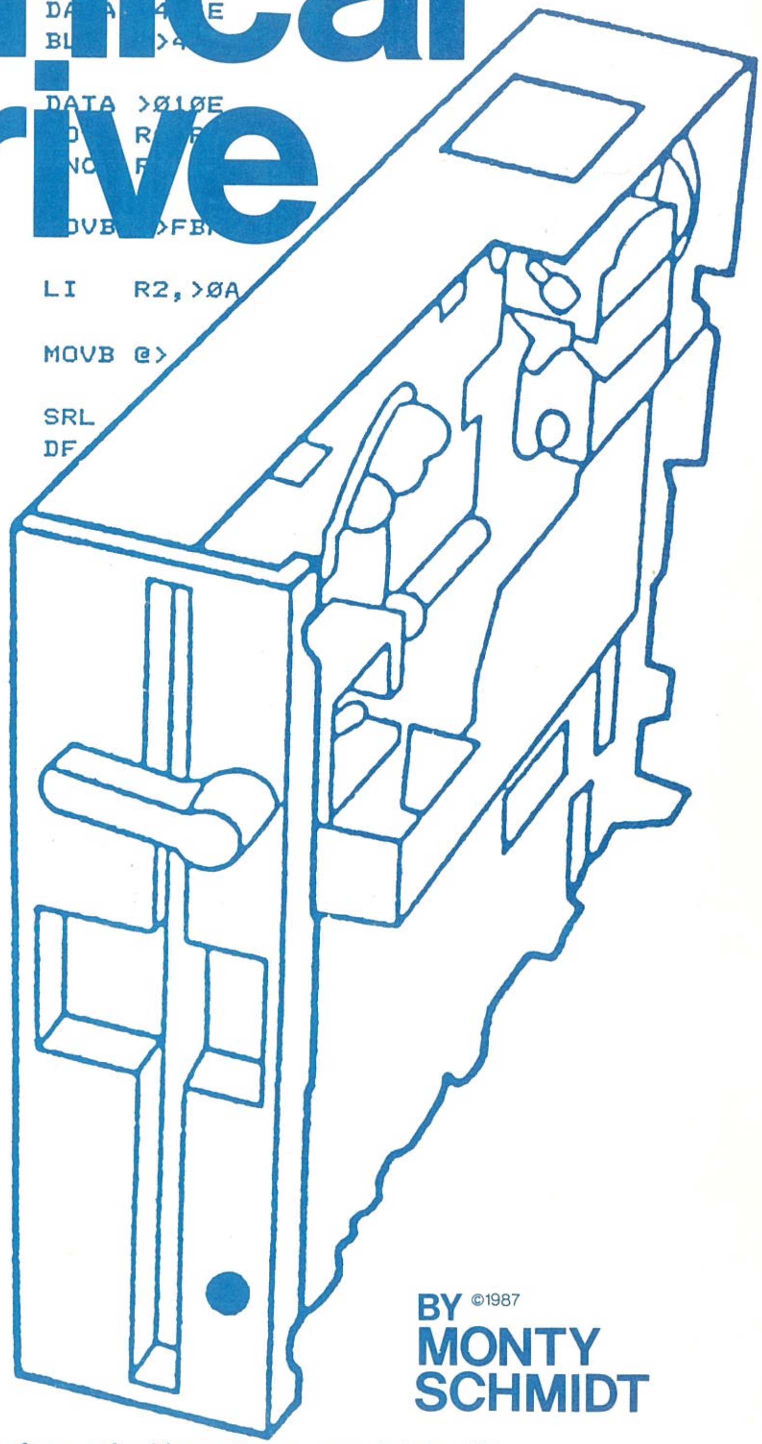
Technical Drive

```
5972    Ø6AØ    BL     @>4B76
5974    4B76
5976    C1ØØ    MOV    RØ,R4
5978    1328    JEQ    >59CA
[illegible]
4B7Ø
84      Ø1ØE    DATA   >Ø1ØE
86
88
        DØ      MOVB   >FB
598C    FBFE
598E    Ø2Ø2    LI     R2,>ØA
599Ø    ØAØØ
5992    DØEF    MOVB   @>
5994    FBFE
5996    Ø983    SRL
5998    Ø648    DF
599A    Ø429
599C    ØØ5A
599E    Ø1Ø2
59AØ    DØ2F
59A2    FBFE
59A4    C1CØ
59A6    Ø24Ø
59A8    Ø8ØØ
59AA    51CØ
59AC    Ø987
59AE    Ø587
59BØ    Ø287
59B2    ØØØ2
59B4    16Ø2
59B6    Ø227
59B8    ØØØ3
59BA    Ø287
59BC    ØØØ8
59BE    1AØ2
59CØ    Ø227
59C2    FF81
59C4    ØA4Ø

59C6    E1CØ
59C8    1ØØ4
59CA    Ø4C2
59CC    Ø4C6
59CE    Ø4C3
59DØ    Ø4C7
59D2    1ØØF
```



TI 99/4A SECRETS
DSRs
RS232
MINI MEMORY
CORCOMP CLOCK
DISK CONTROLLER

BY ©1987
MONTY SCHMIDT

```
** Record #Ø for of directory routine **

59D4    Ø225    AI     R5,>ØØFF
59D6    ØØFF
59D8    Ø6AØ    BL     @>4658
59DA    4658
59DC    4D4E    DATA   >4D4E
```

TECHNICAL DRIVE

Technical Drive

A new book by Monty Schmidt. Published in North America by Ryte Data.

Ryte Data
MILLENNIUM COMPUTERS
210 MOUNTAIN STREET,
HALIBURTON, ONTARIO K0M 1S0

To my parents, who've always believed in me and my dreams no matter how bizarre they were. I love you.

Thanks to all my friends who offered inspiration and support, Gundy, Eddie, Foxer, Dix, Sticks, Skarv, Bart, George, Steve, Emery, Mike-man, Barry, Charlie, and Phil(wherever he may be).

Special thanks to Russ Hirschy and Richard Horn who gave me my start in computers, Joe Dillie who helped me acquire my first computer, and Bruce Ryan who made this book possible.

Lastly, to three very special people who mean the world to me, Green Eyes, The Swami, and The Muscle Potato, "I finally did it guys!"

TABLE OF CONTENTS

INTRODUCTION

A lack of "hard core" information has been one of the biggest drawbacks of being a TI-99/4A user for many years. In order to learn anything about the code inside of the machine, people had to spend long hours disassembling and figuring out how it worked. What was found could then be traded to others who had done the same. Thus, bits and pieces of information became known, but a need still existed for more precise and accurate information. With this information the machine could be put to its full potential.

Detailed information on the internal operating system became available when the book "TI Intern" was released by Heiner Martin of West Germany. This book is an excellent reference, but a gap still exists in the information available. Peripherals for the TI-99/4A contain vast amounts of code and to really understand how they work and utilize them to their fullest potential one must know what is inside their ROM's. Hopefully this book will help fill that gap.

In the following pages you will find detailed, commented,

assembly language listings of the Corcomp 9900 clock card, the Texas Instruments RS232 card, and the Texas Instruments Disk Controller card. A map of VDP memory use by the Disk Controller, a review of the format of TI disks, and descriptions of the various low level access routines used by the cards can be found in the section on the Disk Controller card. Also included is commented source code of the DSRLNK utility used to communicate between the computer and it's peripherals. It is not a necessity to understand this routine but will give you a better understanding of peripherals as a whole if you do. Last but not least, you will find various support routines to access the cards at a "direct" level. These routines skip the DSRLNK routine and thus allow functions which perform faster and perhaps could not even be done without internal knowledge of the card.

I have made great effort to insure the accuracy of the knowledge contained in this book. This is not a guarantee, however, that everything is correct. Since I have had to "reverse engineer" this code, it is my own interpretation of what is actually happening. At any rate I think you will find this book informative and a useful tool in the development of new and better software!

Peripherals and Device Service Routines (DSR's), an overview.

When the TI-99/4A was designed, Texas Instruments did not know what peripherals would be added to it in the future. Thus, they needed a flexible way of interfacing new devices such as, RS232 interfaces, disk controllers, and clocks, without internal hardware modification of the computer. The Communications Register Unit(CRU) and the memory space from >4000 - >5FFF fulfill this requirement.

The CRU bits in the console are bits which can be written to, or read from, depending on how they are defined. They are used for such purposes as peripheral enable/disable, device control, bank switching of memory, and data transfer to and from peripherals. There are 4K of CRU bits and of these 4K, the last 1.9K (addresses >1000 - >1FFE) are used for peripherals attached to the I/O port. Each peripheral is assigned 128 CRU bits for its use. These definitions are listed in the table below.

Communications Register Unit	
>0000-03FE	CRU TMS 9901 space
>0404-10FE	Test bits, various third party cards
>1100-11FE	Disk Controller
>1200-12FE	Home Security
>1300-13FE	Primary RS232
>1400-14FE	Modem
>1500-15FE	Secondary RS232
>1600-16FE	Digital Cassette
>1700-17FE	Hex-bus (tm)
>1800-18FE	Thermal printer
>1900-19FE	EPROM programmer
>1A00-1AFE	Student Typing
>1B00-1BFE	Unassigned
>1C00-1CFE	Video Controller Card
>1D00-1DFE	IEEE 448 Controller Card, also used by the CorComp 9900 Clock and Triple-Tech cards.
>1E00-1EFE	Unassigned
>1F00-1FFE	P-Code Card

From the table it can be seen that many peripherals were planned for the 99/4A. They never appeared however, due to the fact that the

99/4A was pulled from the market before they were put into production.

The base CRU address of each peripheral is the memory enable bit. Setting the bit to 1 turns the device ROM/RAM on and resetting it to Ø turns it off. When the bit is set the device ROM will be "paged in" to the memory space >4ØØØ - >5FFF. At this time this area can be accessed just like any other ROM in the console.

For more information on CRU access and DSR's consult the Texas Instruments Editor/Assembler Manual.

