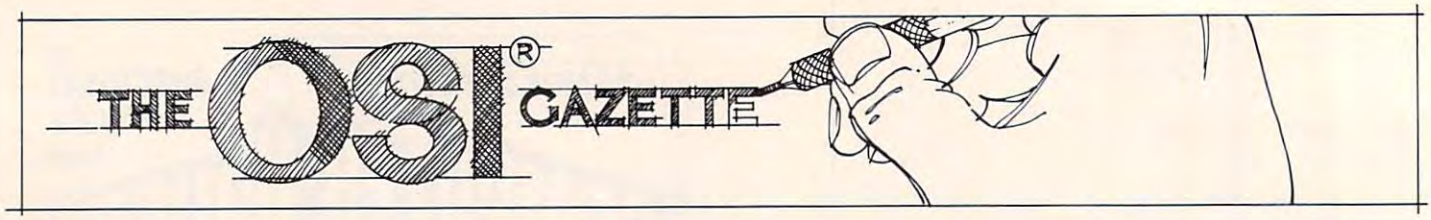



## Part II:

# A Small Operating System: OS65D, The Disk Routines

T. R. Berger
Coon Rapids, MN

*Editor's Note: Part I appeared last month. Here, the author presents a map of the disk routines. — RTM*

Let's turn to track zero. Exactly one ms. after the index hole a two byte address is recorded on the disk in high byte-low order. This address is read by the ROM on boot. It is the start address for loading track zero into memory. Next comes the number of pages in track zero. Finally, that many pages of data are written on the track. There are no track start or stop markings. After track zero is loaded, the computer always jumps to $2200. Hopefully, track zero has been loaded in that vicinity. It would appear that OSI did not think the track zero format over very carefully.

### Subroutine Descriptions

Most of the disk routines are self-explanatory. Because these routines are far more involved than those in the kernel, many more flow charts are needed. Let's run through the memory map in order, commenting on special properties of certain subroutines.

The timing routines at $2678, $267A, and $26A2 are independent of the system clock. The wait time in the routines at $2700, $289F, and $28A4 should be divided by *T* if the system clock is T MHZ.

OS65D does not use binary track numbers, but BASIC does. Thus BASIC uses $26A6, but OS65D enters this routine at $26BC with the BCD track number in the accumulator. With a binary track number in the accumulator, this routine may be entered at $26A9. It will move the disk head over the correct track after some error checking.

The sequence beginning at $2728 may be viewed as the standard startup to read or write a track or sector. It puts the head on the disk, finds the index hole, then initializes the disk data ACIA.

The EXAMINE command uses $2739 to load the entire contents of a track into memory without regard to error checking, track formatting, or sectoring. This type of command is only possible with the asynchronous data format used by OSI. If you crash a track, this command can prove invaluable in retrieving what may remain. I view this routine as a utility. It should reside on the disk and not in memory, unless needed. The initialize routine at $2768 used on a full disk falls in the same category. Such programs as these should be transient, i.e. only called when needed.

The major "Save a Sector" routine begins at $27D7. It uses the data in $265E-$2661. Most of OS65D's disk data is stored in page zero. Because Zpage is swapped out when BASIC comes in, the most important data is repeated in $265C-$2662. BASIC passes its values to these latter locations. LOAD and SAVE routines must then move this data to Zpage. Since OS65D can put information directly into Zpage, it puts the save vector into $FE, $FF directly, entering the Save routine at $27E1. Except when SAVE or CALL are used, all saving is done in Sector one for 12 ($OC) pages on 8" floppies and for eight pages on minifloppies. After a write, the sector is reread and compared with memory. If the comparison fails, the sector is reread again. This may occur up to four times. If comparison still fails, another attempt is made to write the sector. If comparison fails after four rereads again, the operation is aborted with Error #2. To my recollection, I've never seen Error #2 occur. It might happen on an old worn disk, on a midnight special, or with a very dirty head.

The major "Read a Sector" routine is $295D. It uses data in $265-E-$2662. Again OS65D may enter this routine at $2967 if the load vector at $FE, $FF has been set. This program tries to read a sector seven times. The only error check (other than sector seek errors which abort immediately) is a parity check for each byte. If, after seven tries, a read still fails, then the head is moved down then up one track. This whole process may be repeated up to four times before Error #1 is reported. This error also seems to be very rare.

Both read and save routines use the sector seeking routine at $28C4 which, in turn, calls $2998.

# OSI TRS-80 COLOR-80 OSI

**GALAXIAN** - 4K - One of the fastest and finest arcade games ever written for the OSI, this one features rows of hard-hitting evasive dogfighting aliens thirsty for your blood. For those who loved (and tired of) Alien Invaders. Specify system — A bargain at $9.95 OSI

**LABYRINTH** - 8K - This has a display background similar to MINOS as the action takes place in a realistic maze seen from ground level. This is, however, a real time monster hunt as you track down and shoot mobile monsters on foot. Checking out and testing this one was the most fun I've had in years! — $13.95. OSI

**THE AARDVARK JOURNAL**

**FOR OSI USERS** — This is a bi-monthly tutorial journal running only articles about OSI systems. Every issue contains programs customized for OSI, tutorials on how to use and modify the system, and reviews of OSI related products. In the last two years we have run articles like these!

1) A tutorial on Machine Code for BASIC programmers.
2) Complete listings of two word processors for BASIC IN ROM machines.
3) Moving the Directory off track 12.
4) Listings for 20 game programs for the OSI.
5) How to write high speed BASIC — and lots more —

Vol. 1 (1980) 6 back issues - $9.00
Vol. 2 (1981) 4 back issues and subscription for 2 additional issues - $9.00.

**ADVENTURES!!!**

**For OSI, TRS-80, and COLOR-80.** These Adventures are written in BASIC, are full featured, fast action, full plotted adventures that take 30-50 hours to play. (Adventures are interactive fantasies. It's like reading a book except that you are the main character as you give the computer commands like "Look in the Coffin" and "Light the torch".)

Adventures require 8K on an OSI and 16K on COLOR-80 and TRS-80. They sell for $14.95 each.

**ESCAPE FROM MARS** (by Rodger Olsen)
This ADVENTURE takes place on the RED PLANT. You'll have to explore a Martian city and deal with possibly hostile aliens to survive this one. A good first adventure.

**PYRAMID** (by Rodger Olsen)
This is our most challenging ADVENTURE. It is a treasure hunt in a pyramid full of problems. Exciting and tough!

**TREK ADVENTURE** (by Bob Retelle)
This one takes place aboard a familiar starship. The crew has left for good reasons - but they forgot to take you, and now you are in deep trouble.

**DEATH SHIP** (by Rodger Olsen)
Our first and original ADVENTURE, this one takes place aboard a cruise ship - but it ain't the Love Boat.

**VAMPIRE CASTLE** (by Mike Bassman)
This is a contest between you and old Drac - and it's getting a little dark outside. $14.95 each.

**OSI NEW-NEW-NEW OSI**
**TINY COMPILER**

The easy way to speed in your programs. The tiny compiler lets you write and debug your program in Basic and then automatically compiles a Machine Code version that runs from 50-150 times faster. The tiny compiler generates relocatable, native, transportable machine code that can be run on any 6502 system.

It does have some limitations. It is memory hungry — 8K is the minimum sized system that can run the Compiler. It also handles only a limited subset of Basic — about 20 keywords including FOR, NEXT, IF THEN, GOSUB, GOTO, RETURN, END, STOP, USR(X), PEEK, POKE, -, =, *, /, , <,> , Variable names A-Z, and Integer Numbers from 0-64K.

