and TPOWRTI, respectively. Software used in this area will be similar to floppy data transfer software.

Western Digital allows single or multiple sector data transfer operations. When executing a multiple sector operation it should be noted that the device driver must keep the multiple sector count (count of sectors that have read or written). The reason for this is that the Western Digital, in multiple operation mode, continues to read sectors until the program issues a force interrupt instruction or the Western Digital chip times out when it cannot find the next sector because it does not exist. This time-out will occur after not finding the next sector after 5 index pulses or 5 tape blocks going by. If this time-out happens tape repositioning would be necessary.

In some systems, timing considerations dictate that sector counting cannot be done in software. This being the case, Irwin recommends that single sector operations be used. The programming strategy would be to set up a large buffer to transfer data to/from the tape and then issue single sector data transfer commands to the Western Digital chip in succession between sectors. The Western Digital chip will accept a read or write next sector command between the end of a previous sector and the beginning of the next sector with the tape block formatted with no sector interweave.

Write Track
Used in low-level device driver, TPOFRMFL, to format a single block. Software implementation similar to that for a floppy disk.

Force Interrupt
This command is used to force hardware interrupts to the processor. It is used in TPOREDI and TPOWRTI to terminate multiple sector data transfer operations. It is also used to "awaken" the WD chip in error and hangup situations.

Read ID
Used in low-level device driver TPOID to find next ID on tape for tape positioning.

5.20 Tape Positioning with the Western Digital

Bit 1 of the type 1 status register continuously reflects the condition of the index line. To position the tape, the tape positioning routine needs only to count the passing tape blocks. This can be done by monitoring the index status bit and counting the index pulses coming from the tape drive.

6.00 Considerations for the NEC 765 Controller.

This section discusses the special considerations needed when using the NEC 765 floppy disk controller. The following is a list of the NEC 765 commands used in the TPO routines and the command parameters:

Recalibrate	- drive unit select
Specify	- step rate, head load time, head unload time, DMA mode
Sense Drive Status	- drive unit select, head select zero,
Seek	- drive unit select, head select zero, cylinder number
Read Data, Write Data	- drive unit select, head select zero, cylinder number, head number zero, sector number, sector length, last sector operation, VCO sync time, DTL - user defined data length
Read ID	- drive unit select, head select zero, select MFM mode
Format a Track	- drive unit select, head select zero, sector length, sector/track, gap length, format data constant

6.10 Using the NEC 765 Commands

These commands are used in the following ways:

Recalibrate
The tape drive executes a simulated recalibration which is a good test to see if drive is "awake". The NEC track register is set to zero which alleviates programming problems when the software gets confused as to which track it thinks it is on. This command is used in TPOONL and TPORECAL low-level device drivers.

Sense Interrupt Status
The NEC chip sends back an interrupt after the completion of a command, a change in status of the ready line, or during the execution phase in non-DMA mode. When an interrupt is acknowledged, program control should pass to the software interrupt handling routine. Interrupts not reset by inherent command operation must be reset by the sense interrupt command. Sense interrupt status is generally used after a seek or a recalibrate command which returns completion information and present cylinder (track) number. This information is useful to verify that the proper command was sent to the tape drive.

It should be noted that Irwin has observed multiple processor hardware interrupts after Recalibrate commands with non-contigious drive addresses, such as having two drives addressed 0 and 2 with no drives existing for drive selects 1 and 3 (either floppy or tape drive). When this happens interrupts get nested, and the software is unable to handle the interrupts. Therefore, it is a good idea to to execute multiple Sense Interrupt commands until you get an invalid interrupt response. This technique will always clear out the interrupt queue.

Specify
The Specify command is used by TPOINI and TPOTRM to initialize the step rate, head load and unload time, and DMA mode.

Sense Drive Status
The Sense Drive Status command is used to monitor the status of the Track 0 and Write Protect lines of the Irwin tape drive. This command is used throughout the low-level device drivers.

Seek
The Seek command is used to issue a command pulse train to cause the Irwin tape drive to execute its set of commands. This command is issued by TPOCOMM low-level device driver. Because the NEC 765 has an internal track register, not accessible to the host processor, the application program must monitor the track number the NEC is currently "on". (When the NEC is "on" a track, the internal track register contains that track number, but the track number does not correspond to the tape track number.) The reason for this is that the program can determine the correct track number for the NEC chip to seek to when a command is to be issued. For example, if the NEC 765 is "on" track 24 and you wish to issue a pause command to the Irwin drive, 3 pulses, then you would seek to track 27.

To keep the track register in the NEC 765 valid (no track number greater than 77 or less than 0), Irwin recommends a seeking philosophy of keeping the track number as close to 38 as possible (38 is 1/2 the distance to track 77). Therefore, if the present track is less than 38, then seek to the present track number + n pulses for the desired command. If the track is greater than 38 seek to the present track number-n pulses.

Read Data/Write Data
These two commands perform the actual reading and writing of sectors in a tape block. They are used in low level device drivers TPOREDI and TPOWRTI, respectively. Software used in this area will be similar to floppy data transfer software.

Read ID
Similar to Read and Write Data. This command is used in TPOID to find next id on tape and in TPONDX to count 2 index pulses.

Format a Track
Used in the low-level device driver, TPOFRMFL, to format a single block. Software implementation is similar to that for a floppy disk.

6.20 Tape Positioning with the NEC 765

There are two ways to perform tape positioning with the NEC 765.

The first method of tape positioning involves using the Read ID command while the tape is at read/write speed (39ips) and picking up the next block/sector header that goes under the tape.

The second method of tape positioning involves moving the tape at high speed (70ips), when data cannot be read, and then issuing a Read ID command. After two revolutions of the disk, or in this case, two blocks passing under the tape head, the NEC 765 will time-out with an error condition resulting from the fact that the controller could not read an ID at the high tape speed. Not being concerned with the error condition, the tape positioning routine can count the two tape blocks that went by, then reissue the Read ID command if it wishes to let another two tape blocks go by, and so on until the proper block count comes up.

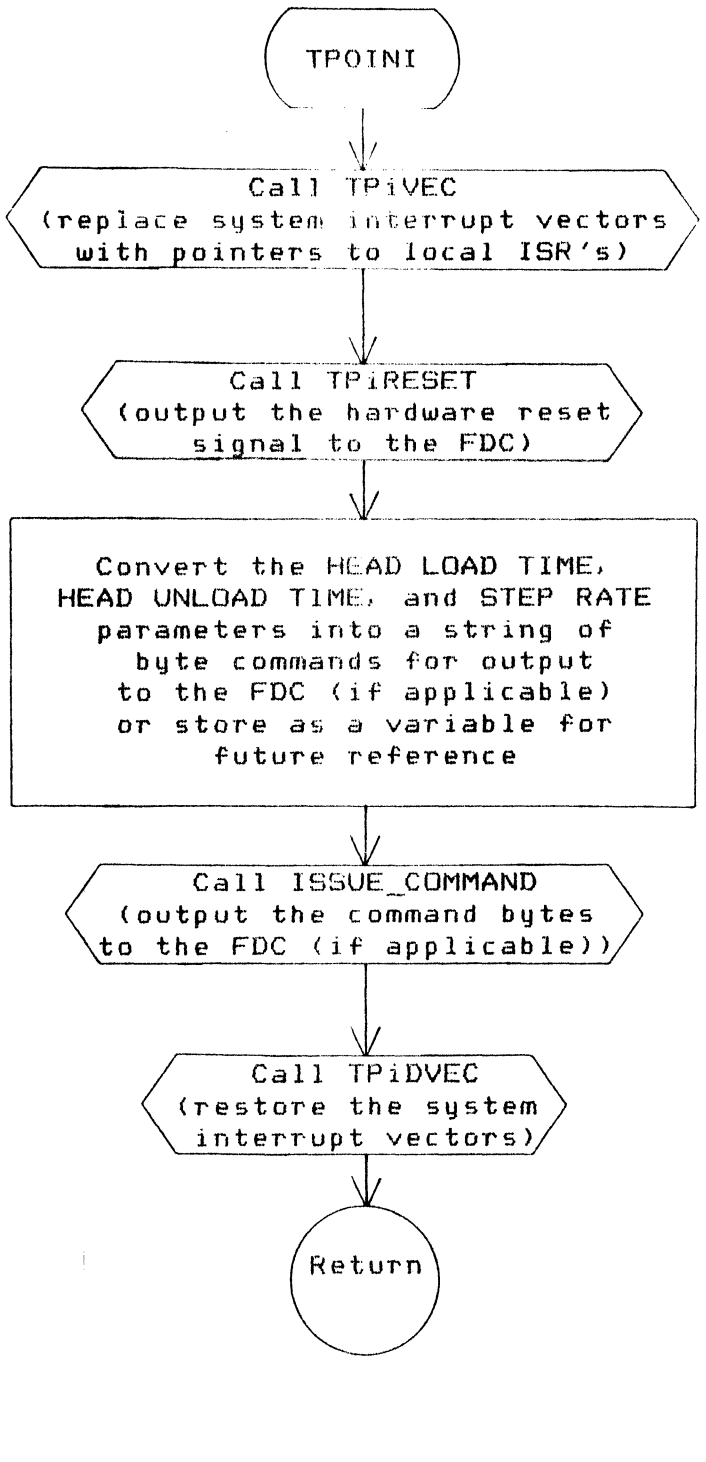
6.30 Programming Problems with the NEC 765

There are two problem areas with the NEC controller chip as far as software programming is concerned. The first is the number of hardware interrupts sent during a Recalibrate command. This is discussed in the description of the NEC 765 Recalibrate command. The other problem has to do with keeping track of the current track number the NEC 765 thinks it is on. This stems out of the problem that the internal track register in the NEC 765 is not accessible by the host hardware or software. Problems arise when the NEC 765 receives Recalibrate or Reset commands from unknown sources (like IBM PC-DOS ROM BIOS interrupt 13) and the internal track register gets out of synchronization with the software track register.

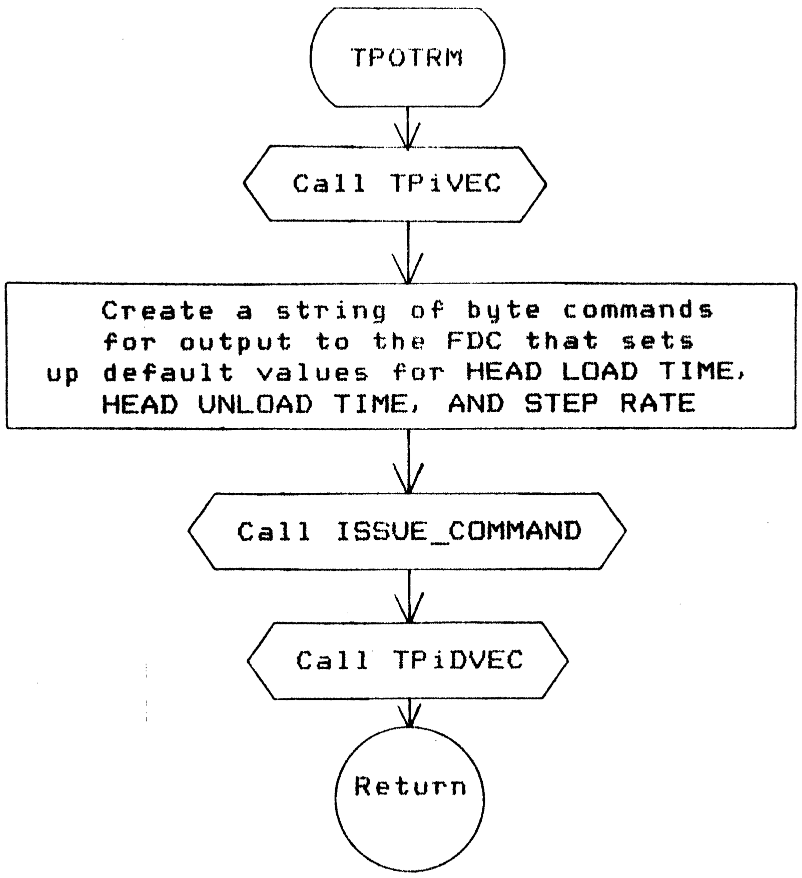
7.00 Low-Level Device Driver Flowcharts

The flowcharts on the following pages outline the low-level device drivers described above. An attempt was made to make these flow-charts controller independent. Specific implementations of this outline will differ slightly.

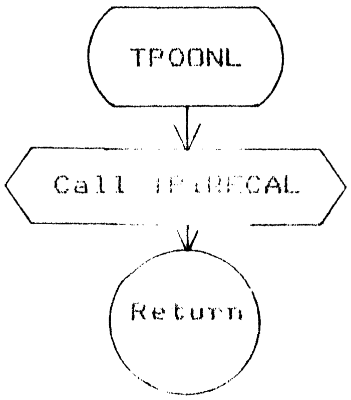
TPOINI(load time, unload time, step rate, io gap)

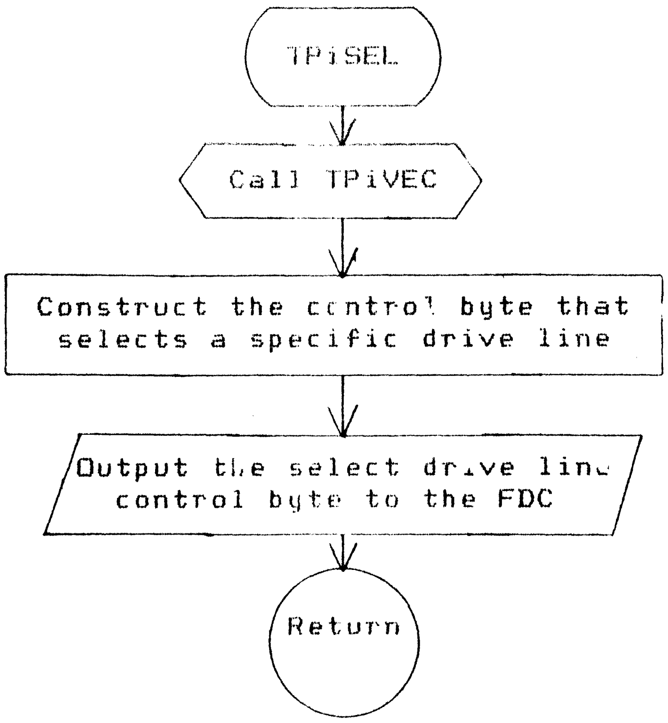

TPOINI
Call TPiVEC (replace system interrupt vectors with pointers to local ISR's)
Call TPiRESET (output the hardware reset signal to the FDC)
Convert the HEAD LOAD TIME, HEAD UNLOAD TIME, and STEP RATE parameters into a string of byte commands for output to the FDC (if applicable) or store as a variable for future reference
Call ISSUE_COMMAND (output the command bytes to the FDC (if applicable))
Call TPiDVEC (restore the system interrupt vectors)
Return