This is a disassembled and commented version of the Mini-Memory DSRLNK routine. It should not vary much from the Editor/Assembler version of the DSRLNK routine other than the register locations of the workspaces used.

```
*****************************************************************
*                                                               *
*            Source code for Mini Memory DSRLNK routine         *
*                                                               *
*                Disassembled and commented 2/25/86             *
*                                                               *
*                          by Monty Schmidt                     *
*                                                               *
*****************************************************************


6000 AA00  DATA >AA01                  ** Data for DSRLNK routine

** VSBR vector

602C 7092  DATA >7092                  ** Workspace for VSBR routine
602E 6140  DATA >6140                  ** Entry address for VSBR routine

** DSRLNK vector

6038 7098  DATA >7098                  ** Workspace for DSRLNK routine
603A 61E4  DATA >603A                  ** Entry address for DSRLNK routine
6056 2000  DATA >2000                  ** Data for DSRLNK routine
6058 2E00  DATA >2E00                  ** Data for DSRLNK routine

** Mini Memory VSBR routine **

6140 06A0  BL   @>6172                 ** Set up VDPWA
6142 6172
6144 DB60  MOVB @>8800,@>0002(R13)     ** Move byte from VDP to MSB of R1 o
6146 8800                              ** old workspace
6148 0002
614A 0380  RTWP                        ** Return

** Subroutine to set VDPWA for VSBR routine **

6172 04C1  CLR  R1                     ** So routine knows its a read
6174 C09D  MOV  *R13,R2                ** Move R0 of old WS to R1
6176 D820  MOVB @>7097,@>8C02          ** Move LSbyte of R2 to VDP write
6178 7097                              ** address
617A 8C02
617C E081  SOC  R1,R2                  ** Set bit if its a write
617E D802  MOVB R2,@>8C02              ** Move MSbyte of R2 to write addrs.
6180 8C02
6182 C06D  MOV  @>0002(R13),R1         ** Put old R1 in new R1
6184 0002
6186 C0AD  MOV  @>0004(R13),R2         ** Put old R2 in new R2
6188 0004
618A 045B  RT                          ** Return

** Mini Memory DSRLNK routine **
```

```
61E4 C17E  MOV  *R14+,R5          ** Get DATA statement after DSRLNK
61E6 53E0  SZCB @>6056,R15        ** Clear out all bits in old Status
61E8 6056                         ** byte  except bit 2
61EA C020  MOV  @>8356,R0         ** Put pointer to name length in R0
61EC 8356
61EE C240  MOV  R0,R9             ** Duplicate it in R9
61F0 0229  AI   R9,>FFF8          ** Subtract 8 (Point to status)
61F2 FFF8
61F4 0420  BLWP @>602C            ** Get length of file name
61F6 602C
61F8 D0C1  MOVB R1,R3             ** Put in MSbyte of R3
61FA 0983  SRL  R3,8              ** Shift so number in 16 bits of R3
61FC 0704  SETO R4                ** Put -1 in R4
61FE 0202  LI   R2,>708A          ** Put Name Buffer Pointer in R2
6200 708A
6202 0580  INC  R0                ** Points to next byte in name
6204 0584  INC  R4                ** # bytes moved so far
6206 80C4  C    R4,R3             ** Done moving it yet?
6208 1306  JEQ  >6216             ** Yes, Leave loop
620A 0420  BLWP @>602C            ** Get byte of Name out of PAB
620C 602C
620E DC81  MOVB R1,*R2+           ** Put it in Name Buffer
6210 9801  CB   R1,@>6058         ** Is the Byte a "."
6212 6058
6214 16F6  JNE  >6202             ** No, Do it again
6216 C104  MOV  R4,R4             ** Yes, is it the first byte?
6218 1352  JEQ  >62BE             ** Yes, goto Error
621A 0284  CI   R4,>0007          ** Is the name greater than 8 bytes
621C 0007
621E 154F  JGT  >62BE             ** Yes, Goto Error
6220 04E0  CLR  @>83D0            ** Clear ROM Search Pointer
6222 83D0
6224 C804  MOV  R4,@>8354         ** Put length in  >8355
6226 8354
6228 C804  MOV  R4,@>7034         ** and in device name length of MM
622A 7034
622C 0584  INC  R4                ** Add 1 to R4
622E A804  A    R4,@>8356         ** Add it to Device Pointer
6230 8356
6232 C820  MOV  @>8356,@>7036     ** Move it to MM Device Pointer
6234 8356
6236 7036
6238 02E0  LWPI >83E0             ** Load GPLWS
623A 83E0
623C 04C1  CLR  R1                ** Clear out R1
                                  ** Time to check peripherals
623E 020C  LI   R12,>0F00         ** Set up for first one
6240 0F00
6242 C30C  MOV  R12,R12           ** Is R12=0?
6244 1301  JEQ  >6248             ** Yes?, jump over turning off ROM
6246 1E00  SBZ  >00               ** Turn off peripheral ROM
6248 022C  AI   R12,>0100         ** Point to next DSR ROM
624A 0100
624C 04E0  CLR  @>83D0            ** Clear ROM search pointer
624E 83D0
6250 028C  CI   R12,>2000         ** Are we at last ROM?
6252 2000
6254 1332  JEQ  >62BA             ** Yes?, then report error
6256 C80C  MOV  R12,@>83D0        ** Put ROM base address in ROM
6258 83D0                         ** search pointer
```

```
625A 1D00   SBO   >00
625C 0202   LI    R2,>4000
625E 4000
6260 9812   CB    *R2,@>6000
6262 6000
6264 16EE   JNE   >6242
6266 A0A0   A     @>70A2,R2
6268 70A2
626A 1003   JMP   >6272
626C C0A0   MOV   @>83D2,R2
626E 83D2
6270 1D00   SBO   >00
6272 C092   MOV   *R2,R2
6274 13E6   JEQ   >6242
6276 C802   MOV   R2,@>83D2
6278 83D2
627A 05C2   INCT  R2
627C C272   MOV   *R2+,R9
627E D160   MOVB  @>8355,R5
6280 8355
6282 1309   JEQ   >6296
6284 9C85   CB    R5,*R2+
6286 16F2   JNE   >626C
6288 0985   SRL   R5,8
628A 0206   LI    R6,>708A
628C 708A
628E 9CB6   CB    *R6+,*R2+
6290 16ED   JNE   >626C
6292 0605   DEC   R5
6294 16FC   JNE   >628E
6296 0581   INC   R1
6298 C801   MOV   R1,@>7038
629A 7038
629C C809   MOV   R9,@>7032
629E 7032
62A0 C80C   MOV   R12,@>7030
62A2 7030
62A4 0699   BL    *R9
62A6 10E2   JMP   >626C
62A8 1E00   SBZ   >00
62AA 02E0   LWPI  >7098
62AC 7098
62AE C009   MOV   R9,R0
62B0 0420   BLWP  @>602C
62B2 602C
62B4 09D1   SRL   R1,13
62B6 1604   JNE   >62C0
62B8 0380   RTWP
62BA 02E0   LWPI  >7098
62BC 7098
62BE 04C1   CLR   R1
62C0 06C1   SWPB  R1

62C2 D741   MOVB  R1,*R13
62C4 F3E0   SOCB  @>6056,R15
62C6 6056
62C8 0380   RTWP
```

```
** Turn on page in ROM bit
** Put Peripheral ROM pntr. in R2

** Check to see if first byte is >AA
** (Valid identifier)
** No?, then check another ROM
** Add R5(DATA statement)


** Move ROM Search Pointer 2 into R2

** Turn on Peripheral ROM
** Is pointer to next field 0?
** Yes?, then no more valid names
** Put R2 in Search Pointer 2 (Next
** valid device address)
** Point to entry address
** Put entry point in R9
** Get length of name looking for

** 0 length?, then jump
** Compare length with DSR length
** Not equal?, check next field name
** Put length in 16 bits
** Pointer to name buffer in R6

** Compare name
** Not equal?, then jump
** Done yet?
** No?, check another character
** Name matched so put 1 in R1
** Put it in Version # of DSR

** Put the entry point of the routine
** in peripheral ROM in >7032 of MM
** Put CRU address in CRU address of
** Mini-Memory
** Jump to the subroutine in DSR ROM
** Returned here?, ERROR! Jump!
** Turn off ROM
** Load DSRLNK Workspace

** Point to status byte
** Read it

** Check for ERROR
** Go report error
** Return without error
** Load DSRLNK workspace

** Clear MSB of R0 -Calling Workspace
** Move error into MSB of R0 of
** Calling Workspace

** Set bit 2 of old status byte

** Return with ERROR
```

The Corcomp 9900 Clock peripheral was introduced in 1985. It has a battery backed clock and allows the 99/4A owner to read and write the time and date. The functions are accessed by opening a file with the name "CLOCK" and writing and reading strings from the peripheral. The string format used is not very flexible and only allows access to the time and date at the same time. Time can only be accessed in a 24 hour mode, the date is accessed in the format 00/00/00, and the day of the week is referred to as a number.

By dis-assembling the DSR ROM we can learn how to access the clock directly and thus by some intelligent programming, add functions which were not included in the DSR. Functions could include setting or reading individual fields, a 12 hour mode for time, new formats for the date, and interrupt display of time.