TINY COMPILER is written in Basic. It can be modified and augmented by the user. It comes with a 20 page manual.
TINY COMPILER — $19.95 on tape or disk OSI

**SUPERDISK II**

This disk contains a new BEXEC* that boots up with a numbered directory and which allows creation, deletion and renaming of files without calling other programs. It also contains a slight modification to BASIC to allow 14 character file names.

The disk contains a disk manager that contains a disk packer, a hex/dec calculator and several other utilities.

It also has a full screen editor (in machine code on C2P/C4)) that makes corrections a snap. We'll also toss in renumbering and program search programs — and sell the whole thing for —

SUPERDISK II $29.95 (5¼") OSI

**BARE BOARDS FOR OSI C1P**

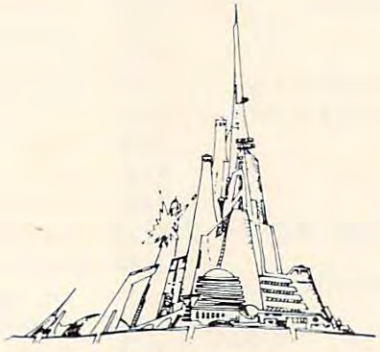
**MEMORY BOARDS!!!** — for the C1P — and they contain parallel ports!

Aardvarks new memory board supports 8K of 2114's and has provision for a PIA to give a parallel ports! It sells as a bare board for $29.95. When assembled, the board plugs into the expansion connector on the 600 board. Available now!

**PROM BURNER FOR THE C1P** — Burns single supply 2716's. Bare board — $24.95.

**MOTHER BOARD** — Expand your expansion connector from one to five connectors or use it to adapt our C1P boards to your C4/8P. - $14.95.

**16K RAM BOARD FOR C1P** — This one does not have a parallel port, but it does support 16K of 2114's. Bare Board $39.95.



**WORD PROCESSING THE EASY WAY— WITH MAXI-PROS**

This is a line-oriented word processor designed for the office that doesn't want to send every new girl out for training in how to type a letter.

It has automatic right and left margin justification and lets you vary the width and margins during printing. It has automatic pagination and automatic page numbering. It will print any text single, double or triple spaced and has text centering commands. It will make any number of multiple copies or chain files together to print an entire disk of data at one time.

MAXI-PROS has both global and line edit capability and the polled keyboard versions contain a corrected keyboard routine that make the OSI keyboard decode as a standard typewriter keyboard.

MAXI-PROS also has sophisticated file capabibilities. It can access a file for names and addresses, stop for inputs, and print form letters. It has file merging capabilities so that it can store and combine paragraphs and pages in any order.

Best of all, it is in BASIC (OS65D 51/4" or 8" disk) so that it can be easily adapted to any printer or printing job and so that it can be sold for a measly price.
MAXI-PROS — $39.95. Specify 5¼ or 8" disk.

**SUPPORT ROMS FOR BASIC IN ROM MACHINES** — C1S/C2S. This ROM adds line edit functions, software selectable scroll windows, bell support, choice of OSI or standard keyboard routines, two callable screen clears, and software support for 32-64 characters per line video. Has one character command to switch model 2 C1P from 24 to 48 character line. When installed in C2 or C4 (C2S) requires installation of additional chip. C1P requires only a jumper change. — $39.95
C1E/C2E similar to above but with extended machine code monitor. — $59.95 OSI

**ARCADE GAMES FOR OSI, COLOR-80 AND TRS-80 (8K OSI, 16K TRS-80 AND COLOR-80)**

**TIMETREK** - A REAL TIME, REAL GRAPHICS STARTRECK. See your torpedoes hit and watch your instruments work in real time. No more unrealistic scrolling displays! $14.95.

**STARFIGHTER** - This one man space war game pits you against spacecruisers, battlewagons, and one man fighters, you have the view from your cockpit window, a real time working instrument panel, and your wits. Another real time goody. $9.95

**BATTLEFLEET** - This grown up version of Battleship is the toughest thinking game available on OSI or 80 computers. There is no luck involved as you seek out the computers hidden fleet. A topographical toughie. $9.95

**QUEST** - A NEW IDEA IN ADVENTURE GAMES! Different from all the others, Quest is played on a computer generated mape of Alesia. Your job is to gather men and supplies by combat, bargaining, exploration of ruins and temples and outright banditry. When your force is strong enough, you attack the Citadel of Moorlock in a life or death battle to the finish. Playable in 2 to 5 hours, this one is different every time.
16K COLOR-80 OR TRS-80 ONLY. $14.95

***Please specify system on all orders***

**This is only a partial listing of what we have to offer. We offer over 120 games, ROMS, and data sheets for OSI systems and many games and utilities for COLOR-80 and TRS-80. Send $1.00 for our catalog.**


**AARDVARK TECHNICAL SERVICES, LTD.**
**2352 S. Commerce, Walled Lake, MI 48088**
**(313) 669-3110**


OSI COLOR-80

Further, they both use a dual purpose routine at $2905. If the accumulator is zero on entry, this routine reads to memory. If it is nonzero, then the routine compares with memory. The actual read and compare loops within this routine are separate. With 8" floppies and a 1 MHZ clock, the 6502 is not fast enough to get from one disk byte to the next if the read and compare loops are combined into one. As it stands, the compare loop just barely returns in time for the next comparison. With a 2 MHZ clock there is plenty of time.

I view the sector directory routines at $29F3 and $2A41 as utilities. They do not need to be resident in memory.

Machine language routines may access the disk directly. For example, to write a sector, locations $265E-$2662 should be assigned correct values. The following segment of code will write a sector to the disk.

```
10 JSR $26A6 ;Move head to track
20 JSR $2754 ;Engage head, find start of track
30 JSR $27D7 ;Write sector
40 JSR $2761 ;Disengage head
50 RTS
```

If the write address is already in $FE, $FF then $27D7 may be entered at $27E1. In this case, lines 20-40 may be replaced by JSR $2CA7, a kernel routine.

To read a sector, again assign correct values to $265E-$2662 then perform the following.

```
10 JSR $26A6 ;Move head to track
20 JSR $2754 ;Engage head, find start of track
30 JSR $295D ;Read sector
40 JSR $2761 ;Disengage head
50 RTS
```

If the read address is already in $FE, $FF then $295D may be entered at $2967. In this case, lines 20-40 may be replaced by the kernel routine:

```
JSR $2B1A
```

When we discuss the I/O section of OS65D we will see additional ways to read from and write to the disk.