TPOTRM()

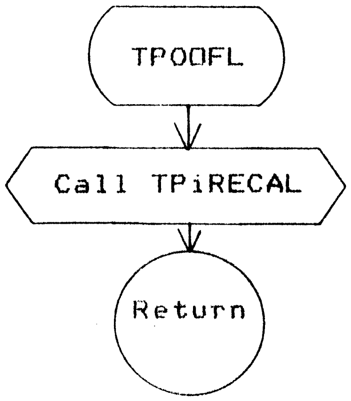

TPOTRM
Call TPiVEC
Create a string of byte commands for output to the FDC that sets up default values for HEAD LOAD TIME, HEAD UNLOAD TIME, AND STEP RATE
Call ISSUE_COMMAND
Call TPiDVEC
Return

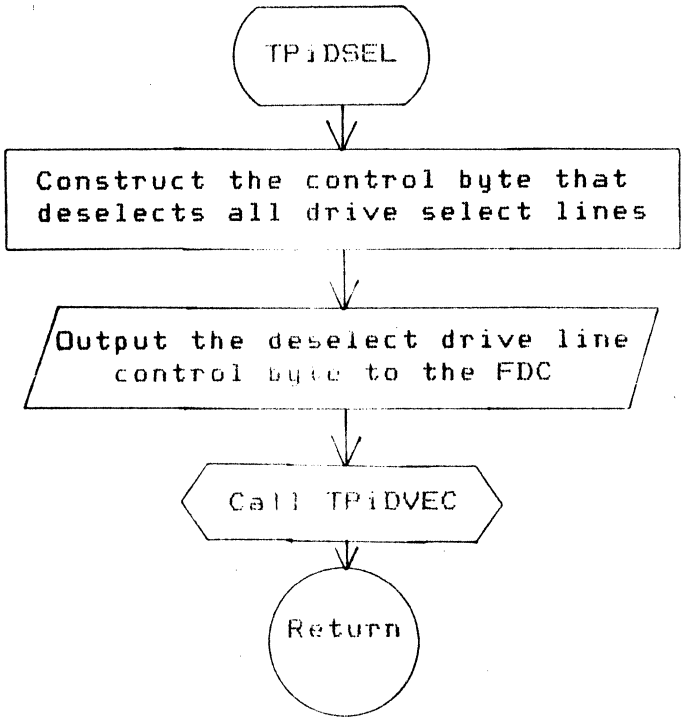
TPOONL(drive)



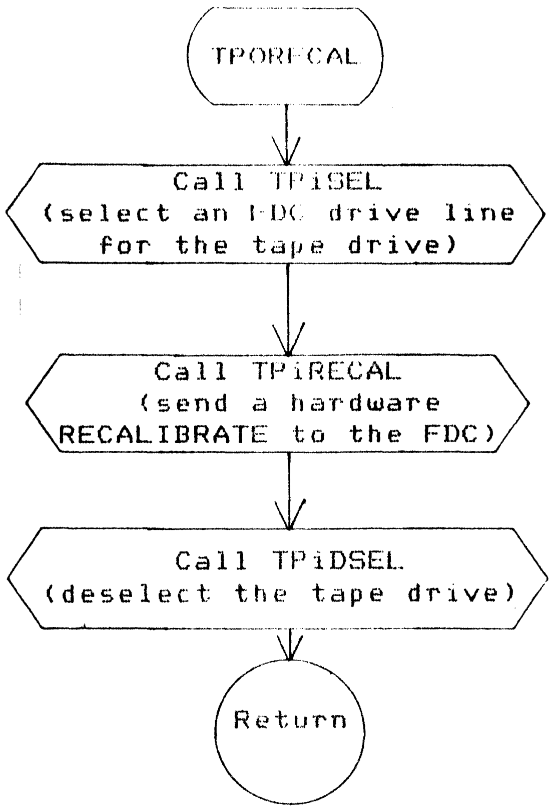

TPiSEL
Call TPiVEC
Construct the control byte that selects a specific drive line
Output the select drive line control byte to the FDC
Return

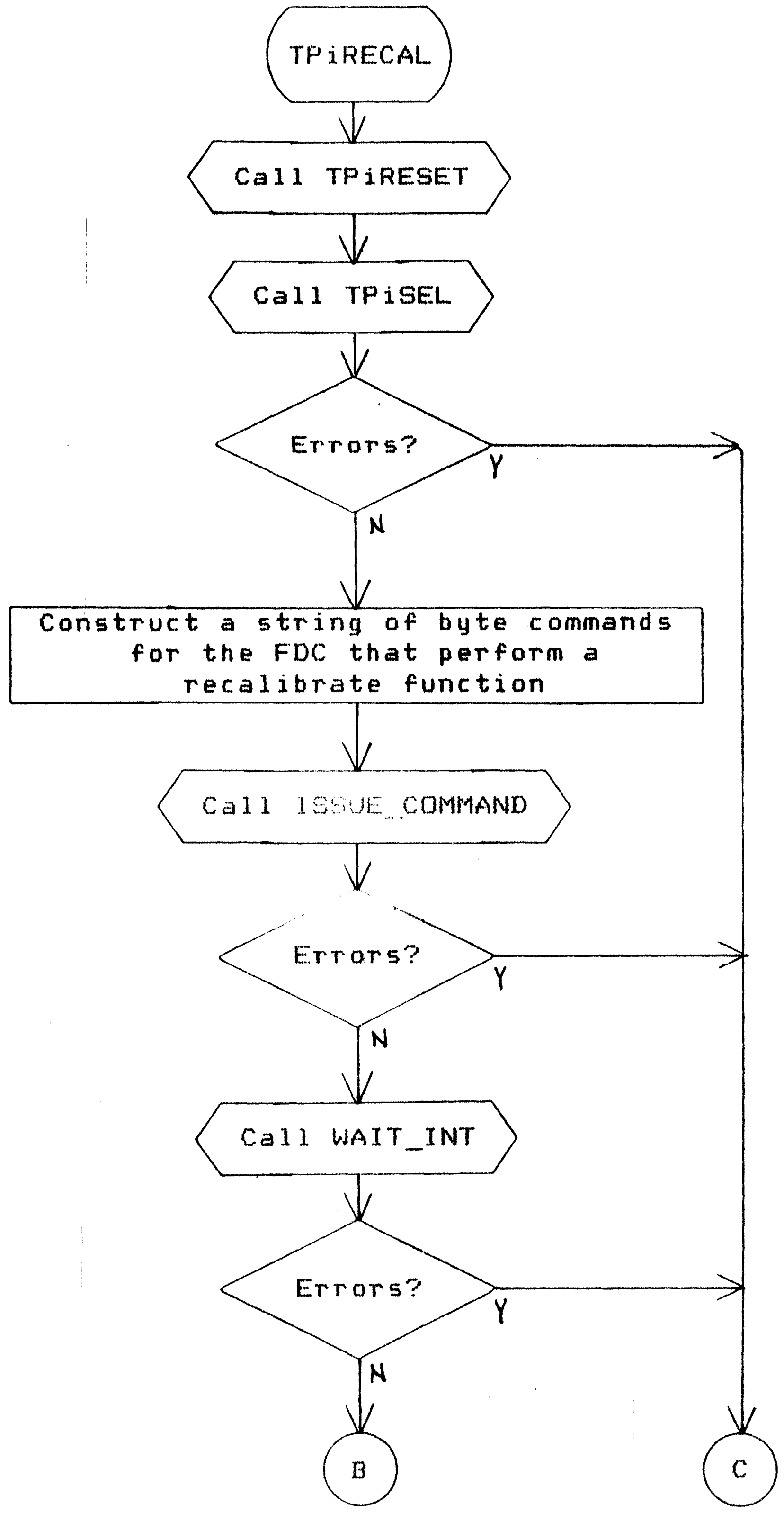
TPOOFL(drive)

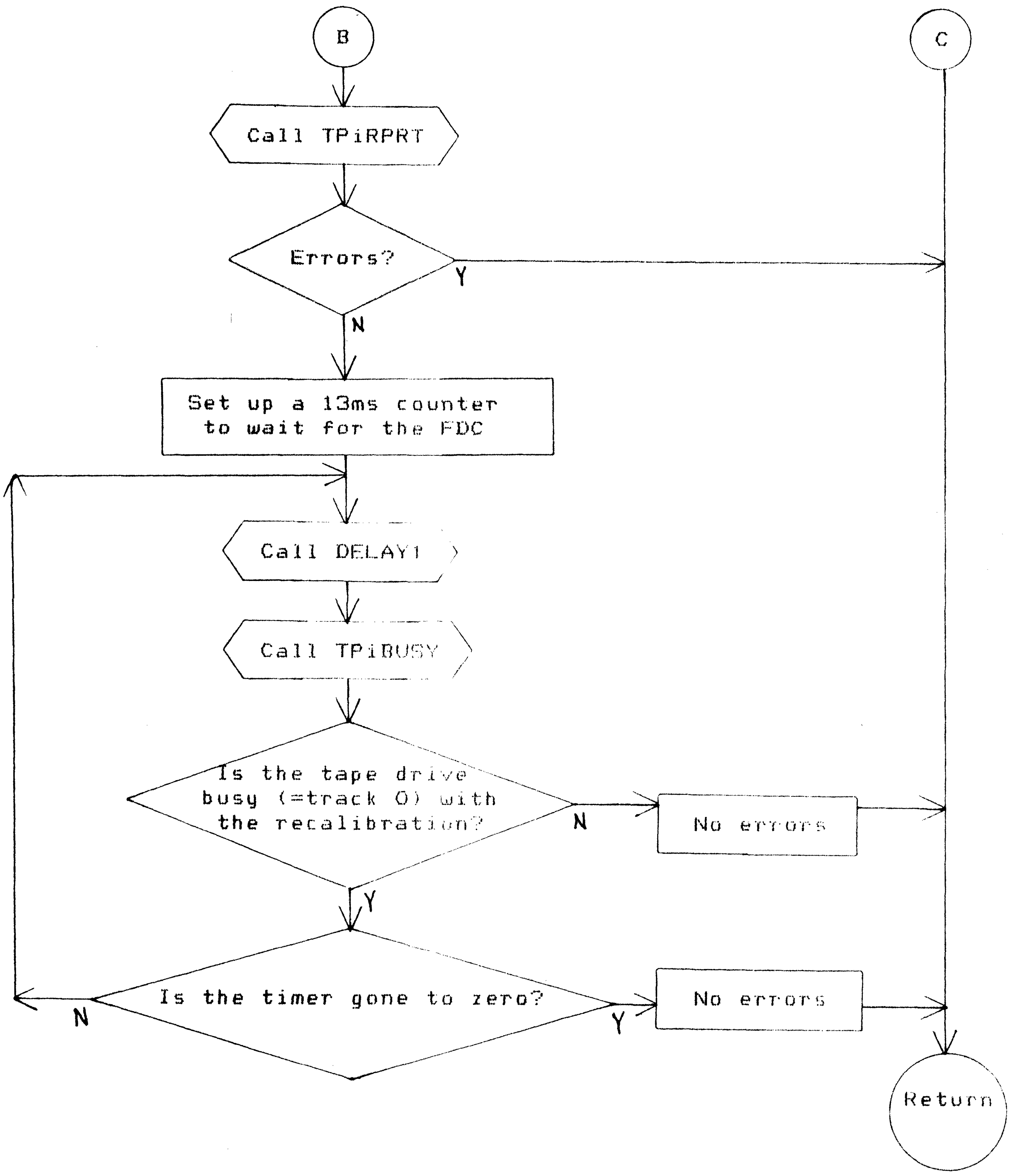



TPiDSEL
Construct the control byte that deselects all drive select lines
Output the deselect drive line control byte to the FDC
Call TPiDVEC
Return

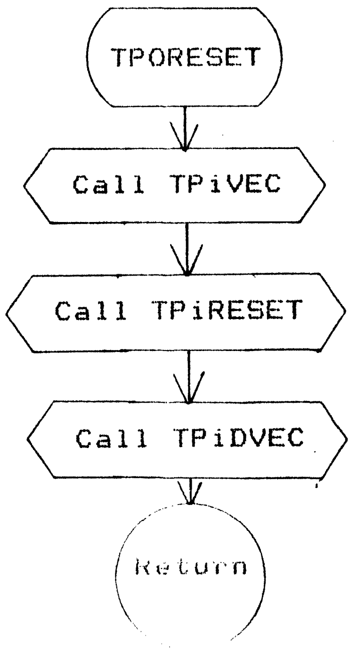
TPORECAL(drive)

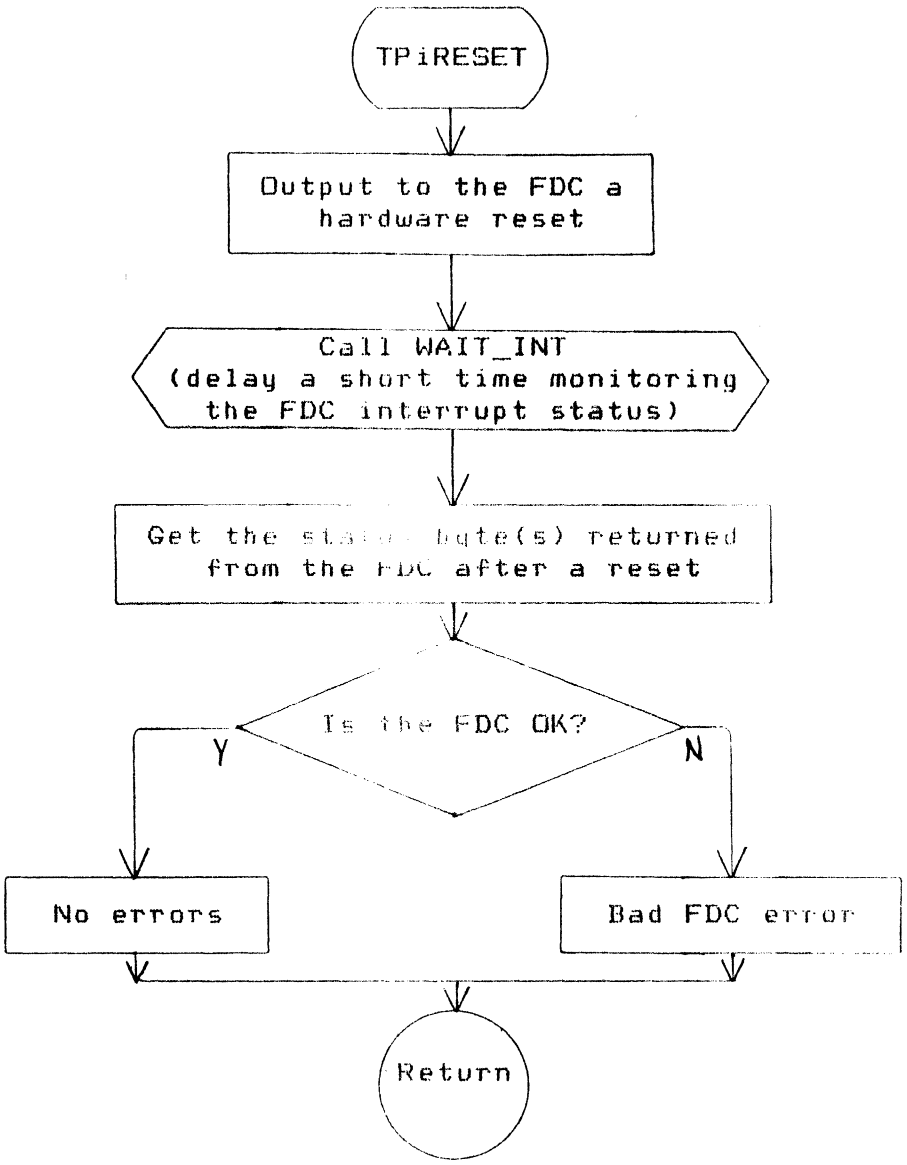

TPORECAL
Call TPiSEL (select an FDC drive line for the tape drive)
Call TPiRECAL (send a hardware RECALIBRATE to the FDC)
Call TPiDSEL (deselect the tape drive)
Return


