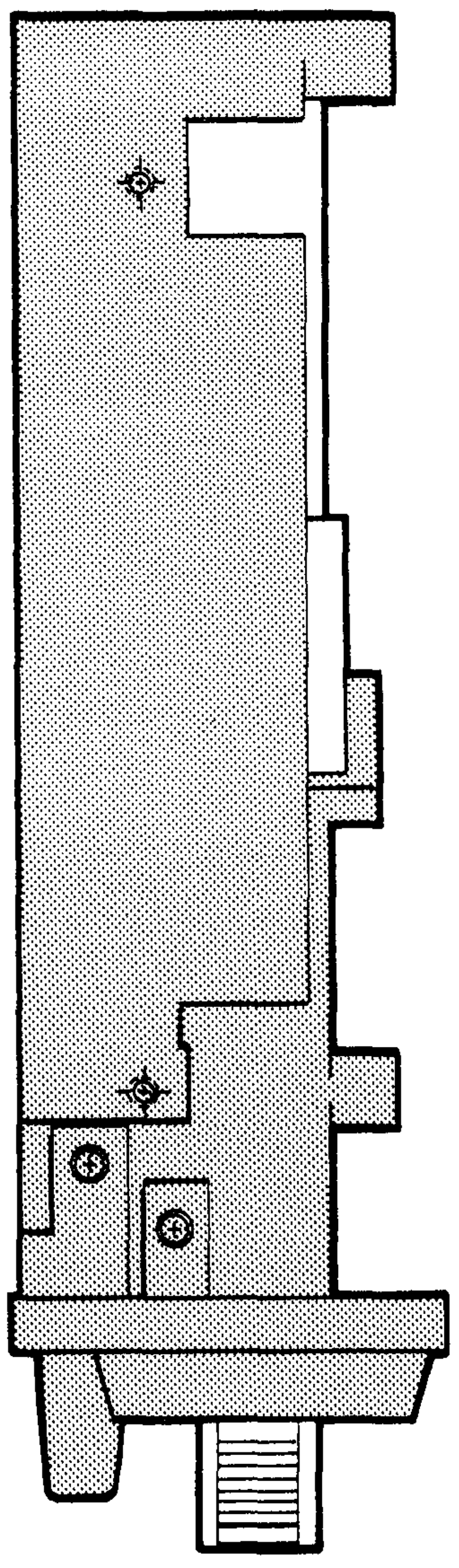
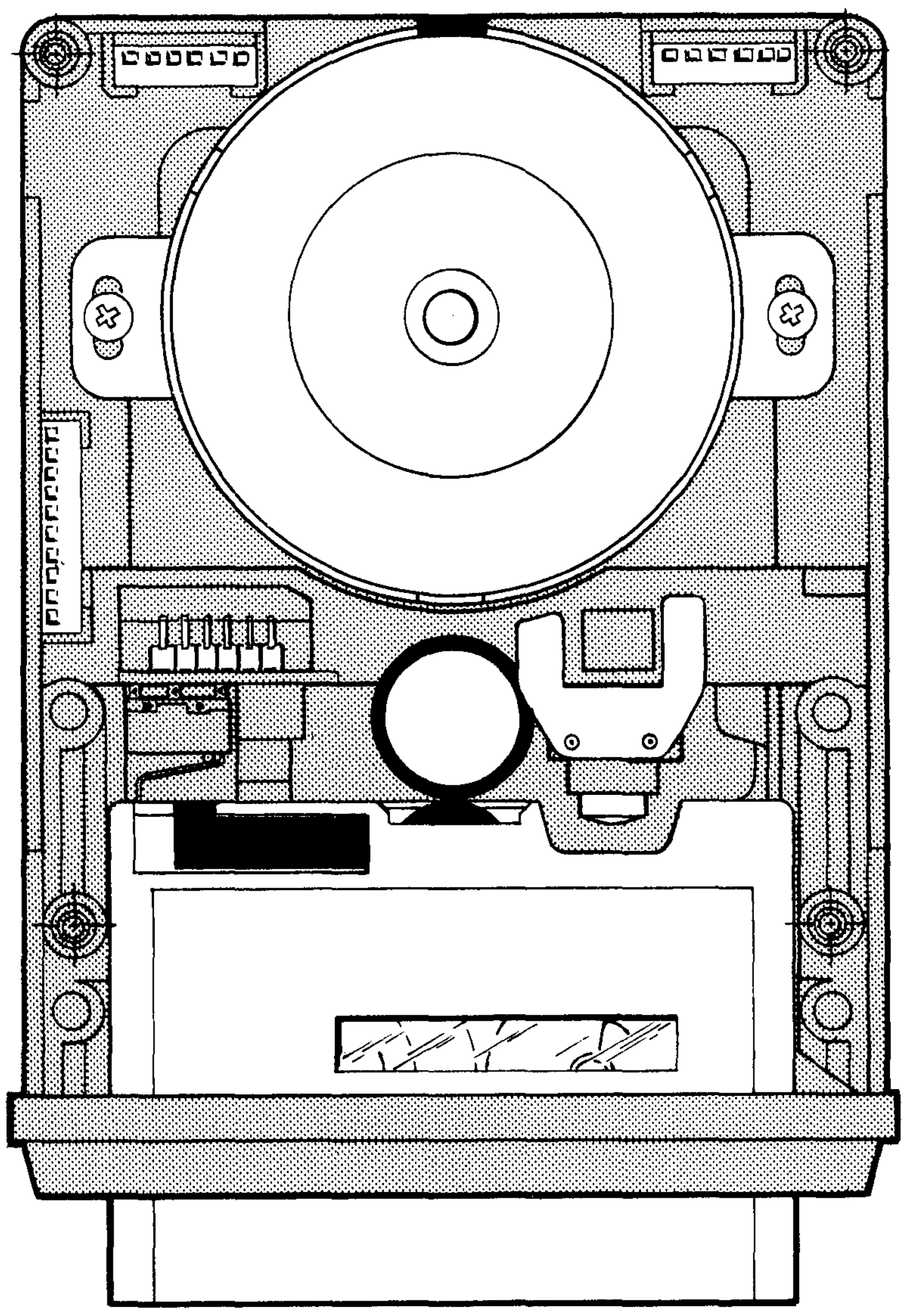
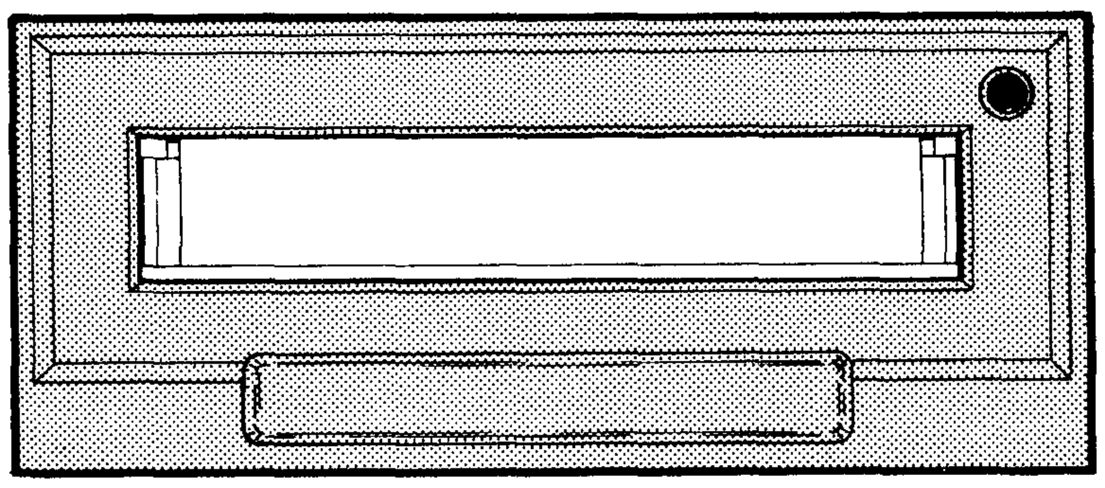
Tape Drive Software Cookbook







IRWIN MAGNETICS

Subject: IRWIN TAPE DRIVE SOFTWARE COOKBOOK

Number: AN-001

Date: September, 1984

1.0 Introduction

This document is intended to be used as a "cookbook" to help you design and test software for the Irwin 110, 210, and 310 tape drives.

We assume that you are familiar with the operation of the floppy disk interface, the disk controller, and other hardware elements of the system. Accordingly, there is no effort to describe any hardware beyond what is required to understand its interaction with the Irwin tape drives. We also assume that you have read the Irwin Tape Drive OEM Manual which defines the terminology used in this document.

Existing Irwin application software used to format tapes, and for image or file-oriented dump/restore operations is referred to often throughout this document.

2.0 Tape Operation and Format

This section describes the three operating modes of the 110/210. Next, the track layout, the block format, and the sector format of the cartridge tape are discussed. Also included in this section is the description of the format of Block 0. Finally, tape and head positioning are discussed.

---- 2.1.0 Tape Operating Modes

The tape drive has three modes of operation: Streaming, Start/Stop, and In-place Update.

--------- 2.1.1 Streaming Mode

The streaming mode is characterized by constant, end-to-end tape motion while reading or writing data. This mode provides the best data throughput since no time is wasted starting and stopping the tape. For example, in this mode 10.35 megabytes of data can be stored on the tape in only eight minutes.

Programs written to support this mode are more difficult to design since they must accommodate continuous read or write operations, where data is read or recorded as a "stream". This usually requires implementing special program procedures to prevent gaps in data when writing, or data overruns when reading.

--------- 2.1.2 Start/Stop Mode

The Start/Stop mode permits starting and stopping the tape anywhere without losing storage efficiency. In conventional tape drives, an inter-record gap (IRG) is produced each time the tape is stopped and restarted. As the number of starts and stops increases, storage efficiency decreases because of the wasted space represented by the IRGs.

The Irwin tape drives avoid this problem by using a formatted tape with identifiable, fixed-length blocks located along its length. Since all blocks are identified, the tape can be positioned to the start of any data block on the tape with no loss of storage capacity.

The Start/Stop mode is useful when processor overhead is such that streaming is impractical. Using the Start/Stop mode permits you to position the tape at a specific tape block to selectively read or write one or more blocks, to retry errors, or to update data in-place.

--------- 2.1.3 In-place Update Mode

The tape drive also permits in-place updating, allowing you to selectively write to any sector in any block on the tape. This mode is a particularly beneficial feature of the Irwin drive because it provides a random-access capability. Using the In-place Update mode, you can maintain and update individual sectors of directories, bad block maps, tape ID blocks, and other files on the tape.

---- 2.2 Tape Servo-writing

Tapes used in the drive must have a servo pattern recorded on them before they can be used to store data. The drives have an internal servo-writer that is implemented using a special firmware module and the drive's read/write head.

To guarantee successful servo-writing, you must first completely erase the tape using a bulk eraser. Failure to do this may leave residual data on the tape (previously recorded servo patterns, for example) that may confuse the drive's track following circuitry.

---- 2.3.0 Tape Format and Organization

During servo-writing the tape is physically and logically divided into tape blocks. Each block occupies 11.95 inches of tape and provides an unformatted capacity of 9,562 bytes. An unformatted tape block is analogous to an unformatted track on a floppy disk. Servo information used for head positioning is recorded at the beginning of each block. This information is for internal drive functions and is not user-accessible.

To a floppy disk controller, the tape "looks" just like a floppy diskette with a lot more tracks. To maintain compatibility, the host software must format each tape block so that it "looks" like a track on the floppy. This includes recording fields such as an index gap, header ID's, header gaps, data fields, CRC's, and so forth, on the tape. Since the format fields are just like those used on the floppy, the part of the format program that produces the data will be very similar to programs used to format floppy diskettes. After formatting, each tape block contains sectors just like the floppy track.

The design of the drive allows you to use just about any tape format you choose. You are free to vary the number of sectors per block as well as the number of bytes per sector. However, most floppy controllers impose restrictions that tend to make some formats better than others. A format with eight sectors per block and 1,024 bytes per sector provides the lowest overhead and maximum space available for data. This is especially true when the drive is being used with floppy controllers that use the NEC 765 or Western Digital 179X chip sets. Accordingly, all of the application software written by Irwin uses this format.

For details regarding disk/tape formats, refer to the NEC 765 and Western Digital 179x specifications and application notes.

When formatting data fields, we recommend using a 6Dh "fill character". The 6Dh pattern is the most difficult MFM pattern to read and consequently provides a read integrity test for the tape media and read channel electronics.

To format the tape with the 5.25" MFM data format standard, we recommend using Write Track commands.

--------- 2.3.1 Tape Format

Irwin has established the following format specifications for the tape cartridge. We use this format extensively because it maximizes the amount of space available for data by minimizing ID header overhead.

The tape is divided into 8 tracks numbered 0 through 7. The tracks are organized on the tape in a serpentine fashion, with even numbered tracks recorded in the forward direction (beginning-of-tape (BOT) to end-of-tape (EOT)), and odd numbered tracks recorded in the reverse direction (EOT to BOT).

Each track is divided into 158 blocks, which are also called cylinders. While blocks and cylinders represent the same areas, blocks are the logical divisions of the tape and cylinders are the physical divisions of the tape. Cylinders are numbered 0 through 157 on each track. Blocks are numbered 0 through 1,263 across the entire tape. (See Figure 1 - Tape Layout, and Figure 2 - Track Layout.)

Each block is further divided into 8 sectors, numbered 1 through 8. The format of each data sector is mini-floppy compatible. Each sector has a 1,024 byte data area, to provide a block capacity of 8K bytes. The total capacity of a tape track is 158 X 8K bytes, or 1.294MB. This translates to a tape capacity of 8 X 1.294MB, or 10.355MB. (See Section 2.3.2 - "Block Format".)

The tape format parameters are stored in block 0, which is the very first block on the tape. Format parameters include the version number of the formatting program, the date of formatting, 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 current volume number and the total volume limit, the time and date of the last tape change, and the track and cylinder of the tape's first free block. Every parameter is duplicated for redundancy checking. (See Section 2.3.3 - "Format Parameters - Block 0".)

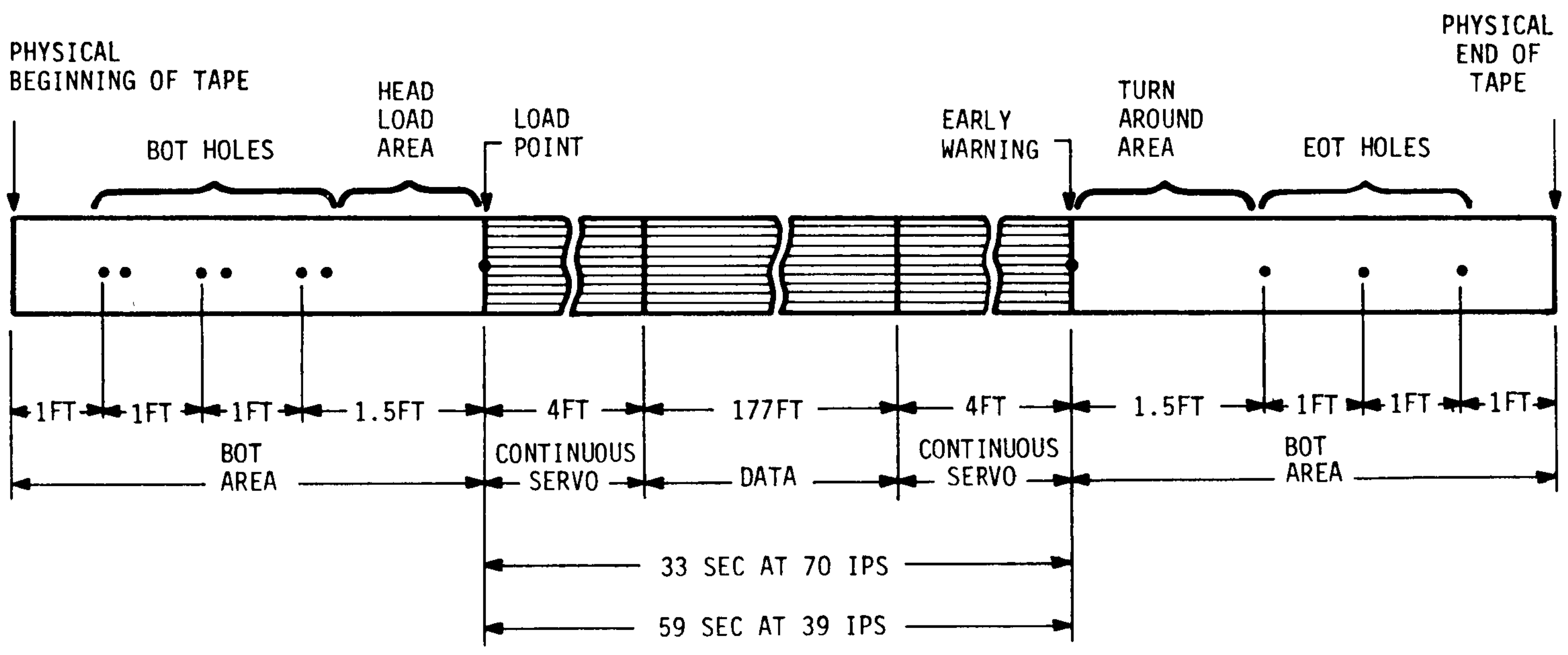


Figure 1 - Tape Layout

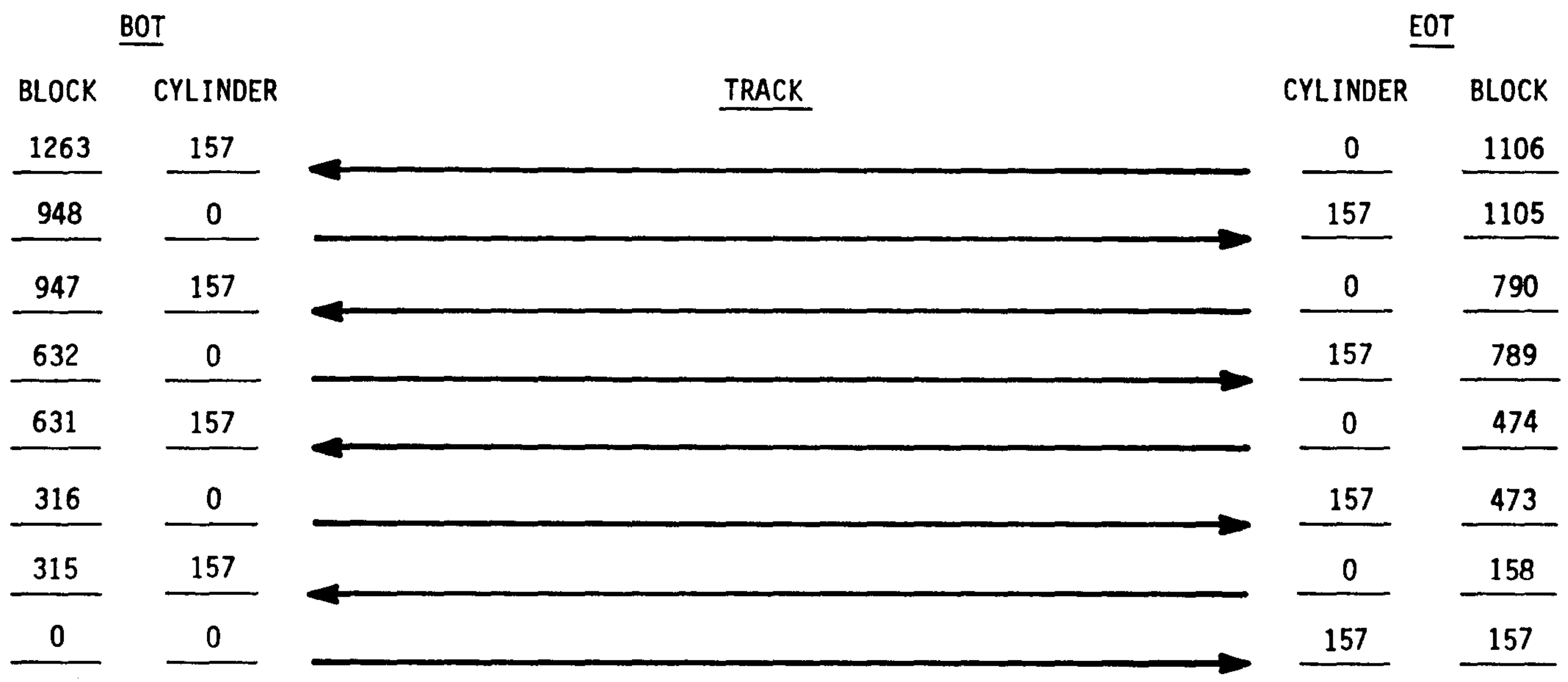


Figure 2 - Track Layout

--------- 2.3.2 Block Format

Each tape block is formatted as shown in Figure 3 - Block Layout. The top section of Figure 3 shows how each block is divided into the block header and 8 data sectors. The lower portion of the figure expands Sector 5 to illustrate the format of an individual sector and its header.

Block Header

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

The following pattern is then repeated for each of the eight sectors.

Sector Format

Number of Bytes	Hex Value	
12	00	
3	A1	(IDAM - ID Address Mark)
1	FE	
1	nn	(Cylinder #; nn = 0 - 157)
1	00	
1	nn	(Sector #; nn = 1 - 8)
1	03	(Bytes per Sector Flag)
2	CRC	
22	4E	
12	00	
3	A1	(DAM - Data Address Mark)
1	FB	
1024	xx	(Data (6Dh suggested))
2	CRC	
54	4E	

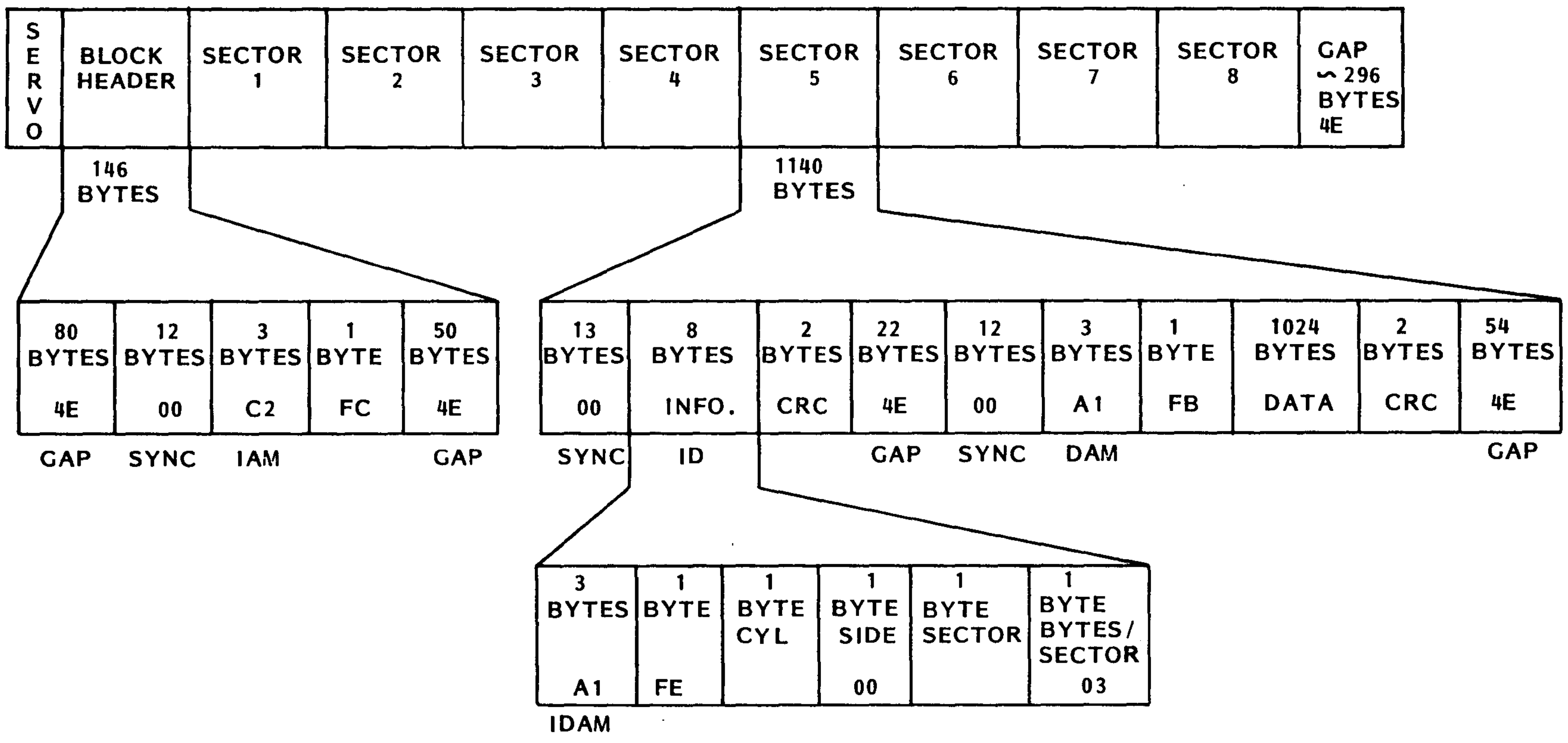


Figure 3 - Block Layout

--------- 2.3.3 Format Parameters - Block 0

Irwin application programs use the first block on the tape (cylinder 0 of track 0) to store format parameters and the bad block map. All of this information is stored in duplicate immediately after formatting the tape. Sector 1 contains the tape's format parameters. Sectors 2 and 3 contain the bad block map and Sector 4 contains nulls. Sectors 5 through 8 are duplicates of sectors 1 through 4.

Although this format is optional, we recommend that you use it to insure interchangeability between products from different manufacturers.

The suggested format parameters are listed in Table 1 - Block 0, Sector 0 Layout.

The bad block map is 1,264 bytes long (one byte for each block on the tape) and occupies all of sector 2 and a small portion of sector 3. Initially, every byte in the table is set to 00h. When a bad block is discovered while formatting the tape or while using the tape, the corresponding byte in the bad block map is changed to FFh.

An alternate approach is to use the bad block map to identify defective sectors. With this approach, any non-zero byte identifies a defective block, while individual bits within the byte identify which of the eight sectors is defective.

Table 1 - Block 0, Sector 0 Layout

Description	Bytes	Data Type
Format program name and version number	0-36	ASCII
Date of formatting (Get 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)	83-89	ASCII
Cylinder of first free block	90-91	decimal
Track of first free block	92-93	decimal
Reserved for application program use	959-1023	

---- 2.4.0 Tape Positioning

The tape can be moved at either of two speeds: 39ips when actually reading or writing data, or 70ips while positioning the tape to a desired data block.

Specific data blocks can be located by counting Index pulses. Index pulses are generated by the tape drive when the embedded servo area, located between each tape block, passes under the tape head. The pulses are coupled from the tape to the floppy controller and are usually made available to the host. When the current block and desired block addresses are both known, positioning is accomplished by issuing a Move Physically Forward (or Reverse) command and counting Index pulses until the tape is within six or seven blocks of the desired position. Stopping the tape takes about 400 milliseconds, which translates to about three tape blocks. By issuing a Move Logically Forward (or Reverse) command the desired position can be approached at 39ips. By reading and interpreting the tape's address headers recorded on tape by your formatting program the desired position can be located exactly.