**References:**

1. Jefferson Harman, "IBM Compatible Disk Drives", Byte October 1979, p. 100
2. Ira Rampil, "A Floppy Disk Tutorial", Byte December 1977, p. 24
3. Les Solomon, "BASICS of Computer Disk Systems", Popular Electronics November 1980, p. 53

---

**MAP – OS65D DISK HANDLER**

**DISK-MEMORY DATA**

| | |
|---|---|
| 265C | DRIVE NUMBER |
| 265D | CURRENT BCD TRACK NUMBER |
| 265E | SECTOR NUMBER |
| 265F | PAGE LENGTH OF SECTOR |
| 2660 | LOW BYTE LOAD/SAVE VECTOR |
| 2661 | HIGH BYTE LOAD/SAVE VECTOR |
| 2662 | BINARY TRACK NUMBER |

**DISK-Z PAGE**

| | |
|---|---|
| E5 | LAST TRACK OF FILE BEING HANDLED |
| F6 | NUMBER OF RETRIES ON WRITE |
| F7 | NUMBER OF HEAD MOVE RETRIES ON READ |
| F8 | NUMBER OR READ RETRIES BEFORE HEAD MOVE |
| F9 | SECTOR COUNT |
| FA | TARGET TRACK NUMBER ON SEEK |
| FB | SECTOR NUMBER READ ON DISK |
| FC | STACK POINTER (IN $29F3) |
| FD | SECTOR PAGE COUNT (IN $27D7) |
| FE | SYSTEM POINTER. USED AS |
| FF | LOAD AND SAVE VECTOR BY DISK |

---

**Subroutines – OS65D Disk Handler**

| | |
|---|---|
| 2663 | Home the Disk. Move the disk head to track 0. |
| 2678 | Wait 12 ms. |
| 267A | Wait X ms. |
| 2683 | Step up one track toward track 76. |
| 268A | Step down one track toward track 0. |
| 26A2 | Wait 8 ms. |
| 26A6 | Fetch binary track number from 2662 then: |
| 26A9 | Convert track number to BCD then: |
| 26BC | Check for track 0-76 BCD, check for drive ready, move disk head to track, adjust head current, and if an error occurs, abort and send an error message via 2A4B. |
| 2700 | Wait 20Y + 7 microseconds (1 MHZ clock). |
| 2708 | Adjust head current. |
| 271D | Find trailing edge of index hole. |
| 2728 | Engage head then: |
| 272B | Find index hole then: |
| 272E | Initialize disk ACIA. |
| 2739 | Engage head, read from index hole full around to index hole, then quit. |
| 2754 | Head down. |
| 2761 | Head up. |
| 2768 | Initialize full disk. |
| 277D | Initialize one track. |
| 27C2 | Send a byte to the disk. |
| 27CD | Fetch a byte from the disk. |
| 27D7 | Fetch sector save vectors then: |
| 27E1 | Save a sector. |
| 289F | Wait 800($FA) microseconds. |
| 28A4 | Wait 100Y microseconds. |
| 28B0 | Fetch a byte from the disk. Abort with an error message if over the index hole. |
| 28C4 | Find the end of the sector preceding the one in 265E. |
| 2905 | Read a sector to or compare a sector with memory. |
| 295D | Fetch disk read vector then: |
| 2967 | Read and reread a sector to memory, quit if successful or the full number of retries are exhausted. |
| 2998 | Find the end of the present sector. |
| 29C6 | Select the drive in 265C then: |
| 29DA | Check if the drive is ready. |
| 29EB | 8 drive select data bytes. |
| 29F3 | Output a sector directory. |
| 2A41 | Output subroutine for 29F3. |

# A Correction For Progressive Computing Chess 1.9


Dave Leskin
Calgary, Canada


Progressive Computing, based in Windsor Ontario, is an excellent source of OSI software with prompt and courteous service; however, there is a major error in their tape version of "Chess 1.9". This error is found in the opening tables. If you try the following sequence of moves you can determine if your copy of "Chess 1.9" has this error too. Note that the last move by the computer is illegal. Microchess notation in brackets.

| | Computer (White) | Human (Black) |
|---|---|---|
| 1 | P-K4 (13-33) | P-K4 (63-43) |
| 2 | N-KB3 (01-22) | N-QB3 (76-55) |
| 3 | B-QN5 (02-46) | N-KB3 (71-52) |
| 4 | B-KN5 (05-41) ??? | |

As you see the "B-KN5" jumps right over the Queen Pawn at 14. To solve this problem I changed the program so that the Queen Knight was moved from 06 to 25 instead. This results in a "Four Knights Game" which is a common opening used by many players. Just follow the steps listed below to effect the change.

### The Changes

**1.** Load "Chess 1.9"

**2.** Press "D" to enter monitor once the board appears.

**3.** Press ".0B34"

**4.** Press "/" to enter the data mode

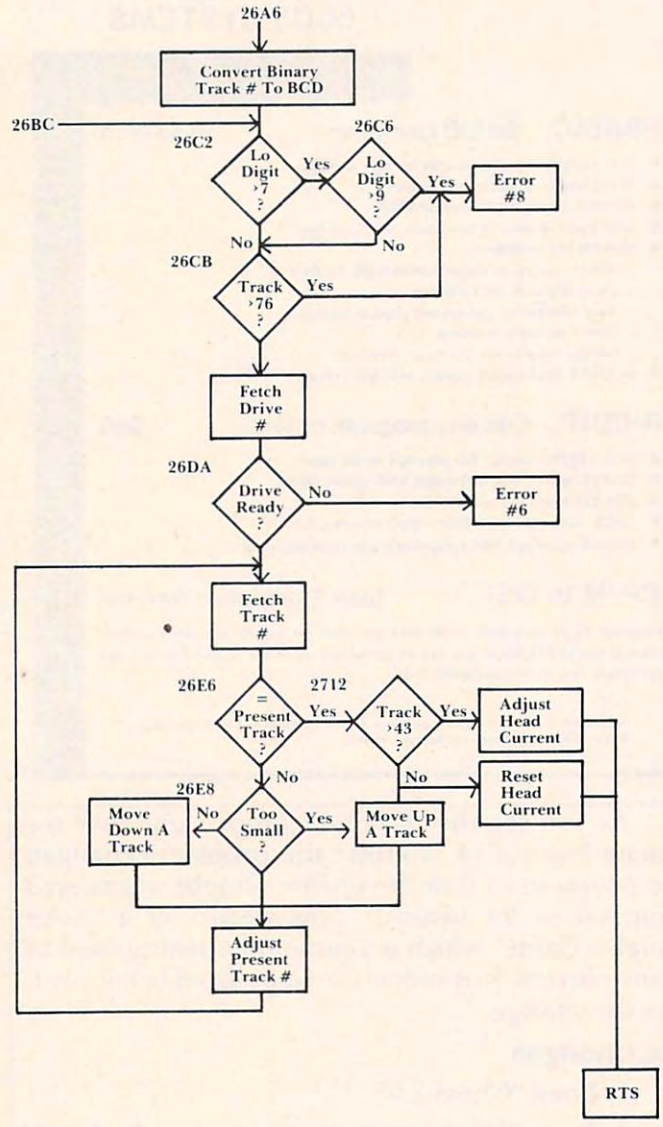
**5.** Press "25" which refers to the square that the piece will be moved to