TPiRECAL
Call TPiRESET
Call TPiSEL
Errors?
Y
N
Construct a string of byte commands for the FDC that perform a recalibrate function
Call ISSUE_COMMAND
Errors?
Y
N
Call WAIT_INT
Errors?
Y
N
B
C

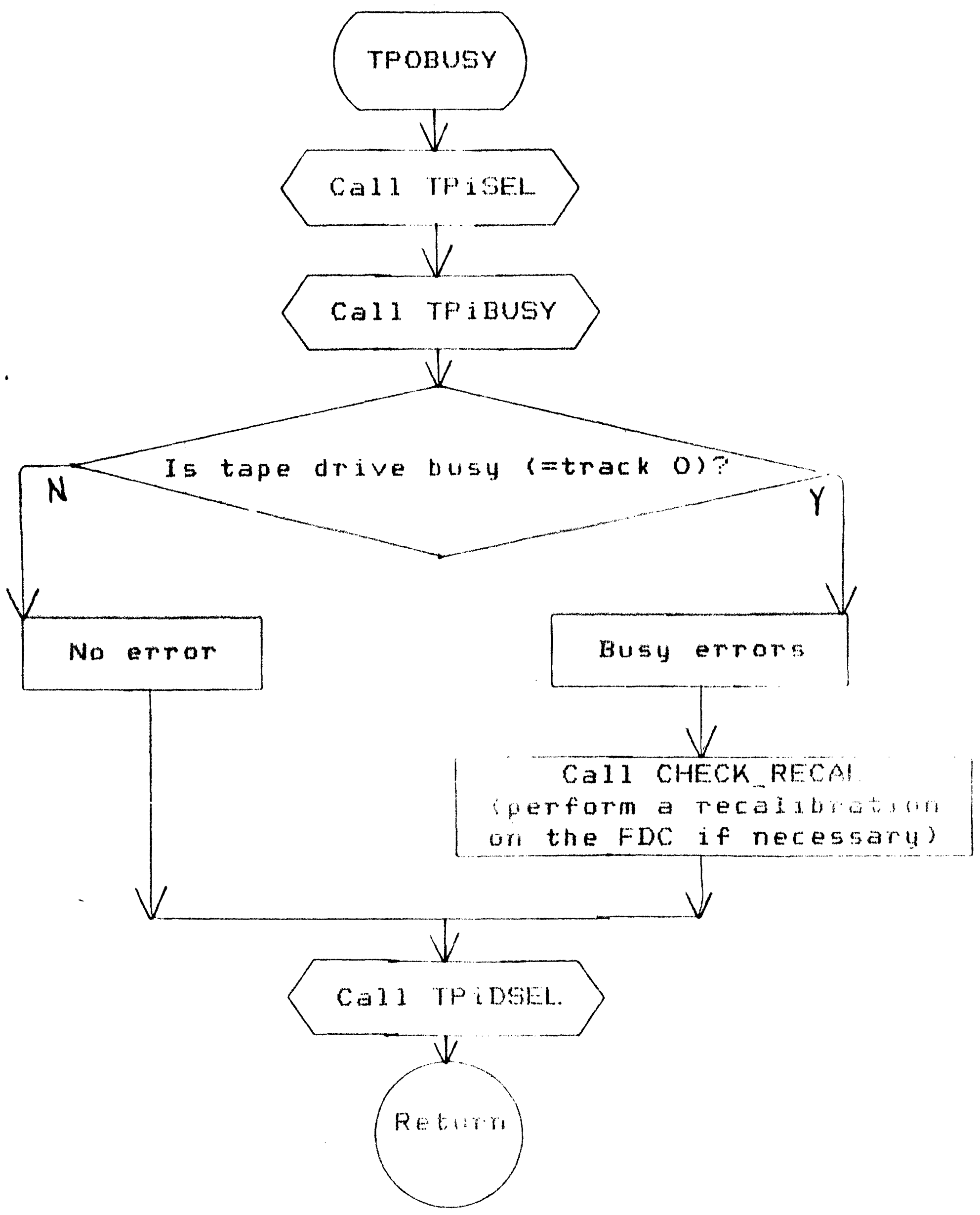

B
C
Call TPiRPRT
Errors?
Y
N
Set up a 13ms counter to wait for the FDC
Call DELAY1
Call TPiBUSY
Is the tape drive busy (=track 0) with the recalibration?
N
No errors
Y
Is the timer gone to zero?
N
Y
No errors
Return

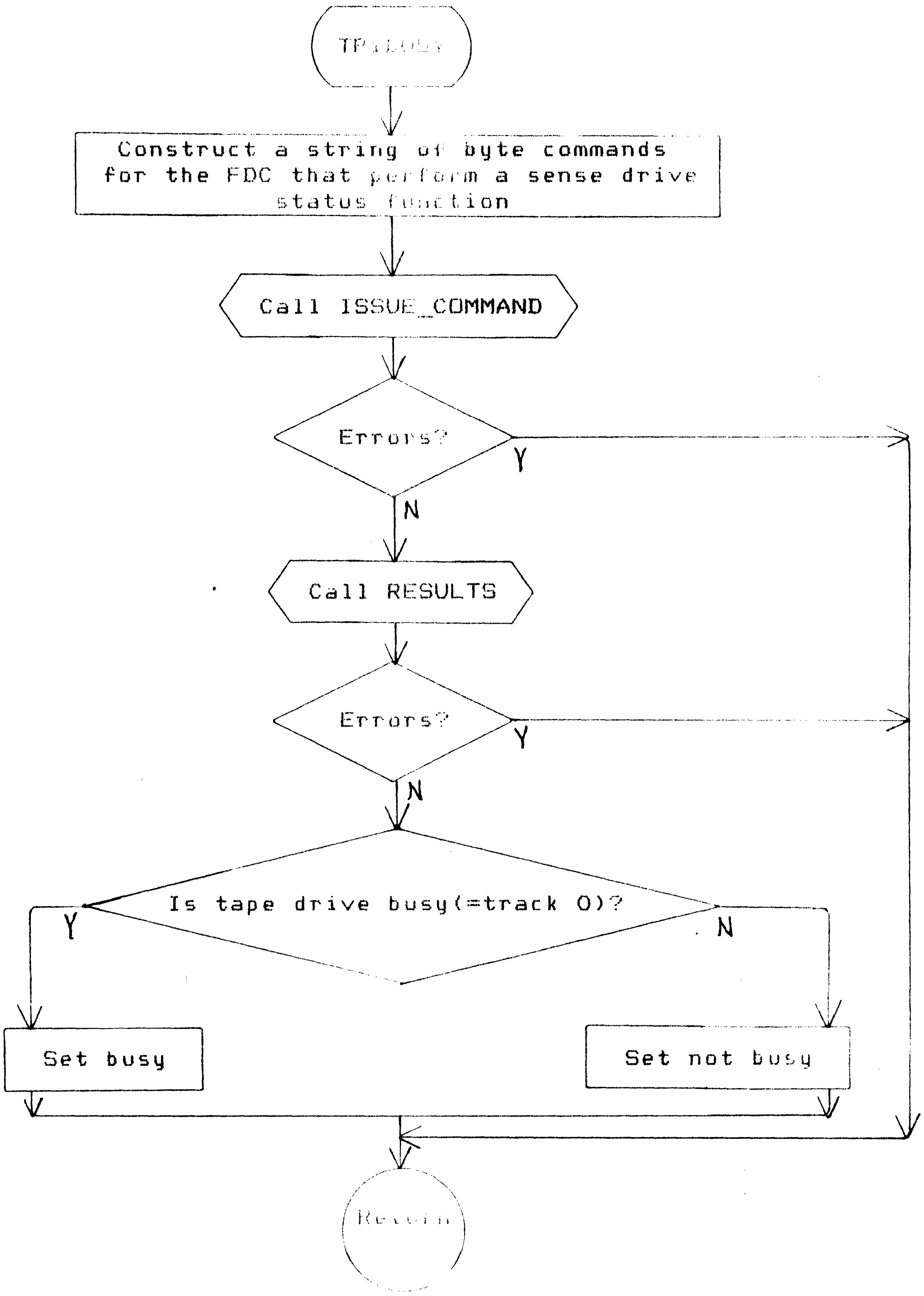
TPORESET()



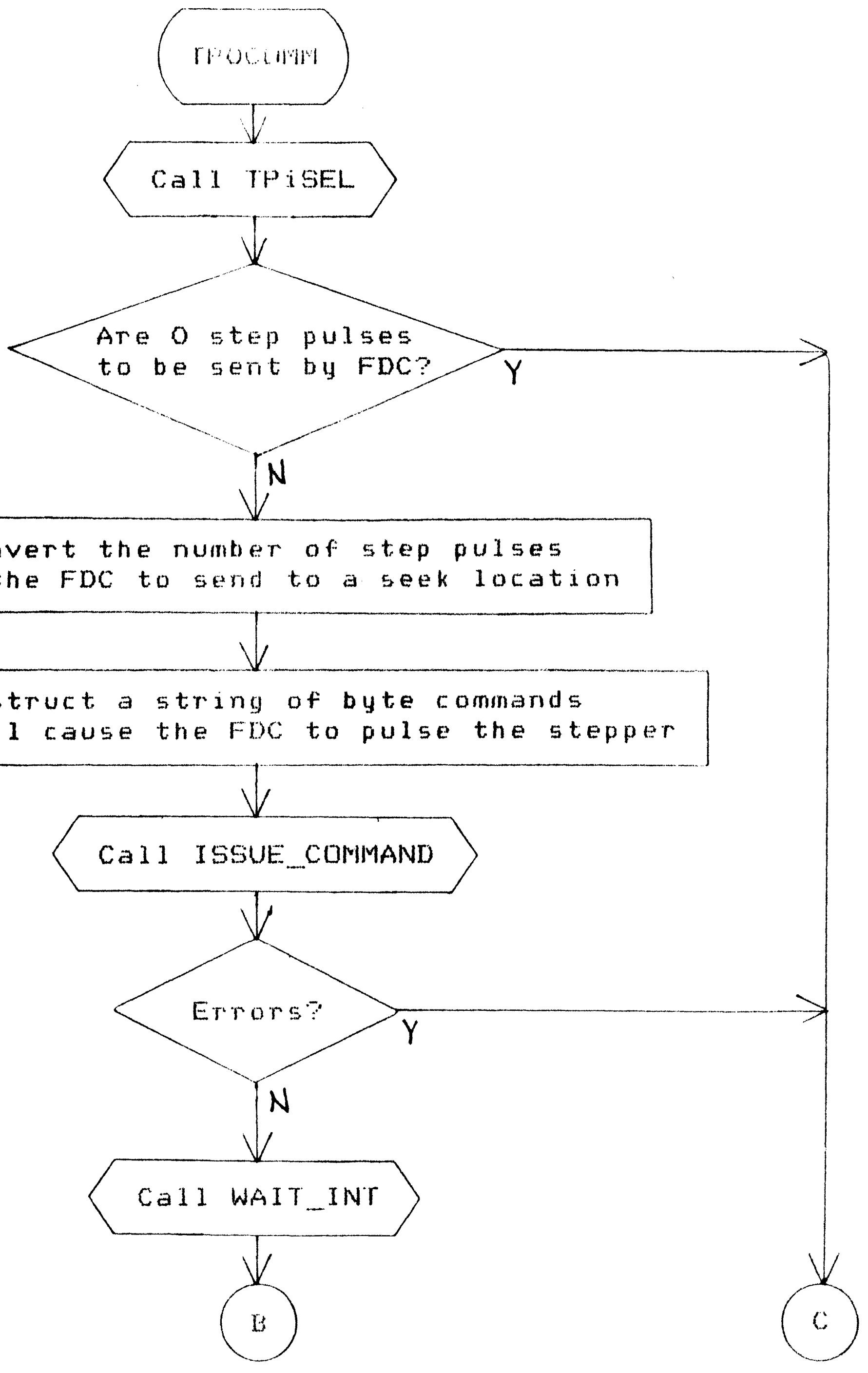

TPiRESET
Output to the FDC a hardware reset
Call WAIT_INT (delay a short time monitoring the FDC interrupt status)
Get the status byte(s) returned from the FDC after a reset
Is the FDC OK?
Y
N
No errors
Bad FDC error
Return