Two subroutines follow the DSR which show how to write to and read from the clock directly. These routines can also be used to access the Corcomp Triple-Tech card since the same DSR is used in this peripheral.

```
*****************************************************************************
*                                                                           *
*              Source code for CORCOMP 9900 Clock card                      *
*                                                                           *
*                Dissassembled and commented 3/20/86                        *
*                                                                           *
*                          by Monty Schmidt                                 *
*                                                                           *
*****************************************************************************

4000 AA01   DATA >AA01                  ** Valid DSR identifier and version
4002 0000   DATA >0000                  ** Not used in DSR calls
4004 4038   DATA >4038                  ** Address of Power up link
4006 0000   DATA >0000                  ** Not used in DSR calls
4008 403E   DATA >403E                  ** DSRLNK address
400A 0000   DATA >0000                  ** Not used in DSR calls
400C 0000   DATA >0000                  ** INTLNK zero, no interrupt rtn.
400E 0000   DATA >0000                  ** Not used in DSR calls
4010 0460   DATA >0460                  ** ???? Test routine perhaps
4012 430C   DATA >434C                  ** Points to infinite loop Test rtn?
4014 2843   TEXT '(C) COPYRIGHT 1985'
4016 2920
4018 434F
401A 5059
401C 5249
401E 4748
4020 5420
4022 3139
4024 3835
4026 2042   TEXT ' BY CORCOMP, INC.'
4028 5920
402A 434F
402C 5243
402E 4F4D
4030 502C
4032 2049
4034 4E43
4036 2E00   EVEN
4038 0000   DATA >0000                  ** Linkage set to 0: only 1 power up
403A 4062   DATA >4062                  ** Entry point of power up routine
403C 0000   DATA >0000                  ** Name length set to 0
403E 0000   DATA >0000                  ** Linkage to next device field-none
4040 40A4   DATA >40A4                  ** Entry point of device
            BYTE >05                    ** Name length of device
4042 0543   TEXT 'CLOCK'                ** Device name
4044 4C4F
4046 434B
            BYTE >01                    ** ???????????
4048 0113   BYTE >13                    ** Number of characters to read
            BYTE >08                    ** Mask byte for write
404A 0804   BYTE >04                    ** Maximum allowable opcode
            BYTE >30                    ** ASCII offset for numbers
404C 30C0   BYTE >C0                    ** Enable byte
            BYTE >10                    ** Mask byte for write
404E 1040   BYTE >40                    ** ???????????
            BYTE >60                    ** ???????????
4050 60E0   BYTE >E0                    ** Mask for status and enable byte
4052 2C2F   TEXT ':/,'
```

```
4054 3A00  EVEN
4056 000A  DATA 10                      ** Constants for write routine
4058 0004  DATA 4                       **
405A 40F2  DATA >40F2                   ** Open routine address
405C 4080  DATA >4082                   ** Close routine address
405E 423A  DATA >423A                   ** Read routine address
4060 4134  DATA >4134                   ** Write routine address

** Power up routine **

4062 C18C  MOV  R12,R6                  ** Copy CRU address into R6
4064 045B  RT                           ** Return


** Error Codes **

4066 0201  LI   R1,>4000                ** Bad Open Attribute
4068 4000
406A 1008  JMP  $+>12           >407C
406C 0201  LI   R1,>6000                ** Illegal Operation
406E 6000
4070 1005  JMP  $+>0C           >407C
4072 0201  LI   R1,>8000                ** Out of Table or Buffer space
4074 8000
4076 1002  JMP  $+>06           >407C
 4078 0201  LI   R1,>C000                ** Attempt to read past end of file
407A C000
407C F901  SOCB R1,@>FF6B(R4)           ** Set Status Bit in DSR area
407E FF6B


** Close Opcode Routine **

4080 06A0  BL   @>4112                  ** Set up address to PAB Status Byt
4082 4112
4084 4001  DATA >4001
4086 DBE4  MOVB @>FF6B(R4),@>FFFE(R15) ** Write DSR Status Byte to VDP PA
4088 FF6B
408A FFFE
408C 06A0  BL   @>4112                  ** Set up address to PAB Char count
408E 4112
4090 4005  DATA >4005
4092 DBE4  MOVB @>FF6F(R4),@>FFFE(R15) ** Move DSR char cnt to PAB char c
4094 FF6F
4096 FFFE
4098 05E4  INCT @>FF86(R4)              ** INCT the return address
409A FF86
409C 04C8  CLR  R8
409E C2E4  MOV  @>FF86(R4),R11          ** Move return address into R11
40A0 FF86
40A2 045B  RT                           ** and go back!

** DSR Routine **

40A4 02A4  STWP R4                      ** Store GLPWSP pointer in R4
40A6 C90B  MOV  R11,@>FF86(R4)          ** Save return address in DSR area
40A8 FF86
40AA C184  MOV  R4,R6                   ** Move GPLWSP pointer into R6
40AC 0226  AI   R6,>FF78                ** Make it point to >8358 of DSR
40AE FF78                               ** area
```

```
40B0 0205  LI   R5,>0007                ** Clear out 7 words of the DSR area
40B2 0007
40B4 04F6  CLR  *R6+
40B6 0605  DEC  R5                      ** Done yet?
40B8 16FD  JNE  $->04           >40B4   ** Nope?, do it again
40BA 06A0  BL   @>4112                  ** Set VDPWA to beginning of PAB
40BC 4112
40BE 0000  DATA >0000
40C0 0205  LI   R5,>000A                ** We're going to get 10 bytes
40C2 000A
40C4 C184  MOV  R4,R6                   ** Put GPLWSP in R6
40C6 0226  AI   R6,>FF6A                ** Point to DSR area in Scratch Pad
40C8 FF6A
40CA DDAF  MOVB @>FBFE(R15),*R6+        ** Move byte from PAB to Scratch Pa
40CC FBFE
40CE 0605  DEC  R5                      ** Done Yet?
40D0 16FC  JNE  $->06           >40CA   ** nope?, Do it again
40D2 5920  SZCB @>4051,@>FF6B(R4)       ** Clear out bottom 5 bits of stat
40D4 4051                               ** byte. Set to defaults
40D6 FF6B
40D8 9824  CB   @>FF6A(R4),@>404B       ** Is this valid opcode?
40DA FF6A
40DC 404B
40DE 1202  JLE  $+>06           >40E4   ** Yes, then keep going
40E0 0460  B    @>406C                  ** Nope!, Return an error
40E2 406C
40E4 D164  MOVB @>FF6A(R4),R5           ** Put opcode in MSbyte of R5
40E6 FF6A
40E8 0985  SRL  R5,8                    ** Put it in low byte
40EA 0A15  SLA  R5,1                    ** Multiply it by 2
40EC C165  MOV  @>405A(R5),R5           ** Get address from opcode table
40EE 405A
40F0 0455  B    *R5                     ** Jump to the correct opcode rtn.

** Open Opcode Routine **

40F2 D0A4  MOVB @>FF6E(R4),R2           ** Move @ DSR Logical length into
40F4 FF6E
40F6 1609  JNE  $+>14           >410A   ** If its not 0 then don't alter it
40F8 06A0  BL   @>4112                  ** Set up VDP pointer to logical
40FA 4112                               ** length
40FC 4004  DATA >4004
40FE 0202  LI   R2,>1300                ** 19 Chars for length
4100 1300
4102 D902  MOVB R2,@>FF6E(R4)           ** Put it in DSR logical length byt
4104 FF6E
4106 DBC2  MOVB R2,@>FFFE(R15)          ** Put it in PAB logical length byt
4108 FFFE
410A D064  MOVB @>FF6B(R4),R1           ** Move status byte to R1
410C FF6B
410E 0460  B    @>4080                  ** Return
4110 4080


** This routine sets up the VDPWA for a read or write from the PAB.  A   **
** data statement is passed and used as follows: >40 in MSByte is a     **
** write, >00 is a read, LSByte is offset into PAB.                      **

4112 C064  MOV  @>FF76(R4),R1           ** Put device pointer in R1
4114 FF76
```

```
4116 6064  S    @>FF74(R4),R1          ** Subtract the device name length
4118 FF74
411A 0221  AI   R1,>FFF6               ** Subtract 10 (Point to start
411C FFF6                              ** of PAB)
411E A07B  A    *R11+,R1               ** Add data statement to R1
4120 D7E4  MOVB @>0003(R4),*R15        ** Move LSByte of R1 into VDPWA
4122 0003
4124 1000  NOP                         ** Wait
4126 D7C1  MOVB R1,*R15                ** Move MSByte of R1 into VDPWA
4128 045B  RT                          ** Go back

** This routine is a delay routine. The data statement passed is the     **
** number of times to execute the delay loop.                           **

412A C07B  MOV  *R11+,R1               ** Get number of times to loop
412C 1000  NOP                         ** Wait
412E 0601  DEC  R1                     ** Done yet?
4130 16FD  JNE  $->04          >412C   ** No?, loop again.
4132 045B  RT                          ** Go back

** Write Opcode Routine **

4134 C064  MOV  @>FF6C(R4),R1          ** Data Buffer Address Pointer
4136 FF6C                              ** into R1
4138 06A0  BL   @>4120                 ** Enter in middle of sub to set
413A 4120                              ** VDPWA to the data buffer in VDP
413C D1E4  MOVB @>FF6F(R4),R7          ** Move Chr count to MSbyte R7
413E FF6F
4140 0987  SRL  R7,8                   ** Put it in LSbyte R7
4142 0287  CI   R7,>0013               ** Compare to See if 19 bytes
4144 0013
                                       ** Where's the Jump if Greater?


** Get data from PAB for write

4146 D1AF  MOVB @>FBFE(R15),R6         ** Get day of week
4148 FBFE
414A 06C6  SWPB R6                     ** Switch em
414C D1AF  MOVB @>FBFE(R15),R6         ** Read another
414E FBFE
4150 91A0  CB   @>4052,R6              ** Check to see if it's a ','
4152 4052
4154 168E  JNE  $->E2          >4072   ** No? Then return an error
4156 06C6  SWPB R6                     ** Put day back in High Byte
4158 D906  MOVB R6,@>FF7E(R4)          ** Put it in DSR area >835E
415A FF7E
415C D92F  MOVB @>FBFE(R15),@>FF82(R4) ** Get first digit of month
415E FBFE
4160 FF82
4162 D92F  MOVB @>FBFE(R15),@>FF81(R4) ** Get second digit
4164 FBFE
4166 FF81
4168 D1AF  MOVB @>FBFE(R15),R6         ** Get rid of slash
416A FBFE
416C 1000  NOP                         ** wait
416E D92F  MOVB @>FBFE(R15),@>FF80(R4) ** Get first digit of day
4170 FBFE
4172 FF80
```

```
4174 D92F  MOVB @>FBFE(R15),@>FF7F(R4) ** Get second digit of day
4176 FBFE
4178 FF7F
417A D1AF  MOVB @>FBFE(R15),R6         ** Get rid of slash
417C FBFE
417E 1000  NOP                         ** wait
4180 D1AF  MOVB @>FBFE(R15),R6         ** Get first digit of year
4182 FBFE
4184 D906  MOVB R6,@>FF84(R4)          ** Move it into DSR area
4186 FF84
4188 0986  SRL  R6,8                   ** Move it into 16 bits
418A 39A0  MPY  @>4056,R6              ** Multiply by 10
418C 4056
418E D16F  MOVB @>FBFE(R15),R5         ** Get second digit of year
4190 FBFE
4192 D905  MOVB R5,@>FF83(R4)          ** Put it in DSR area
4194 FF83
4196 0985  SRL  R5,8                   ** Put it in 16 bits
4198 A1C5  A    R5,R7                  ** Add it to First digit * 10
419A 04C6  CLR  R6                     ** Set up for divide
419C 3DA0  DIV  @>4058,R6              ** Divide by 4
419E 4058
41A0 C1C7  MOV  R7,R7                  ** Check to see if a leap year
41A2 1603  JNE  $+>08           >41AA  ** If its not then jump
41A4 F920  SOCB @>404B,@>FF80(R4)      ** Set bit 5 of first digit of day
41A6 404B                              ** byte
41A8 FF80
41AA D1AF  MOVB @>FBFE(R15),R6         ** Get the comma
41AC FBFE
41AE 91A0  CB   @>4052,R6              ** Is it a comma?
41B0 4052
41B2 1302  JEQ  $+>06           >41B8  ** Yes?, Keep going
41B4 0460  B    @>4072                 ** Report the error
41B6 4072
41B8 D92F  MOVB @>FBFE(R15),@>FF7D(R4) ** Get first digit of hour
41BA FBFE
41BC FF7D
41BE F920  SOCB @>404A,@>FF7D(R4)      ** Set most significant bit of the
41C0 404A                              ** low nybble
41C2 FF7D
41C4 D92F  MOVB @>FBFE(R15),@>FF7C(R4) ** Get second digit of hour
41C6 FBFE
41C8 FF7C
41CA D1AF  MOVB @>FBFE(R15),R6         ** Get rid of colon
41CC FBFE
41CE 1000  NOP                         ** Wait
41D0 D92F  MOVB @>FBFE(R15),@>FF7B(R4) ** Get first digit of minutes
41D2 FBFE
41D4 FF7B
41D6 D92F  MOVB @>FBFE(R15),@>FF7A(R4) ** Get second digit of minutes
41D8 FBFE
41DA FF7A
41DC D1AF  MOVB @>FBFE(R15),R6         ** Get rid of colon
41DE FBFE
41E0 1000  NOP                         ** Wait
41E2 D92F  MOVB @>FBFE(R15),@>FF79(R4) ** Get first digit of seconds
41E4 FBFE
41E6 FF79
41E8 D92F  MOVB @>FBFE(R15),@>FF78(R4) ** Get second digit of seconds
41EA FBFE
```

```
41EC FF78
41EE 0206  LI   R6,>000D                ** 13 Bytes to write
41F0 000D
41F2 C144  MOV  R4,R5                   ** Duplicate GPLWS in R5
41F4 0225  AI   R5,>FF84                ** Point to >8364 in DSR area (bytes
41F6 FF84                               ** we want to put in clock)
41F8 0208  LI   R8,>5040                ** Point to enable byte
41FA 5040
41FC D620  MOVB @>404D,*R8              ** Enable the device
41FE 404D
4200 06A0  BL   @>412A                  ** Delay >10 times
4202 412A
4204 0010  DATA >0010
4206 D0D5  MOVB *R5,R3                  ** Move the byte into R3
4208 0243  ANDI R3,>0F00                ** Mask out bits except for Low
420A 0F00                               ** nybble of MSByte
420C C1C6  MOV  R6,R7                   ** Copy R6 into R7
420E 0607  DEC  R7                      ** Subtract one from R7
4210 0A87  SLA  R7,8                    ** Put it in MSByte
4212 F1E0  SOCB @>404D,R7               ** Set Bits 0 and 1 of MSByte
4214 404D
4216 D607  MOVB R7,*R8                  ** Move to enable byte
4218 D803  MOVB R3,@>5000               ** Move to data byte
421A 5000
421C 0247  ANDI R7,>4F00                ** Mask out bits 0,2 and 3 of MSByte
421E 4F00                               ** and all of LSByte
4220 D607  MOVB R7,*R8                  ** Move it to enable byte
4222 F1E0  SOCB @>404E,R7               ** Set bit 3 of MSByte
4224 404E
4226 D607  MOVB R7,*R8                  ** Move it to enable byte
4228 51E0  SZCB @>404E,R7               ** Clear all but bit 3 of MSByte
422A 404E
422C D607  MOVB R7,*R8                  ** Move it to enable byte
422E 0605  DEC  R5                      ** Subtract one from pointer to data
4230 0606  DEC  R6                      ** Done yet?
4232 16E9  JNE  $->2C           >4206   ** Nope?, Do it again
4234 D606  MOVB R6,*R8                  ** Disable write byte
4236 0460  B    @>4080                  ** Go back!
4238 4080

** Read Opcode Routine **

423A 0206  LI   R6,>000D                ** We're going to read 13 bytes
423C 000D
423E C144  MOV  R4,R5                   ** Put GPLWSP in R5
4240 0225  AI   R5,>FF84                ** Point to >8364 in DSR area
4242 FF84
4244 D920  MOVB @>4049,@>FF6F(R4)       ** Put 19 into chr count of DSR ar
4246 4049
4248 FF6F
424A 0208  LI   R8,>5040                ** Enable Byte address in R8
424C 5040
424E D620  MOVB @>404D,*R8              ** Put >C0 into Enable byte
4250 404D
4252 06A0  BL   @>412A                  ** Delay >10 times
4254 412A
4256 0010  DATA >0010
4258 D620  MOVB @>4051,*R8              ** Let it know we're going to read?
425A 4051
425C C1C6  MOV  R6,R7                   ** Copy R6 into R7
```

```
425E 0607  DEC  R7                        ** Subtract 1
4260 0A87  SLA  R7,8                      ** Put it in MSByte
4262 F1E0  SOCB @>4051,R7                 ** Set 3 MSBits
4264 4051
4266 D607  MOVB R7,*R8                    ** Tell it we want another byte
4268 1000  NOP                            ** Wait
426A 1000  NOP
426C D0E0  MOVB @>5000,R3                 ** Get numeric value from data byte
426E 5000
4270 F0E0  SOCB @>404C,R3                 ** Add ascii offset for numbers
4272 404C
4274 D543  MOVB R3,*R5                    ** Move it into DSR area
4276 0605  DEC  R5                        ** Dec DSR area address pointer
4278 0606  DEC  R6                        ** Done yet?
427A 16F0  JNE  $->1E           >425C     ** Nope?, do it again
427C D606  MOVB R6,*R8                    ** Put 0 into enable byte
427E C064  MOV  @>FF6C(R4),R1             ** Address to Pab Data buffer in R1
4280 FF6C
4282 06A0  BL   @>411E                    ** Set up VDPWA
4284 411E
4286 4000  DATA >4000                     ** Gonna write
4288 D1A4  MOVB @>FF7E(R4),R6             ** Get Day of week from DSR area
428A FF7E
428C DBC6  MOVB R6,@>FFFE(R15)            ** Put it in Pab Data buffer
428E FFFE
4290 DBE0  MOVB @>4052,@>FFFE(R15)        ** Put a ',' in there!
4292 4052
4294 FFFE
4296 DBE4  MOVB @>FF82(R4),@>FFFE(R15) ** Move number of month in there
4298 FF82
429A FFFE
429C DBE4  MOVB @>FF81(R4),@>FFFE(R15) ** Second digit of month
429E FF81
42A0 FFFE
42A2 DBE0  MOVB @>4053,@>FFFE(R15)        ** Put in a '/'
42A4 4053
42A6 FFFE
42A8 D1A4  MOVB @>FF80(R4),R6             ** Get the day number
42AA FF80
42AC 0246  ANDI R6,>3300                  ** Mask out anything greater than
42AE 3300                                 ** ASCII 3
42B0 DBC6  MOVB R6,@>FFFE(R15)            ** Put in PAB
42B2 FFFE
42B4 DBE4  MOVB @>FF7F(R4),@>FFFE(R15) ** Put in the second digit
42B6 FF7F
42B8 FFFE
42BA DBE0  MOVB @>4053,@>FFFE(R15)        ** Put in another '/'
42BC 4053
42BE FFFE
42C0 DBE4  MOVB @>FF84(R4),@>FFFE(R15) ** Put in the year
42C2 FF84
42C4 FFFE
42C6 DBE4  MOVB @>FF83(R4),@>FFFE(R15) ** Second digit of year
42C8 FF83
42CA FFFE
42CC DBE0  MOVB @>4052,@>FFFE(R15)        ** Time for another ','
42CE 4052
42D0 FFFE
42D2 D1A4  MOVB @>FF7D(R4),R6             ** Get top digit of hour
42D4 FF7D
```

```
42D6 0246  ANDI R6,>3300                ** Mask out anything greater than
42D8 3300                               ** ASCII 3
42DA DBC6  MOVB R6,@>FFFE(R15)          ** Put it in PAB
42DC FFFE
42DE DBE4  MOVB @>FF7C(R4),@>FFFE(R15) ** Put in second digit of hour
42E0 FF7C
42E2 FFFE
42E4 DBE0  MOVB @>4054,@>FFFE(R15)      ** Put in a ':'
42E6 4054
42E8 FFFE
42EA DBE4  MOVB @>FF7B(R4),@>FFFE(R15) ** Put in Minutes
42EC FF7B
42EE FFFE
42F0 DBE4  MOVB @>FF7A(R4),@>FFFE(R15) ** Put in second digit of minutes
42F2 FF7A
42F4 FFFE
42F6 DBE0  MOVB @>4054,@>FFFE(R15)      ** another ':'
42F8 4054
42FA FFFE
42FC DBE4  MOVB @>FF79(R4),@>FFFE(R15) ** Put in seconds
42FE FF79
4300 FFFE
4302 DBE4  MOVB @>FF78(R4),@>FFFE(R15) ** Put in second digit of seconds
4304 FF78
4306 FFFE
4308 0460  B    @>4080                  ** Go back
430A 4080
430C 02E0  LWPI >8300                   ** Load GPLWSP   Test routine
430E 8300                               ** perhaps?
4310 0460  B    @>430C                  ** Keep looping???
4312 430C
```