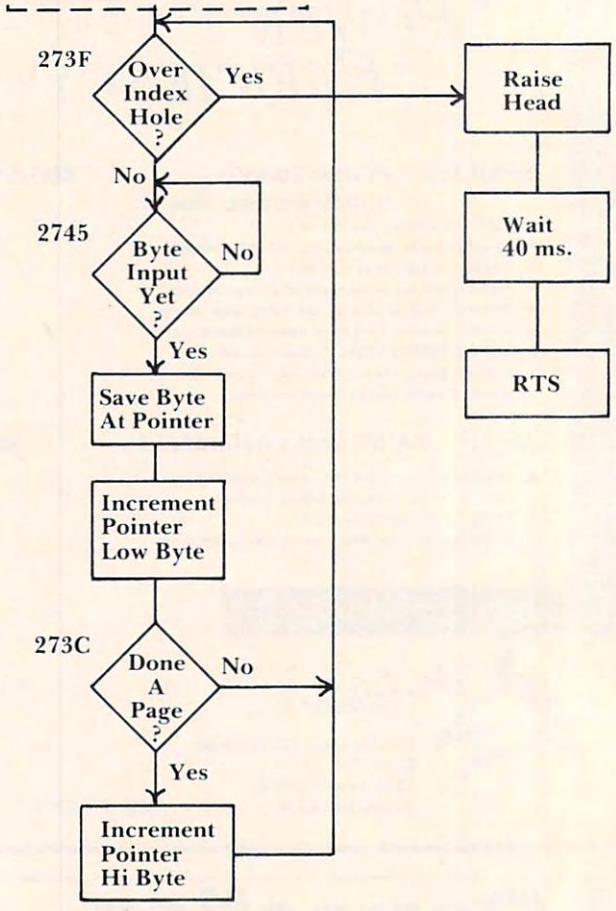
**6.** Press the "RETURN" key

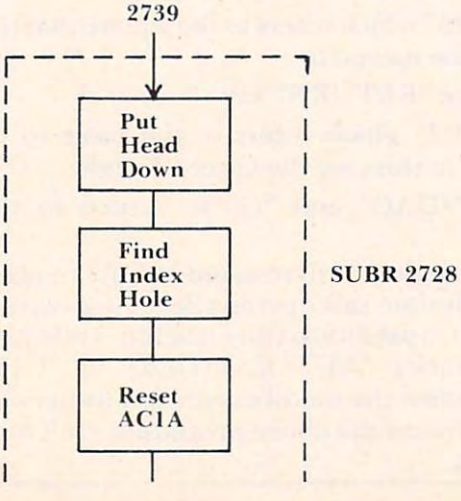
**7.** Press "07" which refers to the piece to be moved — in this case the Queen Knight

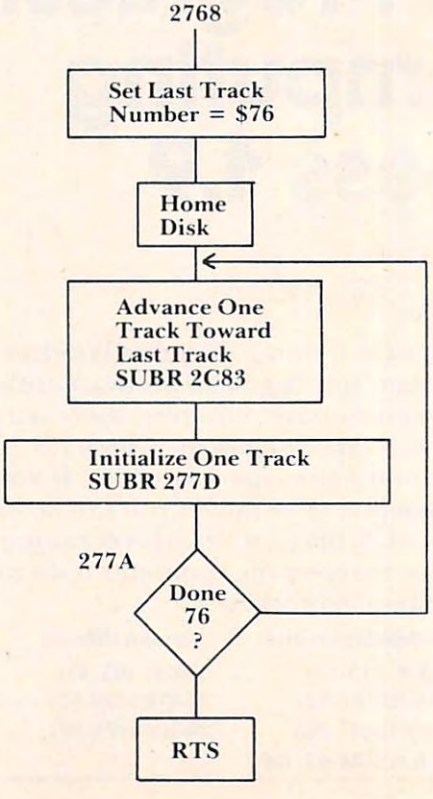
**8.** Press ".03AC" and "G" to return to the program

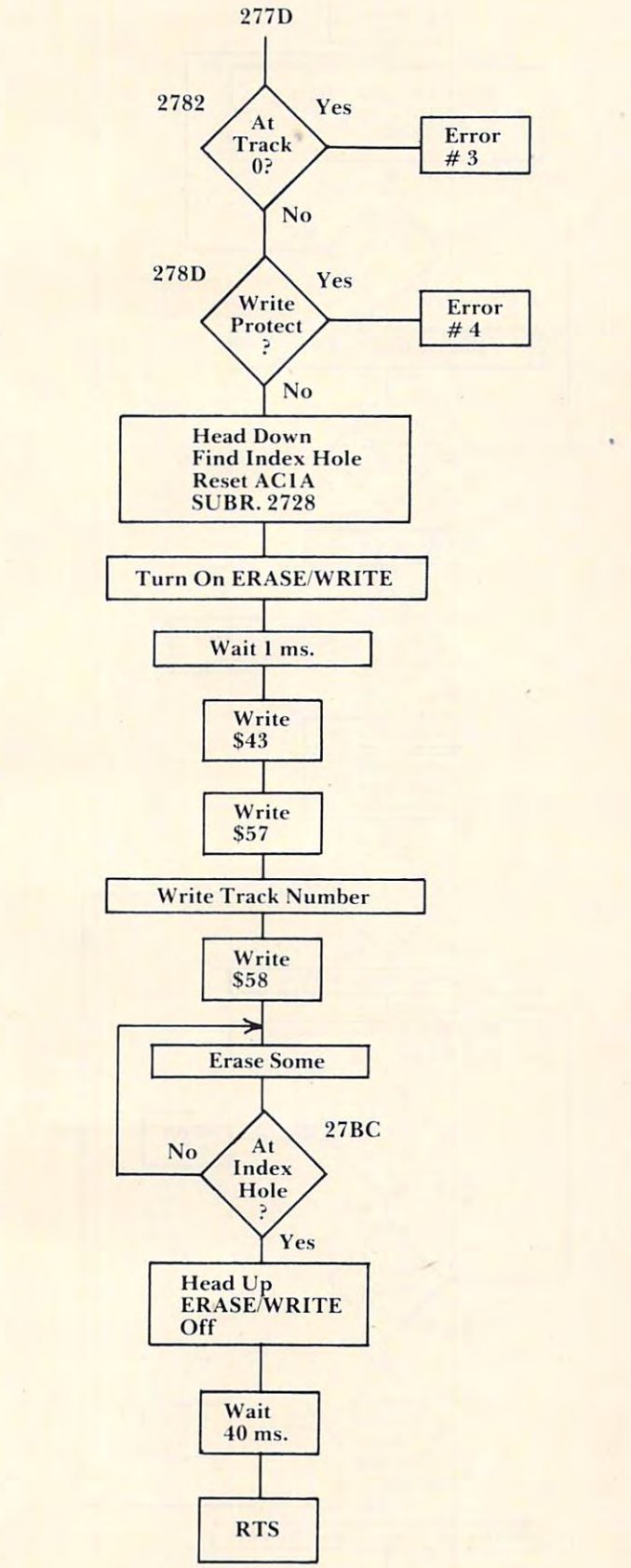
Now the program will respond N-QB3 in place of B-KN5 each time this opening sequence occurs. If you have the capability to store machine code programs (Aardvark's "AUTOLOADER" or "C1E" ROM) then record the modified version (otherwise you'll have to follow the above procedure each time you power up). ©