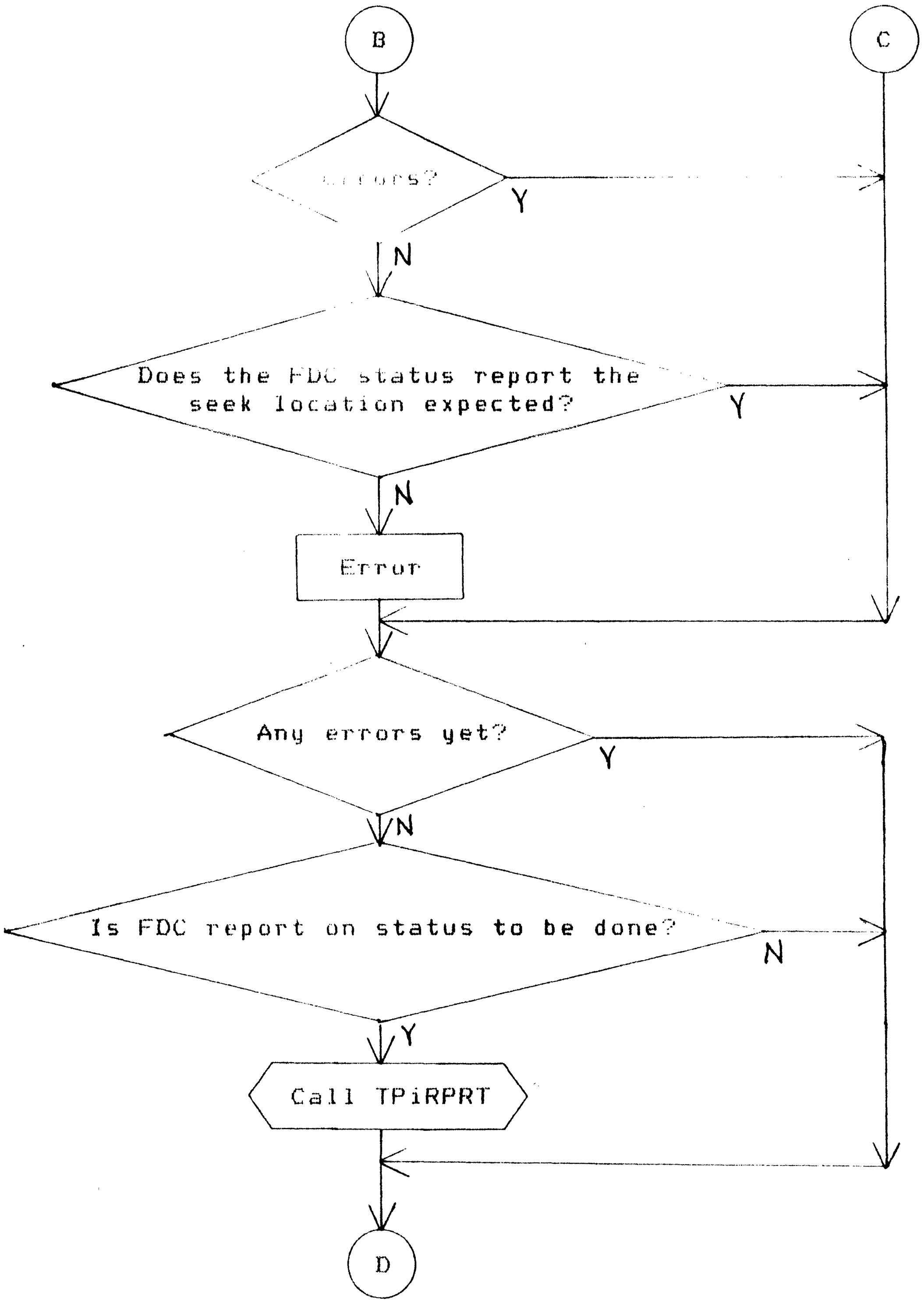
TPOBUSY(drive)

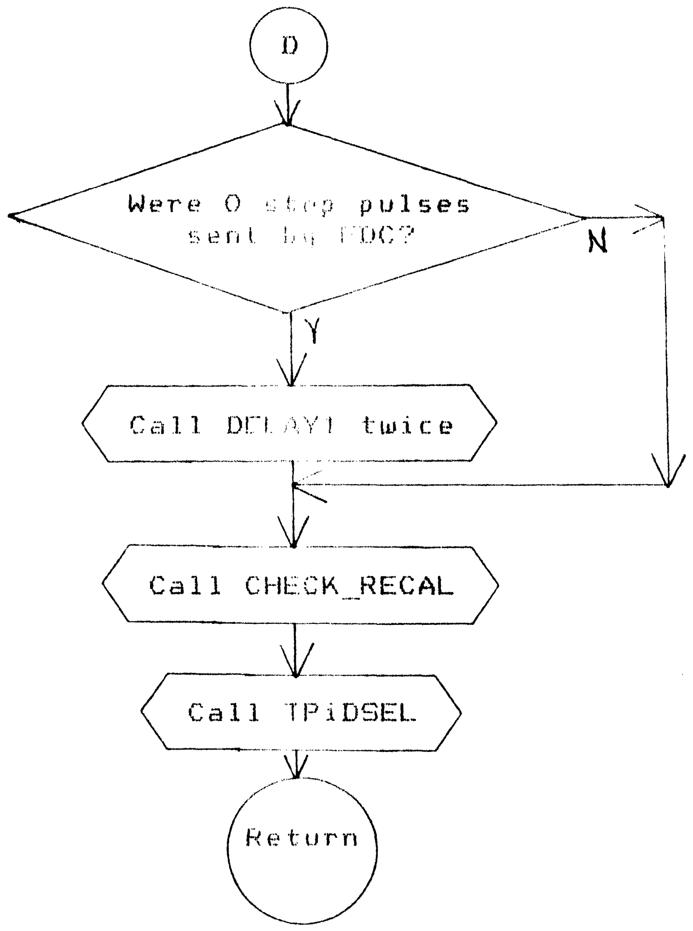

TPOBUSY
Call TPiSEL
Call TPiBUSY
Is tape drive busy (=track 0)?
N
Y
No error
Busy errors
Call CHECK_RECAL (perform a recalibration on the FDC if necessary)
Call TPiDSEL
Return


TPISBUSY
Construct a string of byte commands for the FDC that perform a sense drive status function
Call ISSUE_COMMAND
Errors?
Y
N
Call RESULTS
Errors?
Y
N
Is tape drive busy(=track 0)?
Y
N
Set busy
Set not busy
Return

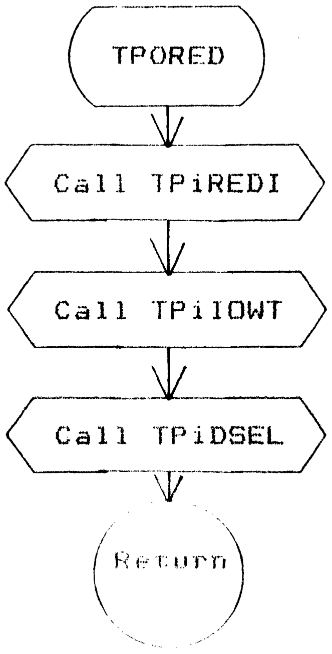
TPOCOMM(drive, steps, wait/status)


TPOCOMM
Call TPiSEL
Are 0 step pulses to be sent by FDC?
Y
N
Convert the number of step pulses for the FDC to send to a seek location
Construct a string of byte commands that will cause the FDC to pulse the stepper
Call ISSUE_COMMAND
Errors?
Y
N
Call WAIT_INT
B
C

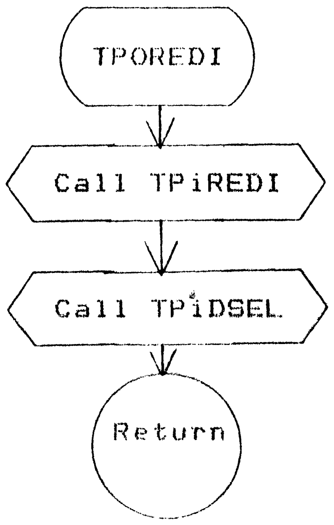

B
C
Errors?
Y
N
Does the FDC status report the seek location expected?
Y
N
Error
Any errors yet?
Y
N
Is FDC report on status to be done?
N
Y
Call TPiRPRT
D

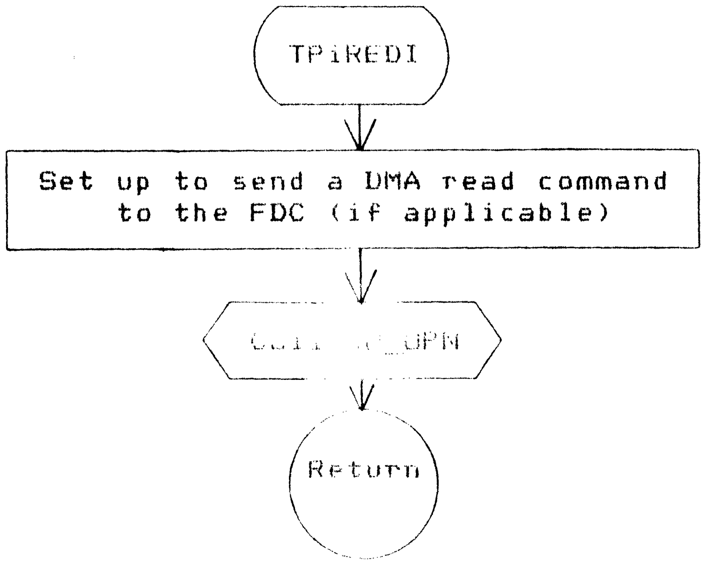

D
Were 0 step pulses sent by FDC?
N
Y
Call DELAY1 twice
Call CHECK_RECAL
Call TPiDSEL
Return

TPORED(drive, buffer address, cylinder, sector, sector count)

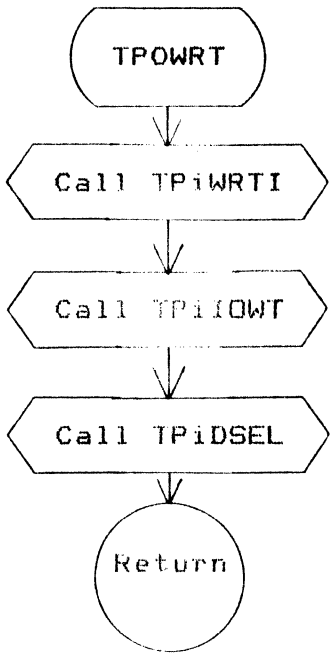


TPOREDI(drive, buffer address, cylinder, sector, sector count)

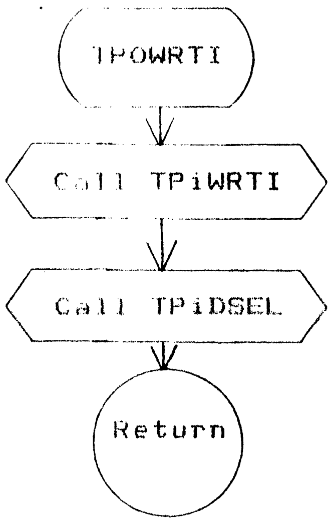


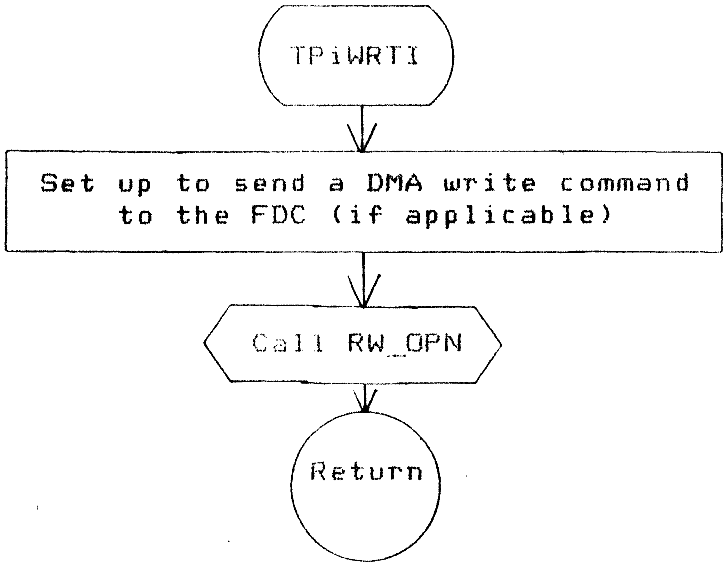

TPiREDI
Set up to send a DMA read command to the FDC (if applicable)
Call [illegible]_OPN
Return

__TPOWRT(drive, buffer address, cylinder, sector, sector count)__

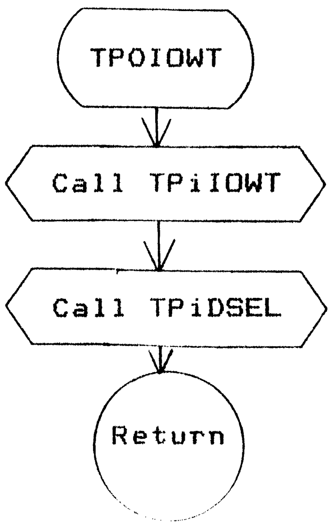


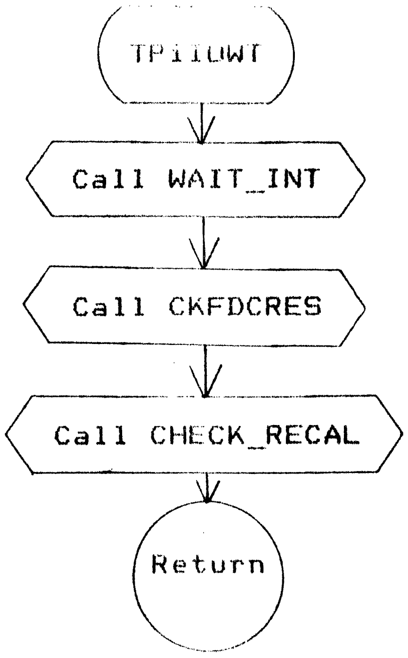
TPOWRTI(drive, buffer address, cylinder, sector, sector count)