As an example, if the tape is at cylinder 4 of an even track and you want to move to cylinder 100, you would issue a Move Physically Forward command, count 90 Index pulses (to get to cylinder 94), then issue a Stop command. During the 400 millisecond stop time, cylinders 95, 96, and 97 would pass beneath the head. Then you would issue a Move Logically Forward command and interpret the cylinder headers to locate cylinder 100. The desired read or write operation would then be initiated.

Because the NEC 765 does not provide an Index line, most controllers using the NEC chip set do not provide a way for the host to monitor the Index signal. Some controllers may incorporate external hardware to monitor this Index signal. (On the other hand, the Western Digital 179x shows Index as a bit "S1" in the Status Register for Type One commands.) When Index is not available, some other positioning technique must be used. One method that has proven successful with the NEC 765 is to repeatedly ask it to find an invalid sector. The chip is designed to try to locate a specified sector for two complete disk revolutions, which is the same as the passing of two tape blocks. If it is unsuccessful, the chip times out. By repeatedly commanding the controller to seek an invalid sector and by counting the number of timeouts, you can simulate an Index pulse counter.

Another method of positioning is to initiate movement in the desired direction and use a timer to time the passage of each block. At 70ips, it takes 192.4 milliseconds for a block to pass beneath the head. While calculating the time you must also account for the 400 milliseconds required to start and stop the tape. This method is less accurate due to cumulative errors in speed and start/stop timing.

When using any of these positioning techniques, you must use a "read header" routine to read and interpret block address headers. It is only by reading these headers that you can be absolutely certain of tape position.

Ideally, after positioning the tape will be only one or two blocks ahead of the desired block. However, unless Index pulses are actually counted, it is possible to overshoot the desired position. If this happens, issue one or more Pause commands to move the tape backward to the desired block. Each Pause command moves the tape back two blocks towards the start of the file (i.e. "against" data).

Each tape block is 13.47 inches long, so at a tape speed of 70ips and with a stop time of 400 milliseconds, it takes about three blocks to stop the tape. Since the tape's start and stop times are equal, it also takes three blocks to restart the tape, making the total start/stop overhead six blocks. Thus, in the forward direction, you should stop the tape at least six blocks short of the desired position. When you are approaching the desired block from the reverse direction, the start/stop times are overlayed and cancel each other. To be safe, you should stop the tape a block past of the desired position, allowing it to coast to a position four blocks past the desired point. (See Figure 4 - Tape Motion Timing.)

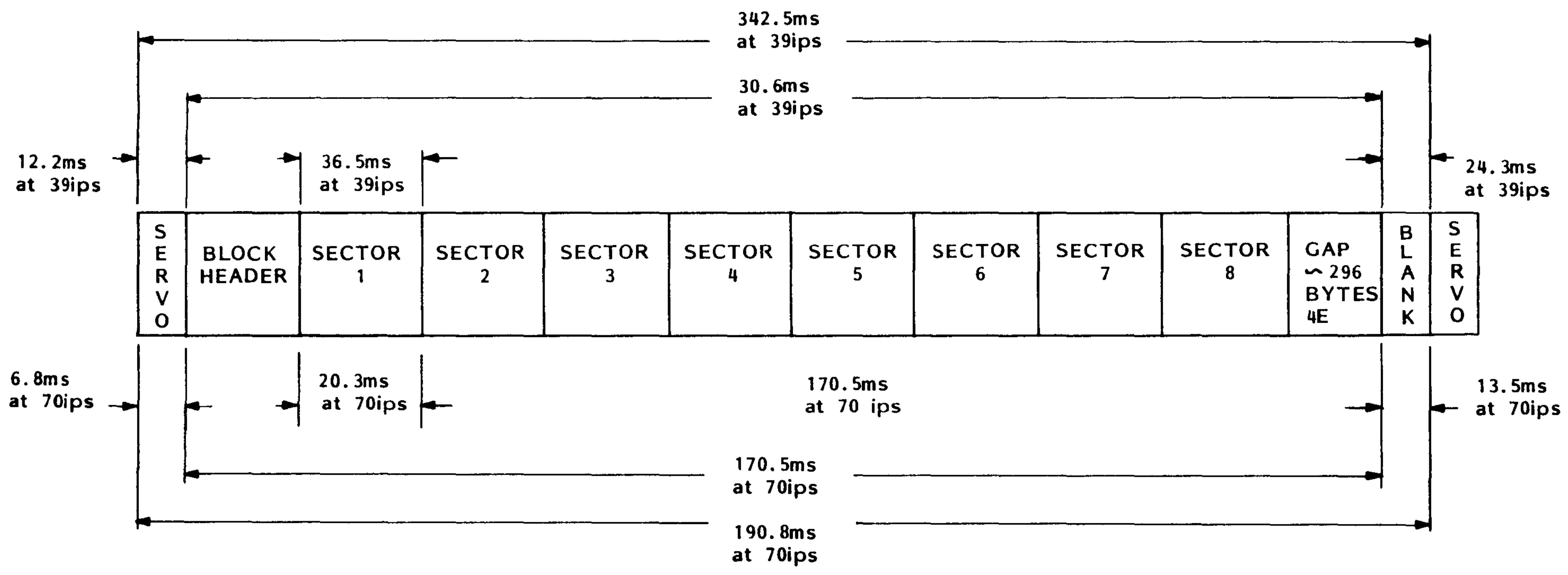


Figure 4 - Tape Motion Timing

--------- 2.4.1 Access Time

At the drive's 39ips read/write speed, it takes 59 seconds to move from BOT to EOT. At the 70ips positioning speed, BOT to EOT takes 33 seconds. Use these figures to calculate access time.

At 59 seconds per track, it takes about eight minutes to read or write the entire tape.

---- 2.5.0 Write Data Verification

Like a floppy disk, the tape drive has no "on the fly" read-after-write capability. Consequently, to insure data integrity, data recorded on tape must be verified by one of two methods. First, verification can be performed by simply rereading the original data and comparing it to the data read from the tape. Secondly, verification can be performed by reading the data from the tape and watching for CRC errors produced by the floppy disk controller.

Since tape positioning is time consuming, you should attempt to make just two passes -- one to record everything that is to be recorded, and one to read it back for verification.

---- 2.6.0 Head Positioning

Proper head positioning is automatically accomplished by the tape drive's microprocessor through the use of servo information on the tape. At the beginning and end of tape head positioning is almost instantaneous because of the continuous servo information. In the middle of the tape the servo patterns occur as small sections between each data block. Thus, track-to-track positioning may require the passing of 2 or 3 blocks. This should be taken into account when preparing for a data transfer from another track by positioning the tape 2 or 3 blocks before the desired block.

3.0 Hardware Considerations

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

---- 3.1.0 Floppy Disk Controller

There are two basic kinds of floppy disk controllers. One type is based around one of the commonly available floppy disk controller chip sets. The second type uses discrete logic, a microprocessor (possibly bit-slice), or both.

Most of the floppy controller chips are functionally the same. The two most popular chips are the Western Digital 179X series and the NEC 765, so the following discussion is confined to these. These chips have a processor interface on one side and a floppy disk interface on the other.

The most important requirement for a floppy disk controller is to have a mechanical and electrical 5.25" interface that is compatible with the Shurgart SA450 standard. Most 5.25" 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 respond to any of the four device selects signals, and does not use the Motor ON, Side Select, or Direction signals.

Mechanically the tape drive has a 5.25" half-high "footprint". Electrically it uses the same interface and power connectors used by SA450-compatible disk drives. It can be connected in a daisy-chain arrangement with floppy disks, and is addressed the same as a floppy. When the tape drive is the last device on the daisy-chain, it must have a terminator resistor pack (SIP style) installed. Otherwise, the terminator pack, which is provided with the drive, must be removed.

Another area of consideration is the write precompensation of the floppy disk controller. The tape drive works best with a write precompensation of 250 nanoseconds for all blocks. Most controllers have an adjustable write precompensation circuit since compensation varies from one type of disk to another. Adjusting write precompensation below the recommended 250 nanosecond specification may increase the number of soft errors, which show up as CRC errors at the controller. In some cases, however, we have been successful reading data that was recorded with only 125 nanoseconds of write precompensation.

The last area of floppy disk controller concern is the MFM Data Separator or Data Recovery circuit. A Data Separator recovers a serial data stream and the appropriate clock bits. This type of electronics is commonly found in a discrete logic or microprocessor-based controller. Most of the chip-based controllers, such as those using the NEC 765 and the Western Digital 179x, have 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 reading the floppy's outside tracks than the tracks on the inside. This is due to the fact that the bit density is higher on the inside tracks.) Poor data recovery technique will have an adverse effect on soft error rate.

--------- 3.1.1 8" Floppy Disk Compatibility Considerations

Eight-inch floppy disk controllers are incompatible with the tape drive but can usually be modified to work. The two devices have incompatible data rates and connecting cables. The data rate used in the eight-inch floppy is 500KHz, unlike the 250KHz data rate used in the tape drive. Also, the data and power connections used in eight-inch drives are mechanically different.

These incompatibilities can usually be resolved by modifying the controller. In most cases the data rate can be reduced by halving the clock frequency (usually with flip-flops) to the controller chip. With the addition of a simple logic circuit, the controller can be made to operate at either of two software-controlled data rates.

Since eight-inch controllers generally use the same I/O lines as the tape drive, all that is usually required is a simple mechanical adapter to adapt the standard 50-pin I/O ribbon cable to the tape drive.

---- 3.2 DMA - Direct Memory Access

In most microcomputers, a single processor is used to transfer data between all peripherals. With disk and tape controllers, the processor is a slave of the controller during data transfers 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 using a single processor, the processor initially does a disk access to retrieve data, then does a tape access to store data. While the processor is reading information from the disk, the tape must remain motionless because the processor cannot do two things at once. With a single-processor design, a "save" operation involves moving data from disk to memory, then from memory to tape as the microprocessor sequentially moves each chunk of data.

Direct Memory Access (DMA) circuits allow the hardware to perform independent operations while the processor's software program is involved with other tasks. Some microcomputers have more than one DMA channel, permitting simultaneous transfers to two different peripherals. With a dual-channel DMA, for example, one channel can be transferring data from hard disk to memory, while the second is transferring data from memory to the tape. With a single-channel DMA, one of the peripherals (tape or disk) typically has DMA capability while the other relies on the host processor to move data between memory and the peripheral.

In some instances, both DMA channels cannot use the same memory buffer simultaneously, so the memory must be partitioned to let the disk load one partition while the tape is unloading the other. When the operation is finished, the devices switch partitions.

The processor's role in this is typically to turn channels on and off and to manage the starting address and size of the buffer transfers. The absolute size of the memory buffer required depends upon the system timing differences between transfers to and from disk and tape.

---- 3.3 Interrupts

Interrupts give a peripheral the ability to capture the attention of the processor and direct it to an associated interrupt service 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 to prevent other peripherals from interrupting the host processor. An interrupt may stop data transfer and require costly time to reposition the tape. You should make sure that unnecessary hardware interrupts are disabled when performing tape data transfers.

For more information on software interrupts, see Section 4.3.2 - "Software Interrupts".

---- 3.4 Controller Accessability

To operate the tape drive using an SA450 interface, low-level device software must allow the hardware to perform primitive floppy disk operations. These operations include selecting the drive, head stepping, transferring data, and controller operation interruption. For these reasons, the software has to have access to the controller's hardware.

The level of hardware accessibility depends upon 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 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 low-level operations as needed to accomplish higher-level functions. While these controllers are generally incompatible with the tape drive, it is sometimes possible to reprogram their ROM to provide the kind of low-level support the tape drive needs.

---- 3.5 Host Memory Requirements

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 minimize the number of start/stop operations.

---- 3.6 Power Supply

It is important that the host power supply provides sufficient power for the tape drive. Software problems and failure to stop at the tape's EOT or BOT mark are some of the problems caused by inadequate supply of power. Power supply problems are the most frequent cause of tape drive failure. (See Irwin Application Note AN-002 - DC Power Considerations for the 110/210.)

4.0 Software Considerations

To minimize the cost of the tape drive subsystem, Irwin designed the tape drive so it interfaces directly to an existing floppy disk controller, rather than using a separate tape controller. Although the tape drive's hardware is totally compatible with floppy hardware, the drive is not intended to be operated by floppy disk software drivers. This requires writing separate software drivers that recognize the tape drive's unique identity.

This section considers the conceptual design of application programs by discussing low-level device drivers, tape drive needs, and application considerations. This section leads you through the design of a typical application program.

---- 4.1 General Software Requirements

While the tape drive is different from a floppy disk drive, many aspects of the software are the same. This is one advantage of using the same controller for both devices. However, since the controller was initially designed for floppy disks, it is necessary to program it to "think" like a tape drive controller for many tape drive operations.

The software has two major areas of responsibility: data transfer and tape positioning. Data transfer operations use software that is practically identical to floppy software, while tape positioning requires a completely new set of modules.

In general, the positioning software needs to convert a floppy disk controller into a tape controller. This involves programming the floppy controller hardware, whether comprised of discrete circuits or controller chip sets, 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 used. A good understanding of the operation of these controller chips is a prerequisite for writing effective low-level device drivers.

---- 4.2.0 Software/Hardware Interaction Concerns

When programming a tape or disk peripheral, the peripheral requires that the proper amount of data be transferred at a specified rate and time. Good overall performance depends on the interaction between hardware and software.

The performance of the tape drive is optimized when data is transferred continuously as a constant "stream". This means that the program must transfer data fast enough to keep the tape moving without causing gaps (during writes) or data overruns (during reads). Continuously streaming data is desirable because this minimizes tape start/stop operations which are very time-consuming.

There are three areas of software/hardware interaction that must be addressed: interrupts, DMA operation, and memory buffers.

--------- 4.2.1 Interrupts

It is necessary to know which interrupts are required by the system and when they will occur. If an untimely interrupt occurs during tape transfers, the tape may not be serviced as needed, requiring you to reposition the tape. To prevent this, interrupts should be disabled during tape data transfers.

You will have to determine which interrupts can and should be disabled. Since the floppy controller interrupt is set up to always vector to the floppy interrupt service routine, you will have to revise the program to vector to either a floppy or a tape drive interrupt handler, depending upon the current mode of the controller.

Other interrupt routines, depending on the operating system, may have to be patched 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 after tape drive operations are completed.

--------- 4.2.2 DMA Operation

DMA is necessary to permit "streaming" operation. Knowing how your system's DMA channel works is a prerequisite to writing effective data transfer routines. Things to consider are the transfer speed of data over the DMA channel(s), setup time, and buffer design. Buffer design factors include the number of buffers to use, buffer size, the buffer address, and buffer speed.

--------- 4.2.3 Memory Buffers

Memory buffer usage is a function of the DMA hardware architecture and the amount of memory available for use as a buffer. If your system doesn't have a DMA channel, then use as large a buffer as you can. A large buffer will let you transfer as much data as possible to or from the tape without interruption, and as a result, minimize the number of starts and stops. If the system has DMA, optimum buffer size depends upon the DMA channel's architecture and speed.

With the aid of DMA and interrupts, a buffer management scheme using overlapped I/O is a good way to minimize tape repositioning. In this scheme, two or more memory buffers are used in the data transfer. One DMA channel continuously loads data into the buffer(s) from the tape or disk, while a second channel unloads data and transfers it to the opposite peripheral. 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 overlapped I/O, use as much memory as possible for the buffer(s). Also, remember that hard disk data transfers, in most cases, will be faster than transfers to or from the tape. Because of this, keep the buffer(s) full of data during disk-to-tape transfers, and empty for tape-to-disk transfers.

It sometimes happens, however, that tape transfers have a higher effective transfer rate. This is primarily caused by the processor overhead associated with working with disk directories, searching for particular files, servicing interrupts, and hard disk errors (retries). If this happens, the tape will have to be stopped while the disk catches up.

To minimize the number of starts and stops, once you are forced to stop the tape it is a good practice to completely fill the buffer (or empty it, in the case of tape-to-disk transfers). This gives the disk a head start on the tape and lets it run longer before things get out of hand again.

---- 4.3.0 Software Design

Most application program design is done using a "top-down" approach. Using this approach, the user's needs are determined first and then software modules are specified, progressing down from the required user functions to the primitive operations known as the low-level device drivers.

Using a floppy controller to control the tape drive requires rethinking this traditional approach. For the tape drive, you must define program operations from both a top-down and a "bottom-up" standpoint.

In the bottom-up approach, the low-level device driver routines are considered first. The modules are then implemented by specifying software modules through hierarchical levels, progressing up to the user interface. Where the two designs meet is where the optimal trade-off between programming goals, user's needs, hardware considerations, and operation speed occurs.

An example of the merged approach can be illustrated by the approach used to design a disk backup program. A disk backup program must permit unsophisticated users to quickly and easily save hard disk files on the tape. This requires a program that is easy to operate, fast, and fail-safe. (The companion "restore" program must also be easy to use, but can take longer since it is used less frequently.)

To get started on such a program, you must first know how the controller hardware works and what it is capable of doing. In addition, you must know how the operating system works, how files are structured, and how to interface to them.

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 are to be done, and how the DMA and memory are to be used.

--------- 4.3.1 Software Transportability

Software transportability is another consideration when writing programs of any kind. Transportability means the ability to export software across hardware, operating system, and file system boundaries. Hardware boundaries are crossed by low-level device routines, usually written in assembler, while other programming is done in a high-level language (such as "C"). Operating systems and file system boundaries can be crossed if you design your program modularly, keeping all system dependent functions in a small number of program modules.

--------- 4.3.2 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 status knowledge 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 controllers and drives. Recalibration causes the tape program to "forget" the contents of the NEC 765's track register. This, in turn, impacts the software's ability to send 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.

---- 4.4 Software Design Example

We begin the software design process with a top-down design procedure to determine the general flow of the program. A block diagram showing the program outline can be very helpful.

Figure 5 - IMAGE Program Block Diagram, shows the organization of the Irwin IMAGE program that performs an image backup from disk to tape and an image restore from tape to disk.

At the top level is the user interface, if any, followed by the main structure of the program. Further down the block diagram are the data handling and manipulation routines. These mid-level modules deal with the operating system being used, the desired data organization on tape, and considerations about the specific hardware used. For these mid-level modules, you should probably move away from the top-down approach, and begin favoring the bottom-up approach. At the bottom level of the program are the low-level I/O and hardware interfacing routines.

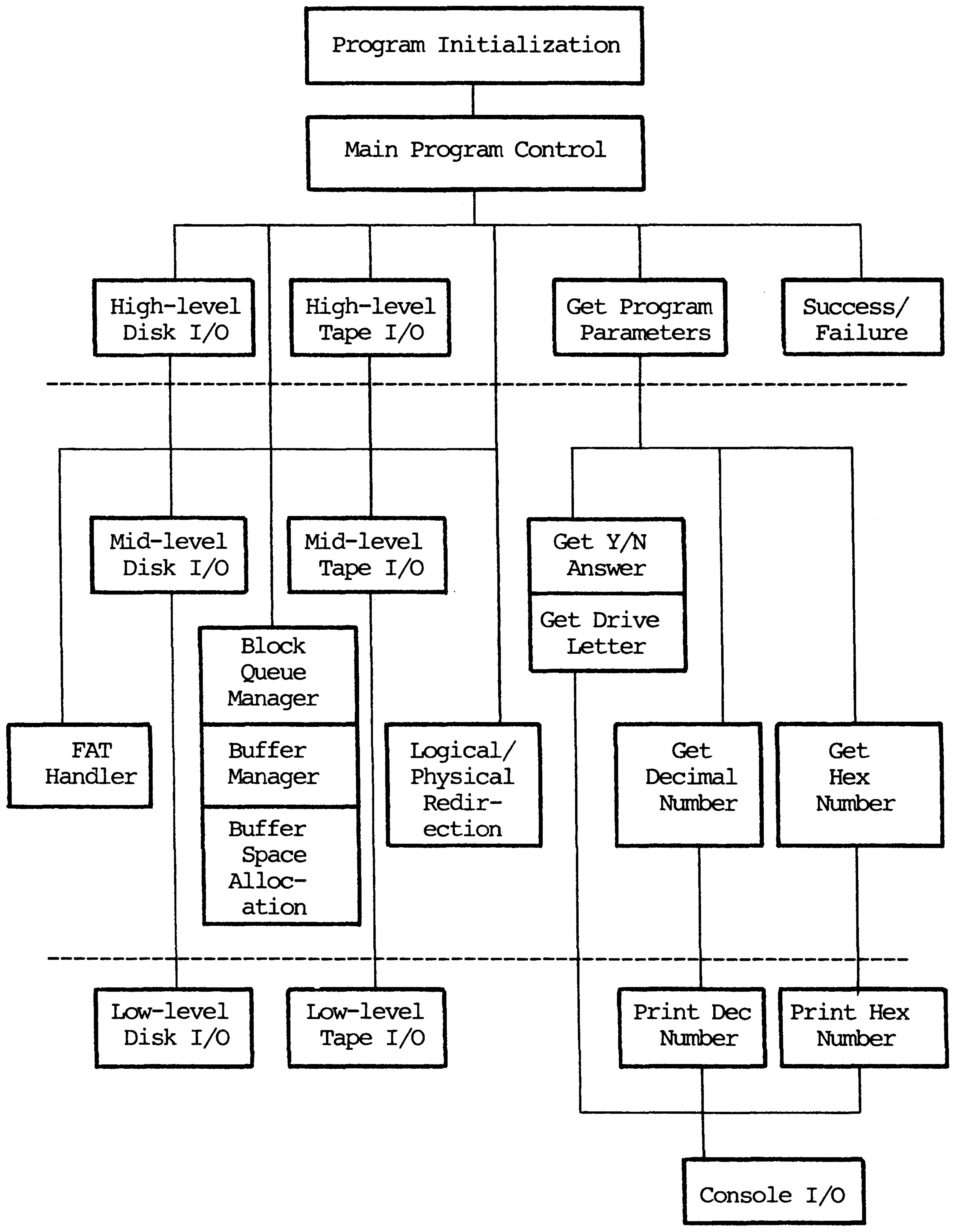


IMAGE Program Block Diagram

In most tape application programs there will be similar main program tasks. Following 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.

User Interface

The user interface routines provide two-way communications by getting any information needed to operate the program, and outputting status or information back to the user. All error messages should be handled through the user interface. Ergonomics and other human factors should be considered 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 any calculations required and pass the results to the calling program. The information can be the number of tapes needed, which bytes to transfer, which flags to set, or 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 provide 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 the tape are done here. These routines manage tape data flow and tape formatting. Most of the application program code will be in these routines.

Buffer Manager

These routines organize and manage the memory buffers and interface to the DMA channel if one is available. Timing requirements are critical and consequently will be a major consideration for these modules.

--------- 4.4.1 Mid-level Tape I/O Routines

The mid-level tape I/O routines allow you to move the application software across operating system and hardware boundaries. These routines are called by the tape file manager and buffer manager and 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 done automatically. On the following pages is a list of the suggested mid-level routines.

TP1ONL

Get Drive Line Number

Calling Parameters

None

Return Parameters

None

Description

Gets the tape drive's physical unit number and stores it for future reference. This routine is called once per program to insure that the controller and tape drive are on-line and operational.

TP1OFL

Remove Drive Line

Calling Parameters

None

Return Parameters

None

Description

Sets the drive off-line. This routine is called once per program.

TP1REDMNT

Read Mount

Calling Parameters

None

Return Parameters

None

Description

Prepares a new tape for reading. This routine assumes that the drive is already on-line. If necessary, the user is asked to insert a tape cartridge. A seek load point operation is performed to position the head to BOT over track 0.

This routine is called once per tape change.

TP1WRTMNT

Write Mount

Calling Parameters

None

Return Parameters

None

Description

Prepares a new tape for writing. This routine is identical to the read mount routine except that the write protect status is checked to see if writes to the tape are allowed. If not, the user is asked to insert a non-protected tape.

TP1RED

Read Block

Calling Parameters

Buffer Address - Where to start storing data read from tape.
Block   - Block number.

Return Parameters

None

Description

Reads a single block of data from tape. All of the tape positioning required to position the desired block under the read/write head is handled internally.

The read block operation will fail if the routine is unable to initiate the read (because of a buffer boundary error or simply a controller error). A failure indicates a fairly serious problem, so a retry would also fail. Any less serious problems will be discovered by the Read Wait routine (TP1REDWT).

When this routine returns, either normally or abnormally, tape motion continues.

TP1REDWT

Read Wait

Calling Parameters

None

Return Parameters

None

Description

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	- The starting address for data to be recorded on tape.
Block	- Block number

Return Parameters

None

Description

Transfers a block of data from the buffer and records it on the tape. Tape positioning required to position the desired block beneath the read/write head is handled internally by the routine.

The write block operation fails if the routine is unable to initiate the write (because of a buffer boundary error or simply a controller error). A failure indicates a fairly serious problem, so a retry would also fail. Any less serious problems will be discovered by the Write Wait routine (TP1WRTWT).

When this routine returns, either normally or abnormally, tape motion continues.

TP1WRTWT

Write Wait

Calling Parameters

None

Return Parameters

None

Description

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

<u>TP1PAUSE</u>

Pause

<u>Calling Parameters</u>

None

<u>Return Parameters</u>

None

<u>Description</u>

Backspaces the tape two blocks, then stops. This routine is used when there must be a temporary pause in the streaming mode to correct for tape/disk timing differences.

<u>TP1STOP</u>

Stop

<u>Calling Parameters</u>

None

<u>Return Parameters</u>

None

<u>Description</u>

Backspaces the tape one block, then stops. 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.

TP1CONT

Continue

Calling Parameters

None

Return Parameters

None

Description

Starts the tape in the logical forward direction . This routine is called after TP1PAUSE or TP1STOP to restart tape motion.

TP1DMNT

Dismount

Calling Parameters

None

Return Parameters

None

Description

Dismounts,   or "unloads", the tape. A seek load point command is issued without waiting for completion.

--------- 4.4.2 Low-Level Device Routines

The low-level device routines are intimately involved with hardware operations. These routines are custom written for the hardware used with any given system. They are typically written in assembler and are called only by the mid-level routines. On the following pages is a list of the suggested low-level routines.

Table 2 - TPO Return Codes

These are the suggested return codes for the low-level (TPO) tape I/O routines. These are the return codes used by Irwin's own applications software. The codes that each routine will return depend upon the implementation of the TPO routines, which is in turn dependent on the configuration of 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 program problem	nn (other)

TPOINI

Controller Initialization

Calling Parameters

Load Time	- Head load time in milliseconds (suggest 4).
Unload Time	- Head unload time in milliseconds (suggest 480).
Step Rate	- Step speed in milliseconds (suggest 6).
I/O Gap	- Gap length to use for read/write (suggest 017h).

Return Parameters

None

Description

Initializes the floppy disk controller for tape usage. The floppy hardware interrupt vectors are saved and replaced with different interrupt vectors for the tape routines. Depending on the controller, the calling parameters are passed on to the controller, saved for reference, or simply ignored.

Any software or hardware initialization that needs to be done once per program should be done by this routine.