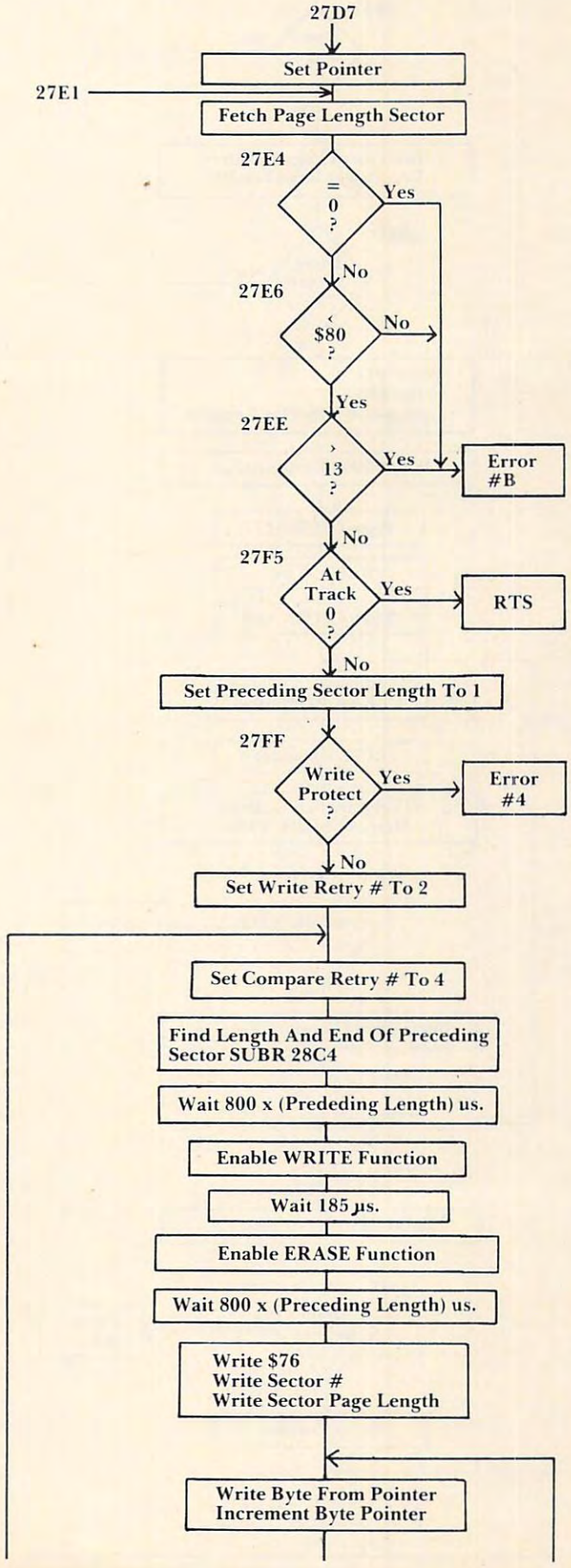

26A6
Convert Binary Track # To BCD
26BC
26C2
Lo Digit >7 ?
Yes
26C6
Lo Digit >9 ?
Yes
Error #8
No
No
26CB
Track >76 ?
Yes
Fetch Drive #
26DA
Drive Ready ?
No
Error #6
Fetch Track #
26E6
= Present Track ?
Yes
2712
Track >43 ?
Yes
Adjust Head Current
No
No
Reset Head Current
26E8
Too Small ?
No
Yes
Move Down A Track
Move Up A Track
Adjust Present Track #
RTS

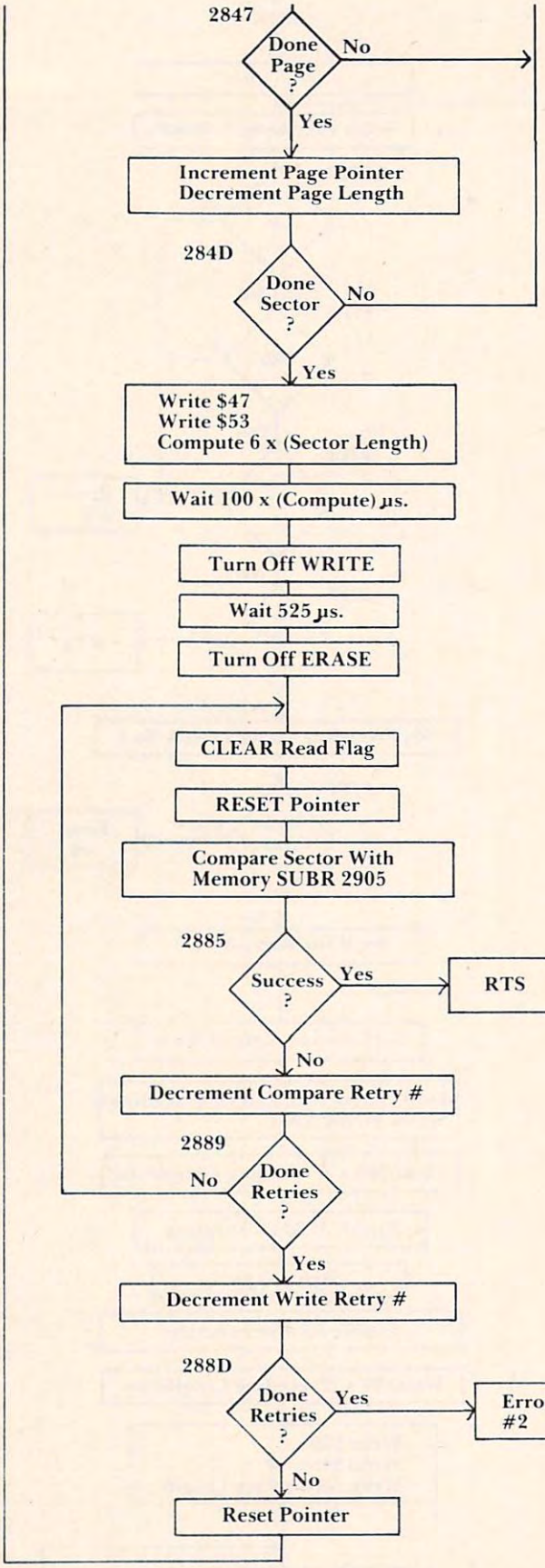

273F
Over Index Hole ?
Yes
Raise Head
Wait 40 ms.
RTS
No
2745
Byte Input Yet ?
No
Yes
Save Byte At Pointer
Increment Pointer Low Byte
273C
Done A Page ?
No
Yes
Increment Pointer Hi Byte