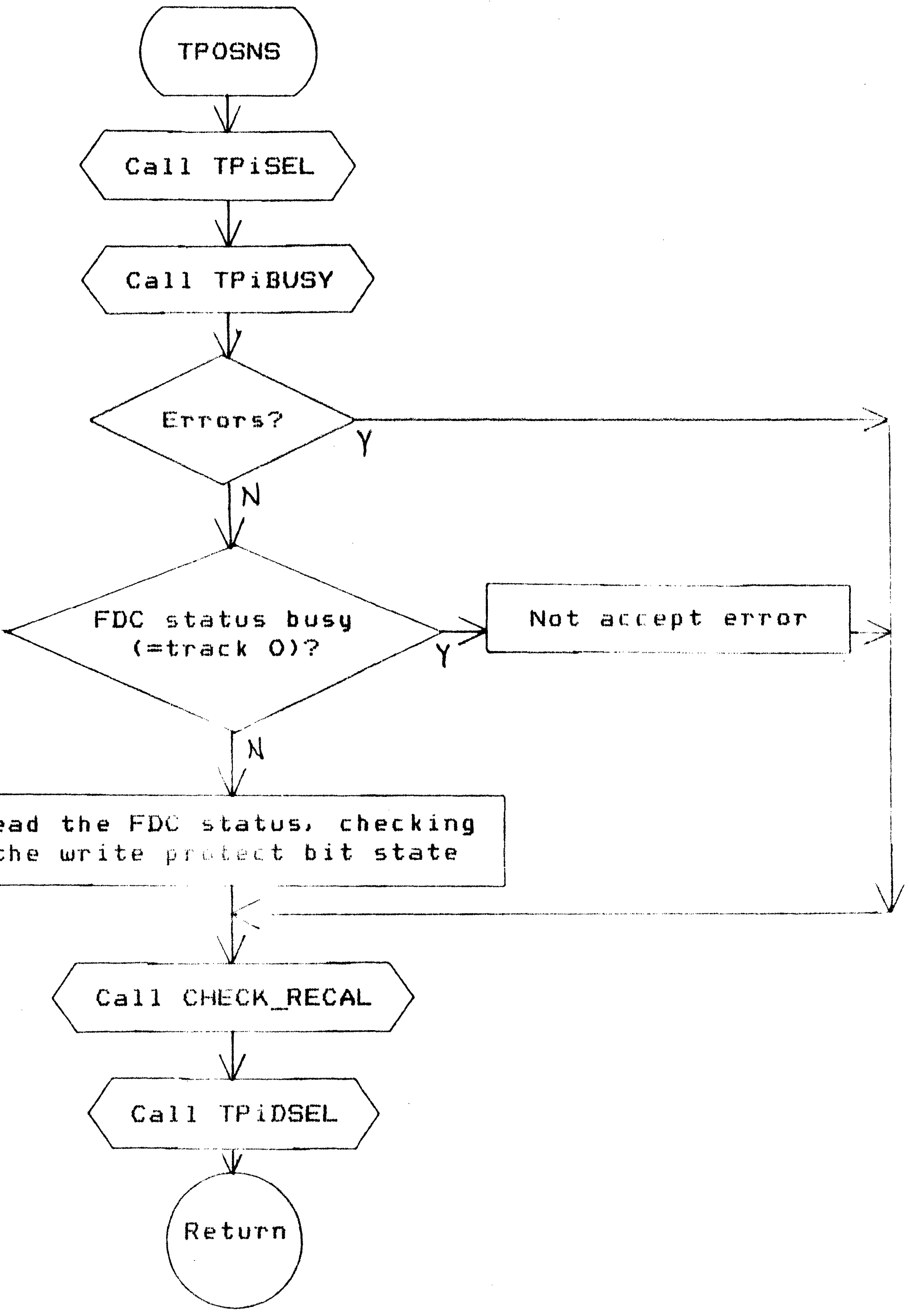

TPiWRTI
Set up to send a DMA write command to the FDC (if applicable)
Call RW_OPN
Return

TPOIOWT(drive)

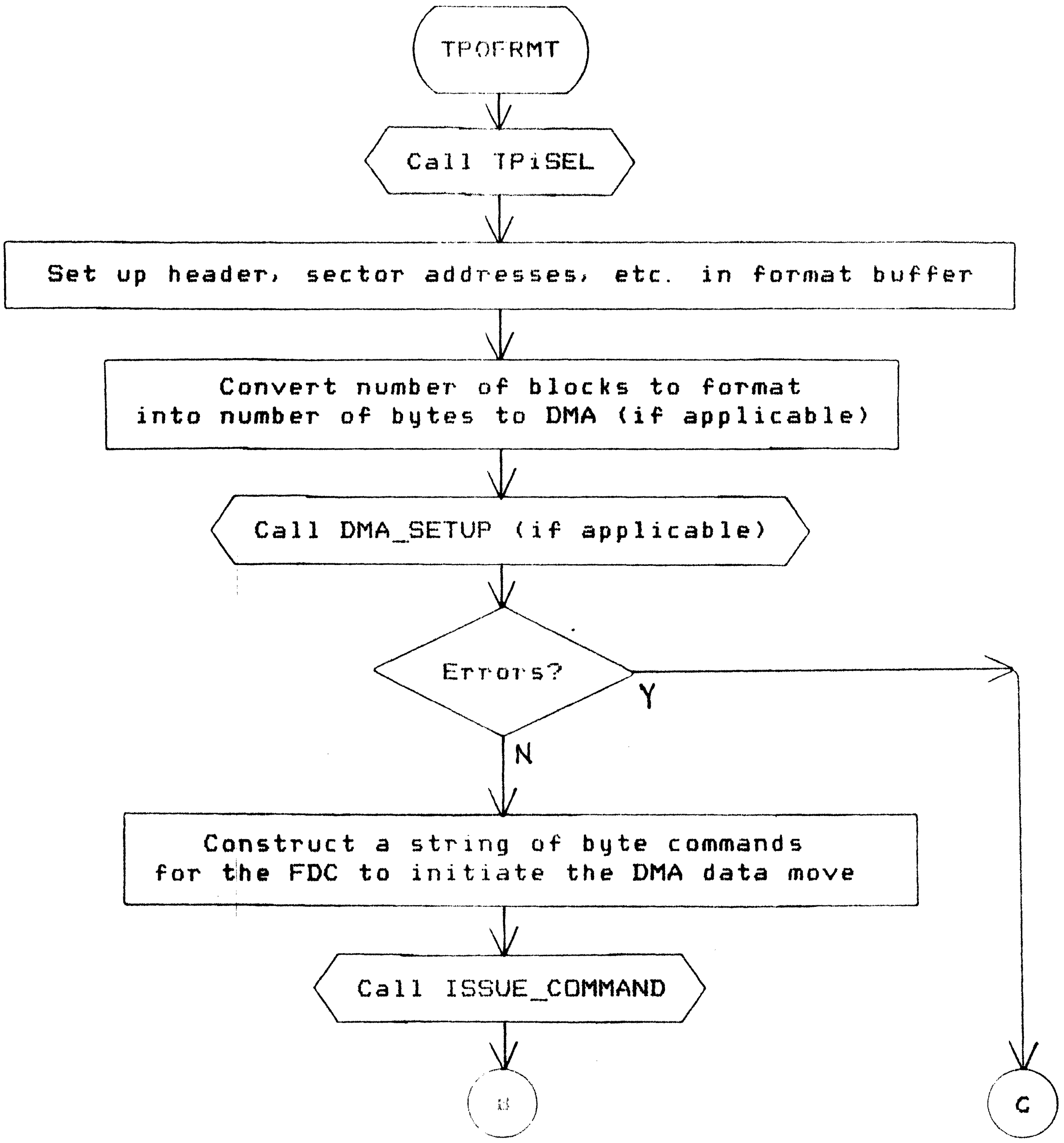


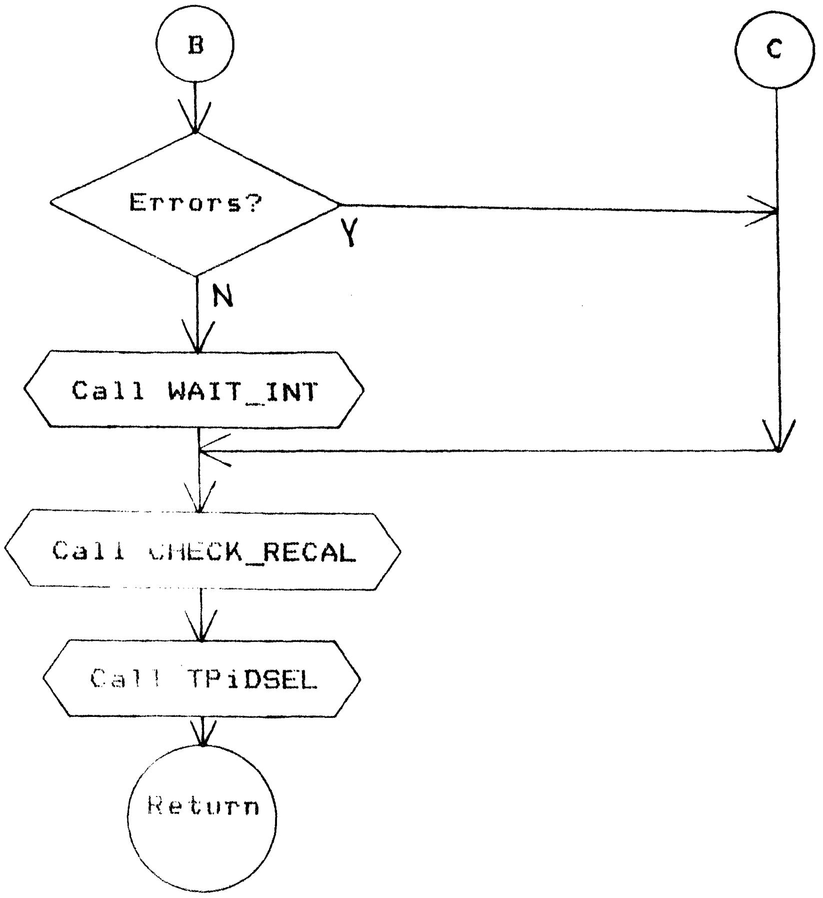


TPOSNS(drive, protected)

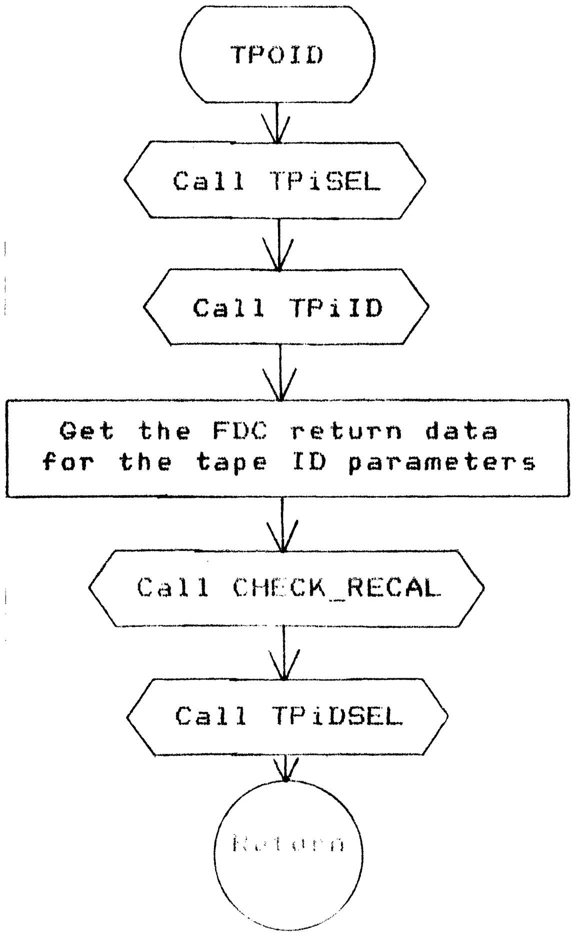


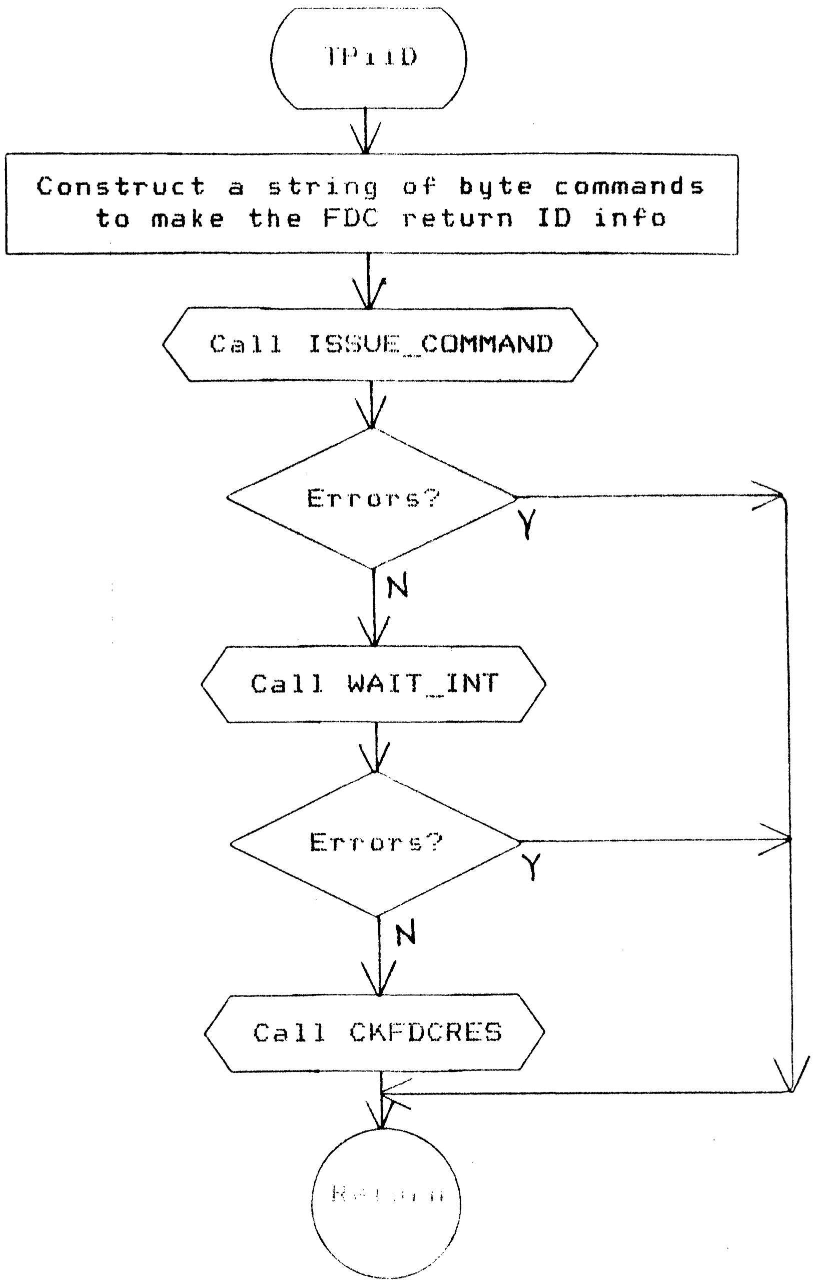
TPOFRMT(drive, buffer address, cylinder, sector length, sector count, format gap)

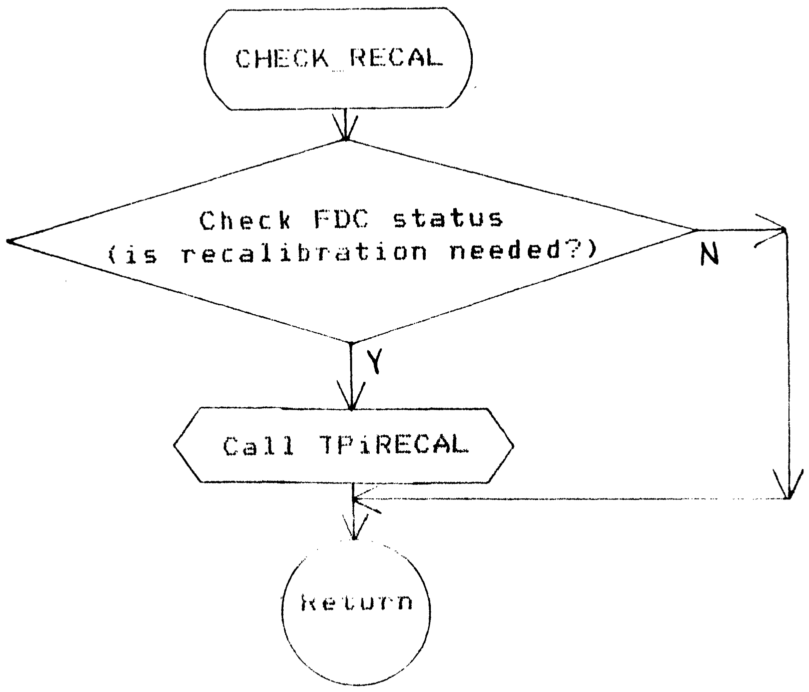

TPOFRMT
Call TPiSEL
Set up header, sector addresses, etc. in format buffer
Convert number of blocks to format into number of bytes to DMA (if applicable)
Call DMA_SETUP (if applicable)
Errors?
Y
N
Construct a string of byte commands for the FDC to initiate the DMA data move
Call ISSUE_COMMAND
B
C