Clock Demo Routines:

```
** These two routines will allow you to read and write to the clock    **
** card.  It uses a 13 byte buffer space to pass the data.  For        **
** example; to write the date Monday, Ø1/Ø1/87 and time 12:ØØ:ØØ place **
** '87Ø1Ø1112ØØØØ' in the BUFFER space.                                **

CRUBAS EQU >1DØØ                    ** CRU base for Clock card
FLFLOP EQU >5Ø4Ø                    ** Enable byte address
DATABT EQU >5ØØØ                    ** Data byte address
ENABLE BYTE >CØ                     ** Control byte
MASK   BYTE >EØ                     ** Control byte
BUFFER BSS 13                       ** Buffer to read and write
OFFSET BYTE >3Ø                     ** Ascii offset for numbers
HRMASK BYTE >Ø8                     ** Control byte
WMASK1 BYTE >1Ø                     ** Control byte
       EVEN
TEN    DATA 1Ø                      ** Data to calculate leap year
FOUR   DATA 4                       ** Data to calculate leap year


** Low level access write routine for clock card **
** Data should be set up as follows **
**
** BUFFER       First digit of year
** BUFFER+1     Second digit of year
** BUFFER+2     First digit of month
** BUFFER+3     Second digit of month
** BUFFER+4     First digit of day
** BUFFER+5     Second digit of day
** BUFFER+6     Day of week
** BUFFER+7     First digit of hour
** BUFFER+8     Second digit of hour
** BUFFER+9     First digit of minute
** BUFFER+1Ø    Second digit of minute
** BUFFER+11    First digit of second
** BUFFER+12    Second digit of second
** All bytes should have >3Ø ascii offset added
** Uses R1,R3,R5,R6,R7,R8,R1Ø,R11,R12

WRITE  MOV R11,R1Ø                  ** Save return address
       LI R12,CRUBAS                ** Set CRU base for clock card
       SBO Ø                        ** Turn on the card
       MOVB @BUFFER,R6              ** Get First digit of year
       SRL R6,8                     ** Put it in 16 bits
       MPY @TEN,R6                  ** Multiply by 1Ø
       MOVB @BUFFER+1,R6            ** Get second digit of year
       SRL R6,8                     ** Put it in 16 bits
       A R6,R7                      ** Add them together
       CLR R6                       ** Set up for division
       DIV @FOUR,R6                 ** Divide it by four
       MOV R7,R7                    ** Is it a leap year?
       JNE NOTLEP                   ** No?, then jump
       SOCB @FOUR+1,@BUFFER+4       ** Set bit 5, 1st digit of day
NOTLEP SOCB @HRMASK,@BUFFER+7       ** Set the MSBit of low nybble
*                                   ** of first digit of hour
       LI R6,13                     ** 13 bytes to write
       LI R5,BUFFER                 ** Address of data to write
       LI R8,FLFLOP                 ** Enable byte address
       MOVB @ENABLE,*R8             ** Enable the card
```

```
       BL @DELAY                        ** Wait 16 loops
       DATA >0010
WRTLP  MOVB *R5+,R3                     ** Get byte to write
       ANDI R3,>0F00                    ** Subtract ascii offset
       MOV R6,R7                        ** Put number of chars in R7
       DEC R7                           ** Toggle LSBit
       SLA R7,8                         ** Put it in 16 bits
       SOCB @ENABLE,R7                  ** Set bits for enable
       MOVB R7,*R8                      ** Write it to enable address
       MOVB R3,@DATABT                  ** Write the byte to card
       ANDI R7,>4F00                    ** Mask bits 0,2,3 of MSByte
       MOVB R7,*R8                      ** Write to enable address
       SOCB @WMASK1,R7                  ** Set bit 3 of MSByte
       MOVB R7,*R8                      ** Write to enable address
       SZCB @WMASK1,R7                  ** Clear all but bit 3 of MSB
       MOVB R7,*R8                      ** Write to enable address
       DEC R6                           ** Done yet?
       JNE WRTLP                        ** No?, write another
       MOVB R6,*R8                      ** Write a >00 to enable
*                                       ** to tell it we're done
       SBZ 0                            ** Turn off the card
       MOV R10,R11                      ** Restore the return address
       RT                               ** Go back!

** Low level access Read routine for clock card **
** Data is returned in following locations **
**
** BUFFER       First digit of year
** BUFFER+1     Second digit of year
** BUFFER+2     First digit of month
** BUFFER+3     Second digit of month
** BUFFER+4     First digit of day
** BUFFER+5     Second digit of day
** BUFFER+6     Day of week
** BUFFER+7     First digit of hour
** BUFFER+8     Second digit of hour
** BUFFER+9     First digit of minute
** BUFFER+10    Second digit of minute
** BUFFER+11    First digit of second
** BUFFER+12    Second digit of second
** All bytes returned with >30 ascii offset added
**
** Uses R1,R3,R6,R7,R8,R12,R11,R10
READ   LI R12,CRUBAS                    ** Set CRU Base for clock card
       MOV R11,R10                      ** Save return address
       SBO 0                            ** Turn on card
       LI R5,BUFFER                     ** Buffer to put data from
*                                       ** card into
       LI R6,>000D                      ** 13 bytes to read
       LI R8,FLFLOP                     ** Location of enable
       MOVB @ENABLE,*R8                 ** Move enable data byte to
*                                       ** enable location
       BL @DELAY                        ** Delay 16 loops
       DATA >0010
       MOVB @MASK,*R8                   ** Move second enable data byte
RDLOOP MOV R6,R7                        ** Move # of bytes to read R7
       DEC R7                           ** Subtract 1 (toggles LSBit
*                                       ** for reading)
       SLA R7,8                         ** Put in high byte
       SOCB @MASK.R7                    ** Add mask byte to it
```

```
        MOVB R7,*R8                     ** Let it know we want a byte
        NOP                             ** Wait
        NOP                             ** Wait some more
          MOVB @DATABT,R3                 ** Get the data from data byte
        SOCB @OFFSET,R3                 ** Add ascii offset
        MOVB R3,*R5+                    ** Put it in the buffer
        DEC R6                          ** Done yet?
        JNE RDLOOP                      ** No?, do it again
        MOVB R6,*R8                     ** Move >ØØ to disable the
*                                       ** enable byte
        SBZ Ø                           ** Turn off the card
        MOVB @BUFFER+4,R6               ** Get first digit of day
        ANDI R6,>33ØØ                   ** Mask out anything greater
*                                       ** than ASCII 3
        MOVB R6,@BUFFER+4               ** Put it back in buffer
        MOVB @BUFFER+7,R6               ** Get first digit of hour
        ANDI R6,>33ØØ                   ** Mask out anything greater
*                                       ** than ASCII 3
        MOVB R6,@BUFFER+7               ** Put it back in buffer
        MOV R1Ø,R11                     ** Restore return address
        RT                              ** Go back!

** Subroutine DELAY
** Pass in delay count after BL instruction
** Counts down and returns

DELAY   MOV *R11+,R1                    ** Get # of loops to delay
DELLP   NOP                             ** Wait
        DEC R1                          ** Done yet?
        JNE DELLP                       ** No?, delay some more
        RT                              ** Go back
        END
```

The Texas Instruments RS232 card allows communication through three different ports. These include two RS232, and one 8-bit parallel port. These allow the TI-99/4A owner to use such equipment as modems, printers, and remote terminals. The commented listing of the Device Service Routine in this card is a little over 2K bytes long. You will notice in the DSR listing that up to four RS232, and two parallel ports can be accessed. These extra ports can be accessed by buying a second RS232 card and having it modified.

In the first part of the code there are two different frequency tables for initializing baud rate. This is due to the fact that there are two different clock speeds at which the computer was designed to run. These are 3 and 2.8 megahertz. To find out which frequency the card should use, the DSR checks the timing byte at location >0000 in the console ROM.