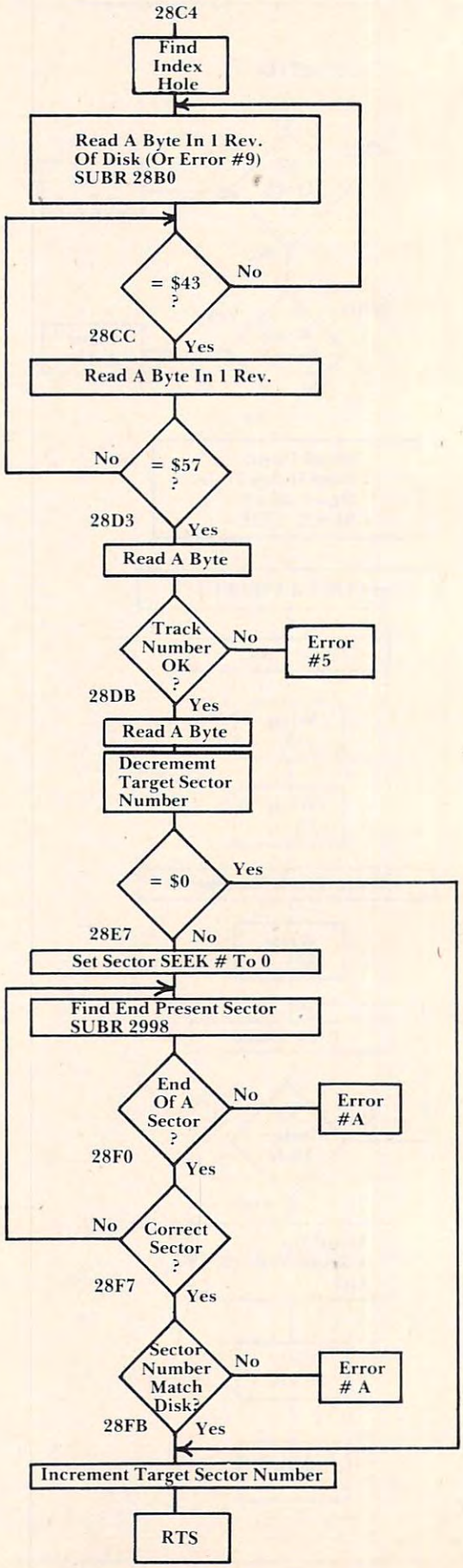

2739
Put Head Down
Find Index Hole
Reset AC1A
SUBR 2728

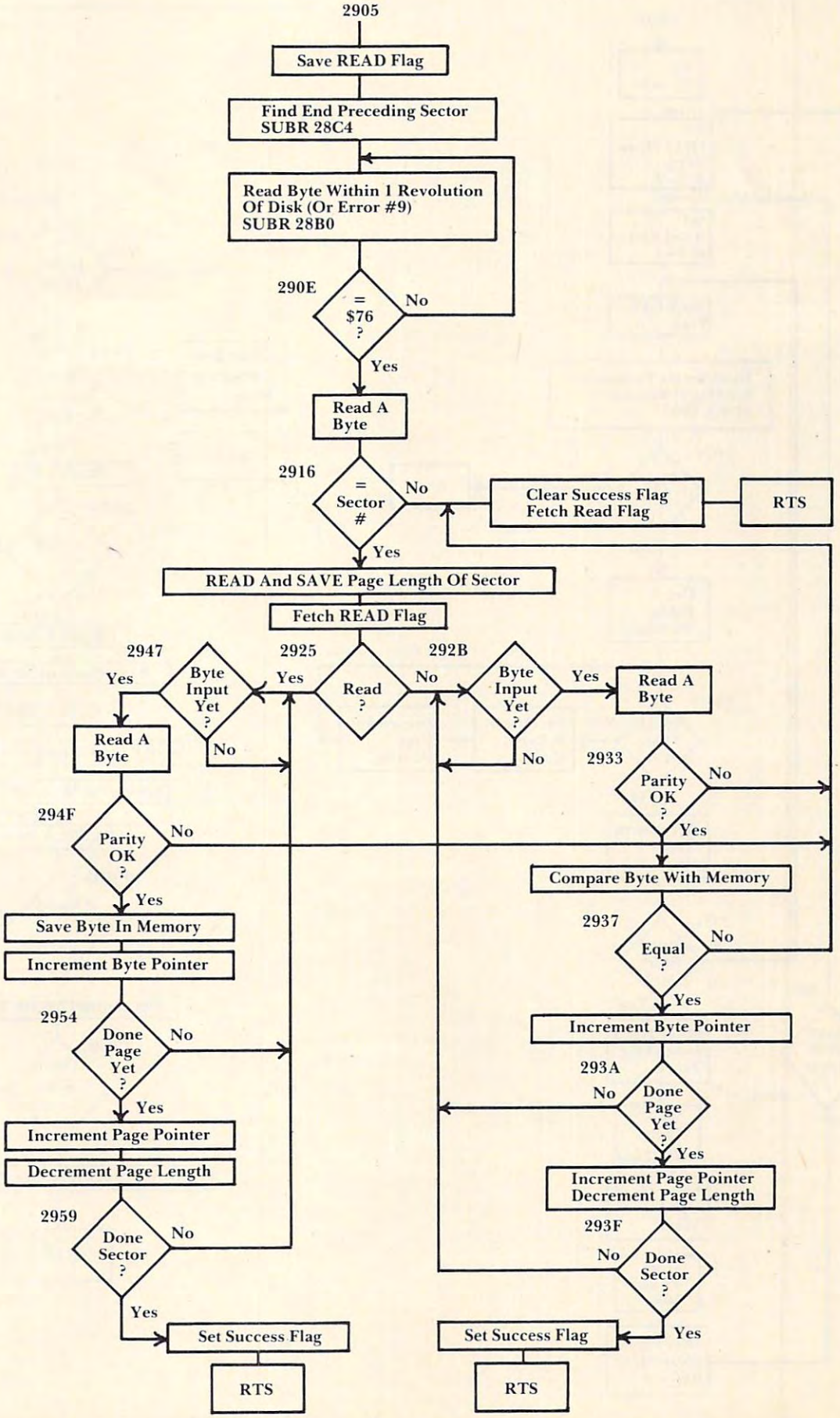

2768
Set Last Track Number = $76
Home Disk
Advance One Track Toward Last Track SUBR 2C83
Initialize One Track SUBR 277D
277A
Done 76 ?
RTS