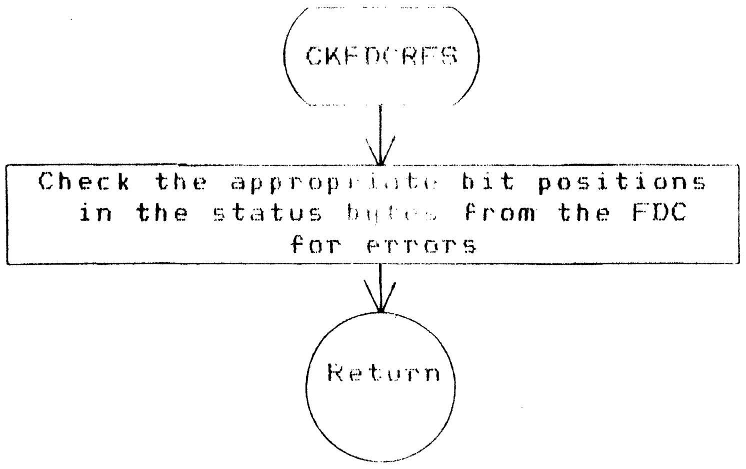

B
C
Errors?
Y
N
Call WAIT_INT
Call CHECK_RECAL
Call TPiDSEL
Return

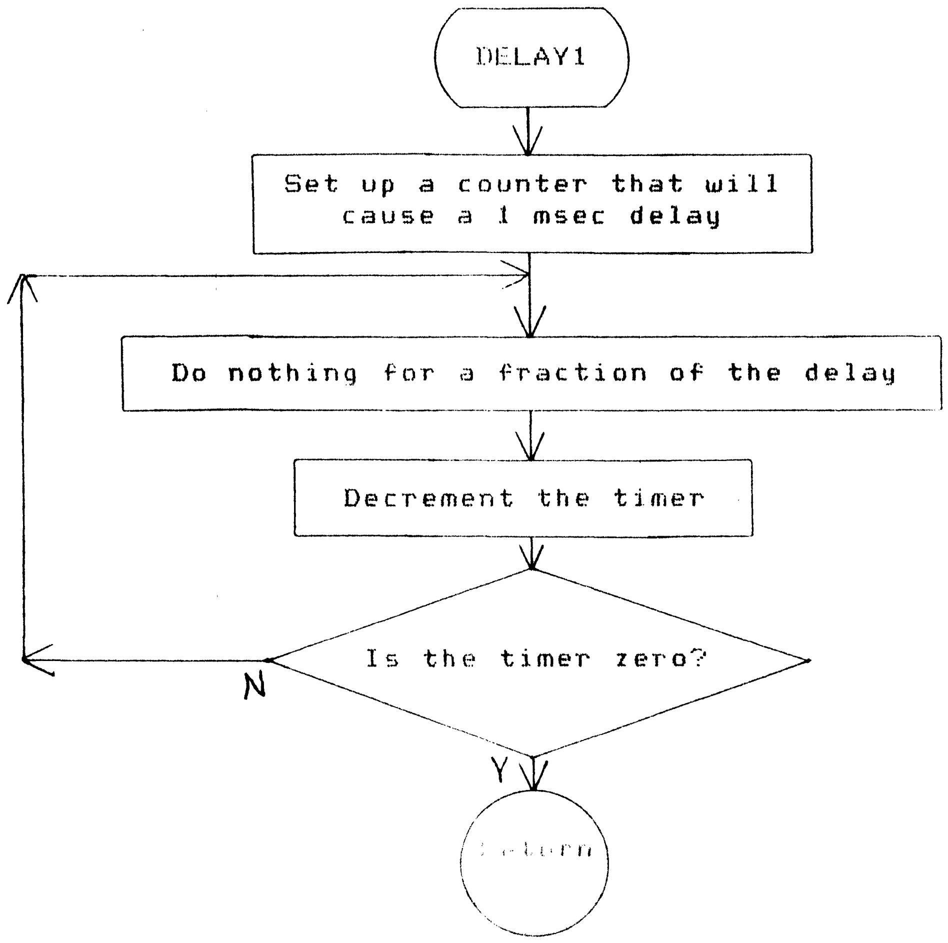
TPOID(drive, cylinder, sector)

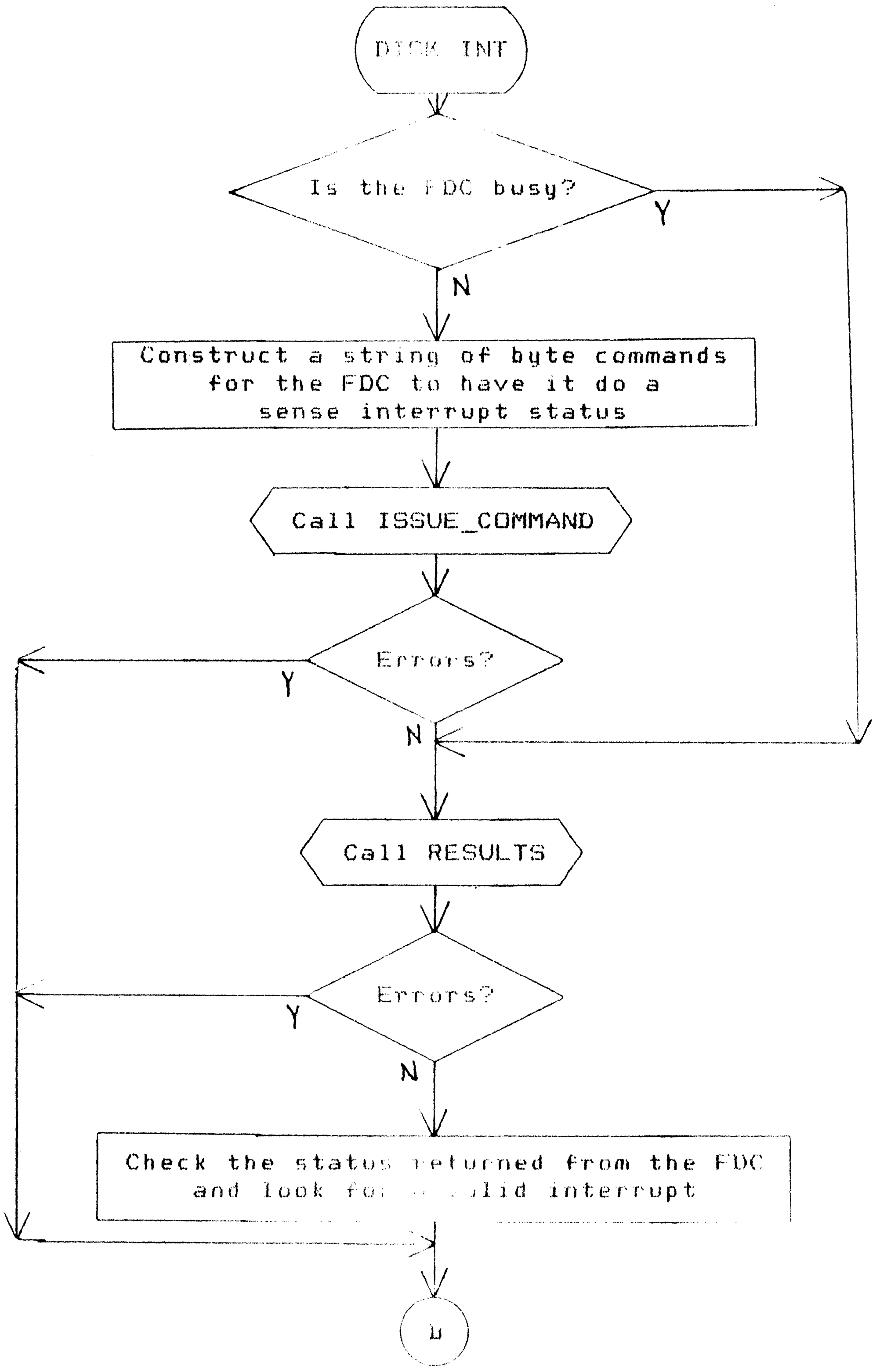


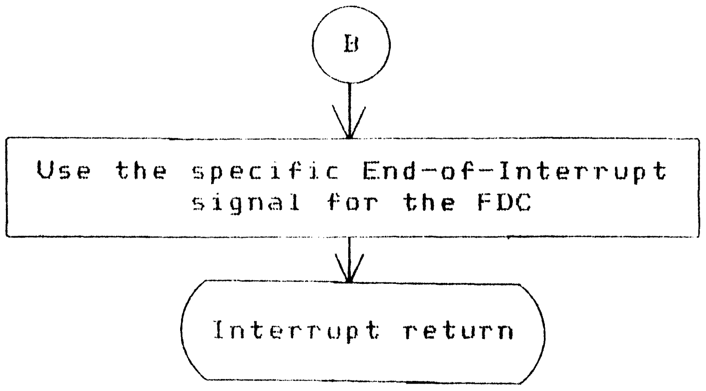

TPIID
Construct a string of byte commands to make the FDC return ID info
Call ISSUE_COMMAND
Errors?
Y
N
Call WAIT_INT
Errors?
Y
N
Call CKFDCRES
Return

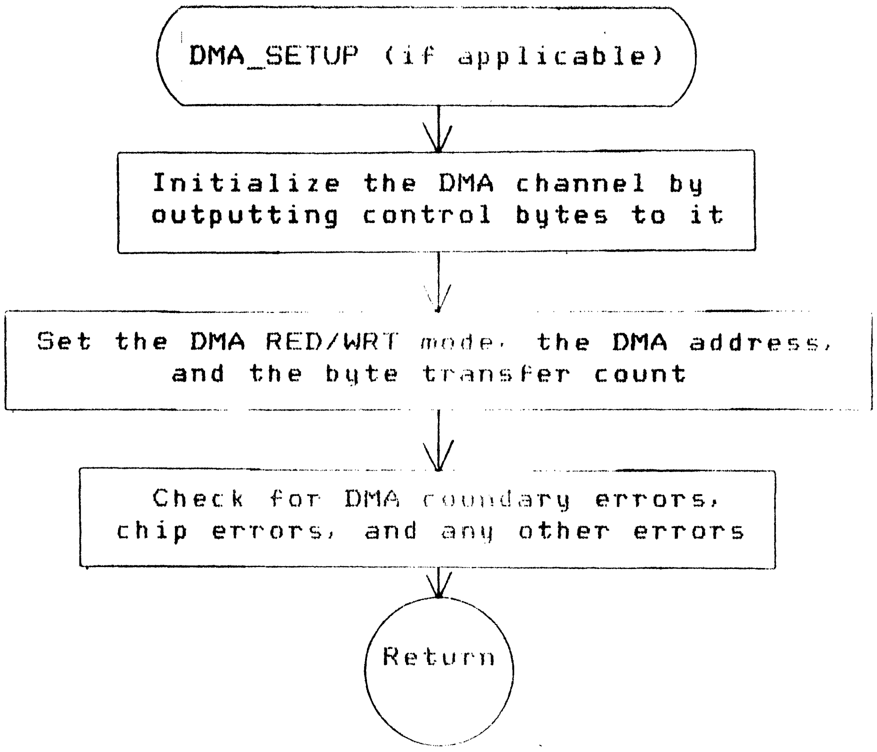

CHECK_RECAL
Check FDC status
(is recalibration needed?)
N
Y
Call TPiRECAL
Return

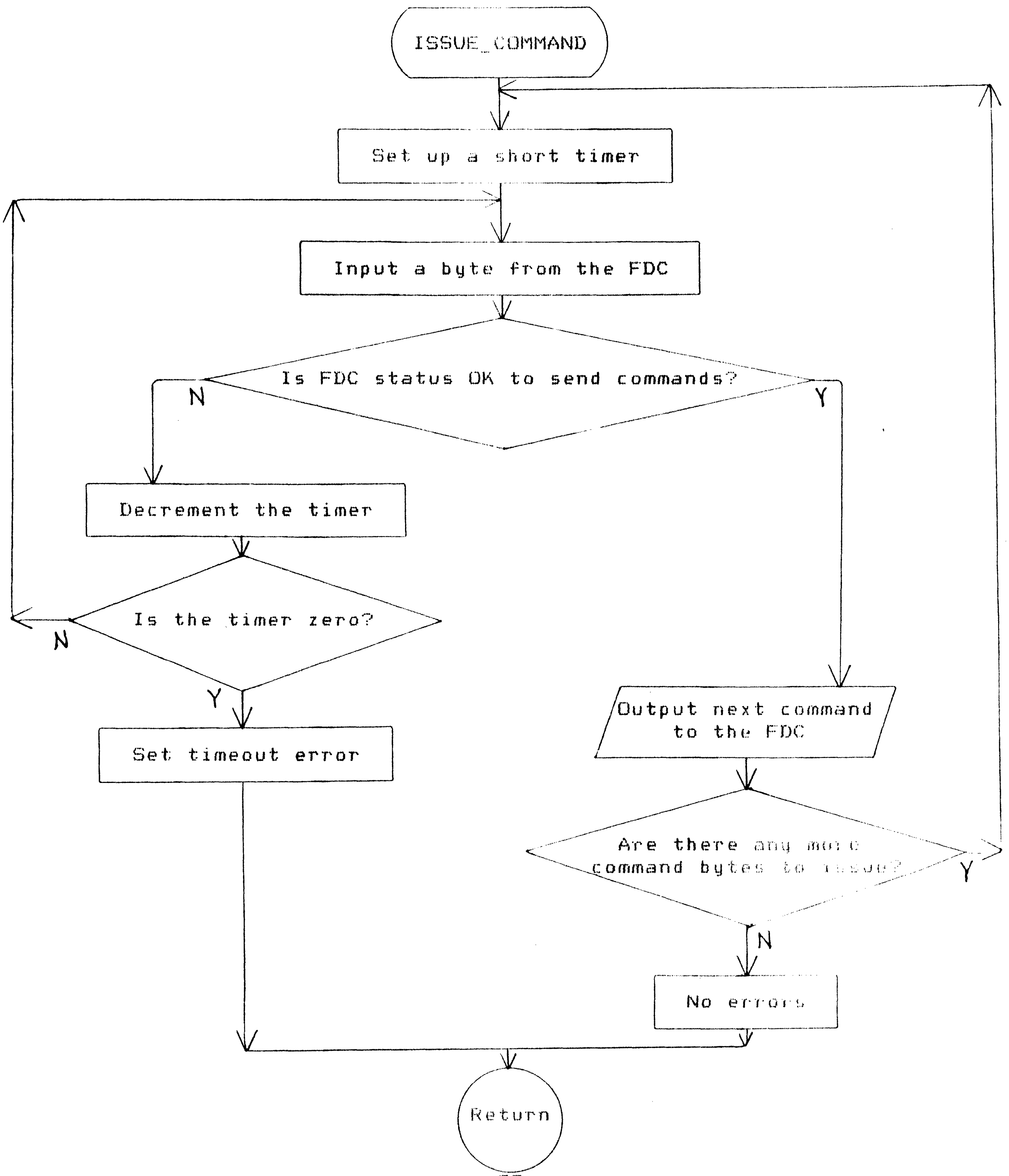

CKFDCRES
Check the appropriate bit positions in the status bytes from the FDC for errors
Return