TPOTRM

Controller Termination

Calling Parameters

None

Return Parameters

None

Description

Terminates the use of the floppy disk controller to perform tape functions. All of the initialization processes are reversed by restoring floppy disk parameters. Most notably, the floppy hardware interrupt vectors are restored.

TPOONL

Drive Select

Calling Parameters

Drive	- Drive number.

Return Parameters

None

Description

Selects or initializes the indicated drive. This routine is performed with a controller reset and as part of 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

Deselects 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 some systems, this routine may do nothing, or not even exist.)

TP0RECAL

Recalibrate

Calling Parameters

Drive	- Drive number.

Return Parameters

None

Description

Attempts to "awaken" the drive by resetting the controller and issuing a recalibrate command to the tape drive. This routine is performed automatically when required to clear a drive error.

TP0RESET

Controller Reset

Calling Parameters

None

Return Parameters

None

Description

Attempts to "awaken" the controller by issuing a controller reset command. This routine is performed automatically when necessary to reset the controller following an error.

TP0BUSY

Check for Busy

Calling Parameters

Drive	- Drive number.

Return Parameters

Busy	- Drive busy flag.

Description

Checks to see if the specified drive is busy.

TP0COMM

Issue Command

Calling Parameters

Drive	- Drive number.
Steps	- Number of step pulses in the command.
Wait/status	- Flag meaning "Wait until end, then report status".

Return Parameters

None

Description

Issues the command which corresponds to the number of step pulses specified. (See Table 3 - Step Pulse Command List on the following page.) If the wait/status flag is set, the routine waits until the command is executed, then returns the status in the return code. Otherwise, it returns with no status.

Table 3 - 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
Enter format mode (turn on second index pulse)	16
Exit format mode (turn off second index pulse)	17
Seek track n (0 <= n <= 7)	20 + n
Write servo	31
Recalibrate	32 +

TP0REDI

Initiate Read

Calling Parameters

Drive	- Drive number.
Buffer Address	- Starting address of buffer for storing data.
Cylinder	- Cylinder number.
Sector	- Sector number.
Sector Count	- Number of sectors to be read.

Return Parameters

None

Description

Initiates a read tape operation. If the system has DMA, the routine returns immediately and reports any errors. If the system does not have DMA, the routine returns only after reading the specified data and saves an error code that is reported later by TP0IOWT.

TP0RED

Read

Calling Parameters

Drive	- Drive number.
Buffer Address	- Starting address of buffer for storing data.
Cylinder	- Cylinder number.
Sector	- Sector number.
Sector Count	- Number of sectors to be read.

Return Parameters

None

Description

Performs a complete read operation by sequentially calling TP0REDI and TP0IOWT. Any errors are returned immediately.

TP0WRTI

Initiate Write

Calling Parameters

Drive	- Drive number.
Buffer Address	- Starting address for data to be recorded.
Cylinder	- Cylinder number.
Sector	- Sector number.
Sector Count	- Number of sectors to be written.

Return Parameters

None

Description

Initiates a write tape operation. If the system has DMA, the routine returns immediately and reports any errors. If the system does not have DMA, the routine returns only after writing the specified data, and saves an error code that is reported later by TP0IOWT.

TP0WRT

Write

Calling Parameters

Drive	- Drive number.
Buffer Address	- Starting address for data to be recorded.
Cylinder	- Cylinder number.
Sector	- Sector number.
Sector Count	- Number of sectors to be written.

Return Parameters

None

Description

Performs a complete write operation by sequentially calling TP0WRTI and TP0IOWT. Any errors are reported immediately.

TP0IOWT

I/O Status (for Wait)

Calling Parameters

Drive	- Drive number.

Return Parameters

None

Description

Waits for the completion of the TP0REDI/TP0WRTI I/O. After completion of the I/O this routine returns and reports any errors to the calling program.

TP0SNS

Write Protect Status

Calling Parameters

Drive	- Drive number.

Return Parameters

Protected	- Write protect status.

Description

Returns the status of the cartridge's write protect tab.

TP0NDX

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 implementations where only an even number of pulses are counted, odd numbers are rounded down.

TP0FRMFL

Fill Format Buffer

Calling Parameters

Buffer Address	- Starting address of the buffer used to store format data
Sector Length	- Length of each data sector.
Sector Count	- Number of sectors per block.
Format Gap	- Gap length actually written (suggest 034h).

Return Parameters

None

Description

Fills the buffer that is used to format one tape block. The buffer must be longer than a tape block, as unformatted tape area varies with tape speed error and timing. (Twice the block length is sufficient, as it depends on the gap length.)

This buffer contains the format information to be written in the data area between the servo information bursts. The gap length specifies the byte time length of the write splice between adjacent sectors.

TP0FRMT

Format

Calling Parameters

Drive	- Drive number.
Buffer Address	- Starting address of the buffer to be used for format data.
Cylinder	- Cylinder number.
Sector Length	- Length of each sector.
Sector Count	- Number of sectors per block.
Format Gap	- Gap length actually written (suggest 034h).

Return Parameters

None

Description

Formats the specified number of sectors in the specified block of the tape. The routine TP0FRMFL must be called to fill the buffer before this routine can be executed. The calling parameters "Sector Count" and "Format Gap" must be the same as used by TP0FRMFL. As in TP0FRMFL, the buffer length must be longer than the block length. (Twice the block length is sufficient, as it depends on the gap length.)

TP0ID

Read ID

Calling Parameters

Drive	- Drive number.

Return Parameters

Cylinder	- Cylinder number.
Sector	- Sector number.

Description

Reads the current ID (which is the sector header) of the next sector to pass under the head. This ID contains the block (or floppy track) number and the sector number.

5.0 Considerations for the Western Digital WD179x and WD279x Series

This section discusses the special considerations needed when using floppy disk controllers based upon the Western Digital series of floppy disk controller chips. The following is a list of the Western Digital commands used in the TP0 routines and the parameters required for each command:

(Parameter options vary between Western Digital Models 1791, 1792, 1793, 1794 and Models 1795 and 1797. The models handle the Side Select options differently.)

Command	Parameters Required
Restore and Seek	1. Load head at beginning of operation 2. Set for no verify 3. Set for 3 millisecond step rate
Read and Write sector	1. Number of sectors to transfer 2. Select side 0, or set sector length 3. Set head delay to 0 4. Set side select update to zero, or disable side compare 5. If write, set data address mark
Write Track	1. Set head delay to zero 2. Set side select update to zero
Force Interrupt	1. Set as needed
Read ID	1. Set head delay to 0 2. Set side select update to zero

---- 5.1 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 connected and working. The controller chip's internal track register is set to zero. This command is used in the TP0ONL and TP0RECAL low-level device drivers.

Seek

The Seek command is used to issue the command pulse train that causes the tape drive to execute one of its commands. This command is issued by the TP0COMM low-level device driver. Since the internal track register is accessible to the software, the way to issue a command is to zero out the track register, load the value (number of pulses) associated with the desired command into the register, then execute the seek command.

Read Sector/Write Sector

These two commands read or write one or multiple sectors in a single tape block. They are used in low-level device routines TP0REDI and TP0WRTI, respectively. Software used in this area will be similar to floppy data transfer software.

The Western Digital chip sets allow single or multiple sector data transfers. When reading or writing more than one sector, your device driver software must keep track of the number of sectors that are read or written. This is because in multi-sector mode the controller continues to read (or write) sectors until the program issues a Force Interrupt instruction or until the Western Digital chip times out. A chip timeout occurs when the desired sector is not found within five tape blocks, which is equivalent to five revolutions of the disk. If this timeout occurs, it will be necessary to reposition the tape since it is now five blocks downstream.

In some systems, processor timing considerations dictate that sector counting cannot be done by the software. When this is the case, we recommend using single sector read/write operations. To take full advantage of this technique, set up a large enough buffer to accomodate all of the data for the entire operation for transfers to or from the tape. Then issue successive single sector data transfer commands to the controller between sectors. The controller chip will accept a read or write sector command between the end of the previous sector and the beginning of the next sector. The read or write sector command is sent during the write splice gap area since the tape blocks are formatted with no sector interleave.

Write Track

Used in low-level device driver, TP0FRMT, to format a single tape block. This implementation of this command is similar to that used for a floppy disk.

Force Interrupt

This command is used to force hardware interrupts to the processor. It is used in TP0REDI and TP0WRTI to terminate multiple sector data transfer operations. It is also used to reinitialize the controller chip following an error or in hang-up situations.

Read ID

Used in low-level device driver TP0ID to find next ID on tape for tape positioning.

---- 5.2 Tape Positioning with the Western Digital Controller

Bit 1 of the Type 1 status register always reflects the status of the Index line. To position the tape, monitor this bit and count the passing tape blocks.

6.0 Considerations for the NEC 765 Controller

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

Command	Parameters Required
Recalibrate	1. Drive unit select
Sense Interrupt Status	1. Drive unit select
	2. Cylinder Number
Specify	1. Step Rate
	2. Head Load Time
	3. Head Unload Time
	4. DMA Mode Yes/No
Sense Drive Status	1. Drive Unit Select
	2. Head Select Zero
Seek	1. Drive Unit Select
	2. Head Select Zero
	3. Cylinder Number
Read Data/Write Data	1. Drive Unit Select
	2. Head Select Zero
	3. Cylinder Number
	4. Head Number Zero
	5. Sector Number
	6. Sector Length
	7. Last Sector Operation
	8. VCO Sync Time
	9. DTL - User-defined Data Length
Read ID	1. Drive Unit Select
	2. Head Select Zero
	3. Select MFM Mode
Format a Track	1. Drive Unit Select
	2. Head Select Zero
	3. Sector Length
	4. Number of Sectors per Track
	5. Gap Length
	6. Format Data Constant

---- 6.1 Using the NEC 765 Commands

The NEC 765 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 track register is set to zero, which is a useful way to start over when the software gets confused as to which track it is on. This command is used in TP0ONL and TP0RECAL low-level device drivers.

Sense Interrupt Status

The controller 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 command operations must be reset by the sense interrupt command. Sense interrupt status is generally used after a seek or a recalibrate command to return completion status information and the present cylinder (floppy track) number. This information is useful to verify that the proper command was sent to the tape drive.

Irwin has observed multiple NEC 765 interrupts following a Recalibrate command if non-contiguous drive select addresses are used. For example, having two drives addressed 0 and 2 with no drives existing for drive selects 1 and 3 (either floppy or tape drive) causes this condition. When this happens, interrupts get nested, and the software is unable to sort them out. Therefore, it is a good idea 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 TP0INI and TP0TRM 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 tape drive. This command is used throughout the low-level device drivers.

Seek

The Seek command is used to issue the command pulse train that causes the tape drive to execute one of its commands. This command is issued by TP0COMM low-level device driver.

Since the controller chip's internal track register is not accessible to the software, the application program must maintain an external register that indicates which track 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. This register is kept to avoid confusion and so that the program can calculate 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 one-half 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, then seek to the present track number - n pulses. This philosophy will avoid the program issuing invalid track seeks to the NEC 765.

Read Data/Write Data

These two commands read or write the sectors in a tape block. They are used in low-level device drivers TP0REDI and TP0WRTI, respectively. Software used in this area will be similar to that used to transfer data to and from floppies.

Read ID

Similar to Read and Write Data. This command is used in TP0ID to find the next ID on tape, and in TP0NDX to count two Index pulses. (See Section 6.2 - "Tape Positioning with the NEC 765.")

Format a Track

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

---- 6.2 Tape Positioning with the NEC 765

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

The first method 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 head.

The second method involves moving the tape at high speed (70ips) and issuing a Read ID command. After two revolutions of the disk, or in this case, after two tape blocks have passed beneath the tape head, the NEC 765 will timeout since it cannot read sector I.D.'s while moving at 70ips. In fact, at 70ips, data reads from the tape drive are inhibited, but Index pulses are not.

The tape positioning routine can equate the timeout to the passage of two tape blocks. Therefore, by reissuing the Read ID command, the required number of blocks can be passed and counted to position the desired block beneath the read/write head.

---- 6.3 Programming Problems with the NEC 765

There are two programming problem areas with the NEC controller chip.

The first problem is the number of hardware interrupts sent during a Recalibrate command. This was discussed previously in the description of the Recalibrate command.

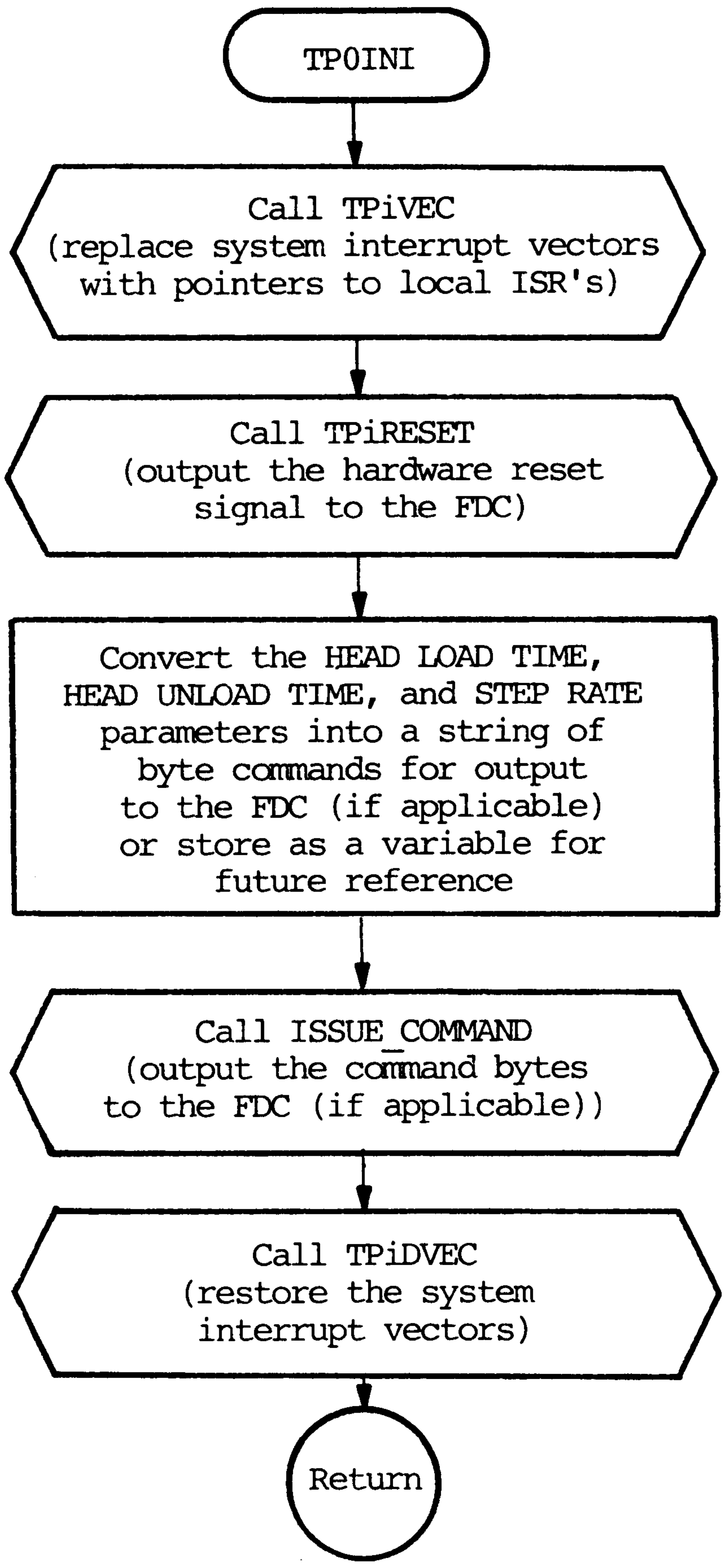
The second problem has to do with keeping track of the contents of the controller's internal track register, since the register is not accessible to either the hardware or software. Problems arise when the controller chip receives Recalibrate or Reset commands from unknown sources (like IBM PC-DOS ROM BIOS Interrupt 13) and the internal track register gets out of synchronization with the software track register. When this occurs, a Recalibrate or Reset command must be issued to re-synchronize the two track registers.

The above problems will cause the drive to execute commands other than those issued. Because the internal track register is different than the software track register, an unexpected number of pulses will be sent to the drive. Since the problems will not be noticed until a command is issued and the unexpected results occur, the only solution to the above problems is careful programming.

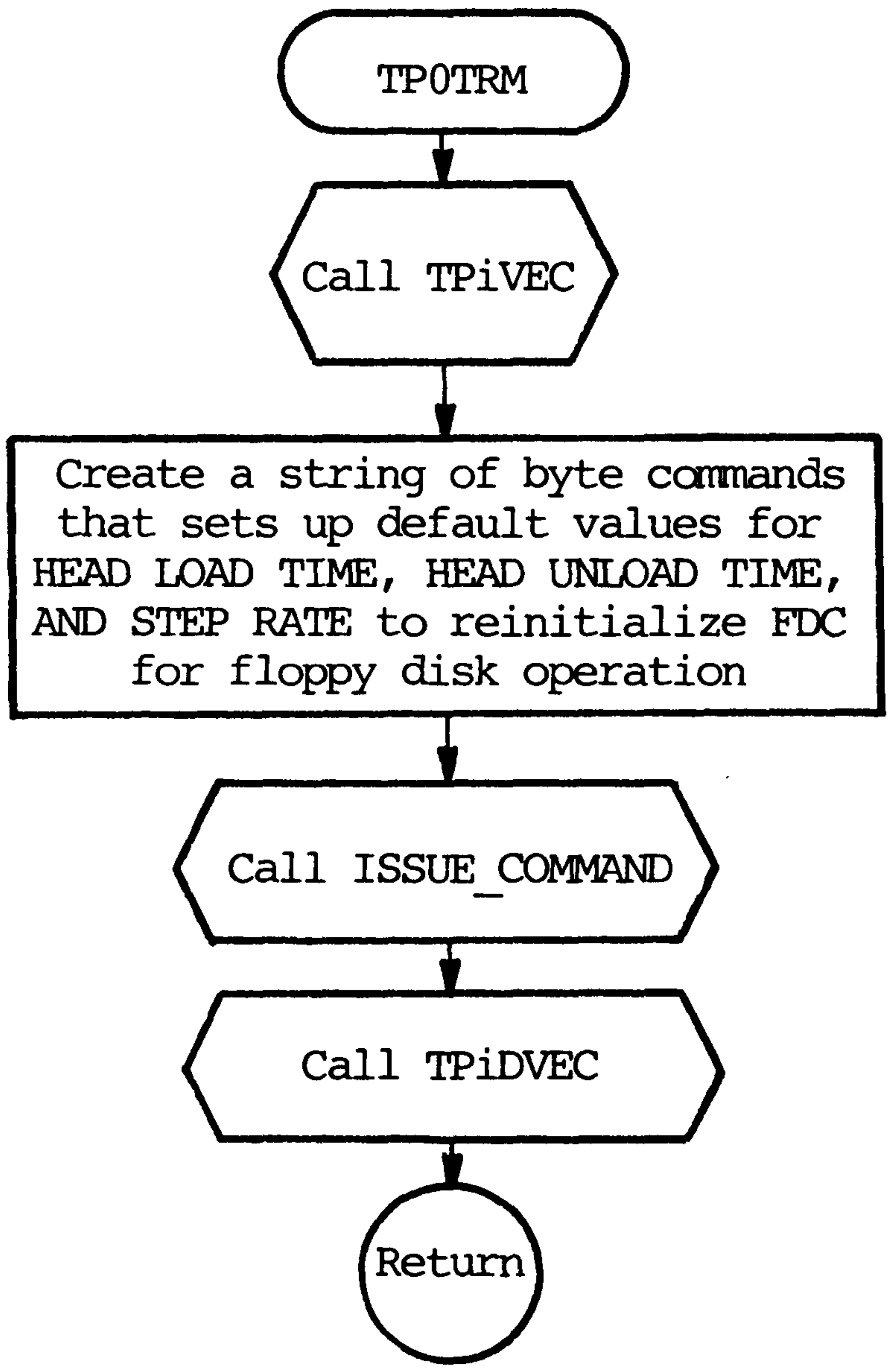
7.0 Low-Level Device Driver Flowcharts

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

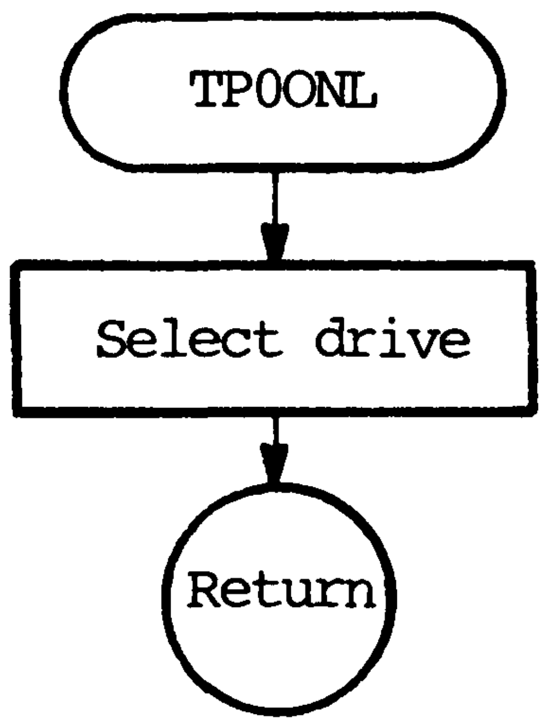
TPOINI(load time, unload time, step rate, io gap)


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

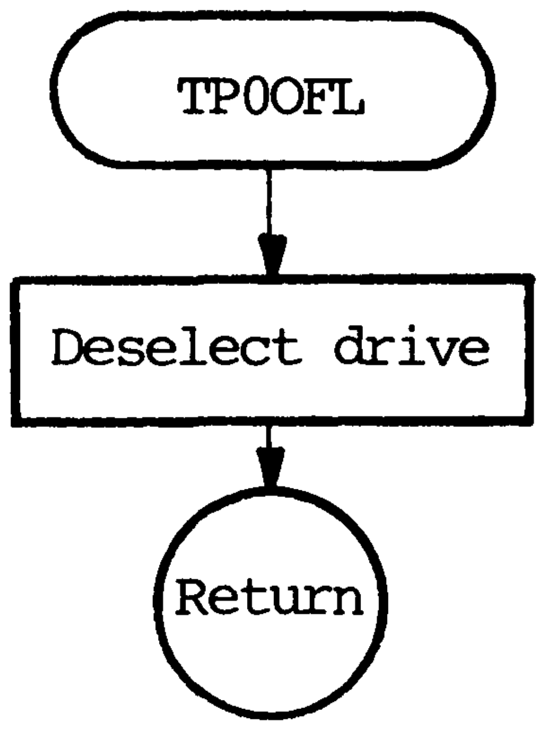
TP0TRM()


TP0TRM
Call TPiVEC
Create a string of byte commands that sets up default values for HEAD LOAD TIME, HEAD UNLOAD TIME, AND STEP RATE to reinitialize FDC for floppy disk operation
Call ISSUE_COMMAND
Call TPiDVEC
Return

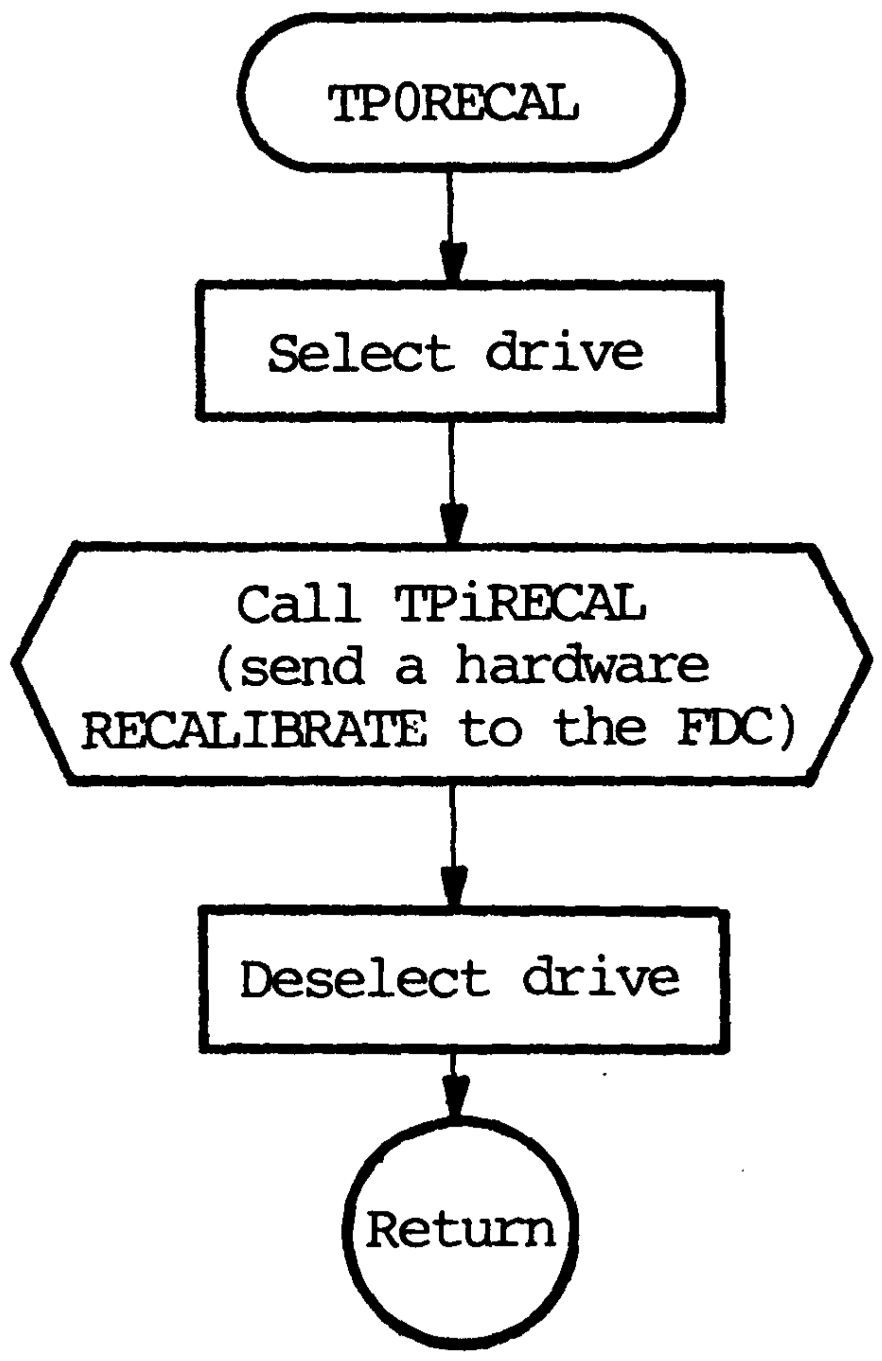
TPOONL(drive)