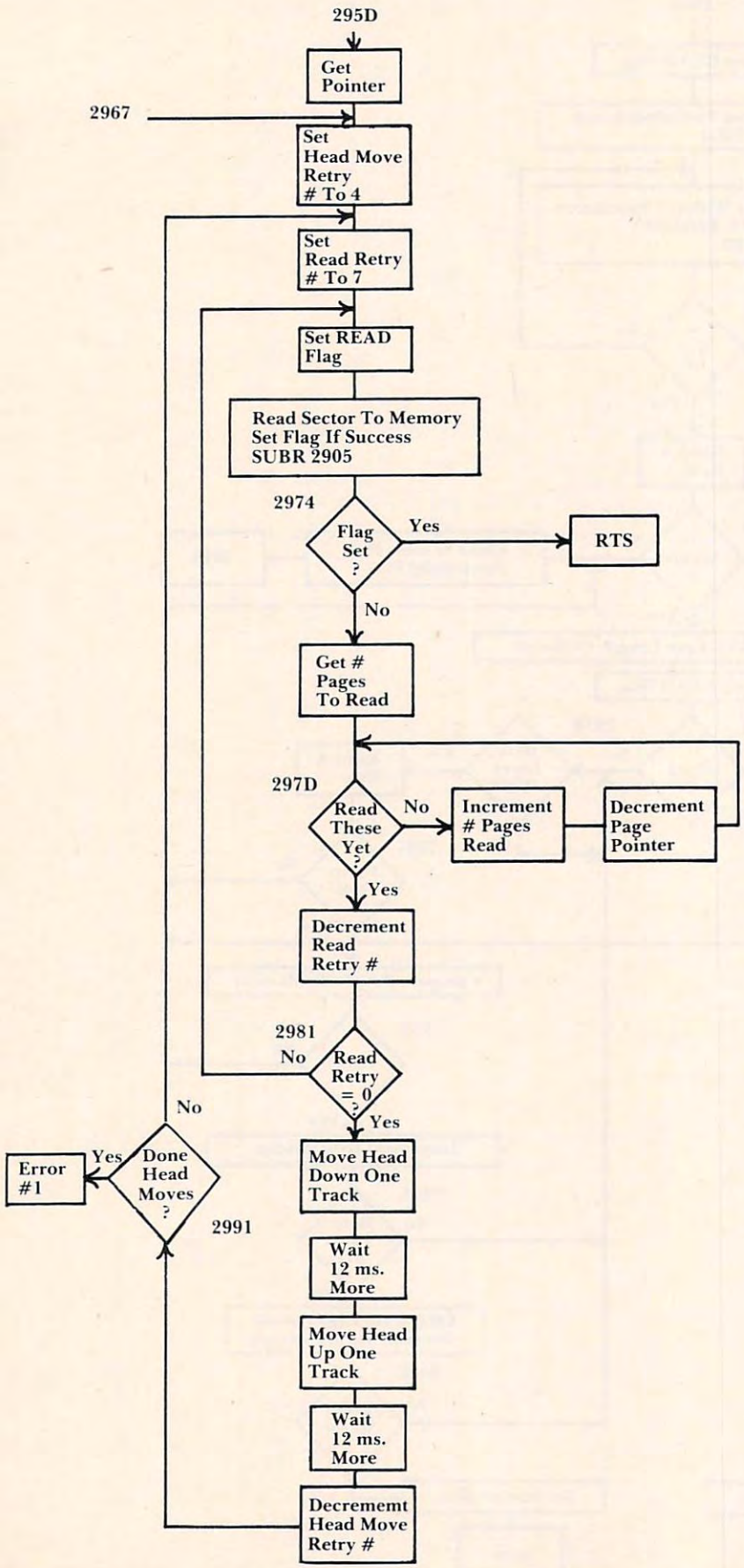

277D
2782
At Track 0?
Yes
Error # 3
No
278D
Write Protect ?
Yes
Error # 4
No
Head Down
Find Index Hole
Reset AC1A
SUBR. 2728
Turn On ERASE/WRITE
Wait 1 ms.
Write $43
Write $57
Write Track Number
Write $58
Erase Some
27BC
No
At Index Hole ?
Yes
Head Up
ERASE/WRITE Off
Wait 40 ms.
RTS

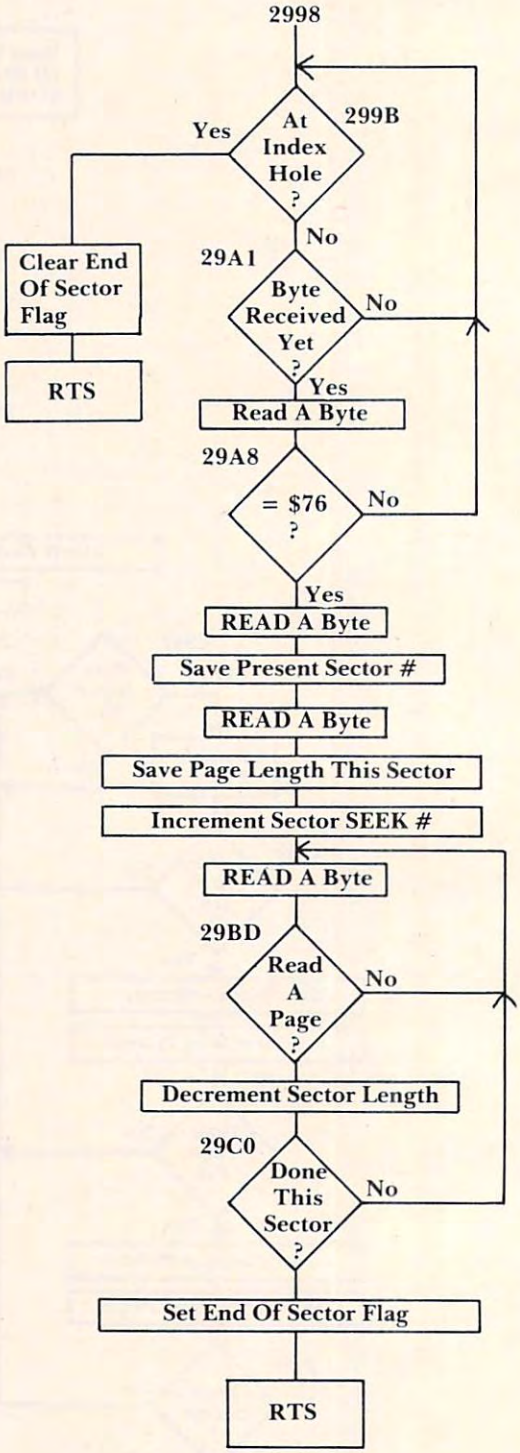

27D7
Set Pointer
27E1
Fetch Page Length Sector
27E4
= 0 ?
Yes
No
27E6
< $80 ?
No
Yes
27EE
> 13 ?
Yes
Error #B
No
27F5
At Track 0 ?
Yes
RTS
No
Set Preceding Sector Length To 1
27FF
Write Protect ?
Yes
Error #4
No
Set Write Retry # To 2
Set Compare Retry # To 4
Find Length And End Of Preceding Sector SUBR 28C4
Wait 800 x (Prededing Length) us.
Enable WRITE Function
Wait 185 µs.
Enable ERASE Function
Wait 800 x (Preceding Length) us.
Write $76
Write Sector #
Write Sector Page Length
Write Byte From Pointer
Increment Byte Pointer

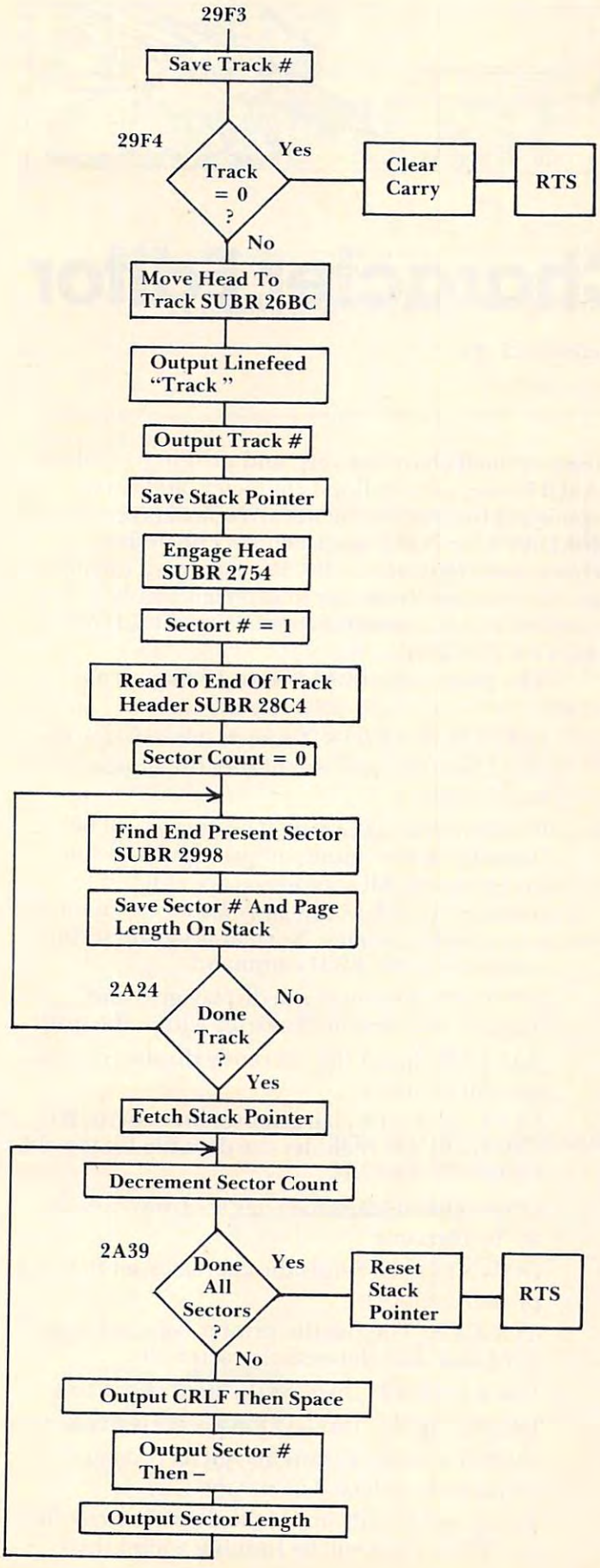

2847
Done Page ?
No
Yes
Increment Page Pointer
Decrement Page Length
284D
Done Sector ?
No
Yes
Write $47
Write $53
Compute 6 x (Sector Length)
Wait 100 x (Compute) µs.
Turn Off WRITE
Wait 525 µs.
Turn Off ERASE
CLEAR Read Flag
RESET Pointer
Compare Sector With Memory SUBR 2905
2885
Success ?
Yes
RTS
No
Decrement Compare Retry #
2889
Done Retries ?
No
Yes
Decrement Write Retry #
288D
Done Retries ?
Yes
Error #2
No
Reset Pointer