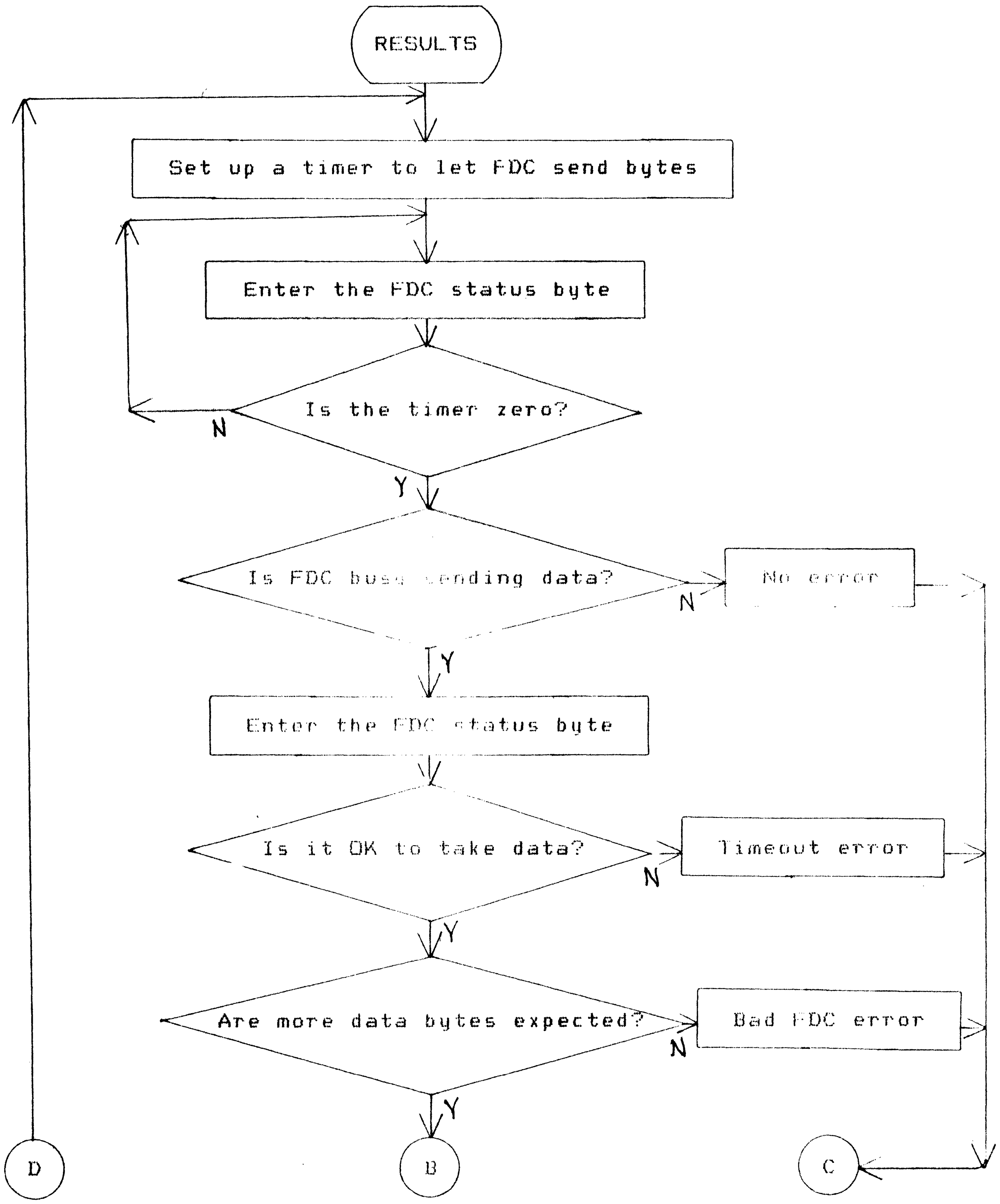

DELAY1
Set up a counter that will cause a 1 msec delay
Do nothing for a fraction of the delay
Decrement the timer
Is the timer zero?
N
Y
Return


DISK INT
Is the FDC busy?
Y
N
Construct a string of byte commands for the FDC to have it do a sense interrupt status
Call ISSUE_COMMAND
Errors?
Y
N
Call RESULTS
Errors?
Y
N
Check the status returned from the FDC and look for a valid interrupt
D

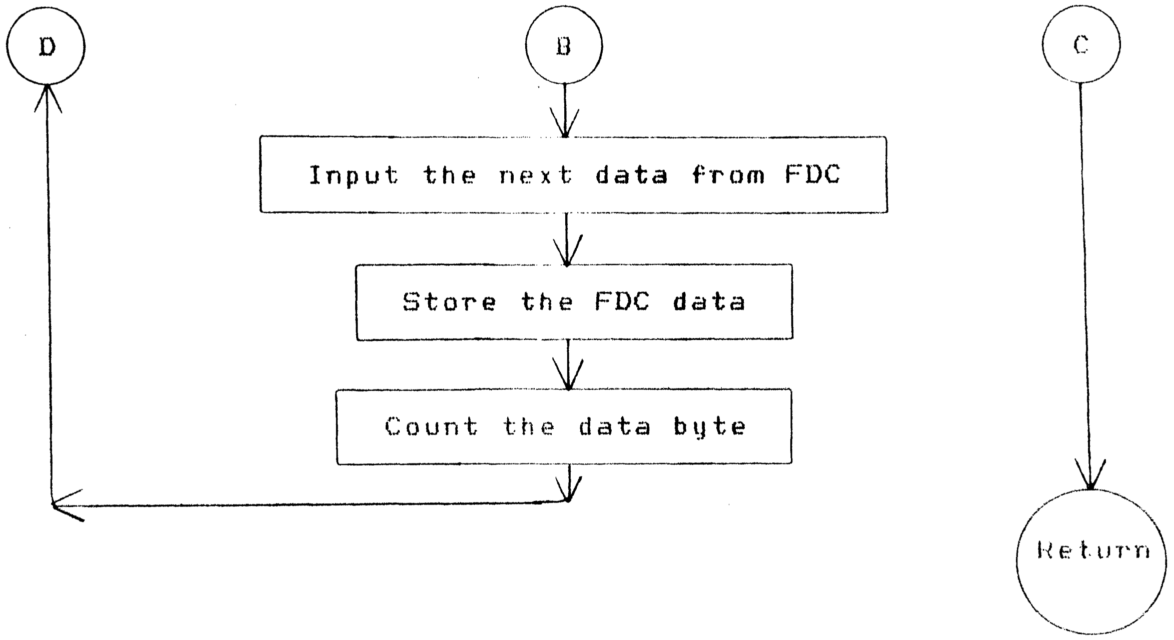

B
Use the specific End-of-Interrupt signal for the FDC
Interrupt return

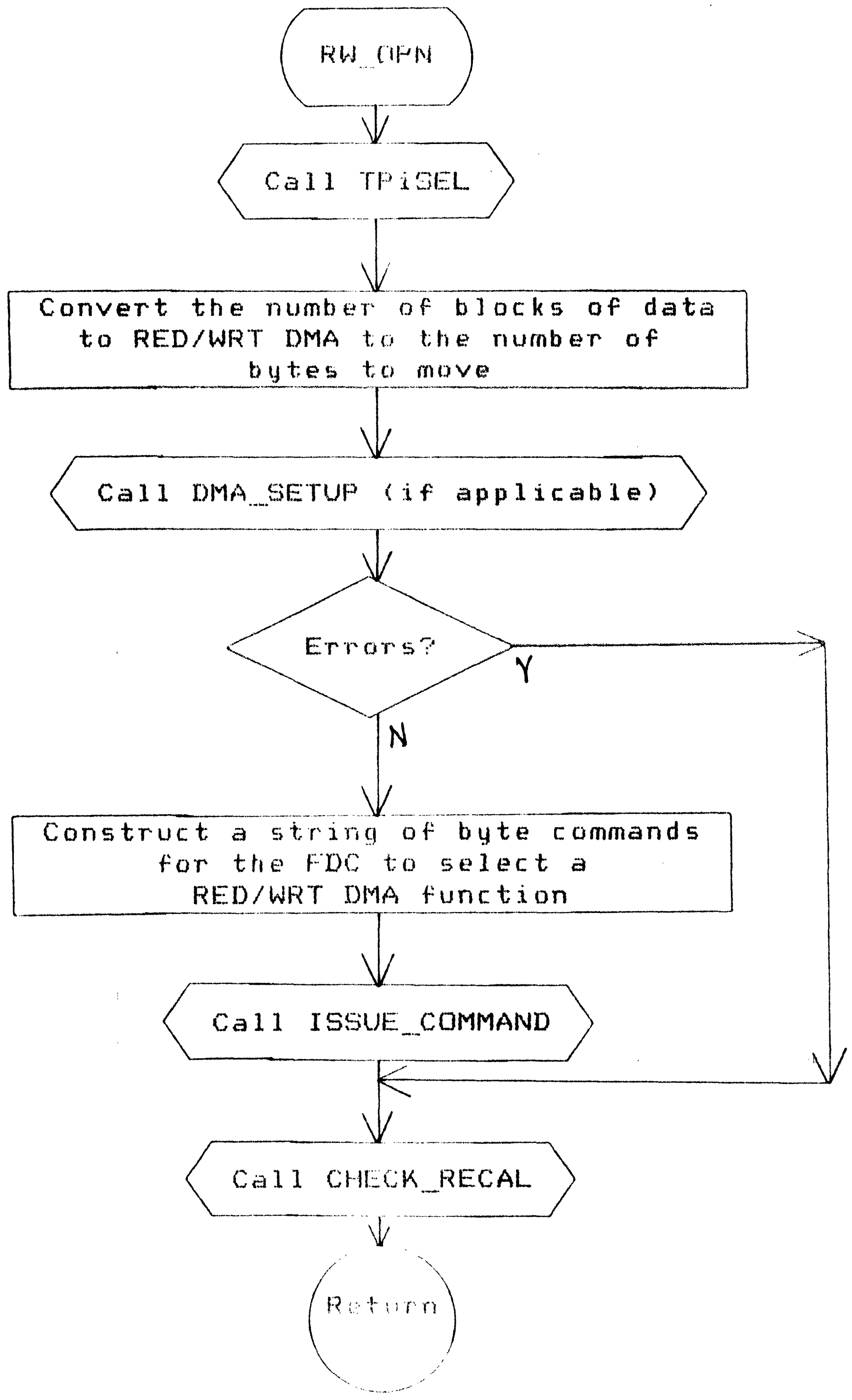

DMA_SETUP (if applicable)
Initialize the DMA channel by outputting control bytes to it
Set the DMA RED/WRT mode, the DMA address, and the byte transfer count
Check for DMA boundary errors, chip errors, and any other errors
Return


ISSUE_COMMAND
Set up a short timer
Input a byte from the FDC
Is FDC status OK to send commands?
N
Y
Decrement the timer
Is the timer zero?
N
Y
Set timeout error
Output next command to the FDC
Are there any more command bytes to issue?
Y
N
No errors
Return