After the listing of the DSR are two sets of code which will allow you to access your T.I. RS232 card directly. One is for writing characters to the parallel (PIO) port and the other is for initialization of, input from, and output to, the RS232 ports. These routines access the card directly and do not require excessive VDP file buffer space.

```
*****************************************************************
*                                                               *
*                  Source code for TI RS232 card                *
*                                                               *
*                  Disassembled and commented 5/15/86           *
*                                                               *
*                  by Monty Schmidt                             *
*                                                               *
*****************************************************************

4000  AA01    BYTE >AA                    ** Valid DSR identifier
              BYTE >01                    ** Version 1
4002  0000    DATA >0000                  ** Not used in DSR calls
4004  4010    DATA >4010                  ** Power up routine link
4006  0000    DATA >0000                  ** Not used in DSR calls
4008  4016    DATA >4016                  ** DSR link
400A  0000    DATA >0000                  ** Not used in DSR calls
400C  406C    DATA >406C                  ** Interrupt routine link
400E  0000    DATA >0000                  ** Not used in DSR calls
4010  0000    DATA >0000                  ** Only one power up no more links
4012  40F4    DATA >40F4                  ** Entry of power up routine
4014  0000    BYTE >00                    ** Name length set to 0
              EVEN
4016  4020    DATA >4020                  ** Linkage to next device field
4018  416E    DATA >416E                  ** Entry point of RS232
401A  0552    BYTE 5                      ** Name length
401C  5332    TEXT 'RS232'                ** Name
401E  3332
4020  402C    DATA >402C                  ** Linkage to next device field
4022  416E    DATA >416E                  ** Entry point of RS232/1
4024  0752    BYTE 7                      ** Name length
4026  5332    TEXT 'RS232/1'              ** Name
4028  3332
402A  2F31
402C  4038    DATA >4038                  ** Linkage to next device field
402E  4174    DATA >4174                  ** Entry point of RS232/2
4030  0752    BYTE 7                      ** Name length
4032  5332    TEXT 'RS232/2'              ** Name
4034  3332
4036  2F32
4038  4040    DATA >4040                  ** Linkage to next device field
403A  415E    DATA >415E                  ** Entry point of PIO
403C  0350    BYTE 3                      ** Name length
403E  494F    TEXT 'PIO'                  ** Name
4040  404A    DATA >404A                  ** Linkage to next device field
4042  415E    DATA >415E                  ** Entry point of PIO/1
4044  0550    BYTE 5                      ** Name length
4046  494F    TEXT 'PIO/1'                ** Name
4048  2F31
404A  4054    DATA >4054                  ** Linkage to next device field
404C  4164    DATA >4164                  ** Entry point of PIO/2
404E  0550    BYTE 5                      ** Name length
4050  494F    TEXT 'PIO/2'                ** Name
4052  2F32
4054  4060    DATA >4060                  ** Linkage to next device field
4056  4180    DATA >4180                  ** Entry point of RS232/3
4058  0752    BYTE 7                      ** Name length
405A  5332    TEXT 'RS232/3'              ** Name
```

```
405C  3332
405E  2F33
4060  0000     DATA >0000                ** No more device fields
4062  417A     DATA >417A                ** Entry point of RS232/4
4064  0752     BYTE 7                    ** Name length
4066  5332     TEXT 'RS232/4'            ** Name
4068  3332
406A  2F34
406C  0000     DATA >0000                ** No more interrupt routines
406E  40D2     DATA >40D2                ** Entry point of interrup rtn.
4070  0000     BYTE >00                  ** Name length set to zero
               EVEN
4072  0800
4074  0303

** Option table **

4076  4543     TEXT 'EC'
4078  4512     DATA >4512                ** Entry point for .EC
407A  4352     TEXT 'CR'
407C  4518     DATA >4518                ** Entry point for .CR
407E  4C46     TEXT 'LF'
4080  451E     DATA >451E                ** Entry point for .LF
4082  4E55     TEXT 'NU'
4084  4524     DATA >4524                ** Entry point for .NU
4086  4441     TEXT 'DA'
4088  4570     DATA >4570                ** Entry point for .DA
408A  4241     TEXT 'BA'
408C  4536     DATA >4536                ** Entry point for .BA
408E  5041     TEXT 'PA'
4090  4540     DATA >4540                ** Entry point for .PA
4092  5457     TEXT 'TW'
4094  4596     DATA >4596                ** Entry point for .TW
4096  4348     TEXT 'CH'
4098  452A     DATA >452A                ** Entry point for .CH
409A  0000     DATA >0000                ** No more options

** Frequency Table **

409C  0028     DATA >0028                ** 2.8 Megahertz
409E  40B6     DATA >40B6                ** Offset
40A0  0030     DATA >0030                ** 3 Megahertz
40A2  40C4     DATA >40C4                ** Offset
40A4  0000     DATA >0000                ** No more frequencies

** Baud table **

40A6  006E     DATA 110                  ** 110 baud
40A8  012C     DATA 300                  ** 300 baud
40AA  0258     DATA 600                  ** 600 baud
40AC  04B0     DATA 1200                 ** 1200 baud
40AE  0960     DATA 2400                 ** 2400 baud
40B0  12C0     DATA 4800                 ** 4800 baud
40B2  2580     DATA 9600                 ** 9600 baud
40B4  0000     DATA >0000                ** End of baud table

** 2.8 MHz baud table **

40B6  8563     DATA >8563                ** 110 baud
40B8  8482     DATA >8482                ** 300 baud
```

```
40BA  8209     DATA >8209                    ** 600 baud
40BC  015B     DATA >015B                    ** 1200 baud
40BE  8082     DATA >8082                    ** 2400 baud
40C0  8041     DATA >8041                    ** 4800 baud
40C2  002B     DATA >002B                    ** 9600 baud

** 3 MHz baud table **

40C4  85AA     DATA >85AA                    ** 110 baud
40C6  849C     DATA >849C                    ** 300 baud
40C8  8271     DATA >8271                    ** 600 baud
40CA  01A1     DATA >01A1                    ** 1200 baud
40CC  809C     DATA >809C                    ** 2400 baud
40CE  804E     DATA >804E                    ** 4800 baud
40D0  8027     DATA >8027                    ** 9600 baud

** Interrupt routine **

40D2  02A4     STWP R4                       ** Put GPLWSP in R4 (Could be
                                             ** INTWSP in R4)
40D4  1D07     SBO  >07                      ** Turn on LED
40D6  C14B     MOV  R11,R5                   ** Save return address
40D8  C18C     MOV  R12,R6                   ** Save CRU address
40DA  022C     AI   R12,>0040                ** Set to RS232/1
40DC  0040
40DE  1F10     TB   >10                      ** Test for character received
40E0  1316     JEQ  $+>2E          >410E     ** Yes?, then jump
40E2  1F1F     TB   >1F                      ** Data set change,timer interrupt
                                             ** transmitter interrupt?
40E4  1306     JEQ  $+>0E          >40F2     ** Yes?, then jump to power up
40E6  022C     AI   R12,>0040                ** Set to RS232/2
40E8  0040
40EA  1F10     TB   >10                      ** Test for character received
40EC  1310     JEQ  $+>22          >410E     ** Yes?, then jump
40EE  1F1F     TB   >1F                      ** Data set change,timer interrupt
                                             ** transmitter interrupt?
40F0  1632     JNE  $+>66          >4156     ** No?, then jump
40F2  C306     MOV  R6,R12                   ** Restore CRU address

** Power up routine **

40F4  C18C     MOV  R12,R6                   ** Save CRU address
40F6  1D07     SBO  >07                      ** Turn on LED
40F8  1D02     SBO  >02                      ** Reset PIO
40FA  1E01     SBZ  >01                      **   "     "
40FC  022C     AI   R12,>0040                ** Set CRU to RS232/1
40FE  0040
4100  1D1F     SBO  >1F                      ** Reset RS232/1
4102  022C     AI   R12,>0040                ** Set CRU to RS232/2
4104  0040
4106  1D1F     SBO  >1F                      ** Reset RS232/2
4108  C306     MOV  R6,R12                   ** Restore CRU address
410A  1E07     SBZ  >07                      ** Turn off LED.
410C  045B     RT                            ** Go back

** Interrupt routine continued (Put char in VDP buffer)

410E  06A0     BL   @>4874                   ** Check for char in buffer
4110  4874
4112  1621     JNE  $+>44          >4156     ** No?, then return
```

```
4114  D064    MOVB @>FF24(R4),R1          ** Get the current count of chars
4116  FF24
4118  B060    AB   @>45F9,R1              ** Add 1 to count
411A  45F9
411C  9901    CB   R1,@>FF22(R4)          ** Too many chars?
411E  FF22
4120  1201    JLE  $+>04          >4124   ** No?, then jump
4122  04C1    CLR  R1
4124  9901    CB   R1,@>FF23(R4)
4126  FF23
4128  1306    JEQ  $+>0E          >4136
412A  3607    STCR R7,8                   ** Get char from port
412C  1F09    TB   >09                    ** Check RCVERR
412E  1607    JNE  $+>10          >413E   ** No?, then jump
4130  0207    LI   R7,>FF00               ** Return >FF code to buffer
4132  FF00
4134  1004    JMP  $+>0A          >413E
4136  0207    LI   R7,>FE00               ** Return >FE code to buffer
4138  FE00
413A  D064    MOVB @>FF24(R4),R1          ** New count = old count
413C  FF24
413E  D901    MOVB R1,@>FF24(R4)          ** Write the count back
4140  FF24
4142  0981    SRL  R1,8                   ** Add count to VDP Buffer ptr.
4144  A064    A    @>FF20(R4),R1
4146  FF20
4148  0241    ANDI R1,>3FFF               ** Mask out 2 MSBits(16K VDP RAM)
414A  3FFF
414C  06A0    BL   @>484E                 ** Set VDPWA to address in R1
414E  484E
4150  4000    DATA >4000                  ** Data for a VDP write
4152  DBC7    MOVB R7,@>FFFE(R15)         ** Write byte from port to buffer
4154  FFFE
4156  1D12    SBO  >12                    ** Set receive interrupt enable
4158  C306    MOV  R6,R12                 ** Restore CRU
415A  1E07    SBZ  >07                    ** Turn off LED
415C  0455    B    *R5                    ** Return

** PIO,PIO/1

415E  0206    LI   R6,>0001               ** Point to card 1
4160  0001
4162  1002    JMP  $+>06          >4168

** PIO/2

4164  0206    LI   R6,>0002               ** Point to card 2
4166  0002
4168  0703    SETO R3                     ** Let routines know it's PIO
416A  04C2    CLR  R2                     ** CRU offset is 0
416C  1011    JMP  $+>24          >4190

** RS232,RS232/1

416E  0206    LI   R6,>0001               ** Point to card 1
4170  0001
4172  1008    JMP  $+>12          >4184

** RS232/2
```

```
4174  0206    LI   R6,>0001                ** Point to card 1
4176  0001
4178  1008    JMP  $+>12          >418A

** RS232/3

417A  0206    LI   R6,>0002                ** Point to card 2
417C  0002
417E  1005    JMP  $+>0C          >418A

** RS232/4

4180  0206    LI   R6,>0002                ** Point to card 2
4182  0002
4184  0202    LI   R2,>0040                ** CRU offset ports 1,3
4186  0040
4188  1002    JMP  $+>06          >418E
418A  0202    LI   R2,>0080                ** CRU offset ports 2,4
418C  0080
418E  04C3    CLR  R3                      ** Let subs know its RS232
4190  02A4    STWP R4                      ** Save GPLWSP in R4
4192  C90B    MOV  R11,@>FF84(R4)          ** Store return address
4194  FF84
4196  8181    C    R1,R6                   ** Check which card
4198  1302    JEQ  $+>06          >419E    ** 1st card?, then jump
419A  0460    B    @>4480                  ** Jump to 2nd card routines
419C  4480
419E  C184    MOV  R4,R6                   ** Put GPLWSP in R6
41A0  0226    AI   R6,>FF78                ** Make it point to DSR use area
41A2  FF78                                 ** of scratch Pad
41A4  0205    LI   R5,>0006                ** Going to clear out 6 words
41A6  0006                                 ** there
41A8  04F6    CLR  *R6+                    ** Clear it
41AA  0605    DEC  R5                      ** Done yet?
41AC  16FD    JNE  $->04          >41A8    ** No?, do it again
41AE  1D07    SBO  >07                     ** Turn on LED