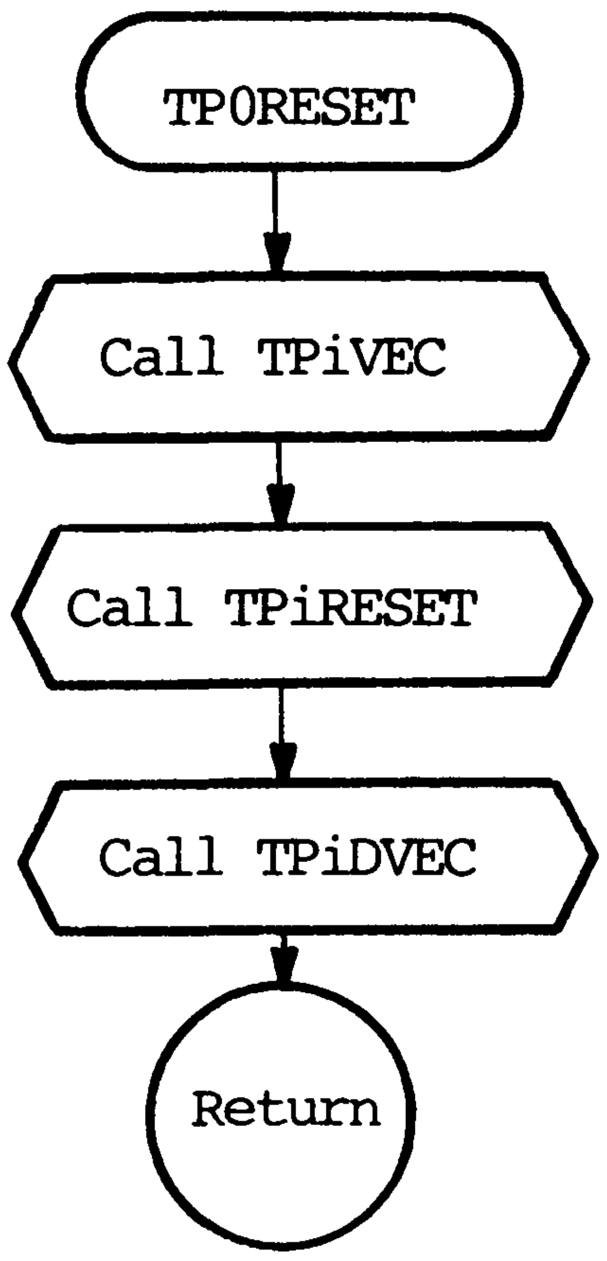
TPOOFL(drive)



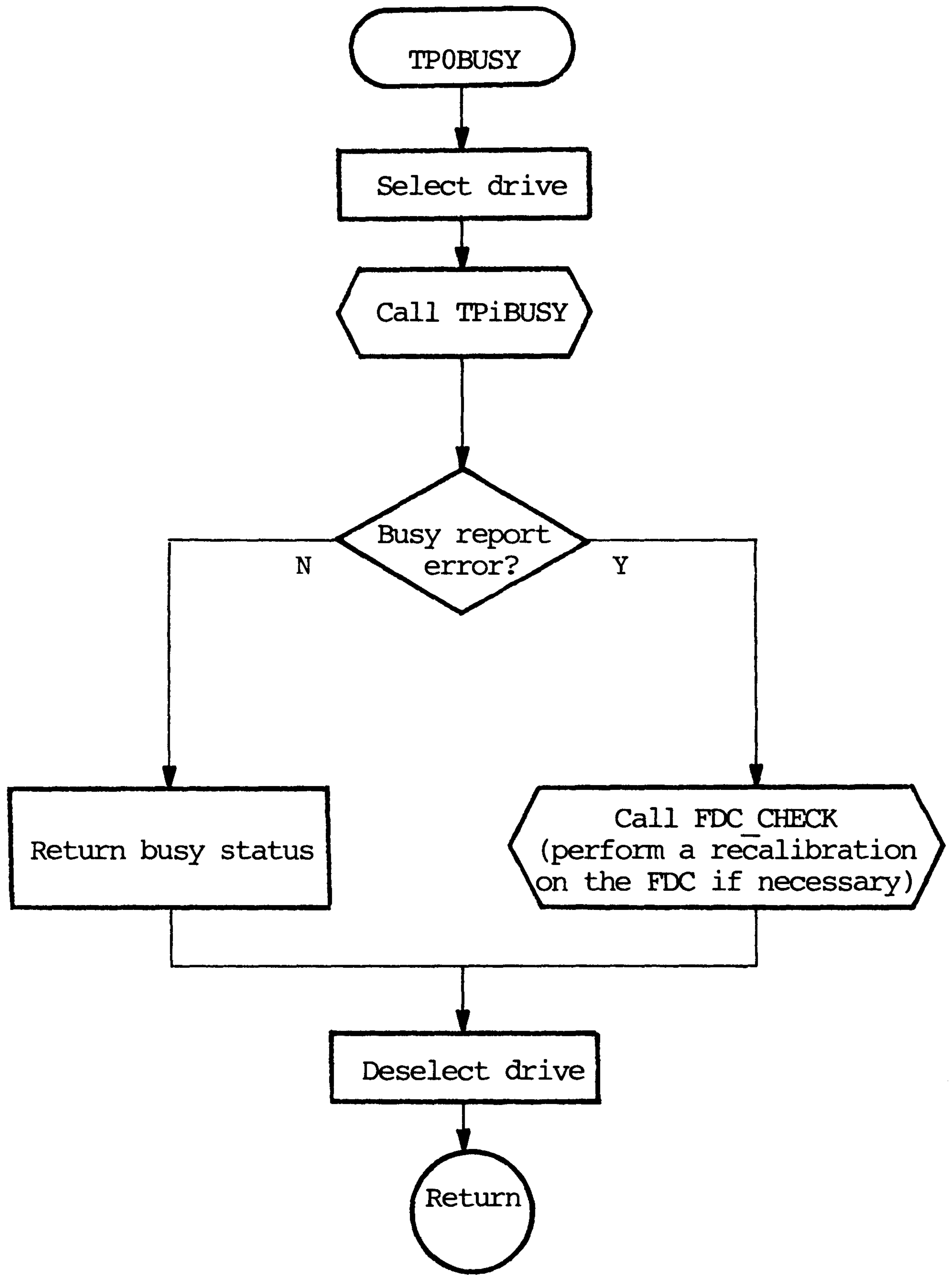
TP0RECAL(drive)



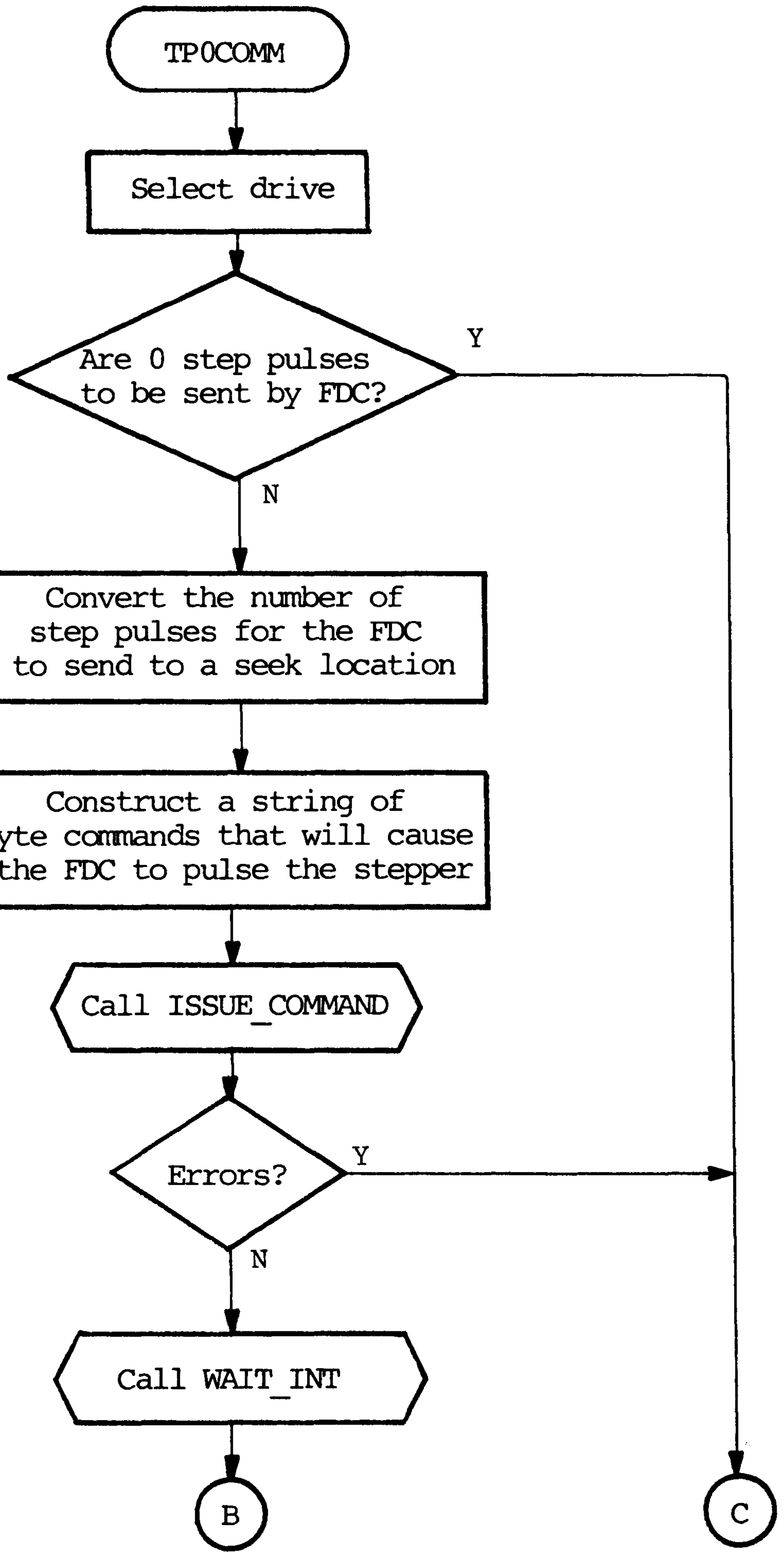
TP0RESET()



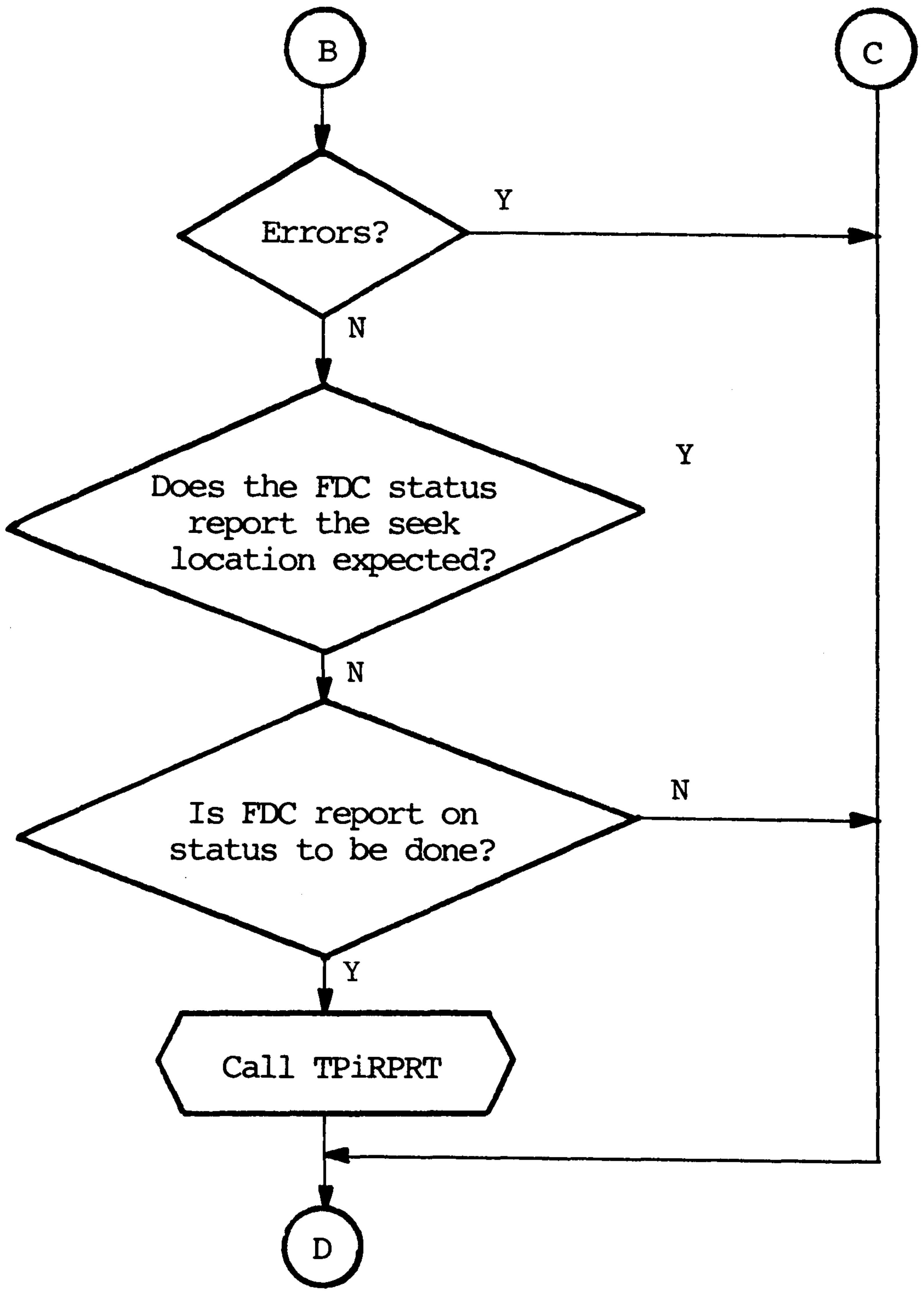
TP0BUSY(drive)

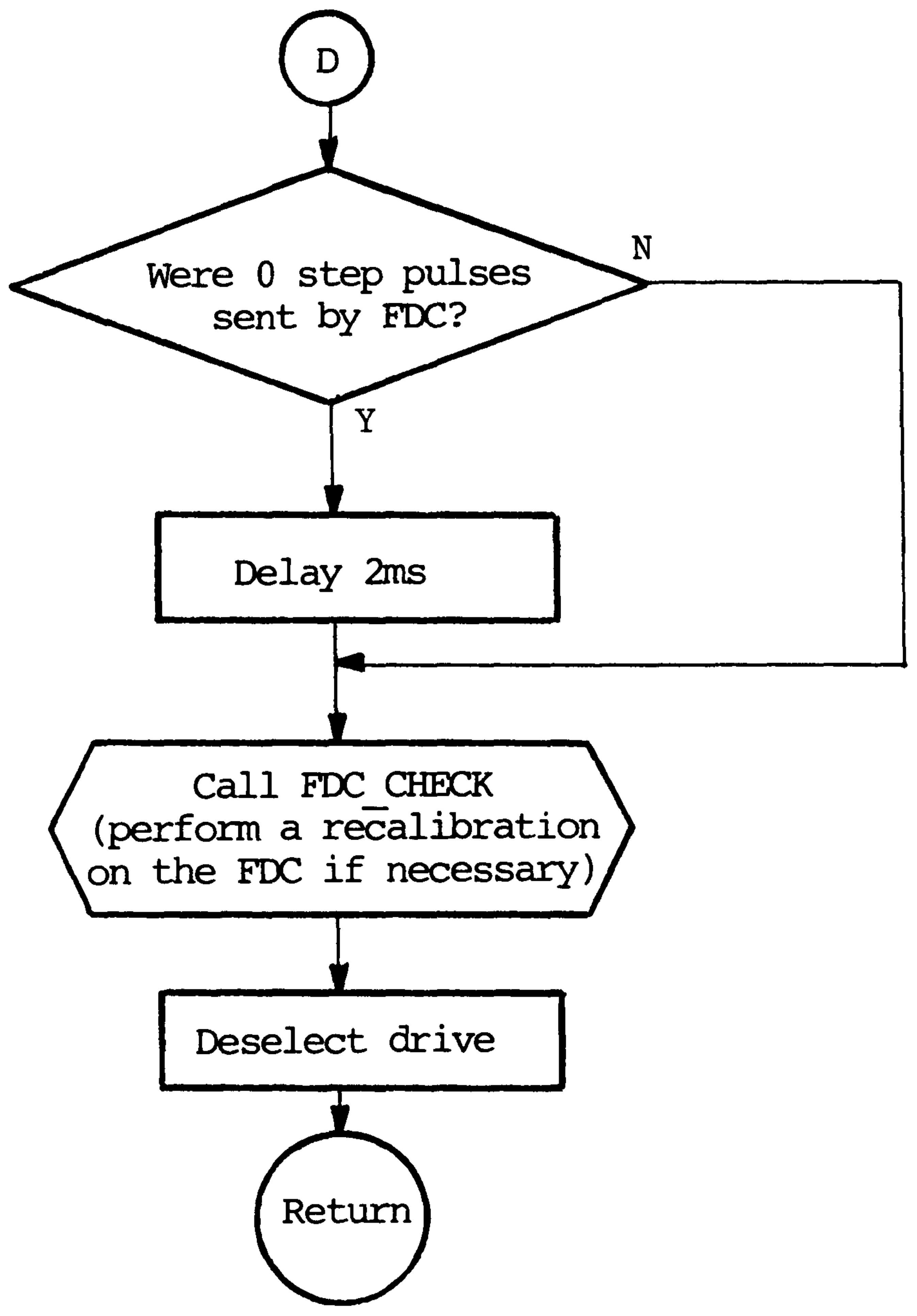


TPOCOMM(drive, steps, wait/status)

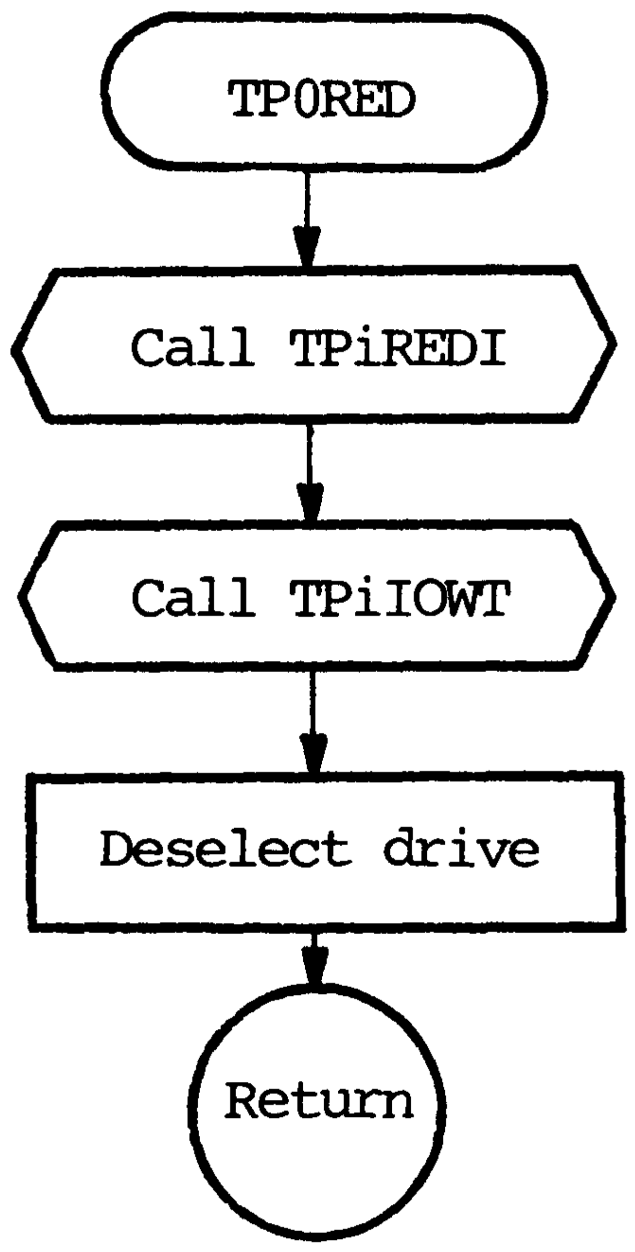


TPOCOMM
Select drive
Are 0 step pulses to be sent by FDC?
Y
N
Convert the number of step pulses for the FDC to send to a seek location
Construct a string of byte commands that will cause the FDC to pulse the stepper
Call ISSUE_COMMAND
Errors?
Y
N
Call WAIT_INT
B
C

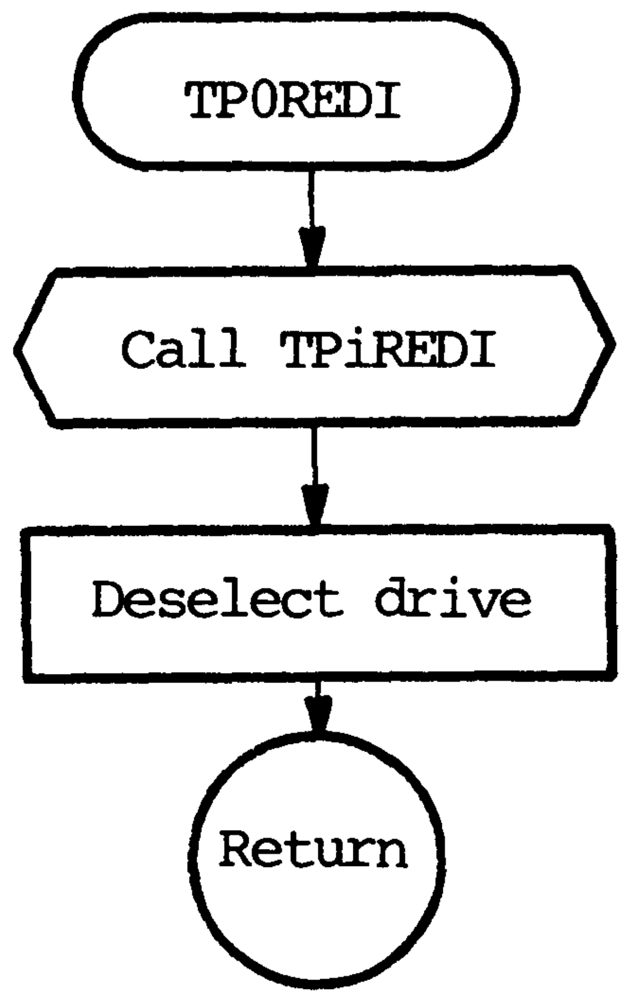

B
C
Errors?
Y
N
Does the FDC status report the seek location expected?
Y
N
Is FDC report on status to be done?
N
Y
Call TPiRPRT
D


D
Were 0 step pulses sent by FDC?
N
Y
Delay 2ms
Call FDC_CHECK (perform a recalibration on the FDC if necessary)
Deselect drive
Return

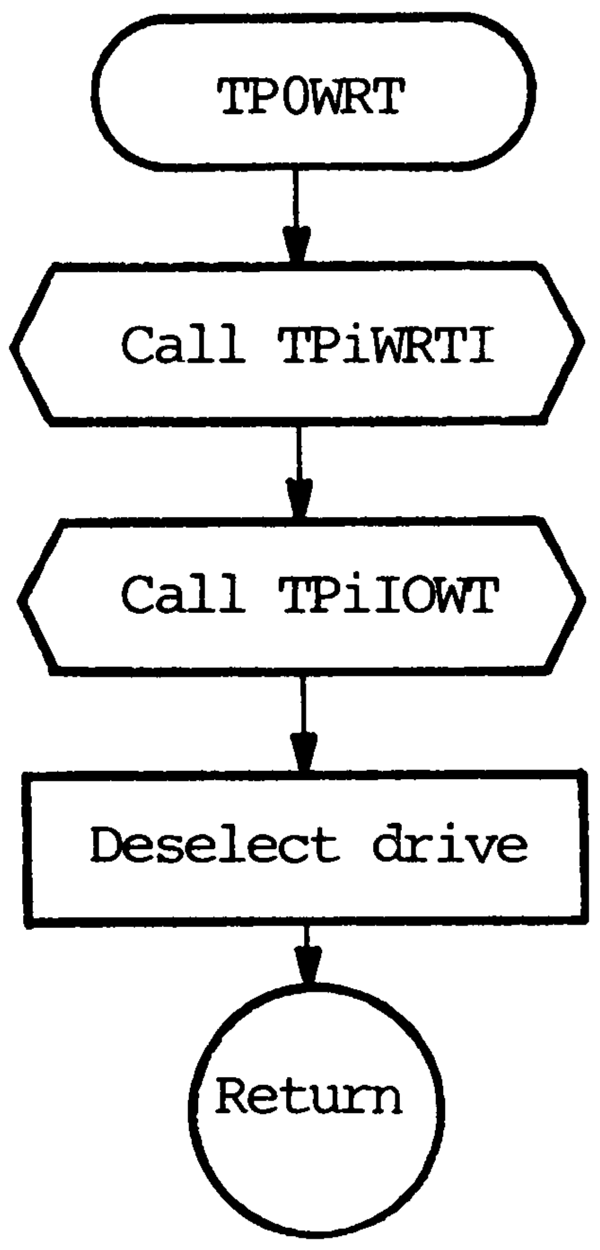
TPORED(drive, buffer address, cylinder, sector, sector count)



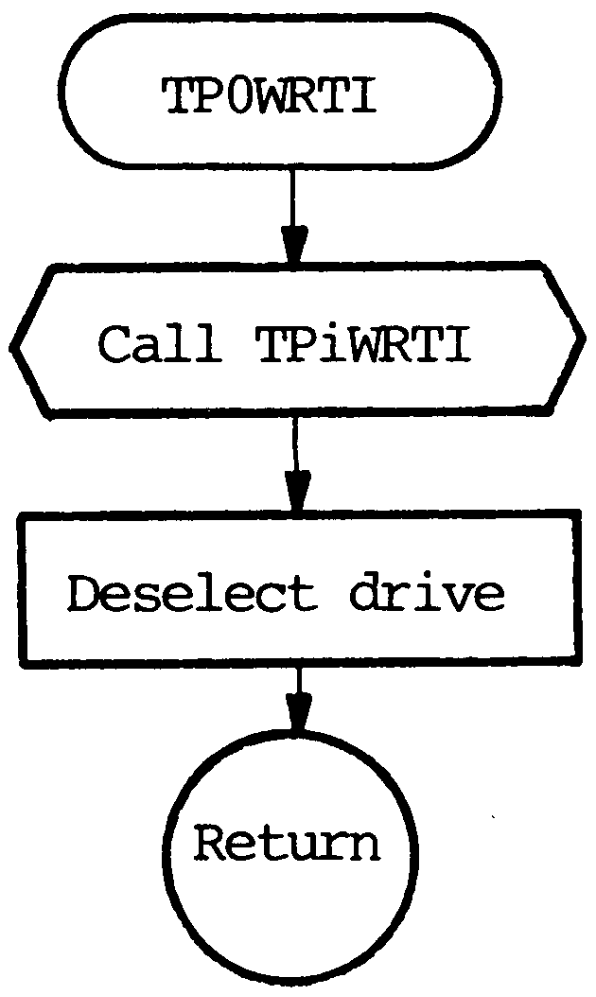
TPOREDI(drive, buffer address, cylinder, sector, sector count)