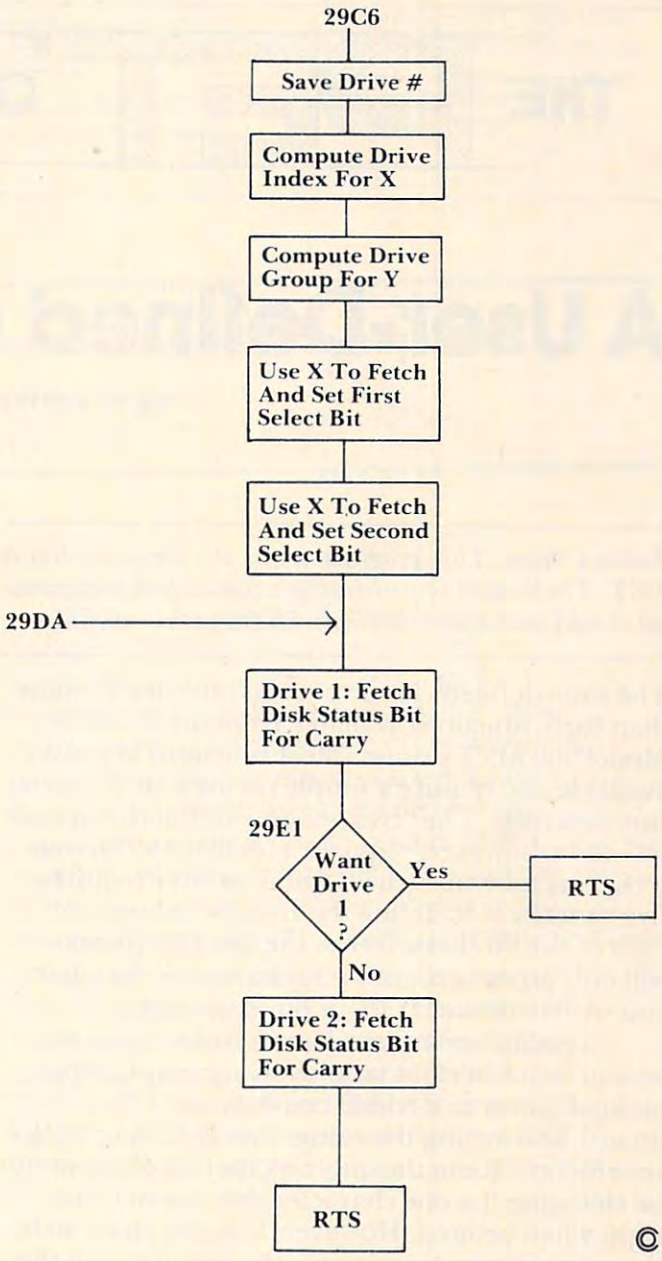

28C4
Find Index Hole
Read A Byte In 1 Rev. Of Disk (Or Error #9) SUBR 28B0
= $43 ?
No
28CC
Yes
Read A Byte In 1 Rev.
No
= $57 ?
28D3
Yes
Read A Byte
Track Number OK ?
No
Error #5
28DB
Yes
Read A Byte
Decrememt Target Sector Number
= $0
Yes
28E7
No
Set Sector SEEK # To 0
Find End Present Sector SUBR 2998
End Of A Sector ?
No
Error #A
28F0
Yes
No
Correct Sector ?
28F7
Yes
Sector Number Match Disk?
No
Error # A
28FB
Yes
Increment Target Sector Number
RTS


2905
Save READ Flag
Find End Preceding Sector
SUBR 28C4
Read Byte Within 1 Revolution
Of Disk (Or Error #9)
SUBR 28B0
290E
= $76 ?
No
Yes
Read A Byte
2916
= Sector #
No
Clear Success Flag
Fetch Read Flag
RTS
Yes
READ And SAVE Page Length Of Sector
Fetch READ Flag
2947
Byte Input Yet ?
Yes
No
2925
Read ?
Yes
292B
No
Byte Input Yet ?
Yes
No
Read A Byte
Read A Byte
2933
Parity OK ?
No
Yes
294F
Parity OK ?
No
Yes
Compare Byte With Memory
2937
Equal ?
No
Yes
Save Byte In Memory
Increment Byte Pointer
Increment Byte Pointer
2954
Done Page Yet ?
No
Yes
293A
No
Done Page Yet ?
Yes
Increment Page Pointer
Decrement Page Length
Increment Page Pointer
Decrement Page Length
2959
Done Sector ?
No
Yes
293F
No
Done Sector ?
Yes
Set Success Flag
RTS
Set Success Flag
RTS


295D
Get Pointer
2967
Set Head Move Retry # To 4
Set Read Retry # To 7
Set READ Flag
Read Sector To Memory Set Flag If Success SUBR 2905
2974
Flag Set ?
Yes
RTS
No
Get # Pages To Read
297D
Read These Yet ?
No
Increment # Pages Read
Decrement Page Pointer
Yes
Decrement Read Retry #
2981
No
Read Retry = 0 ?
Yes
No
Done Head Moves ?
Yes
Error #1
2991
Move Head Down One Track
Wait 12 ms. More
Move Head Up One Track
Wait 12 ms. More
Decrememt Head Move Retry #


2998
299B
At Index Hole ?
Yes
No
Clear End Of Sector Flag
RTS
29A1
Byte Received Yet ?
No
Yes
Read A Byte
29A8
= $76 ?
No
Yes
READ A Byte
Save Present Sector #
READ A Byte
Save Page Length This Sector
Increment Sector SEEK #
READ A Byte
29BD
Read A Page ?
No
Decrement Sector Length
29C0
Done This Sector ?
No
Set End Of Sector Flag
RTS


29F3
Save Track #
29F4
Track = 0 ?
Yes
Clear Carry
RTS
No
Move Head To Track SUBR 26BC
Output Linefeed "Track "
Output Track #
Save Stack Pointer
Engage Head SUBR 2754
Sectort # = 1
Read To End Of Track Header SUBR 28C4
Sector Count = 0
Find End Present Sector SUBR 2998
Save Sector # And Page Length On Stack
2A24
Done Track ?
No
Yes
Fetch Stack Pointer
Decrement Sector Count
2A39
Done All Sectors ?
Yes
Reset Stack Pointer
RTS
No
Output CRLF Then Space
Output Sector # Then –
Output Sector Length


29C6
Save Drive #
Compute Drive Index For X
Compute Drive Group For Y
Use X To Fetch And Set First Select Bit
Use X To Fetch And Set Second Select Bit
29DA
Drive 1: Fetch Disk Status Bit For Carry
29E1
Want Drive 1 ?
Yes
RTS
No
Drive 2: Fetch Disk Status Bit For Carry
RTS