41B0  A302    A    R2,R12                  ** Add port offset to CRU base
41B2  06A0    BL   @>4842                  ** Set VDPWA to beginning of PAB
41B4  4842
41B6  0000    DATA >0000
41B8  0205    LI   R5,>000A                ** 10 bytes to move
41BA  000A
41BC  C184    MOV  R4,R6                   ** Move GPLWSP to R6
41BE  0226    AI   R6,>FF6A                ** Set to DSR area in Scratch Pad
41C0  FF6A
41C2  DDAF    MOVB @>FBFE(R15),*R6+        ** Move byte from VDP PAB to
41C4  FBFE                                 ** Scratch Pad
41C6  0605    DEC  R5                      ** Done yet?
41C8  16FC    JNE  $->06          >41C2    ** No?, do it again
41CA  5920    SZCB @>460B,@>FF6B(R4)       ** Clear 3 MSBits of Status byte
41CC  460B
41CE  FF6B
41D0  9920    CB   @>40B3,@>FF6A(R4)       ** Is the opcode >80?
41D2  40B3
41D4  FF6A
41D6  1606    JNE  $+>0E          >41E4    ** No?, then jump
41D8  F920    SOCB @>4132,@>FF7D(R4)       ** Set all bits at byte ********
41DA  4132
41DC  FF7D
41DE  5920    SZCB @>40B3,@>FF6A(R4)       ** Clear out top bit of opcode
```

```
41E0  40B3                                    ** Now it's an open opcode
41E2  FF6A
41E4  9824    CB   @>FF6A(R4),@>41A7          ** Check for opcode <= 6
41E6  FF6A
41E8  41A7
41EA  1202    JLE  $+>06            >41F0     ** Yes?, then jump
41EC  0460    B    @>4450                     ** Report illegal operation
41EE  4450
41F0  06A0    BL   @>4490                     ** Check to see if any options se
41F2  4490                                    ** in the PAB
41F4  D164    MOVB @>FF6A(R4),R5              ** Put opcode in MSByte of R5
41F6  FF6A
41F8  0985    SRL  R5,8                       ** Put it in 16 bits
41FA  0A15    SLA  R5,1                       ** Multiply it by 2
41FC  C165    MOV  @>4202(R5),R5              ** Get address from opcode table
41FE  4202
4200  0455    B    *R5                        ** Jump to it!

** Opcode table **

4202  4210    DATA >4210                      ** Open routine address
4204  4464    DATA >4464                      ** Close routine address
4206  4236    DATA >4236                      ** Read routine address
4208  42FA    DATA >42FA                      ** Write routine address
420A  4450    DATA >4450                      ** Restore/Rewind routine address
420C  4338    DATA >4338                      ** Load routine address
420E  43D2    DATA >43D2                      ** Save routine address

** Open opcode routine **

4210  D0A4    MOVB @>FF6E(R4),R2              ** Put DSR logical length in R2
4212  FF6E
4214  1609    JNE  $+>14            >4228     ** If it's not 0 don't change it
4216  06A0    BL   @>4842                     ** Set VDPWA to logical length
4218  4842                                    ** byte in PAB
421A  4004    DATA >4004
421C  0202    LI   R2,>5000                   ** 80 characters for length
421E  5000
4220  D902    MOVB R2,@>FF6E(R4)              ** Put it in scratch pad logical
4222  FF6E                                    ** length byte
4224  DBC2    MOVB R2,@>FFFE(R15)             ** Put it in VDP PAB logical
4226  FFFE                                    ** length byte
4228  D064    MOVB @>FF6B(R4),R1              ** Move status byte to R1
422A  FF6B
422C  2060    COC  @>43CA,R1                  ** Check to see if sequential fil
422E  43CA
4230  1663    JNE  $+>C8            >42F8     ** Yes?, jump
4232  0460    B    @>444A                     ** Report bad open attribute
4234  444A

** Read opcode routine **

4236  0743    ABS  R3                         ** R3 now equal to 1 or 0
4238  5920    SZCB @>4132,@>FF6F(R4)          ** Put 0 in Scratch pad Char cou
423A  4132
423C  FF6F
423E  D1E4    MOVB @>FF6E(R4),R7              ** Move logical record length
4240  FF6E                                    ** to MSByte of R7
4242  C264    MOV  @>FF6C(R4),R9              ** Put data buffer address in R9
4244  FF6C
```

```
4246  06A0    BL   @>4740                ** Check for internal file type
4248  4740
424A  1607    JNE  $+>10          >425A   ** No?, then jump
424C  06A0    BL   @>463A                ** Get a character from port
424E  463A
4250  9187    CB   R7,R6                 ** Is logical record length >=
                                         ** first char received
4252  1402    JHE  $+>06          >4258   ** Yes?, then jump
4254  0460    B    @>4456                ** Report error
4256  4456
4258  C1C6    MOV  R6,R7                 ** Move length of record to R7
425A  0987    SRL  R7,8                  ** Put it in 16 bits
425C  1348    JEQ  $+>92          >42EE   ** Is it's length equal to 0?
                                         ** Yes?, then jump
425E  06A0    BL   @>463A                ** Get a character from port
4260  463A
4262  06A0    BL   @>4740                ** Check for internal format
4264  4740
4266  133A    JEQ  $+>76          >42DC   ** Yes?, then jump
4268  D064    MOVB @>FF78(R4),R1          ** Check for Echo on
426A  FF78
426C  1307    JEQ  $+>10          >427C   ** Yes?, then jump
426E  06A0    BL   @>474A                ** Check for fixed length records
4270  474A
4272  1334    JEQ  $+>6A          >42DC   ** Yes?, then jump
4274  0286    CI   R6,>0D00              ** Check for carriage return char
4276  0D00
4278  1631    JNE  $+>64          >42DC   ** No?, then jump
427A  1039    JMP  $+>74          >42EE   ** Jump to exit of routine
427C  0286    CI   R6,>0D00              ** Check for carriage return char
427E  0D00
4280  1325    JEQ  $+>4C          >42CC   ** Yes?, then jump
4282  0286    CI   R6,>7F00              ** Check for ASCII 127 (DEL/RUB)
4284  7F00
4286  1312    JEQ  $+>26          >42AC   ** Yes?, then jump
4288  0286    CI   R6,>1200              ** Check for ASCII 18
428A  1200
428C  1625    JNE  $+>4C          >42D8   ** No?, then jump
428E  C064    MOV  @>FF6C(R4),R1         ** Load data buffer address in R1
4290  FF6C
4292  06A0    BL   @>4850                ** Point to it in VDP Data buffer
4294  4850
4296  06A0    BL   @>46EE                ** Write out CR,LF,Nulls
4298  46EE
429A  C089    MOV  R9,R2                 ** Put buffer point in R2
429C  60A4    S    @>FF6C(R4),R2         ** Subtract Buffer address
429E  FF6C                               ** (# of bytes so far)
42A0  1003    JMP  $+>08          >42A8
42A2  06A0    BL   @>47DE                ** Write byte from data buffer to
42A4  47DE                               ** the port
42A6  0602    DEC  R2                    ** Done yet?
42A8  16FC    JNE  $->06          >42A2   ** Jump if chars still in buffer
42AA  10D9    JMP  $->4C          >425E   ** Jump back to main loop
42AC  8264    C    @>FF6C(R4),R9         ** Are we at begining of data
42AE  FF6C                               ** buffer
42B0  13D6    JEQ  $->52          >425E   ** Yes?, jump back to main loop
42B2  0587    INC  R7                    ** Add 1 to length of record
42B4  0609    DEC  R9                    ** Subtract 1 from data buffer
                                         ** pointer
42B6  C049    MOV  R9,R1                 ** Put it in R1
```

```
42B8  06A0    BL   @>4850                ** Make VDPWA  Point to last byte
42BA  4850                                ** in PAB data buffer
42BC  06A0    BL   @>47DE                ** Echo received char back to por
42BE  47DE
42C0  0286    CI   R6,>0D00              ** Check for carriage return
42C2  0D00
42C4  16CC    JNE  $->66          >425E   ** No?, jump back to main loop
42C6  06A0    BL   @>4700                ** Write CR,LF,Nulls
42C8  4700
42CA  10C9    JMP  $->6C          >425E   ** Jump to main loop
42CC  06A0    BL   @>474A                ** Check for fixed length records
42CE  474A
42D0  1303    JEQ  $+>08          >42D8   ** Yes?, then jump
42D2  06A0    BL   @>46EE                ** Write CR,LF,Nulls
42D4  46EE
42D6  100B    JMP  $+>18          >42EE   ** Jump to end of routine
42D8  06A0    BL   @>47E6                ** Echo received char back to por
42DA  47E6
42DC  C049    MOV  R9,R1                 ** Move buffer pointer to R1
42DE  06A0    BL   @>484E                ** Set VDPWA to Data Buffer Pntr.
42E0  484E
42E2  4000    DATA >4000
42E4  DBC6    MOVB R6,@>FFFE(R15)        ** Write the char to the buffer
42E6  FFFE
42E8  0589    INC  R9                    ** Add 1 to the pointer
42EA  0607    DEC  R7                    ** Done with a record yet?
42EC  16B8    JNE  $->8E          >425E   ** No?, jump back to main loop
42EE  6264    S    @>FF6C(R4),R9         ** Put # of chars received in R9
42F0  FF6C
42F2  0A89    SLA  R9,8                  ** Put it in MSByte
42F4  D909    MOVB R9,@>FF6F(R4)         ** Move it to Scratch Pad Char
42F6  FF6F                               ** Count
42F8  101D    JMP  $+>3C          >4334   ** Return from card

** Write opcode routine **

42FA  C0C3    MOV  R3,R3                 ** Check for RS232
42FC  1301    JEQ  $+>04          >4300   ** Yes?, then jump
42FE  0703    SETO R3                    ** Set it for PIO
4300  C064    MOV  @>FF6C(R4),R1         ** Put data buffer address in R1
4302  FF6C
4304  06A0    BL   @>4850                ** Set VDPWA to data buffer
4306  4850                               ** address
4308  D1E4    MOVB @>FF6F(R4),R7         ** Move char count to MSByte R7
430A  FF6F
430C  06A0    BL   @>4740                ** Check for internal file type
430E  4740
4310  1603    JNE  $+>08          >4318   ** Nope?, then jump
4312  C187    MOV  R7,R6                 ** Put Char count in MSByte R6
4314  06A0    BL   @>47E6                ** Write char count to port
4316  47E6
4318  0987    SRL  R7,8                  ** Put it in 16 bits
431A  1304    JEQ  $+>0A          >4324   ** 0 chars?, then jump
431C  06A0    BL   @>47DE                ** Write char from data buffer
431E  47DE
4320  0607    DEC  R7                    ** done yet?
4322  16FC    JNE  $->06          >431C   ** no?, do it again
4324  06A0    BL   @>4740                ** Check for internal format
4326  4740
4328  1305    JEQ  $+>0C          >4334   ** Yes?, then jump
```

```
432A  06A0    BL   @>474A                  ** Check for variable length
432C  474A                                 ** records
432E  1302    JEQ  $+>06          >4334    ** No?, then jump
4330  06A0    BL   @>46EE                  ** Check options
4332  46EE
4334  0460    B    @>4464                  ** Return from card
4336  4464

** Load opcode routine **

4338  C024    MOV  @>FF70(R4),R0           ** Move max # of bytes that can
433A  FF70                                 ** be received in R0
433C  06A0    BL   @>47E4                  ** Send out synchronize byte(>16)
433E  47E4                                 ** to let know we're ready
4340  1600    DATA >1600          >4342    ** for transfer
4342  0205    LI   R5,>0007                ** Timeout in 7*>C01C loops
4344  0007
4346  0201    LI   R1,>C01C                ** Timeout value
4348  C01C
434A  06A0    BL   @>4870                  ** Character here yet?
434C  4870
434E  1307    JEQ  $+>10          >435E    ** Yes?, then jump
4350  0601    DEC  R1                      ** Done timing out yet?
4352  16FB    JNE  $->08          >434A    ** No?, do it again
4354  06A0    BL   @>4880                  ** Check for Function 4 key
4356  4880
4358  0605    DEC  R5                      ** Done timing out?
435A  16F5    JNE  $->14          >4346    ** Nope?, then jump
435C  10EF    JMP  $->20          >433C    ** Go send transfer byte again
435E  0709    SETO R9                      ** Preset partial CRC
4360  06A0    BL   @>45C6                  ** Go get char
4362  45C6
4364  C1C6    MOV  R6,R7                   ** Put received byte in R7
4366  06A0    BL   @>45C6                  ** Get another char
4368  45C6
436A  0986    SRL  R6,8                    ** Put it in low byte
436C  E1C6    SOC  R6,R7                   ** Put new byte in LSByte R7
436E  06A0    BL   @>45A0                  ** Get CRC from port into R8
4370  45A0
4372  06A0    BL   @>46B4                  ** Display # of records on screen
4374  46B4
4376  8248    C    R8,R9                   ** Compare Calculated CRC with
                                           ** received CRC
4378  1304    JEQ  $+>0A          >4382    ** Equal?, then jump
437A  06A0    BL   @>47E4                  ** Send a Negative Acknowledge
437C  47E4                                 ** (>15) to the port
437E  1500    DATA >1500          >4380
4380  10EE    JMP  $->22          >435E    ** Go get it again!