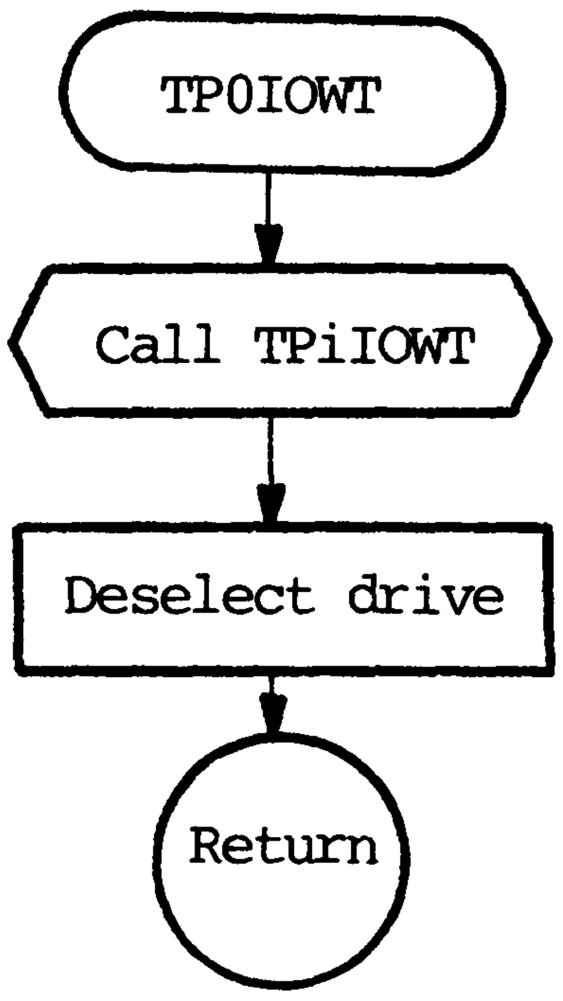
TP0WRT(drive, buffer address, cylinder, sector, sector count)



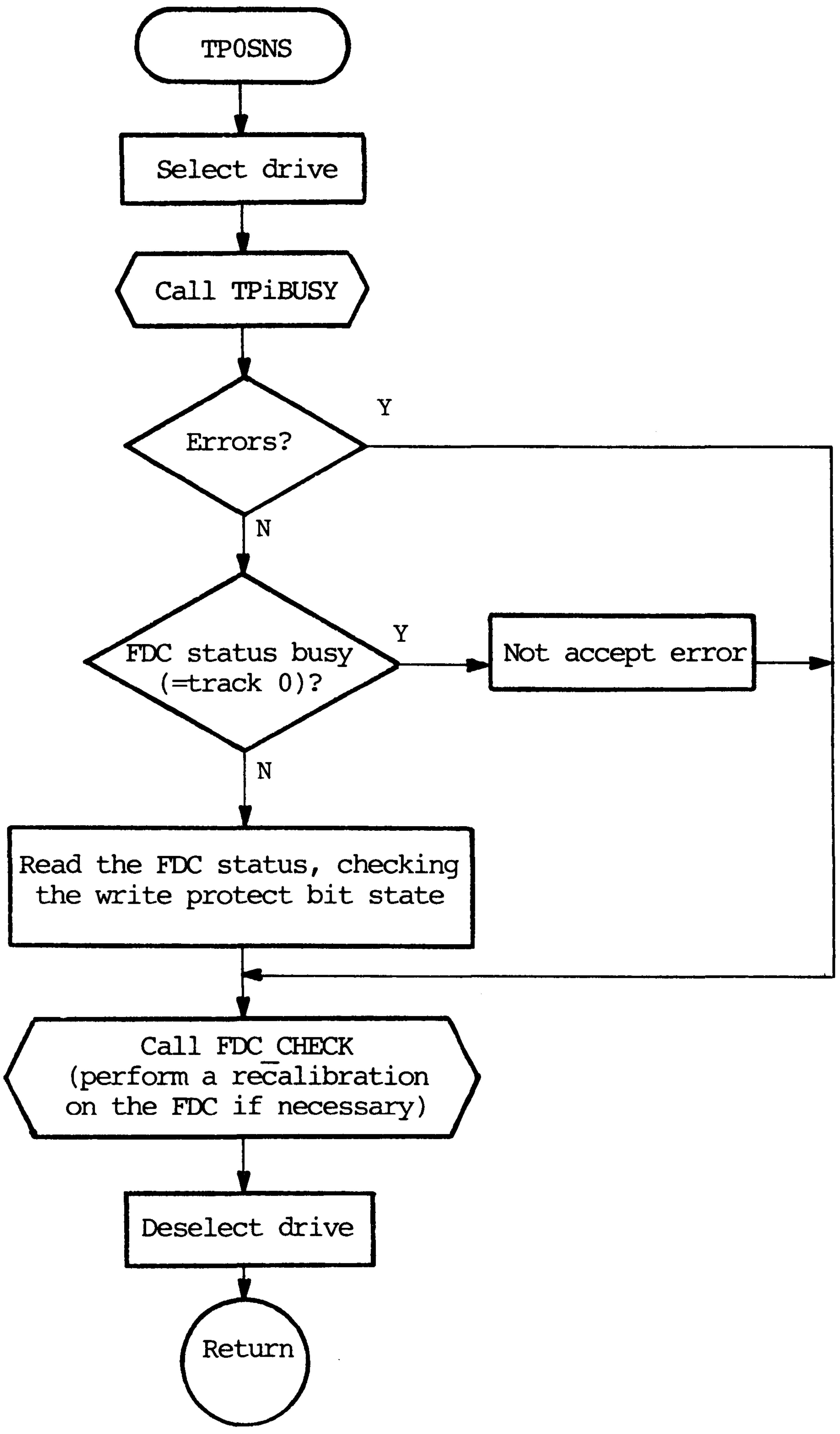
TP0WRTI(drive, buffer address, cylinder, sector, sector count)



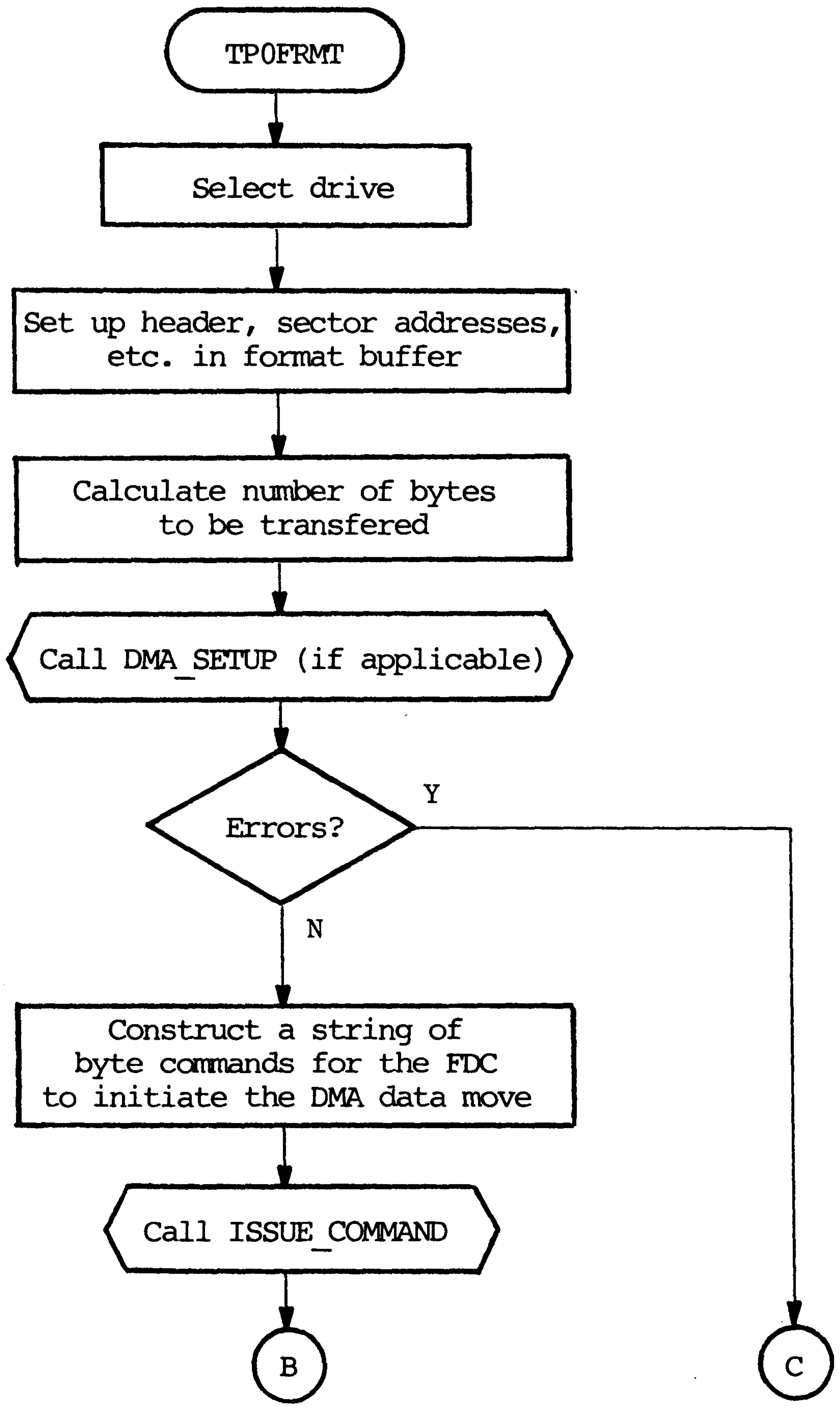
TP0IOWT(drive)

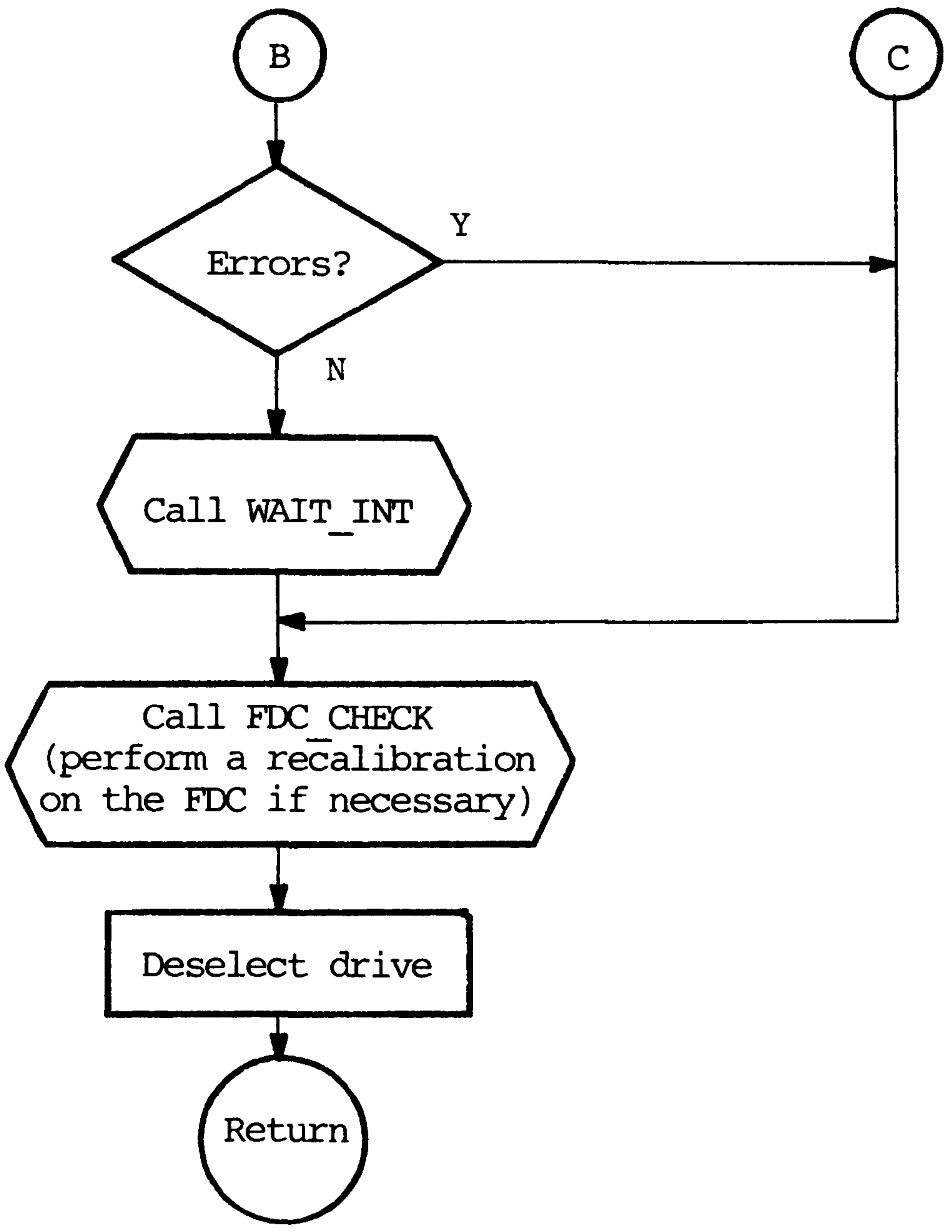


TP0SNS(drive, protected)

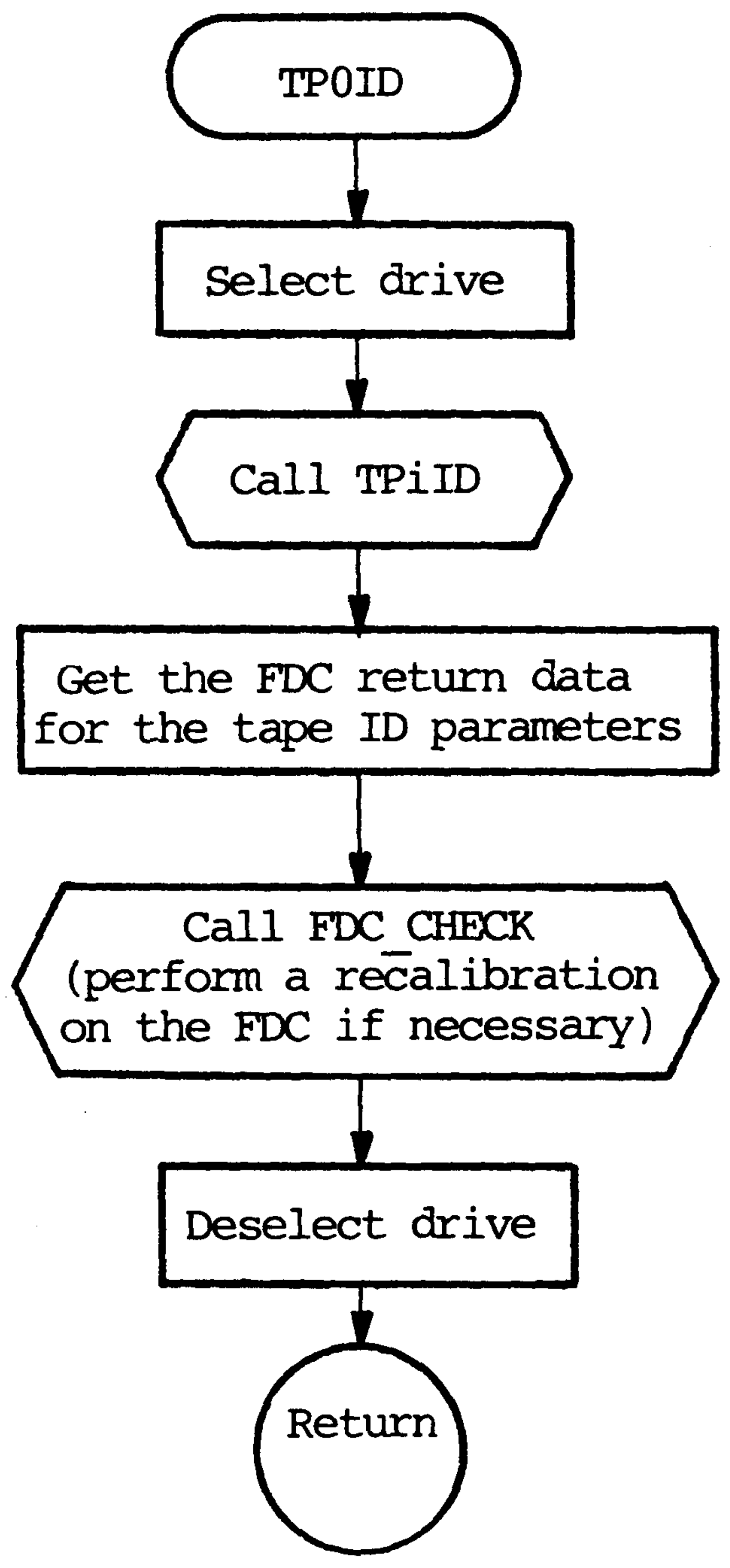

TP0SNS
Select drive
Call TPiBUSY
Errors?
Y
N
FDC status busy (=track 0)?
Y
Not accept error
N
Read the FDC status, checking the write protect bit state
Call FDC_CHECK (perform a recalibration on the FDC if necessary)
Deselect drive
Return

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


TPOFRMT
Select drive
Set up header, sector addresses, etc. in format buffer
Calculate number of bytes to be transfered
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 FDC_CHECK (perform a recalibration on the FDC if necessary)
Deselect drive
Return

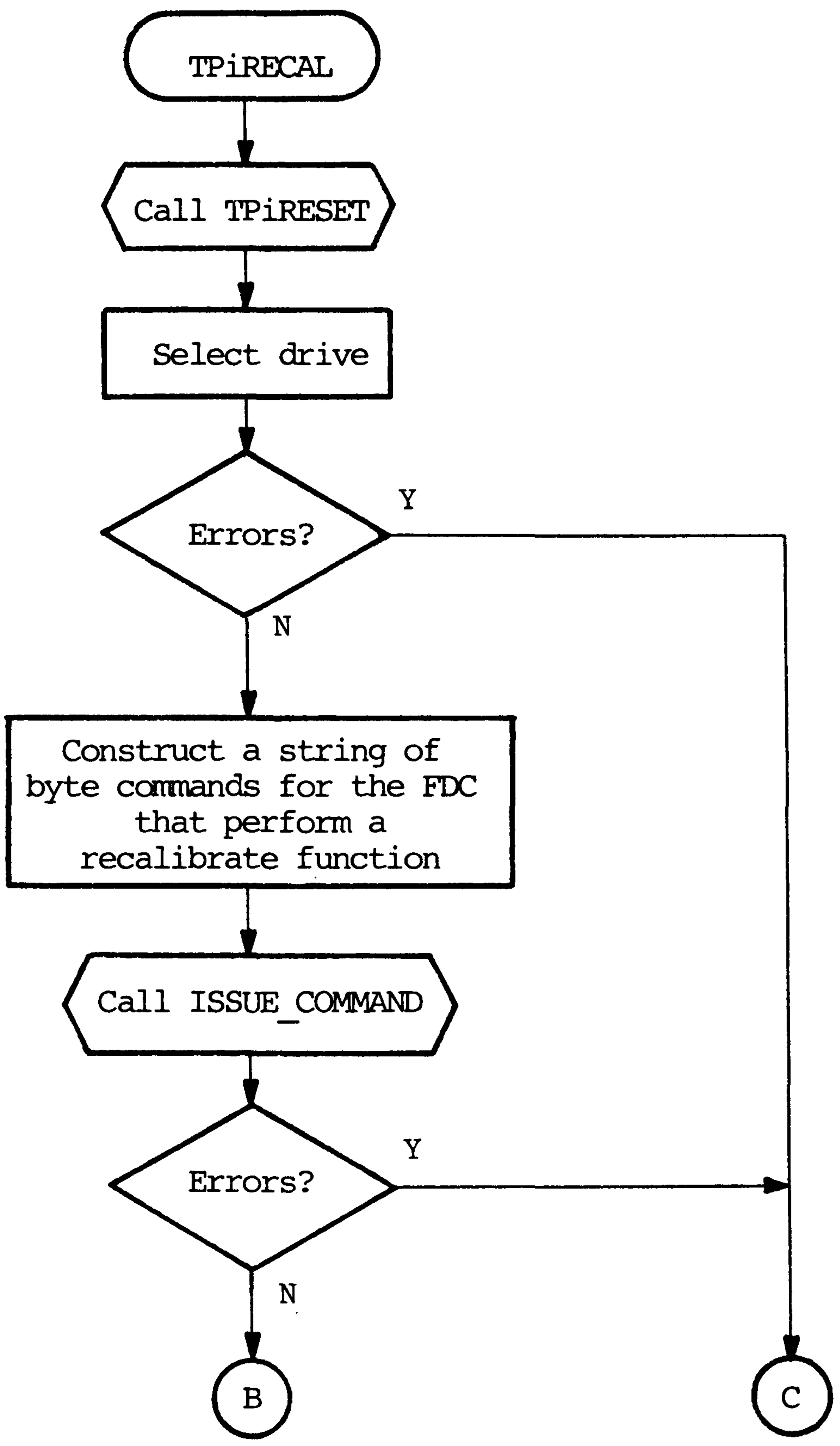
TPOID(drive, cylinder, sector)

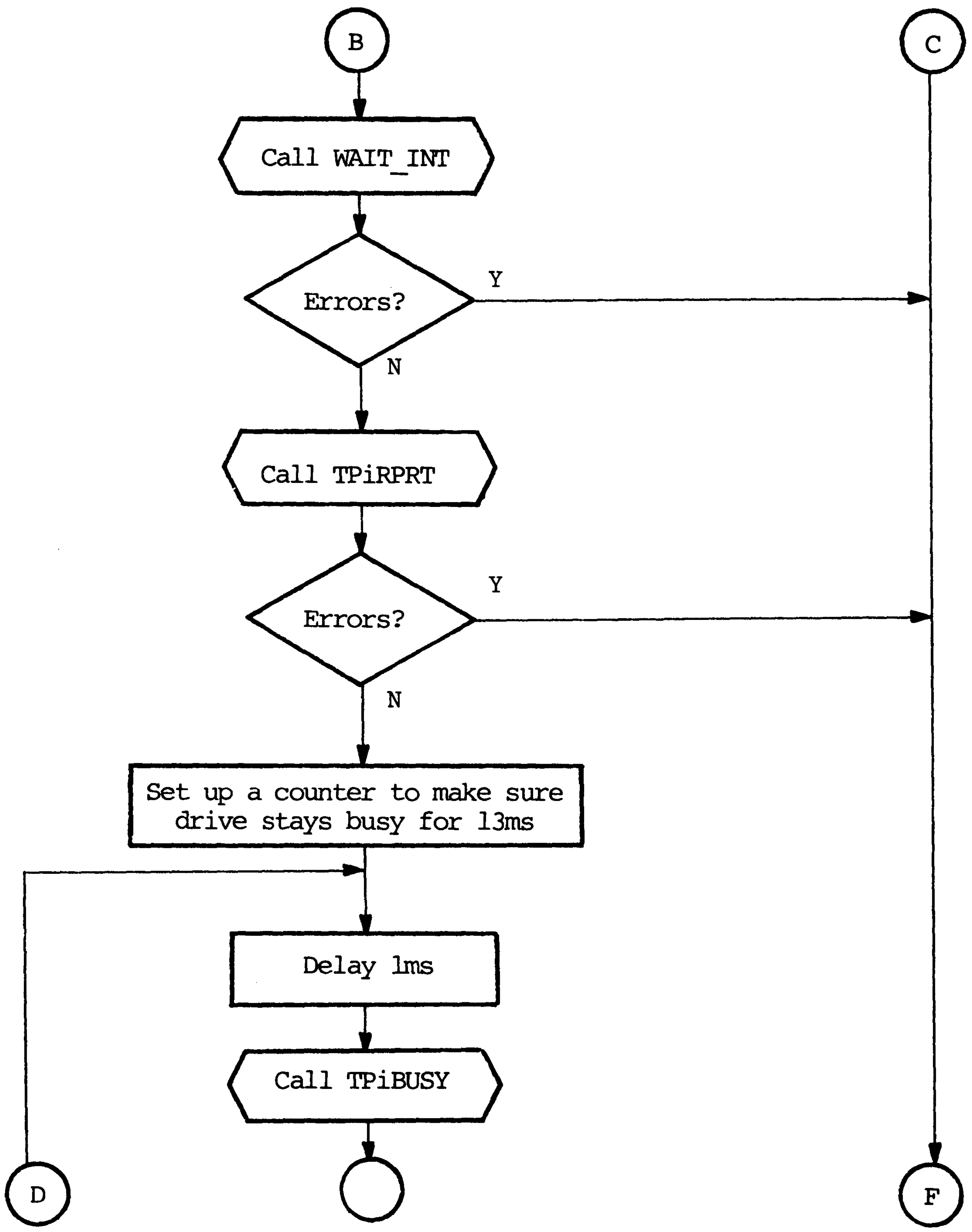

TPOID
Select drive
Call TPiID
Get the FDC return data for the tape ID parameters
Call FDC_CHECK (perform a recalibration on the FDC if necessary)
Deselect drive
Return

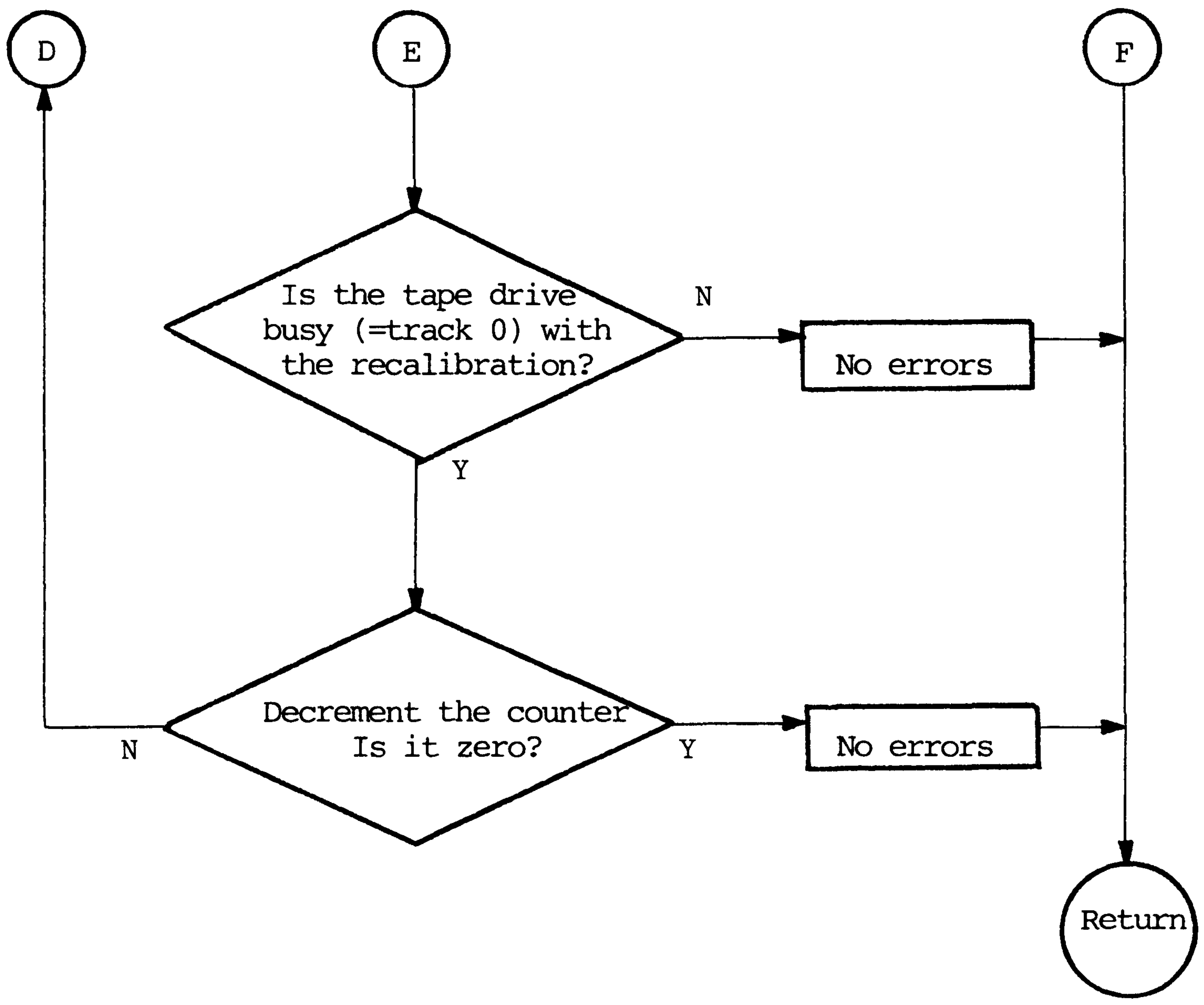
TPiRECAL

Sends recalibrate command to drive and makes sure drive goes busy for 13 ms.