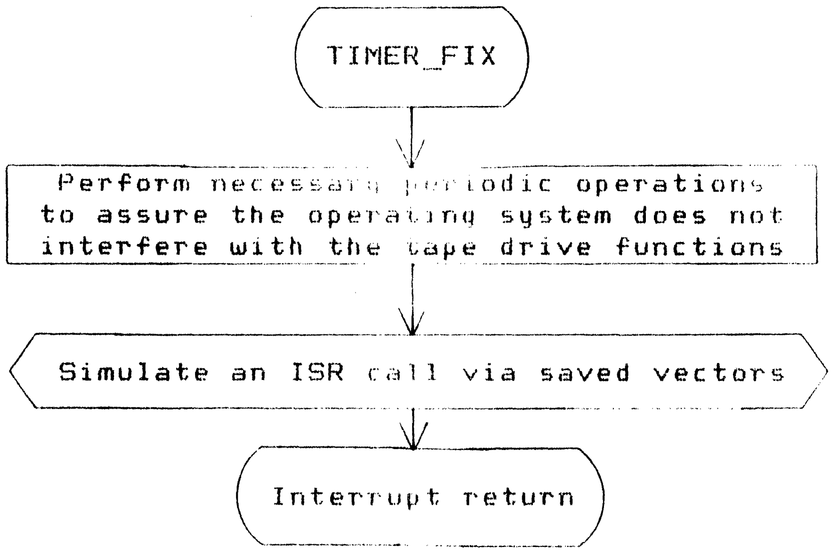


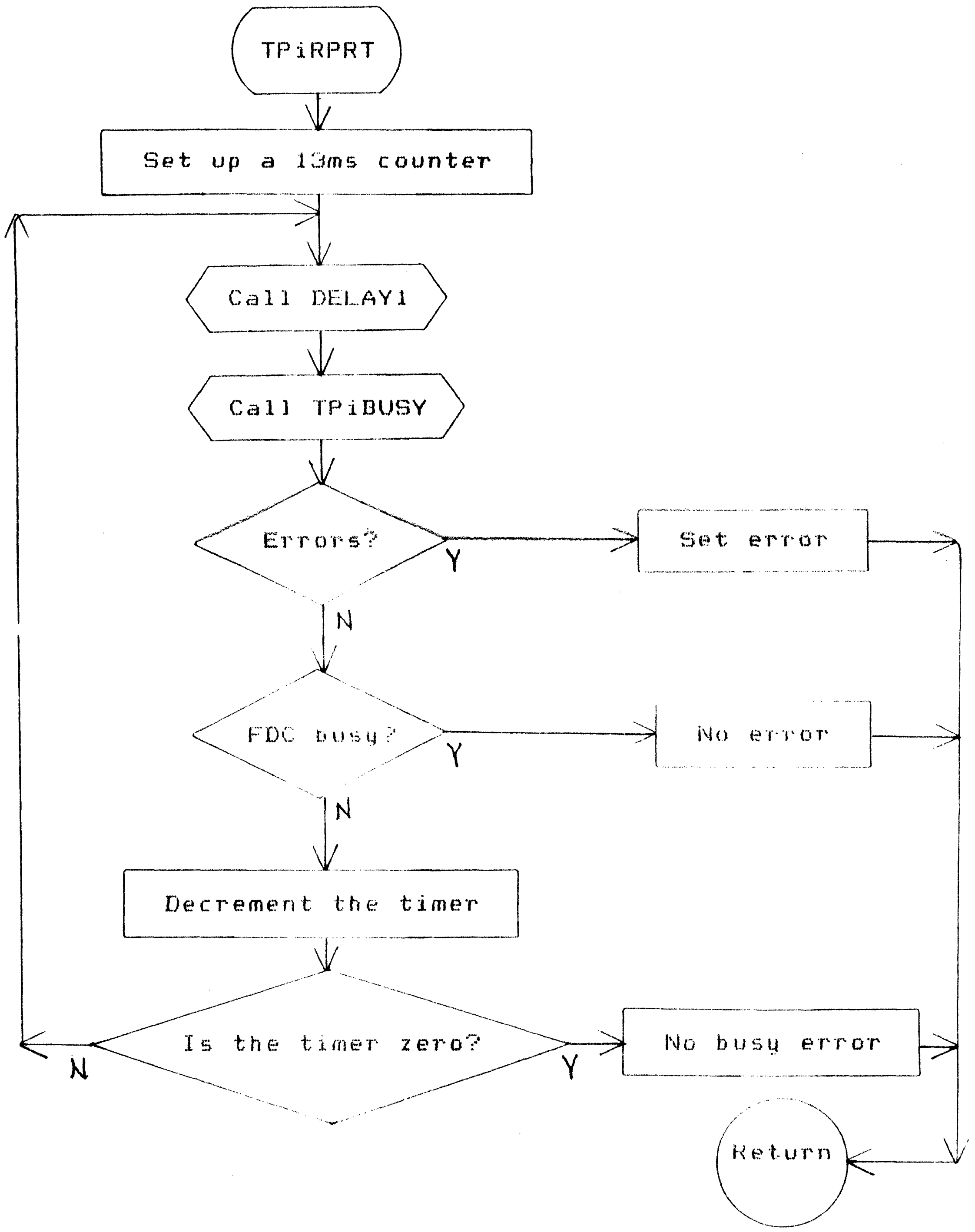
RESULTS
Set up a timer to let FDC send bytes
Enter the FDC status byte
Is the timer zero?
N
Y
Is FDC busy sending data?
N
No error
Y
Enter the FDC status byte
Is it OK to take data?
N
Timeout error
Y
Are more data bytes expected?
N
Bad FDC error
Y
D
B
C

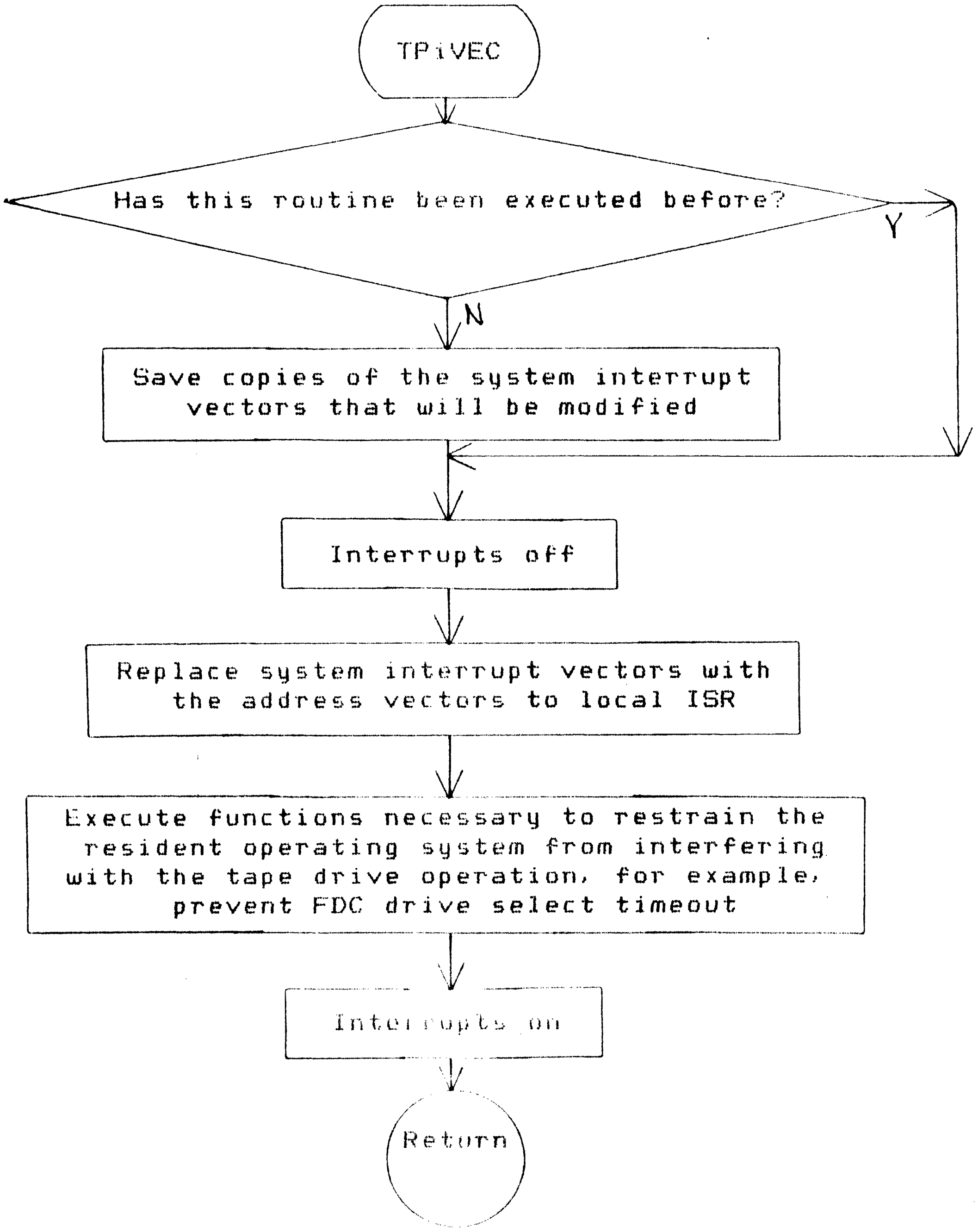

D
B
C
Input the next data from FDC
Store the FDC data
Count the data byte
Return

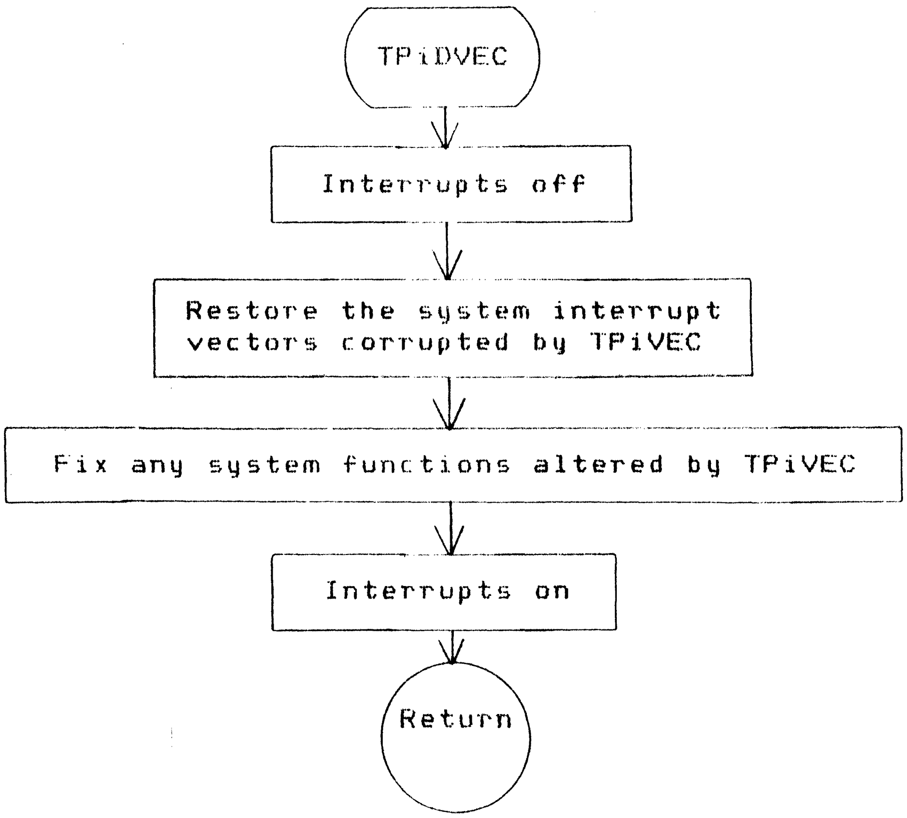


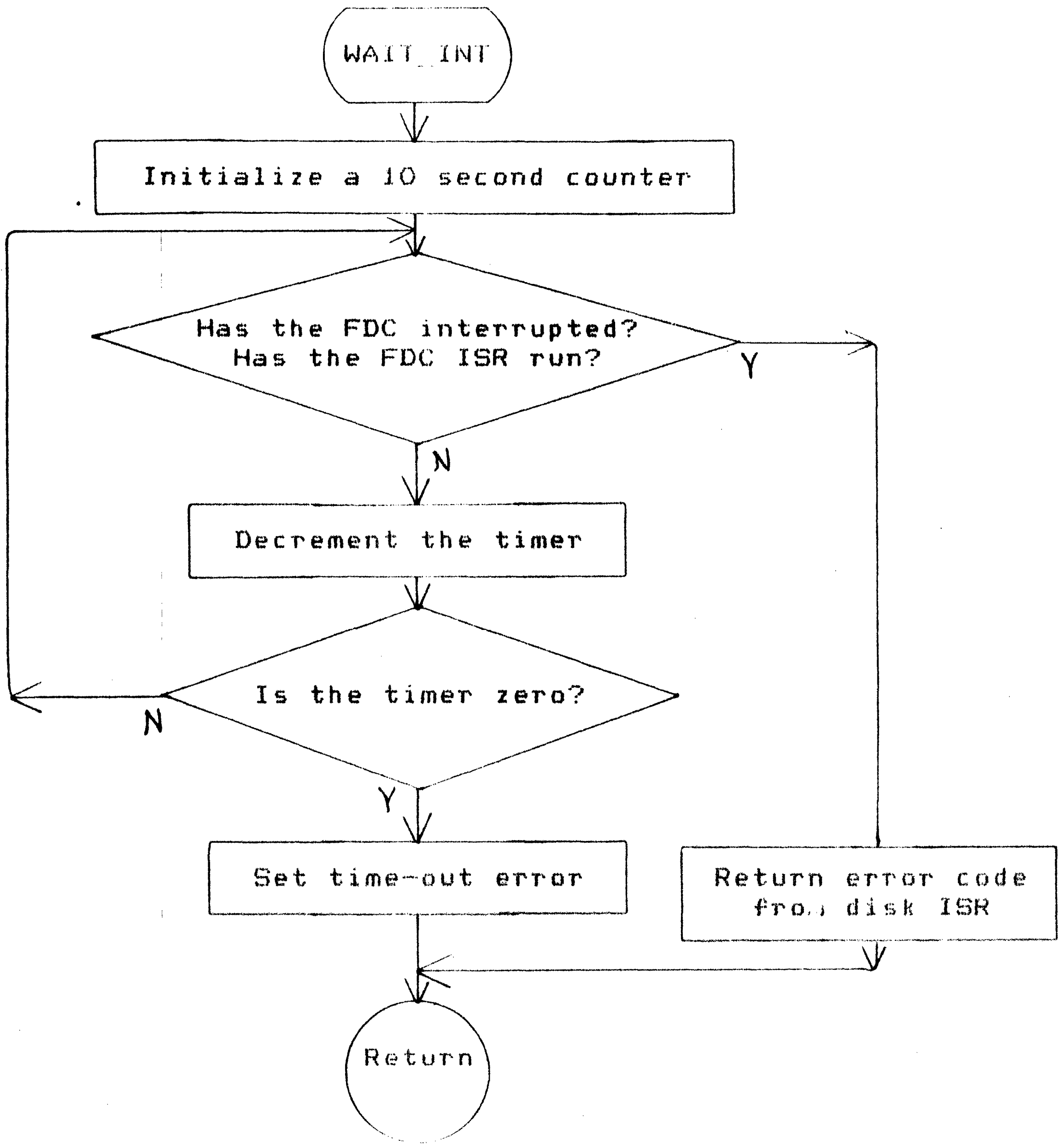
RW_OPN
Call TPiSEL
Convert the number of blocks of data to RED/WRT DMA to the number of bytes to move
Call DMA_SETUP (if applicable)
Errors?
Y
N
Construct a string of byte commands for the FDC to select a RED/WRT DMA function
Call ISSUE_COMMAND
Call CHECK_RECAL
Return


TIMER_FIX
Perform necessary periodic operations to assure the operating system does not interfere with the tape drive functions
Simulate an ISR call via saved vectors
Interrupt return


TPiRPRT
Set up a 13ms counter
Call DELAY1
Call TPiBUSY
Errors?
Y
N
Set error
FDC busy?
Y
N
No error
Decrement the timer
Is the timer zero?
N
Y
No busy error
Return


TPiVEC
Has this routine been executed before?
Y
N
Save copies of the system interrupt vectors that will be modified
Interrupts off
Replace system interrupt vectors with the address vectors to local ISR
Execute functions necessary to restrain the resident operating system from interfering with the tape drive operation, for example, prevent FDC drive select timeout
Interrupts on
Return


TPiDVEC
Interrupts off
Restore the system interrupt vectors corrupted by TPiVEC
Fix any system functions altered by TPiVEC
Interrupts on
Return


WAIT_INT
Initialize a 10 second counter
Has the FDC interrupted?
Has the FDC ISR run?
Y
N
Decrement the timer
Is the timer zero?
N
Y
Set time-out error
Return error code from disk ISR
Return