4382  81C0    C    R0,R7                   ** Is maximum bytes available less
                                           ** than the number to be sent?
4384  1A68    JL   $+>D2          >4456    ** Yes?, return error
4386  06A0    BL   @>47E4                  ** Send an Acknowledge byte(>06)
4388  47E4
438A  0600    DATA >0600
438C  06A0    BL   @>4686                  ** Find how many bytes for curren
438E  4686                                 ** block
4390  0709    SETO R9                      ** Set initial CRC word
4392  C04A    MOV  R10,R1                  ** Put PAB data buff address in R1
4394  06A0    BL   @>484E                  ** Point VDPWA to data buffer
4396  484E
```

```
4398  4000    DATA >4000
439A  06A0    BL   @>45C6              ** Get character from port
439C  45C6
439E  DBC6    MOVB R6,@>FFFE(R15)      ** Write it into VDP
43A0  FFFE
43A2  0607    DEC  R7                  ** Subtract 1 from # of bytes to
                                       ** receive
43A4  16FA    JNE  $->0A         >439A ** Done yet?, No?, go get another
43A6  06A0    BL   @>45A0              ** Get CRC from port
43A8  45A0
43AA  C0C3    MOV  R3,R3               ** Is this RS232?
43AC  1302    JEQ  $+>06         >43B2 ** Yes? then jump
43AE  06A0    BL   @>48A2              ** Delay
43B0  48A2
43B2  8209    C    R9,R8               ** Compare calculated CRC to CRC
                                       ** received from port?
43B4  1306    JEQ  $+>0E         >43C2 ** Are they equal?  then jump
43B6  C1E4    MOV  @>FF80(R4),R7       ** Restore # bytes for block in R
43B8  FF80
43BA  06A0    BL   @>47E4              ** Send a Negative Acknowledge
43BC  47E4
43BE  1500    DATA >1500         >43C0
43C0  10E7    JMP  $->30         >4390 ** Go do it over!
43C2  06A0    BL   @>47E4              ** Send an Acknowledge byte
43C4  47E4
43C6  0600    DATA >0600
43C8  022A    AI   R10,256             ** Add 256 to buffer address
43CA  0100
43CC  C1E4    MOV  @>FF7E(R4),R7       ** Put number of bytes left to
43CE  FF7E                             ** receive in R7
43D0  10DD    JMP  $->44         >438C ** Go get another block
** Save Opcode Routine **

43D2  C04A    MOV  R10,R1              ** Move data buff address to R1
43D4  06A0    BL   @>4850              ** Set VDPWA to buffer address
43D6  4850
43D8  06A0    BL   @>463A              ** Get a char from the port
43DA  463A
43DC  0286    CI   R6,>1600            ** Is it Synchronize byte for
43DE  1600                             ** start of transfer?
43E0  16FB    JNE  $->08         >43D8 ** No?, go check again
43E2  0709    SETO R9                  ** Initial word for CRC
43E4  C0C3    MOV  R3,R3               ** Is this RS232?
43E6  1302    JEQ  $+>06         >43EC ** Yes?, then jump
43E8  06A0    BL   @>48A2              ** Delay
43EA  48A2
43EC  C1A4    MOV  @>FF70(R4),R6       ** Put length of data in R6
43EE  FF70
43F0  06A0    BL   @>45D0              ** Write MSByte to port
43F2  45D0
43F4  06C6    SWPB R6                  ** Switch em!
43F6  06A0    BL   @>45D0              ** Write LSByte to port
43F8  45D0
43FA  06A0    BL   @>45B4              ** Write CRC to port
43FC  45B4
43FE  06A0    BL   @>463A              ** Get a char from the port
4400  463A
4402  0286    CI   R6,>0600            ** Is it Acknowledge byte?
4404  0600
4406  16ED    JNE  $->24         >43E2 ** No?, go do it over!
```

```
4408  C1E4    MOV  @>FF70(R4),R7          ** Move length of file to R7
440A  FF70
440C  06A0    BL   @>4686                 ** Calculate # bytes to send this
440E  4686                                ** block
4410  0709    SETO R9                     ** Initial CRC Value
4412  C04A    MOV  R10,R1                 ** Move Data buffer address to R1
4414  06A0    BL   @>4850                 ** Set VDPWA to buffer
4416  4850
4418  D1AF    MOVB @>FBFE(R15),R6         ** Get byte from buffer
441A  FBFE
441C  06A0    BL   @>45D0                 ** Write the byte to the port
441E  45D0
4420  0607    DEC  R7                     ** Done sending block yet?
4422  16FA    JNE  $->0A          >4418   ** No?, go send another
4424  06A0    BL   @>45B4                 ** Write CRC to the port
4426  45B4
4428  06A0    BL   @>463A                 ** Get char from port
442A  463A
442C  0286    CI   R6,>0600               ** Is it Acknowledge byte?
442E  0600
4430  1307    JEQ  $+>10          >4440   ** Yes?, then jump
4432  C0C3    MOV  R3,R3                  ** Are we using RS232?
4434  1302    JEQ  $+>06          >443A   ** Yes?, then jump
4436  06A0    BL   @>48A2                 ** Delay
4438  48A2
443A  C1E4    MOV  @>FF80(R4),R7          ** Restore # bytes to transfer
443C  FF80                                ** this block
443E  10E8    JMP  $->2E          >4410   ** Go do it all over again!
4440  022A    AI   R10,>0100              ** Add 256 to buffer pointer
4442  0100
4444  C1E4    MOV  @>FF7E(R4),R7          ** Get number of bytes left to
4446  FF7E                                ** transfer
4448  10E1    JMP  $->3C          >440C   ** Go transfer another block!

** Error codes **

444A  0201    LI   R1,>4000               ** Bad open attribute
444C  4000
444E  1008    JMP  $+>12          >4460
4450  0201    LI   R1,>6000               ** Illegal operation
4452  6000
4454  1005    JMP  $+>0C          >4460
4456  0201    LI   R1,>8000               ** Out of table or buffer space
4458  8000
445A  1002    JMP  $+>06          >4460
445C  0201    LI   R1,>C000               ** Attempt to read past end of
445E  C000                                ** file
4460  F901    SOCB R1,@>FF6B(R4)          ** Set status bit in Scratch pad
4462  FF6B
4464  06A0    BL   @>4842                 ** Set VDPWA to status byte in PA
4466  4842
4468  4001    DATA >4001
446A  DBE4    MOVB @>FF6B(R4),@>FFFE(R15)
446C  FF6B                                ** Write Scratch pad Status to
446E  FFFE                                ** VDP PAB status
4470  06A0    BL   @>4842                 ** Set VDPWA to chr count
4472  4842
4474  4005    DATA >4005
4476  DBE4    MOVB @>FF6F(R4),@>FFFE(R1   ** Write scratch pad chr count t
4478  FF6F                                ** VDP PAB chr count
```

```
447A  FFFE
447C  05E4    INCT @>FF84(R4)              ** Fix the return address
447E  FF84
4480  024C    ANDI R12,>FF00              ** Return correct CRU address
4482  FF00
4484  C2E4    MOV  @>FF84(R4),R11         ** Restore return address
4486  FF84
4488  1D02    SBO  >02                    ** Reset PIO
448A  1E01    SBZ  >01
448C  1E07    SBZ  >07                    ** Turn off LED
448E  045B    RT                          ** Return from card
4490  C90B    MOV  R11,@>FF86(R4)         ** Save return address
4492  FF86
4494  06A0    BL   @>4730                 ** Check to see if Save or Load
4496  4730                                ** opcode
4498  1305    JEQ  $+>0C           >44A4  ** Yes?, then jump
449A  0208    LI   R8,>4076               ** Point to start of option table
449C  4076
449E  0201    LI   R1,>B200               ** Default options for Open rtn
44A0  B200
44A2  1004    JMP  $+>0A           >44AC
44A4  0208    LI   R8,>408A               ** Point to middle of option table
44A6  408A                                ** Options before are not allowed
                                          ** for save and load
44A8  0201    LI   R1,>8300               ** Default options for Save, Load
44AA  8300
44AC  0205    LI   R5,>012C               ** Default baud rate
44AE  012C
44B0  C244    MOV  R4,R9                  ** Move GPLWSP to R9
44B2  0229    AI   R9,>FFFA               ** Subtract 6 (point to >83DA,
44B4  FFFA                                ** option byte)
44B6  D641    MOVB R1,*R9                 ** Put in default options
44B8  06A0    BL   @>45F4                 ** Go set default baud rate
44BA  45F4
44BC  D024    MOVB @>FF73(R4),R0          ** Move PAB option length to R0
44BE  FF73
44C0  0980    SRL  R0,8                   ** Put it in 16 bits
44C2  6024    S    @>FF74(R4),R0          ** Subtract device length
44C4  FF74
44C6  1217    JLE  $+>30           >44F6  ** Go set options
44C8  C064    MOV  @>FF76(R4),R1
44CA  FF76
44CC  06A0    BL   @>4850                 ** Set VDPWA to value in R1
44CE  4850
44D0  0706    SETO R6                     ** Initialize byte for search sub.

44D2  C000    MOV  R0,R0                  ** Length of options =0?
44D4  1310    JEQ  $+>22           >44F6  ** Yes?, then no options so jump
44D6  06A0    BL   @>4798                 ** Find '.'
44D8  4798
44DA  2E00    DATA >2E00
44DC  130C    JEQ  $+>1A           >44F6  ** No more options? then jump
44DE  C1C8    MOV  R8,R7                  ** Move option table pointer to R7
44E0  0986    SRL  R6,8                   ** Put char in low byte
44E2  D1AF    MOVB @>FBFE(R15),R6         ** Get second byte of option
44E4  FBFE
44E6  0600    DEC  R0                     ** Subtract 1 from the length
44E8  06C6    SWPB R6                     ** Switch em
44EA  C077    MOV  *R7+,R1                ** Get option from option table
44EC  1311    JEQ  $+>24           >4510  ** Passed last option?, then jump
44EE  C0B7    MOV  *R7+,R2                ** Put entry point of rtn. in R2
```

```
44FØ  8181    C     R1,R6                      ** Is it one we're looking for?
44F2  16FB    JNE   $->Ø8           >44EA     ** No?, go check for another
44F4  Ø452    B     *R2                        ** Branch to option routine
44F6  DØ64    MOVB  @>FF6A(R4),R1              ** Put opcode in MSByte of R1
44F8  FF6A
44FA  1307    JEQ   $+>1Ø            >450A     ** Is it open opcode?
                                               ** Yes?, then jump
44FC  Ø6AØ    BL    @>473Ø                     ** Check to see if Load or Save
44FE  473Ø                                     ** opcode
4500  1606    JNE   $+>ØE            >450E     ** No?, then go return
4502  Ø6AØ    BL    @>46B2                     ** Go write # of records on scree
4504  46B2                                     ** (255)?
4506  C2A4    MOV   @>FF6C(R4),R1Ø             ** Move PAB data buffer address
4508  FF6C                                     ** into R1Ø
450A  Ø6AØ    BL    @>4822                     ** Write the options to the
450C  4822                                     ** control registers
450E  1066    JMP   $+>CE            >45DC     ** Go return
4510  109C    JMP   $->C6            >444A     ** Return bad open attribute
4512  Ø2Ø1    LI    R1,>FF78                   ** .EC ** Load offset into
4514  FF78                                     ** Scratch Pad
4516  100B    JMP   $+>18            >452E
4518  Ø2Ø1    LI    R1,>FF79                   ** .CR ** Load offset into
451A  FF79                                     ** Scratch Pad
451C  1008    JMP   $+>12            >452E
451E  Ø2Ø1    LI    R1,>FF7A                   ** .LF ** Load offset into
4520  FF7A                                     ** Scratch Pad
4522  1005    JMP   $+>ØC            >452E
4524  Ø2Ø1    LI    R1,>FF7C                   ** .NU ** Load offset into
4526  FF7C                                     ** Scratch Pad
4528  1002    JMP   $+>Ø6            >452E
452A  Ø2Ø1    LI    R1,>FF7B                   ** .CH ** Load offset into
452C  FF7B                                     ** Scratch Pad
452E  A044    A     R4,R1                      ** Add offset to GPLWSP
4530  F46Ø    SOCB  @>4132,*R1                 ** Put >FF in the byte
4532  4132
4534  1034    JMP   $+>6A            >459E
4536  CØC3    MOV   R3,R3                      ** .BA ** Check to see if
                                               ** PIO or RS232
4538  1632    JNE   $+>66            >459E     ** PIO?, then jump
453A  Ø6AØ    BL    @>45E2                     ** Go calculate baud rate
453C  45E2
453E  102F    JMP   $+>6Ø            >459E
4540  CØC3    MOV   R3,R3                      ** .PA ** Check to see if
                                               ** PIO or RS232
4542  162D    JNE   $+>5C            >459E     ** PIO?, then jump
4544  Ø6AØ    BL    @>4798                     ** Go find = sign, return char
4546  4798                                     ** after it in MSB R6
4548  3DØØ    DATA  >3DØØ
454A  13E2    JEQ   $->3A            >4510     ** Not found?, then error
454C  566Ø    SZCB  @>4ØA1,*R9                 ** Clear out parity bits
454E  4ØA1
4550  Ø986    SRL   R6,8                       ** Put char in LSByte
4552  Ø286    CI    R6,>ØØ4E                   ** Is it N?
4554  ØØ4E