Called by: TPOONL, TPORECAL, FDC_CHECK.


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

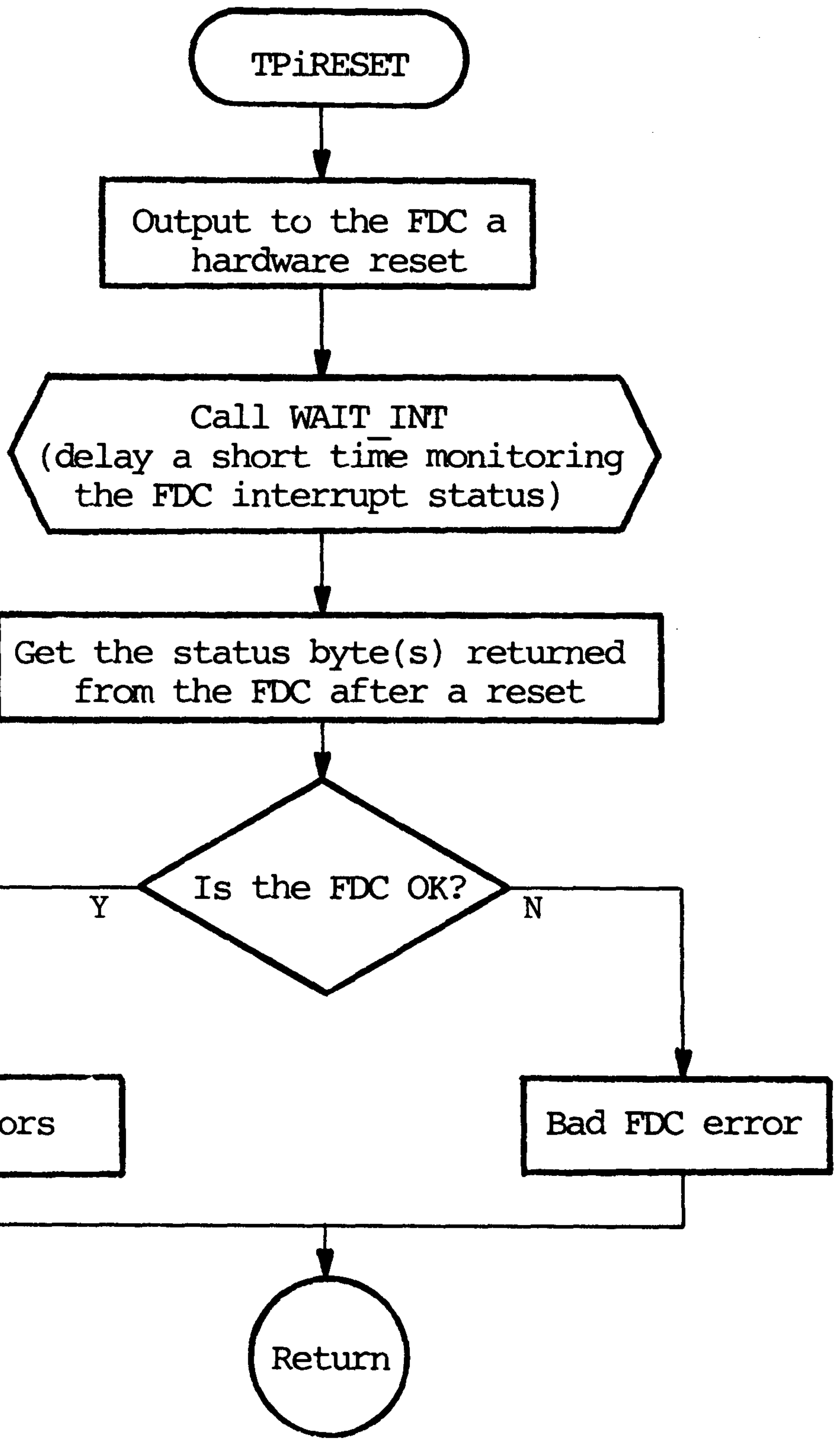

B
C
Call WAIT_INT
Errors?
Y
N
Call TPiRPRT
Errors?
Y
N
Set up a counter to make sure drive stays busy for 13ms
Delay 1ms
Call TPiBUSY
D
F


D
E
F
Is the tape drive busy (=track 0) with the recalibration?
N
No errors
Y
Decrement the counter Is it zero?
N
Y
No errors
Return

TPiRESET

Executes a hardware controller reset (really not applicable to the Western Digital 179x chip).

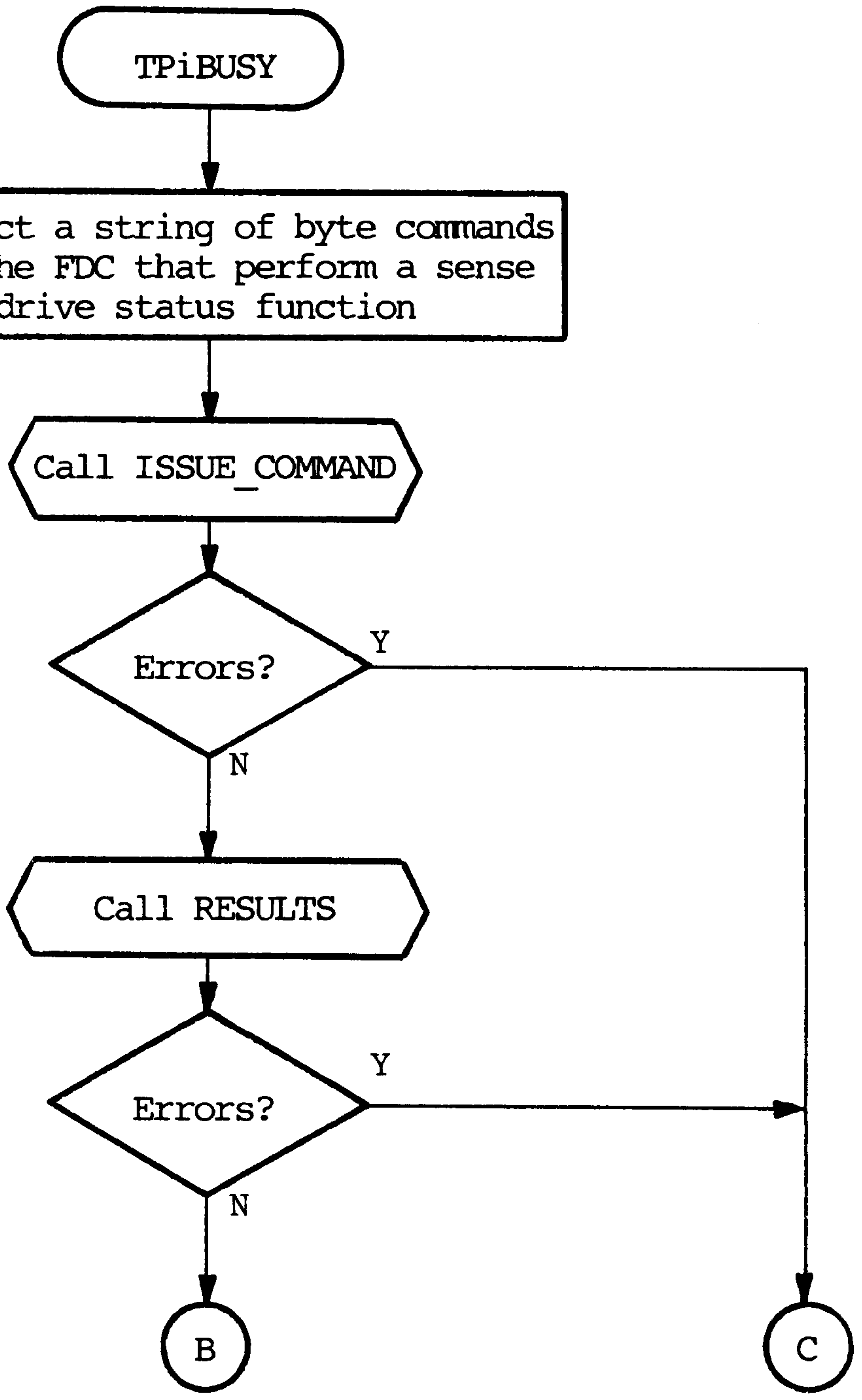
Called by: TP0INI, TP0RESET, TPiRECAL.

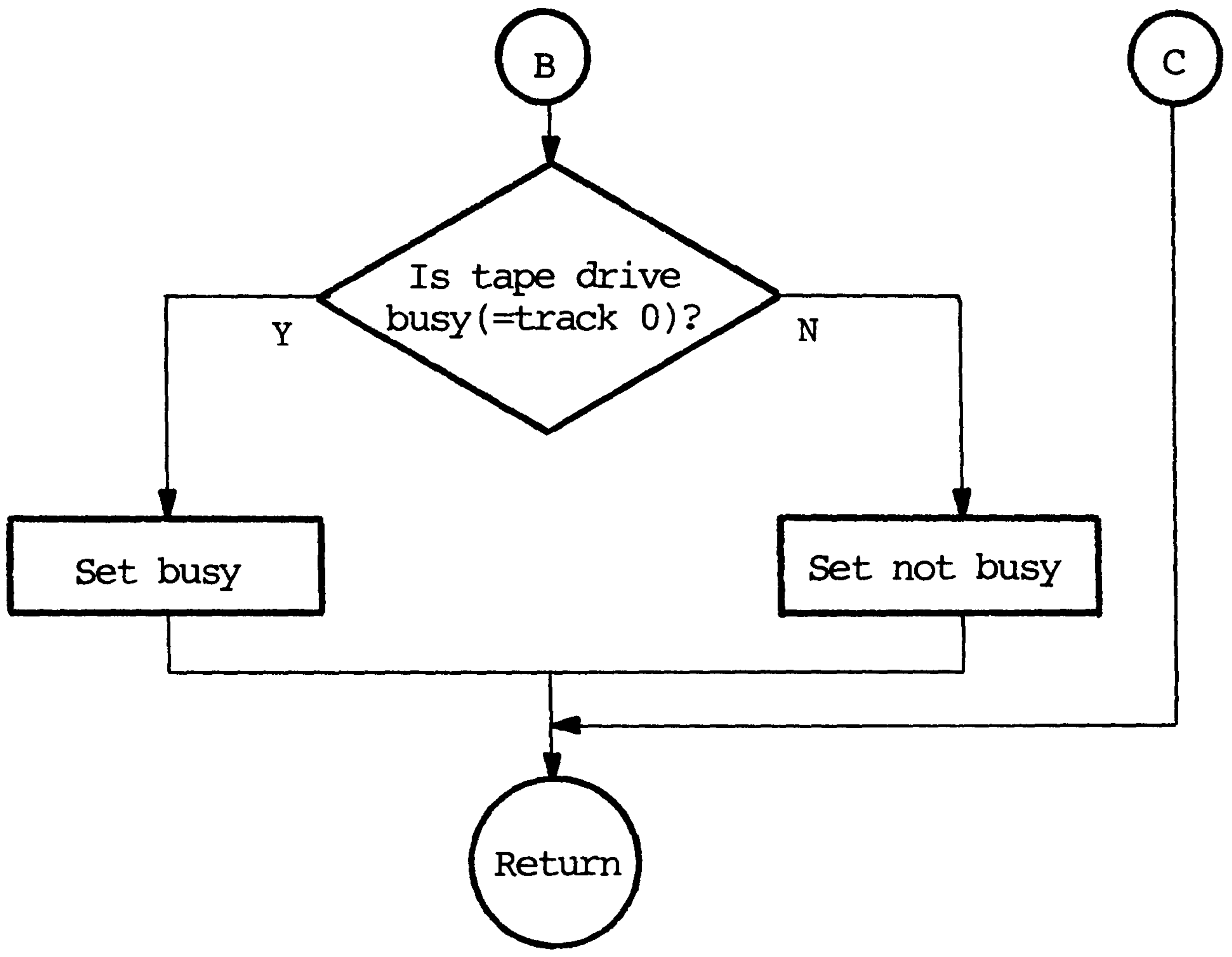

TPiRESET
Output to the FDC a hardware reset
Call WAIT_INT (delay a short time monitoring the FDC interrupt status)
Get the status byte(s) returned from the FDC after a reset
Is the FDC OK?
Y
N
No errors
Bad FDC error
Return

TPiBUSY

Queries controller and returns controller busy status if there is no error condition.

Called by: TP0BUSY, TP0SNS, TPiRPRT, TPiRECAL.

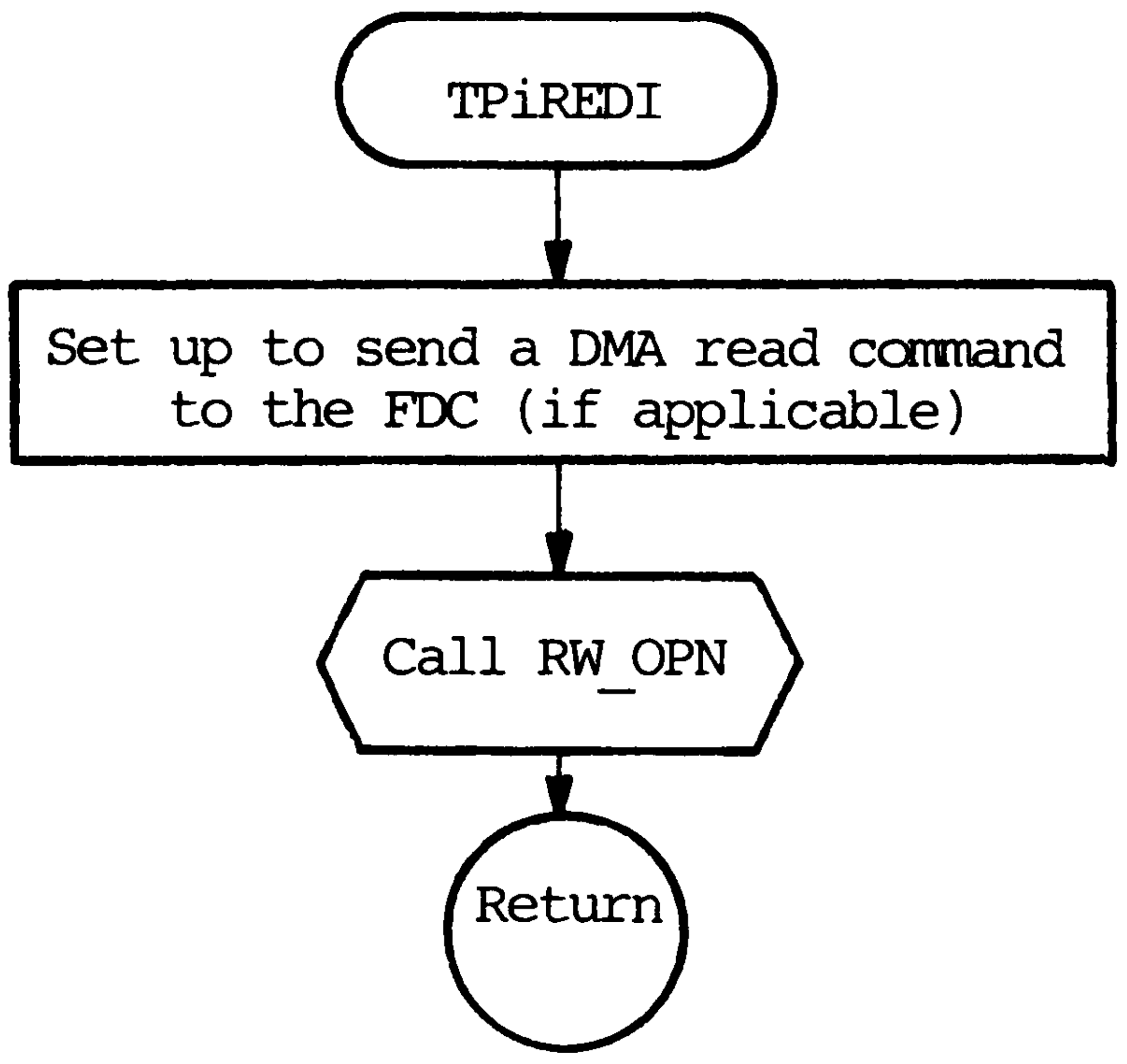

TPiBUSY
Construct a string of byte commands for the FDC that perform a sense drive status function
Call ISSUE_COMMAND
Errors?
Y
N
Call RESULTS
Errors?
Y
N
B
C


B
C
Is tape drive busy(=track 0)?
Y
N
Set busy
Set not busy
Return

TPiREDI

Performs a read operation.

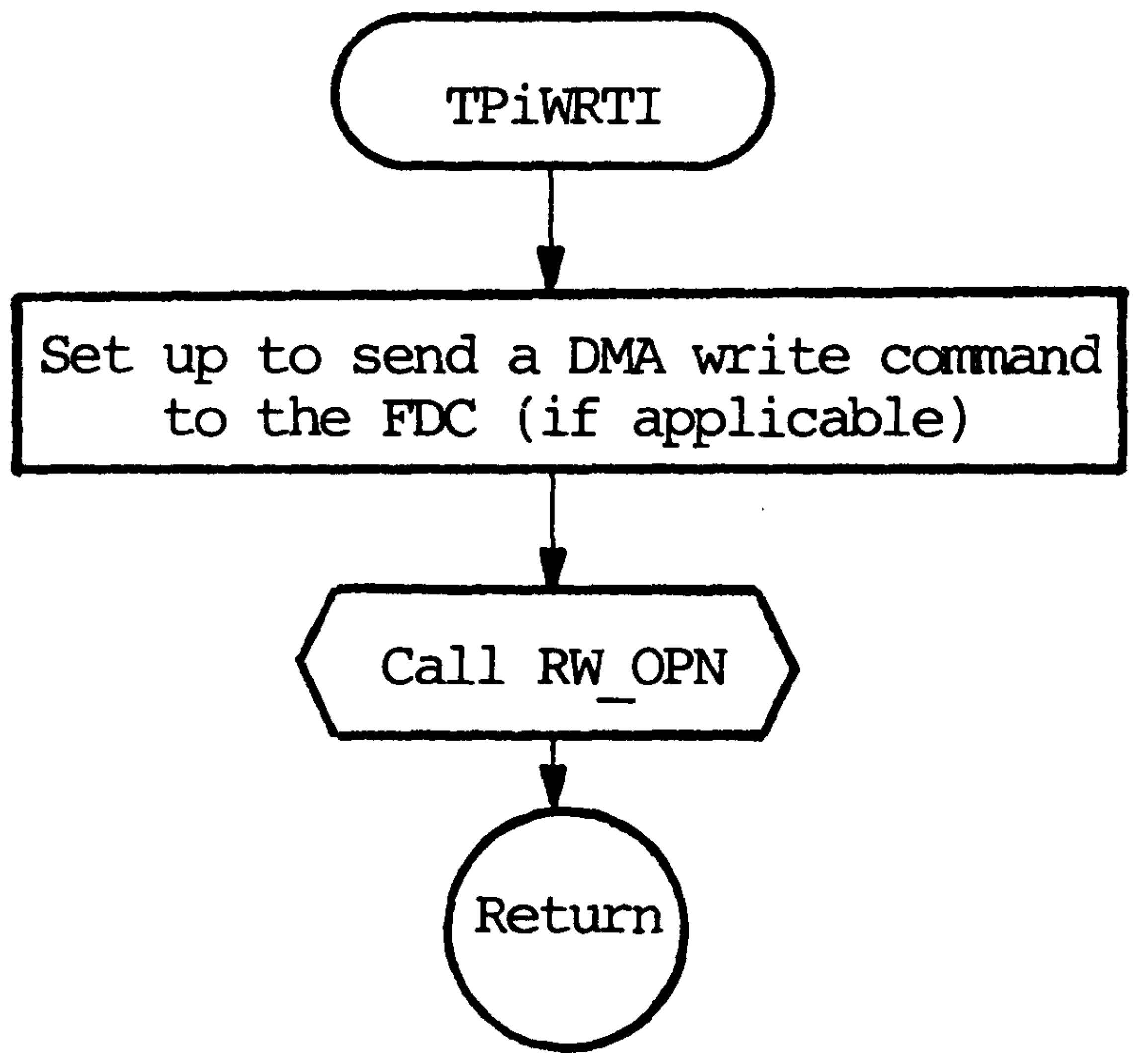
Called by: TPORED, TPOREDI.


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

TPiWRTI

Performs a write operation.

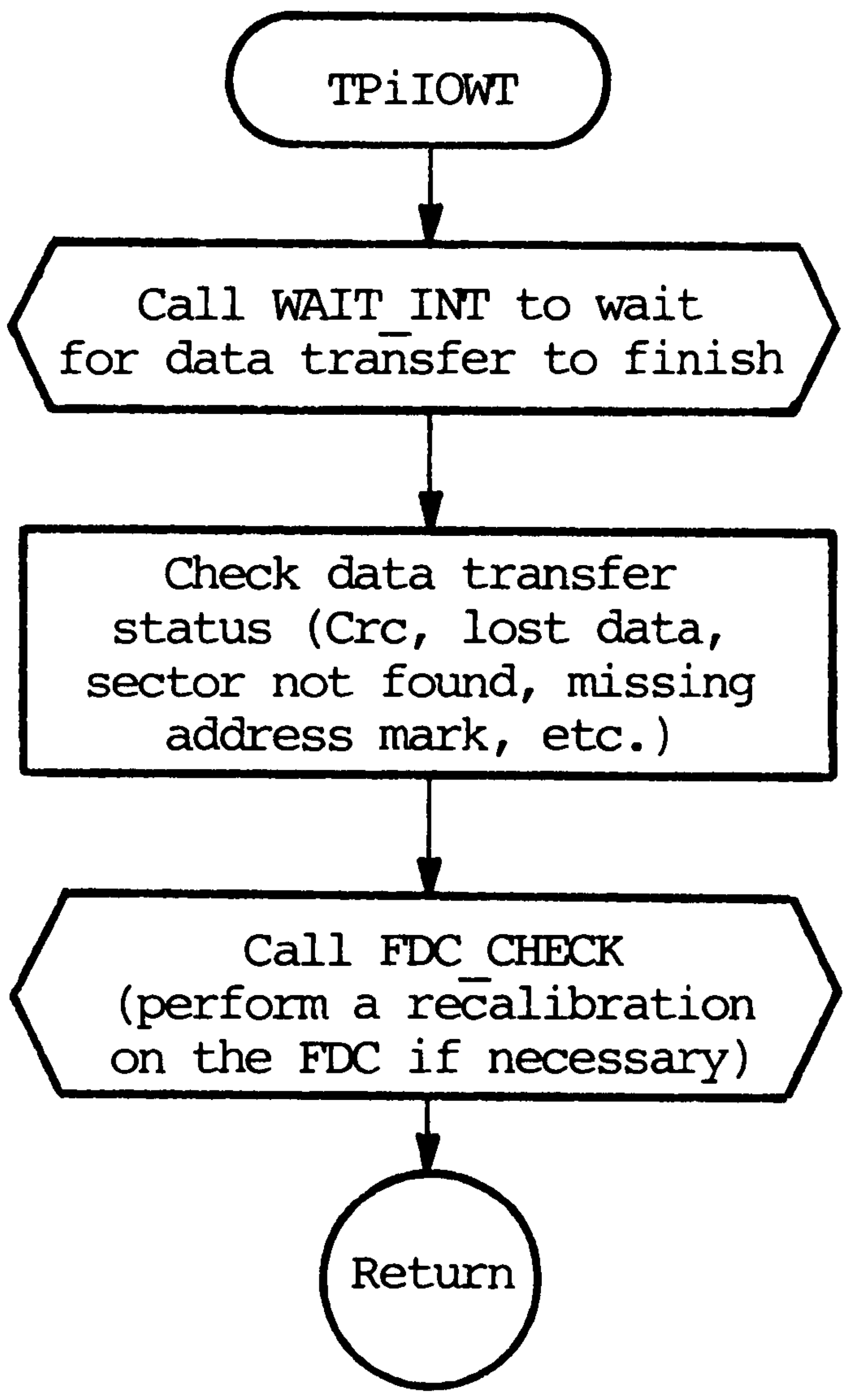
Called by: TPOWRT, TPOWRTI.


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

TPiIOWT

Waits for data transfer operation to finish and checks status.

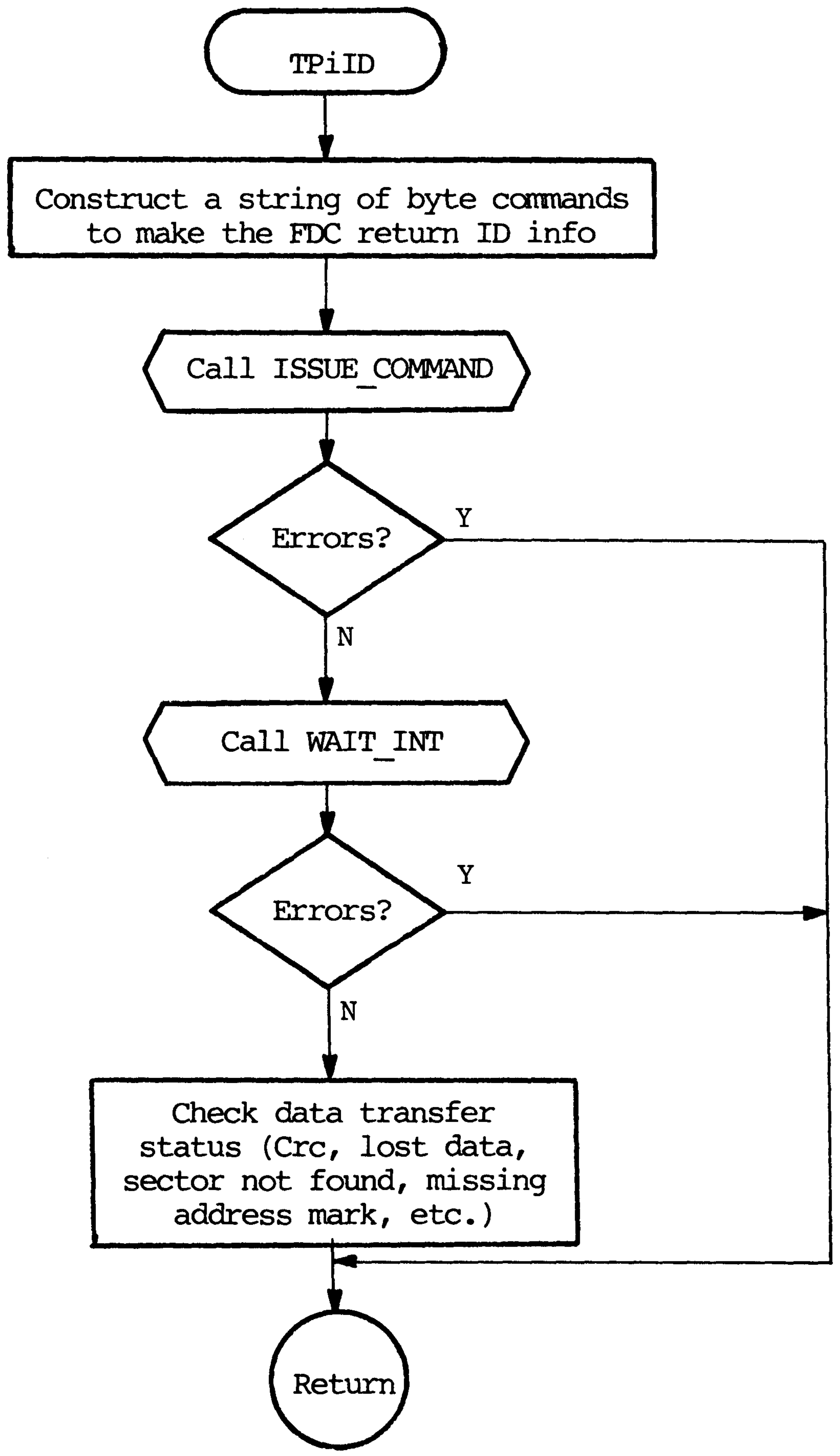
Called by: TPORED, TPOWRT, TPOIOWT.


TPiIOWT
Call WAIT_INT to wait for data transfer to finish
Check data transfer status (Crc, lost data, sector not found, missing address mark, etc.)
Call FDC_CHECK (perform a recalibration on the FDC if necessary)
Return

TPiID

Executes a read next ID command.

Called by: TP0ID.

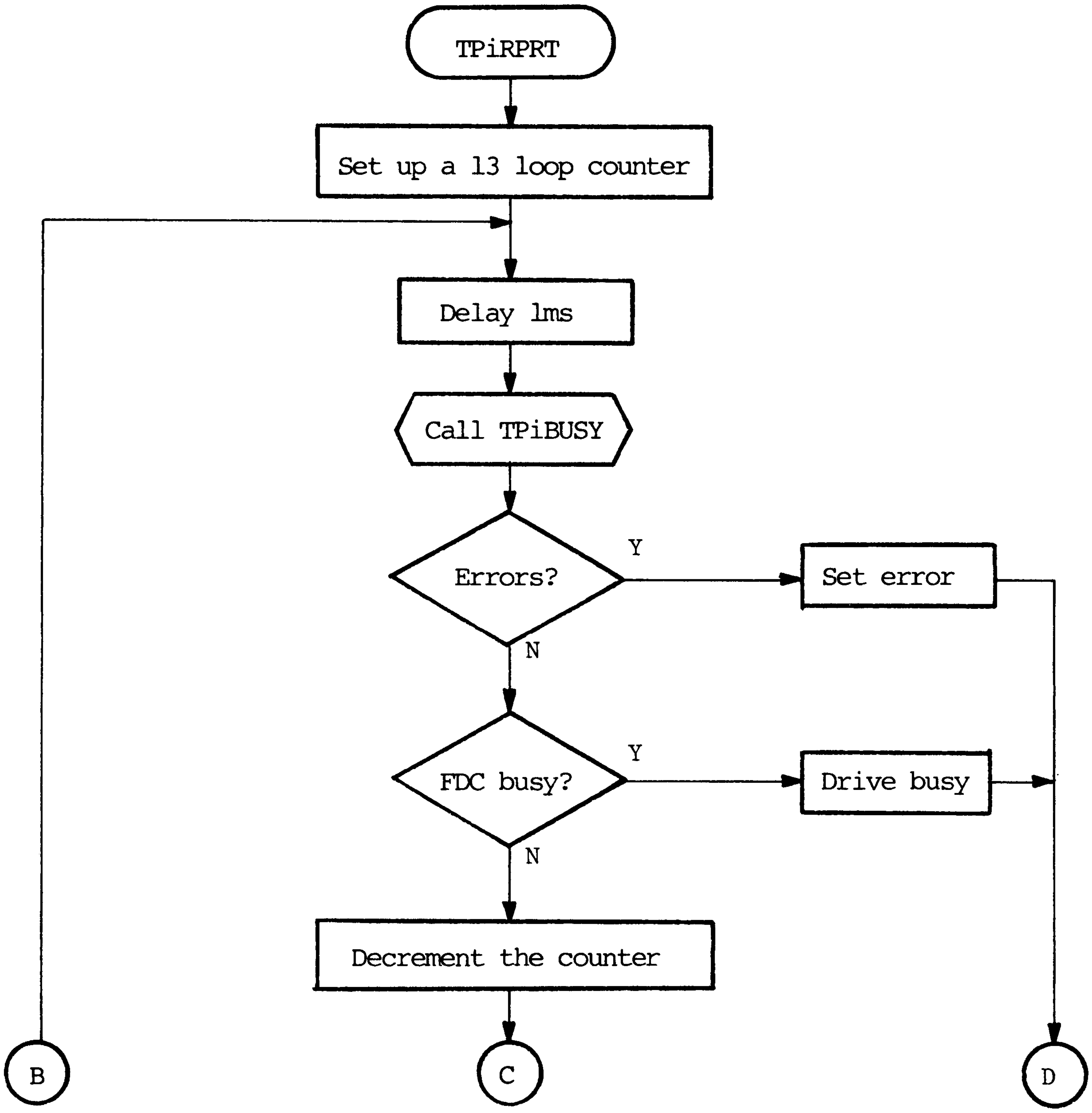


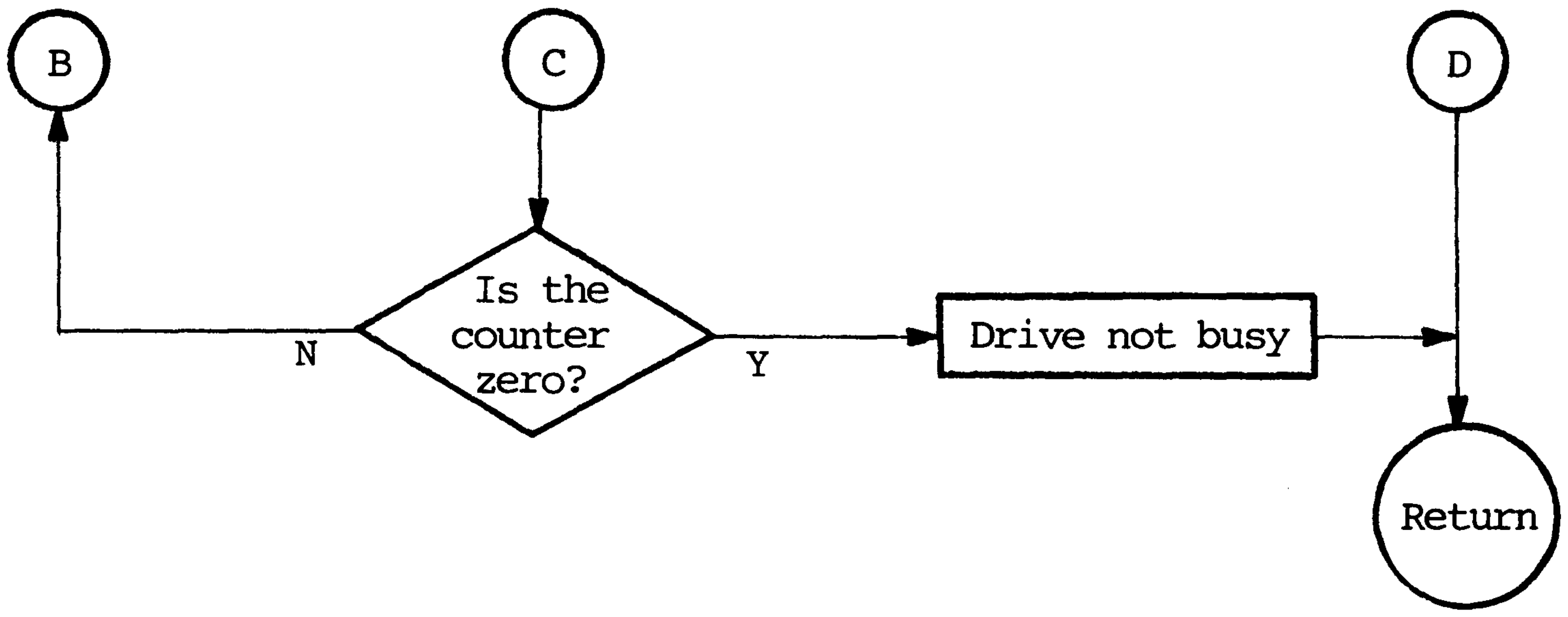
TPiID
Construct a string of byte commands to make the FDC return ID info
Call ISSUE_COMMAND
Errors?
Y
N
Call WAIT_INT
Errors?
Y
N
Check data transfer status (Crc, lost data, sector not found, missing address mark, etc.)
Return

TPiRPRT

Waits for 13ms after issuing of report status command and checks busy status.

Called by: TP0COMM, TPiRECAL.

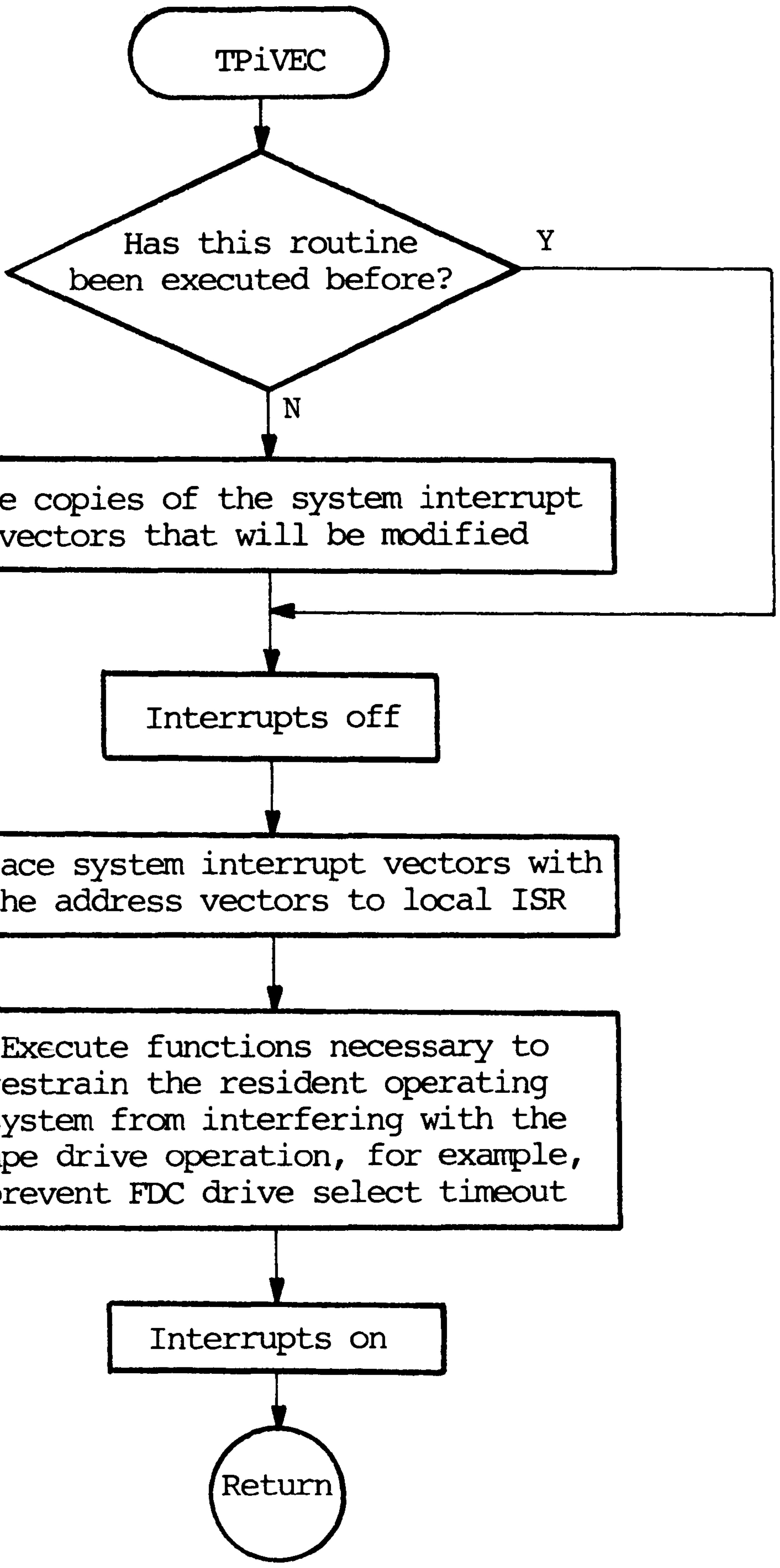



B
C
D
Is the counter zero?
N
Y
Drive not busy
Return

TPiVEC

Sets up tape interrupt service routine (ISR) vector.

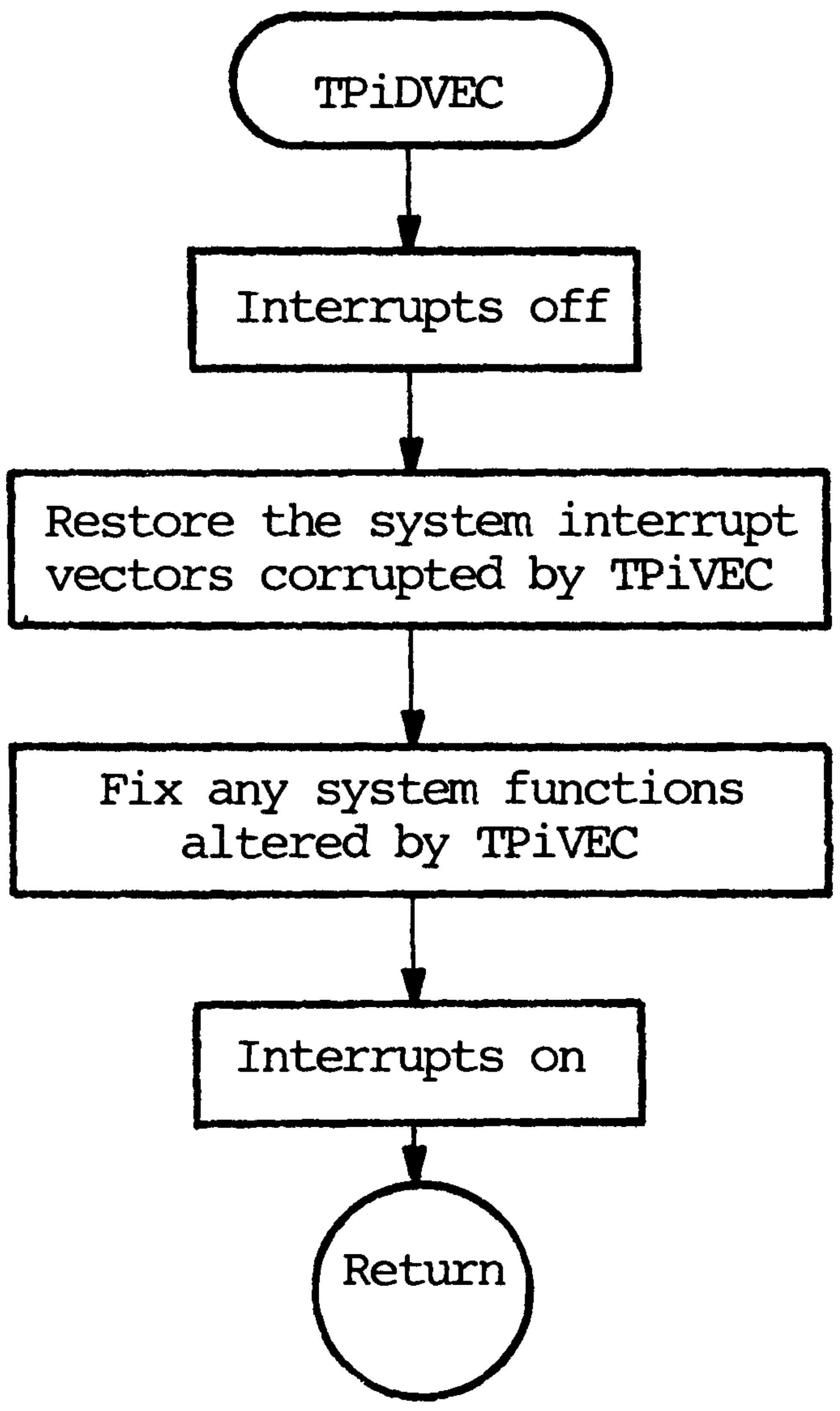
Called by: TP0INI, TP0TRM.


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

Restores system interrupt vectors.

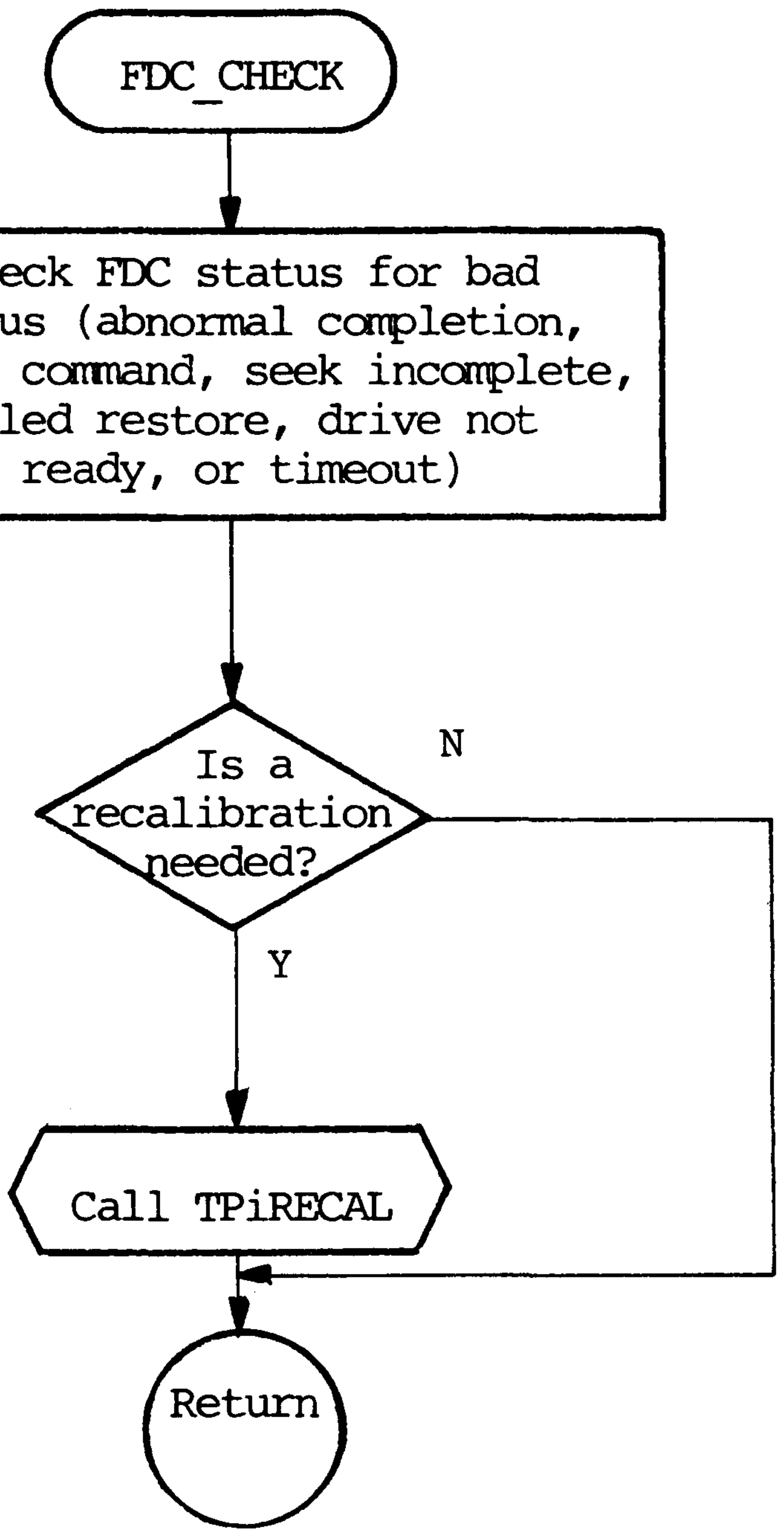
Called by: TP0INI, TP0TRM, TP0RESET.


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

FDC CHECK

Executes a drive recalibrate command if the previous FDC status was bad.

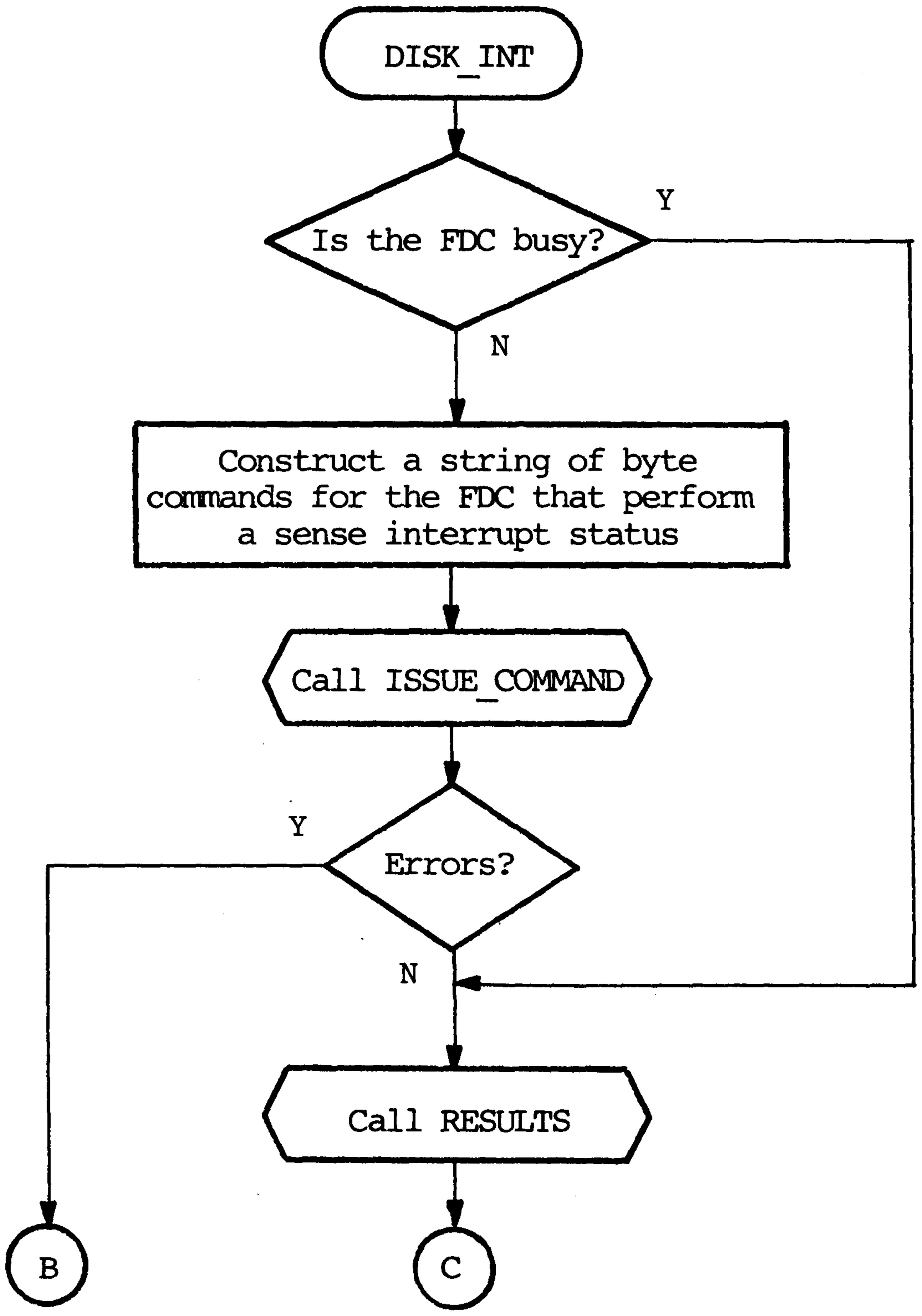
Called by: TPOCOMM, TPOSNS, TPOFRMT, TPOID, TPiBUSY, TPiIOWT, RW_OPN.

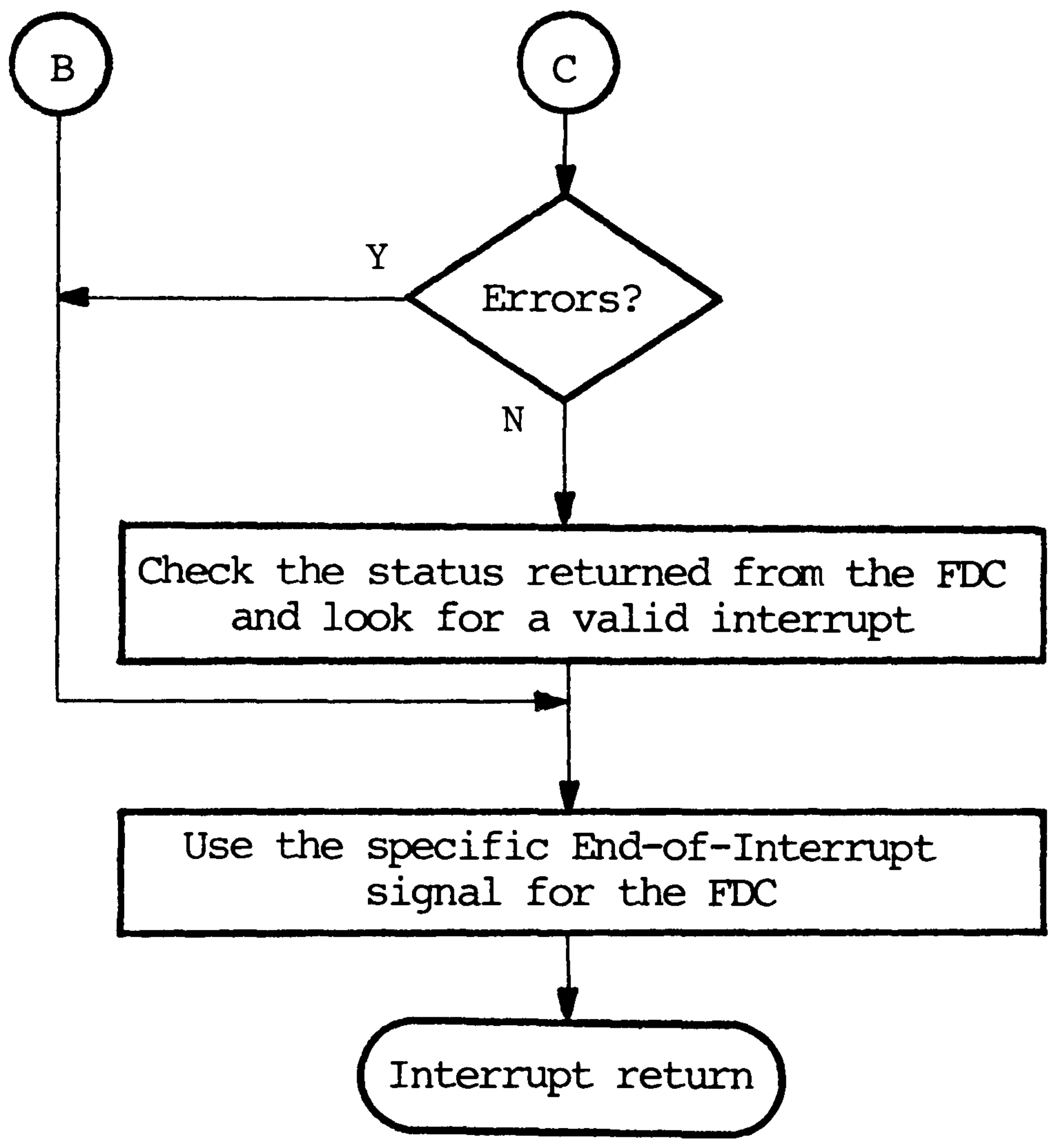

FDC_CHECK
Check FDC status for bad status (abnormal completion, invalid command, seek incomplete, failed restore, drive not ready, or timeout)
Is a recalibration needed?
N
Y
Call TPiRECAL
Return

DISK_INT

Handles FDC interrupts. (Hardware interrupt)

Called by: WAIT_INT.

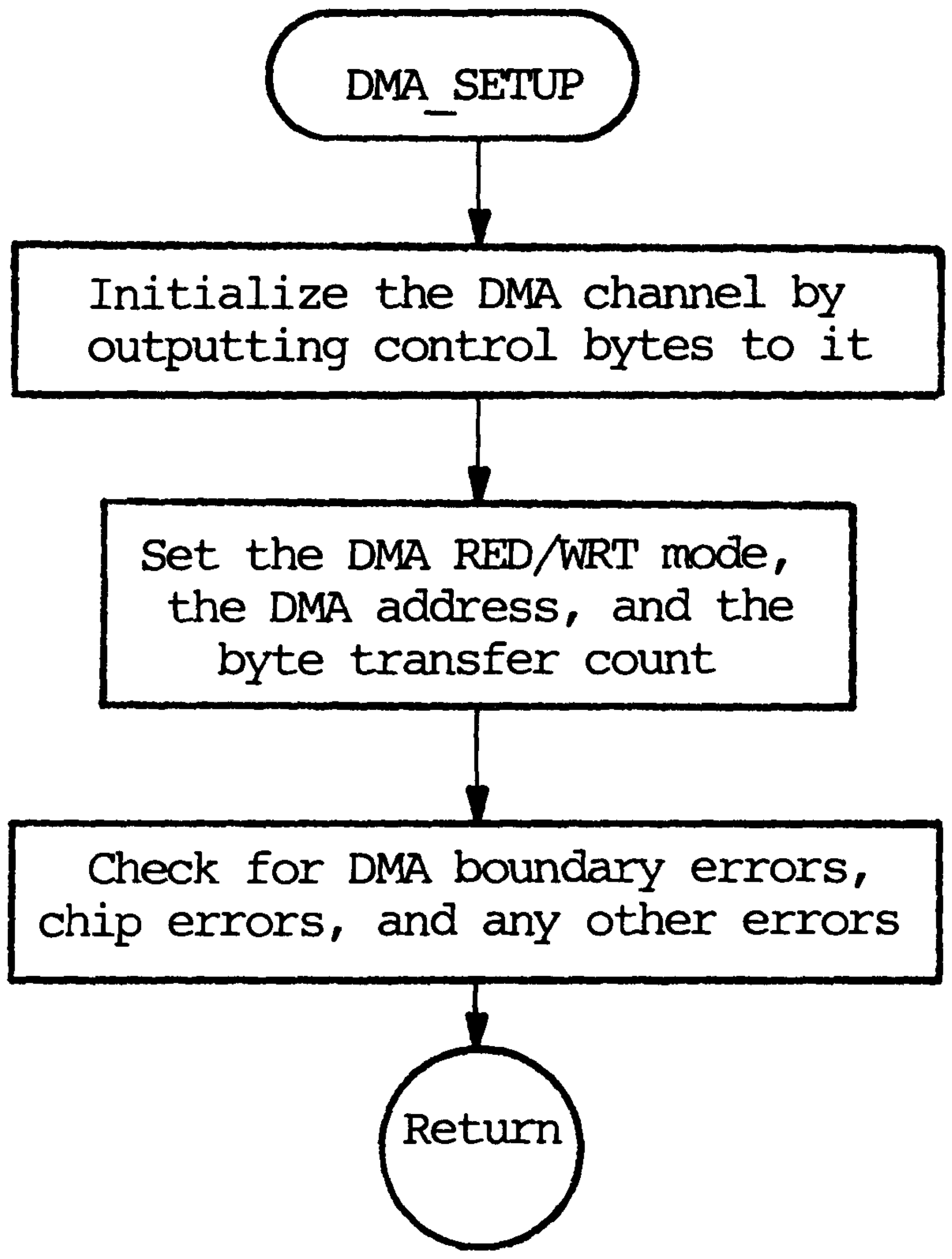

DISK_INT
Is the FDC busy?
Y
N
Construct a string of byte commands for the FDC that perform a sense interrupt status
Call ISSUE_COMMAND
Y
Errors?
N
Call RESULTS
B
C


B
C
Errors?
Y
N
Check the status returned from the FDC and look for a valid interrupt
Use the specific End-of-Interrupt signal for the FDC
Interrupt return

DMA SETUP

Sets up DMA for read or write operation.

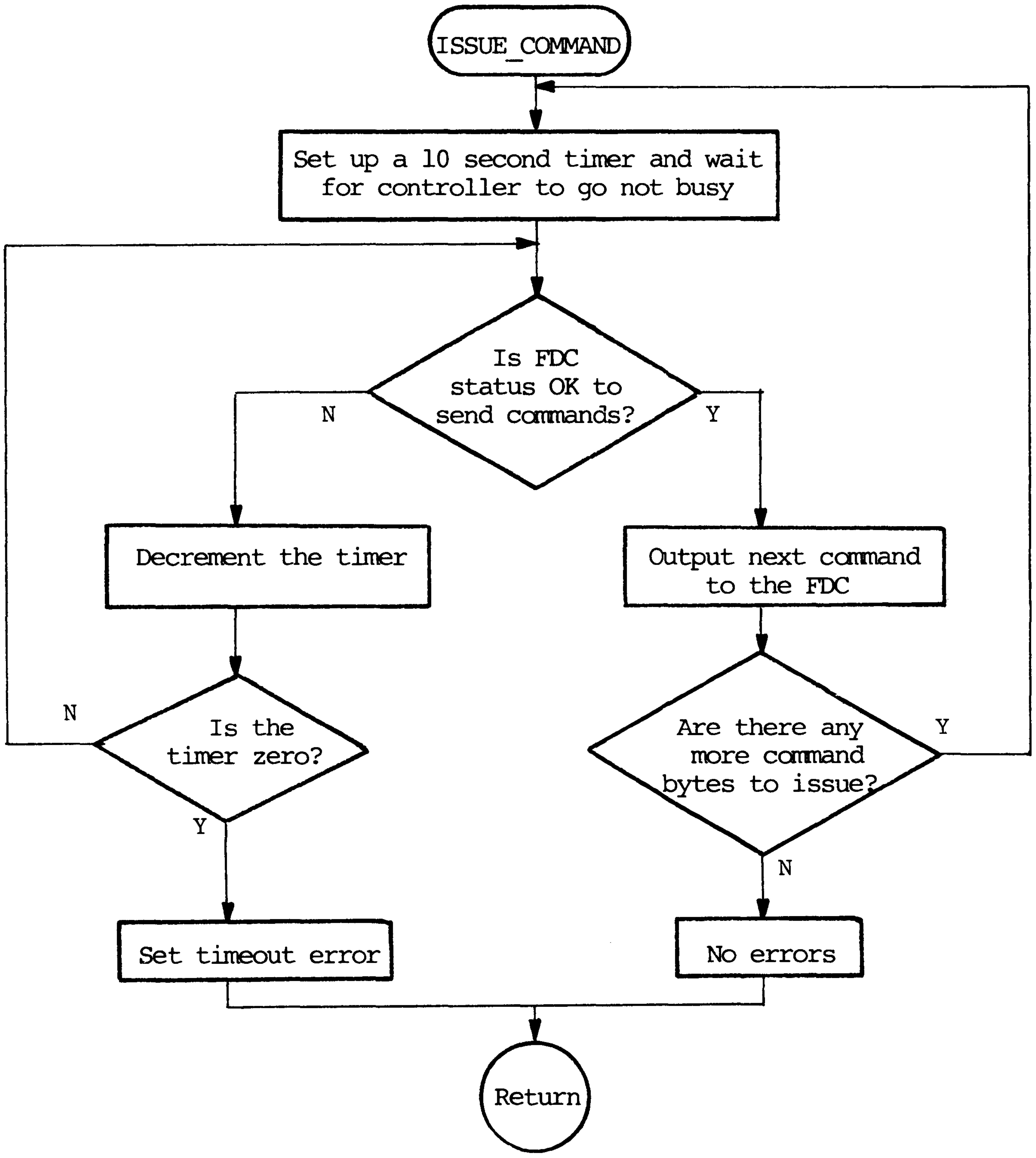
Called by: TP0FRMT, RW_OPN.


DMA_SETUP
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

Sends a train of stepping pulses to the drive to execute a command.

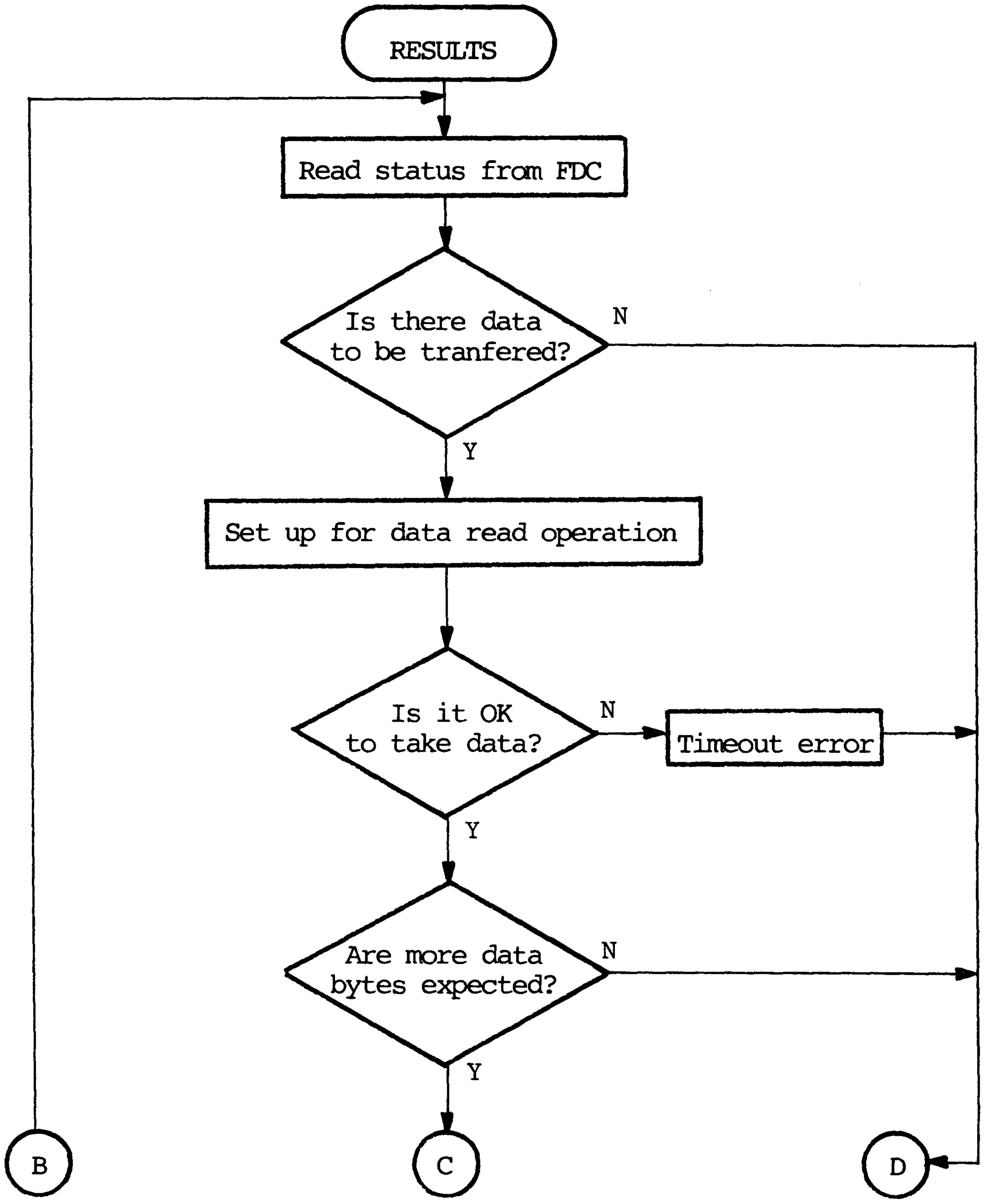
Called by: TP0INI, TP0TRM, TP0COMM, TP0FRMT, TPiRECAL, TPiBUSY, TPiID, DISK_INT, RW_OPN.

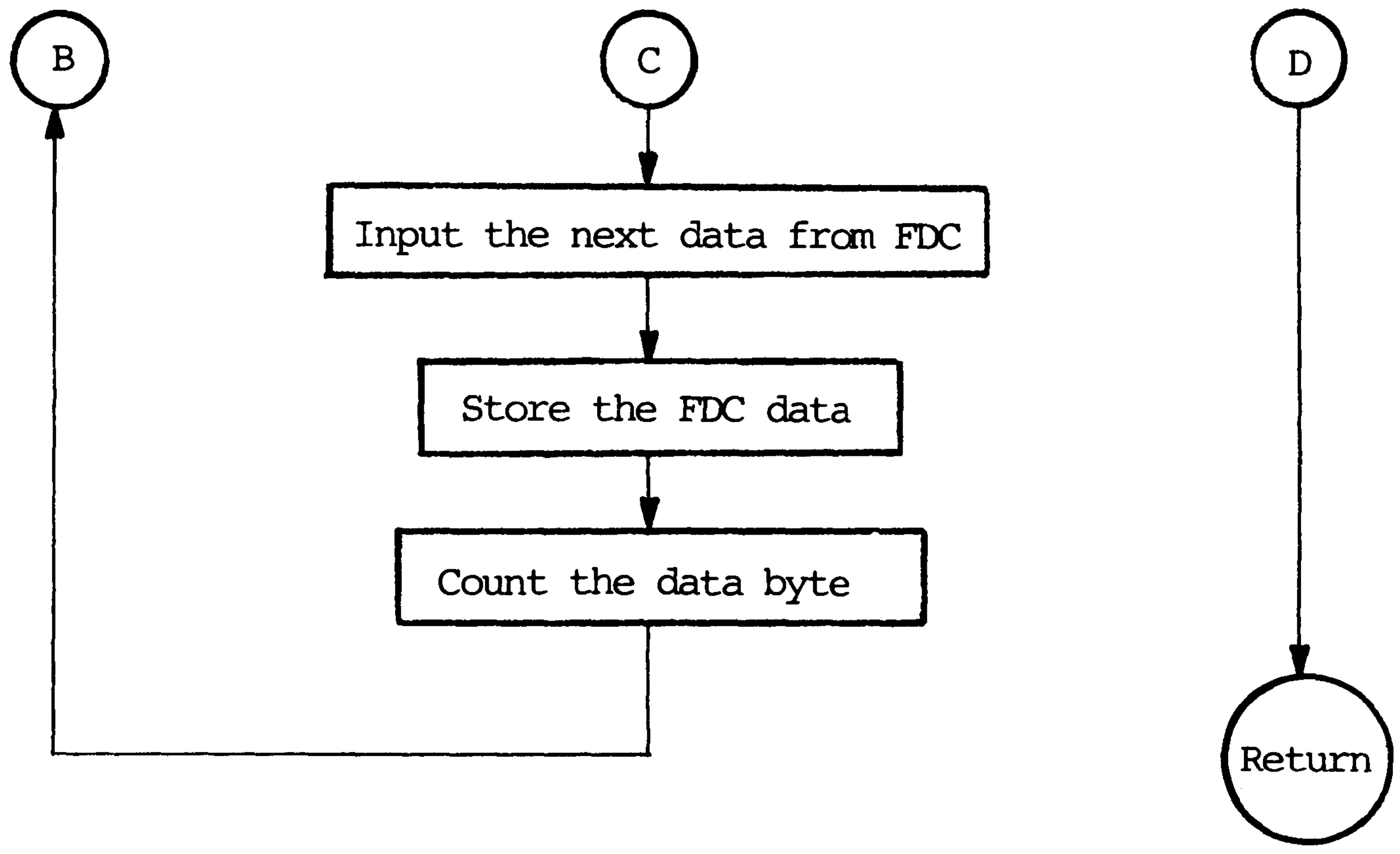

ISSUE_COMMAND
Set up a 10 second timer and wait for controller to go not busy
Is FDC status OK to send commands?
N
Y
Decrement the timer
Output next command to the FDC
Is the timer zero?
N
Y
Are there any more command bytes to issue?
Y
N
Set timeout error
No errors
Return

RESULTS

Reads and stores the results from the FDC status and data.

Called by: TPiBUSY, DISK_INT.

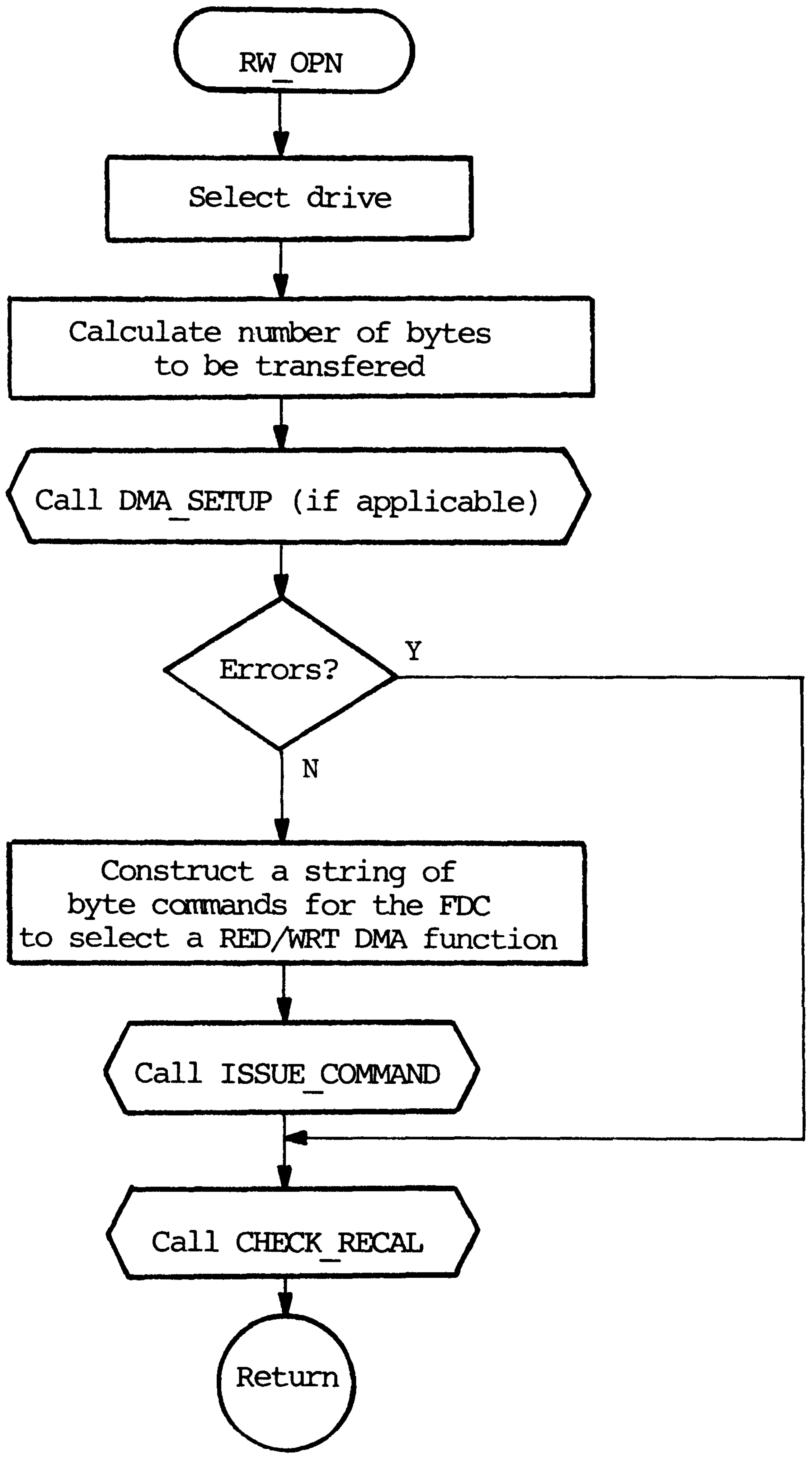

RESULTS
Read status from FDC
Is there data to be tranfered?
N
Y
Set up for data read operation
Is it OK to take data?
N
Timeout error
Y
Are more data bytes expected?
N
Y
B
C
D


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

RW_OPN

Sets up and executes a read or write operation.

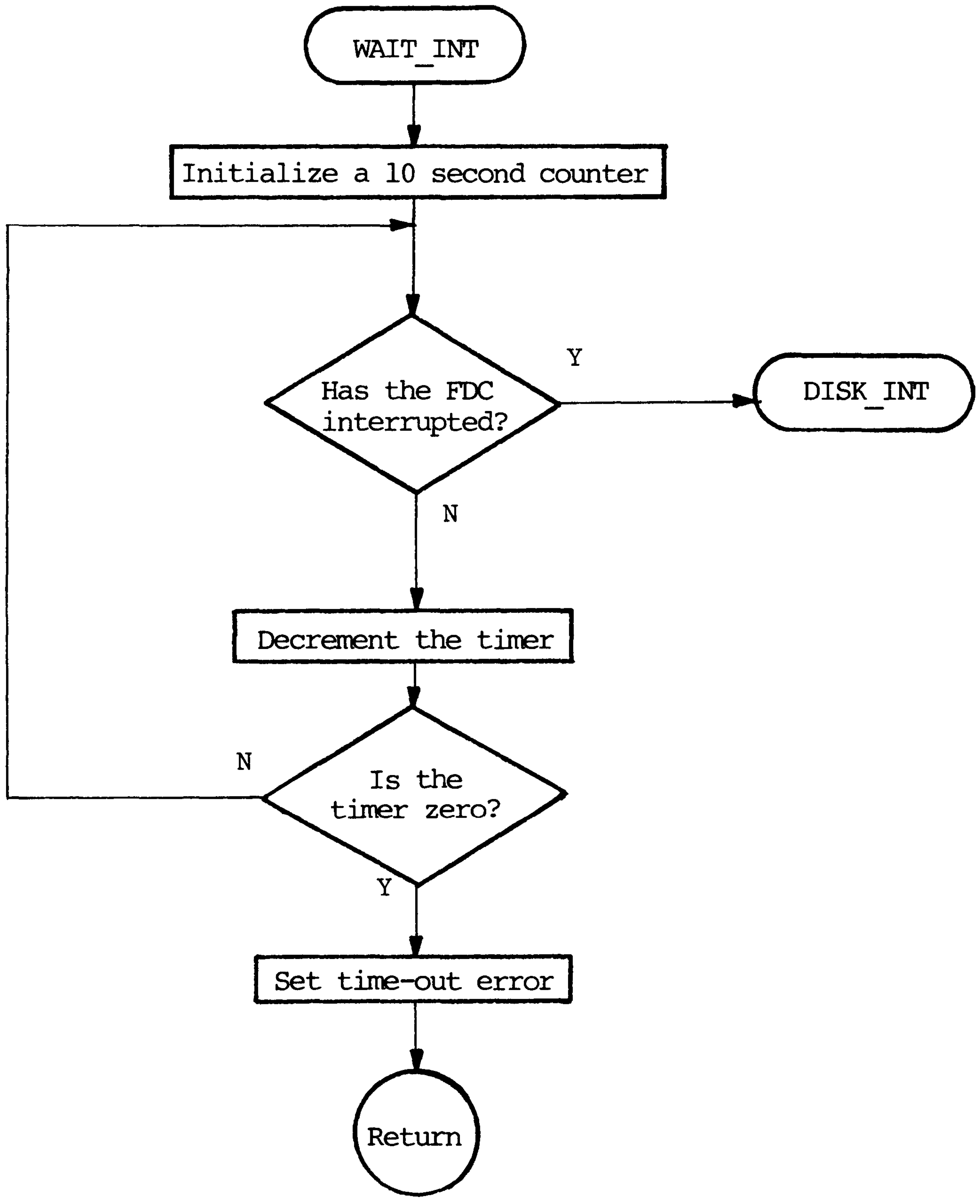
Called by: TPiREDI, TPiWRTI.


RW_OPN
Select drive
Calculate number of bytes to be transfered
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

WAIT INT

Waits for 13ms after issuing of report status command and reports busy status.

Called by: TPOCOMM, TPiRECAL.



A.0 Indexes

A.1 Index

A.2 Index of Tape Driver Routines

A.3 Index of Figures

A.4 Index of Tables