4556  1323    JEQ   $+>48            >459E     ** Yes?, then go return
4558  Ø286    CI    R6,>ØØ45                   ** Is it E?
455A  ØØ45
455C  1306    JEQ   $+>ØE            >456A     ** Yes?, go set even parity
455E  Ø286    CI    R6,>ØØ4F                   ** Is it O?
4560  ØØ4F
```

```
4562  16D6    JNE  $->52          >4510  ** No?, then return bad attribute
4564  F660    SOCB @>40A1,*R9             ** Set bits for odd parity
4566  40A1
4568  101A    JMP  $+>36          >459E
456A  F660    SOCB @>422C,*R9             ** Set bits for even parity
456C  422C
456E  1017    JMP  $+>30          >459E
4570  C0C3    MOV  R3,R3                  ** .DA ** Check to see if
                                          ** PIO or RS232
4572  1615    JNE  $+>2C          >459E   ** PIO?, then jump
4574  06A0    BL   @>4798                 ** Find = sign
4576  4798
4578  3D00    DATA >3D00
457A  13CA    JEQ  $->6A          >4510   ** Not found? go report error
457C  06A0    BL   @>4754                 ** Convert char to numeric value
457E  4754
4580  F660    SOCB @>4074,*R9             ** Set 2 LSBits for default,(8
4582  4074                                ** data bits)
4584  0225    AI   R5,-7                  ** Is data bits option 7?
4586  FFF9
4588  1303    JEQ  $+>08          >4590   ** Yes?, jump
458A  0605    DEC  R5                     ** Is data bits option 8?
458C  16C1    JNE  $->7C          >4510   ** No?, then return error
458E  1002    JMP  $+>06          >4594
4590  5660    SZCB @>45F9,*R9             ** Clear LSBit for 7 data bits
4592  45F9
4594  1004    JMP  $+>0A          >459E
4596  5660    SZCB @>42DC,*R9             ** .TW ** Clear 2 MSBits of optio
4598  42DC                                ** byte
459A  F660    SOCB @>4004,*R9             ** Set bit 1 of option byte
459C  4004
459E  1099    JMP  $->CC          >44D2   ** Go check more options

** Subroutine to get CRC word from the port **

45A0  C90B    MOV  R11,@>FF86(R4)         ** Save return address
45A2  FF86
45A4  06A0    BL   @>463A                 ** Get char, check errors
45A6  463A
45A8  C206    MOV  R6,R8                  ** Put it in R8
45AA  06A0    BL   @>463A                 ** Get char, check errors
45AC  463A
45AE  06C6    SWPB R6                     ** Switch bytes in R6
45B0  E206    SOC  R6,R8                  ** Put 2nd byte in R8
45B2  1014    JMP  $+>2A          >45DC

** Subroutine to write the CRC word to the port **

45B4  C90B    MOV  R11,@>FF86(R4)         ** Save return address
45B6  FF86
45B8  C189    MOV  R9,R6
45BA  06A0    BL   @>47E6                 ** Write char MSByte R6 to port
45BC  47E6
45BE  06C6    SWPB R6                     ** Switch em
45C0  06A0    BL   @>47E6                 ** Write char MSByte R6 to port
45C2  47E6
45C4  100B    JMP  $+>18          >45DC   ** Go restore and return
45C6  C90B    MOV  R11,@>FF86(R4)         ** Save return address
45C8  FF86
45CA  06A0    BL   @>463A                 ** Get char, check errors
```

```
45CC  463A
45CE  1004    JMP   $+>0A          >45D8  ** Go restore and go back
45D0  C90B    MOV   R11,@>FF86(R4)        ** Sae return address
45D2  FF86
45D4  06A0    BL    @>47E6                ** Write char MSByte R6 to port
45D6  47E6
45D8  06A0    BL    @>47C0                ** Calculate CRC
45DA  47C0
45DC  C2E4    MOV   @>FF86(R4),R11        ** Restore return address
45DE  FF86
45E0  045B    RT                          ** Go back
45E2  C90B    MOV   R11,@>FF88(R4)        ** Store return address
45E4  FF88
45E6  06A0    BL    @>4798                ** Look for = Sign
45E8  4798
45EA  3D00    DATA  >3D00
45EC  1391    JEQ   $->DC          >4510  ** Not found?, return error
45EE  06A0    BL    @>4754                ** Convert ascii number to intege
45F0  4754
45F2  1002    JMP   $+>06          >45F8
45F4  C90B    MOV   R11,@>FF88(R4)        ** Save return
45F6  FF88
45F8  0201    LI    R1,>40A6              ** Point to valid baud rate table
45FA  40A6
45FC  04C2    CLR   R2                    ** Clear table offset
45FE  C2F1    MOV   *R1+,R11              ** Get table baud rate
4600  1387    JEQ   $->F0          >4510  ** Passed last one?, return error
4602  82C5    C     R5,R11                ** Is baud = to table baud?
4604  1302    JEQ   $+>06          >460A  ** Yes?, then jump
4606  05C2    INCT  R2                    ** Add 2 to the offset
4608  10FA    JMP   $->0A          >45FE  ** Check again
460A  D2E0    MOVB  @>000C,R11            ** Move clock frequency
460C  000C                                ** into R11
460E  098B    SRL   R11,8                 ** Put it in 16 bits
4610  0201    LI    R1,>409C              ** Point to Frequency table
4612  409C
4614  C171    MOV   *R1+,R5               ** Move frequency to R5
4616  1327    JEQ   $+>50          >4666  ** if no match in table, JMP
4618  82C5    C     R5,R11                ** Check to see if correct
                                          ** frequency
461A  1302    JEQ   $+>06          >4620  ** Yes?, then jump
461C  05C1    INCT  R1                    ** Point to next frequency
461E  10FA    JMP   $->0A          >4614  ** Go check another
4620  A091    A     *R1,R2                ** Add offset pointer to offset
                                          ** in R2
4622  C052    MOV   *R2,R1                ** Get value in R1
4624  1505    JGT   $+>0C          >4630  ** Is MSBit set?, then jump
4626  4660    SOCB  @>4072,*R9            ** Set bit 4 of option byte
4628  4072
462A  0241    ANDI  R1,>7FFF              ** Mask out MSBit
462C  7FFF
462E  1002    JMP   $+>06          >4634
4630  5660    SZCB  @>4072,*R9            ** Clear bit 4 of option byte
4632  4072
4634  C901    MOV   R1,@>FFFE(R4)         ** Move the value to VDP ********
4636  FFFE
4638  1023    JMP   $+>48          >4680  ** Return
463A  C90B    MOV   R11,@>FF88(R4)        ** Save return address
463C  FF88
463E  06A0    BL    @>4870                ** Character here?
```

```
4642  13Ø3    JEQ   $+>Ø8           >464A  ** Yes?, then jump
4644  Ø6AØ    BL    @>488Ø                 ** Check for Function 4 key
4646  488Ø
4648  1ØFA    JMP   $->ØA           >463E
464A  CØC3    MOV   R3,R3                  ** RS232?
464C  16ØE    JNE   $+>1E           >466A  ** No?, then jump
464E  Ø4C6    CLR   R6                     ** Clear out R6
4650  3606    STCR  R6,8                   ** Get the byte
4652  1E12    SBZ   >12                    ** Reset Receive Buffer Reg
4654  1FØB    TB    >ØB                    ** Over Run Error?
4656  1307    JEQ   $+>1Ø           >4666  ** Yes?, report error
4658  1FØC    TB    >ØC                    ** Framing Error?
465A  1305    JEQ   $+>ØC           >4666  ** Yes?, report error
465C  D2E4    MOVB  @>FF7B(R4),R11         ** .CH on?
465E  FF7B
4660  130F    JEQ   $+>2Ø           >4680  ** No?, then jump
4662  1FØA    TB    >ØA                    ** Receive Parity Error?
4664  16ØD    JNE   $+>1C           >4680  ** No?, then jump
4666  Ø46Ø    B     @>445C                 ** Report error
4668  445C
466A  1DØ1    SBO   >Ø1                    ** Tell port we're not busy
466C  1EØ2    SBZ   >Ø2                    **
466E  1FØ2    TB    >Ø2                    ** Check data strobe
4670  1603    JNE   $+>Ø8           >4678  ** Char here?, Yes?, then jump
4672  Ø6AØ    BL    @>488Ø                 ** Check Function 4 Key
4674  488Ø
4676  1ØFB    JMP   $->Ø8           >466E
4678  Ø4C6    CLR   R6                     ** Clear out R6
467A  D1AØ    MOVB  @>5ØØØ,R6              ** Get character
467C  5ØØØ
467E  1DØ2    SBO   >Ø2                    ** Acknowledge we got it
4680  C2E4    MOV   @>FF88(R4),R11         ** Restore return address
4682  FF88
4684  Ø45B    RT                           ** Go back

** Subroutine to calculate the number of bytes to send or receive **
** per block                                                       **

4686  C9ØB    MOV   R11,@>FF88(R4)         ** Save return address
4688  FF88
468A  D1C7    MOVB  R7,R7                  ** Less than 256 bytes left to
                                           ** transfer
468C  1309    JEQ   $+>14           >46AØ  ** Yes?, then jump
468E  Ø6AØ    BL    @>46B4                 ** Write # of recs on the screen
4690  46B4
4692  Ø227    AI    R7,-256                ** Subtract 256 from number of
4694  FFØØ                                 ** bytes left to transfer
4696  C907    MOV   R7,@>FF7E(R4)          ** Save number left to transfer
4698  FF7E
469A  Ø207    LI    R7,256                 ** Going to get 256 bytes
469C  Ø1ØØ
469E  1006    JMP   $+>ØE           >46AC
46AØ  C1C7    MOV   R7,R7                  ** Ø bytes left to transfer?
46A2  1602    JNE   $+>Ø6           >46A8  ** No?, then jump
46A4  Ø46Ø    B     @>4464                 ** Return from card
46A6  4464
46A8  Ø4E4    CLR   @>FF7E(R4)             ** Write Ø bytes left to get
46AA  FF7E
46AC  C907    MOV   R7,@>FF8Ø(R4)          ** Save # bytes for this block
```

```
46AE  FF80
46B0  10E7    JMP  $->30          >4680  ** Return
46B2  0707    SETO R7
46B4  C90B    MOV  R11,@>FF8A(R4)        ** Save return address
46B6  FF8A
46B8  04C1    CLR  R1                    ** Point to screen
46BA  06A0    BL   @>484E                ** Set up VDPWA for screen
46BC  484E
46BE  4000    DATA >4000
46C0  06A0    BL   @>485A                ** Write 15 spaces
46C2  485A
46C4  C087    MOV  R7,R2                 ** # of recs in R2
46C6  0982    SRL  R2,8                  ** Put in 16 bits
46C8  0206    LI   R6,>0064              ** Load R6 with 100
46CA  0064
46CC  04C1    CLR  R1                    ** Get ready for Divide
46CE  3C46    DIV  R6,R1                 ** Divide by 100
46D0  0221    AI   R1,>0030              ** Add ascii offset for #'s
46D2  0030
46D4  0A81    SLA  R1,8                  **
46D6  B064    AB   @>FF72(R4),R1         ** Add PAB screen offset
46D8  FF72
46DA  DBC1    MOVB R1,@>FFFE(R15)        ** Write it to screen
46DC  FFFE
46DE  04C5    CLR  R5                    ** Set up for divide
46E0  3D60    DIV  @>4796,R5             ** Divide by 10
46E2  4796
46E4  C185    MOV  R5,R6                 ** Move integer value to R6
46E6  16F2    JNE  $->1A          >46CC  ** Not zero?, then jump and
                                         ** display another one
46E8  06A0    BL   @>485A                ** Write 15 spaces
46EA  485A
46EC  101E    JMP  $+>3E          >472A  ** Restore and go back
46EE  C90B    MOV  R11,@>FF8A(R4)        ** Save return address
46F0  FF8A
46F2  D2E4    MOVB @>FF79(R4),R11        ** .CR?
46F4  FF79
46F6  1619    JNE  $+>34          >472A  ** Yes, so jump
46F8  06A0    BL   @>47E4                ** Write CR to port
46FA  47E4
46FC  0D00    DATA >0D00
46FE  1002    JMP  $+>06          >4704  ** Jump to Nulls
4700  C90B    MOV  R11,@>FF8A(R4)        ** Save return address
4702  FF8A
4704  D064    MOVB @>FF7C(R4),R1         ** .NU?
4706  FF7C
4708  1307    JEQ  $+>10          >4718  ** No?, then jump
470A  0205    LI   R5,>0006              ** 6 nulls to write
470C  0006
470E  06A0    BL   @>47E4                ** Write Null char
4710  47E4
4712  0000    DATA >0000
4714  0605    DEC  R5                    ** Done yet?
4716  16FB    JNE  $->08          >470E  ** No?, do it again
4718  D064    MOVB @>FF79(R4),R1         ** .CR?
471A  FF79
471C  1606    JNE  $+>0E          >472A  ** Yes?, then jump
471E  D064    MOVB @>FF7A(R4),R1         ** .LF?
4720  FF7A
4722  1603    JNE  $+>08          >472A  ** Yes?, then jump
```

```
4724  06A0    BL   @>47E4                ** Write LF to port
4726  47E4
4728  0A00    DATA >0A00
472A  C2E4    MOV  @>FF8A(R4),R11        ** Restore return address
472C  FF8A
472E  045B    RT                         ** Go back
** Subroutine to check opcode for Load or Save **

4730  D064    MOVB @>FF6A(R4),R1         ** Move opcode into R1
4732  FF6A
4734  0981    SRL  R1,8                  ** Put in 16 bits
4736  0221    AI   R1,>FFFB              ** Subtract 5
4738  FFFB
473A  1301    JEQ  $+>04          >473E  ** If zero return
473C  0601    DEC  R1                    ** Subtract 1
473E  045B    RT                         ** Go back!
4740  D064    MOVB @>FF6B(R4),R1         ** Move flag status to MSByte R1
4742  FF6B
4744  2060    COC  @>4072,R1             ** check for Internal file type?
4746  4072
4748  045B    RT                         ** Go back

** Subroutine to check for variable length records **

474A  D064    MOVB @>FF6B(R4),R1         ** Move flag status to MSByte R1
474C  FF6B
474E  0241    ANDI R1,>1000              ** Mask all bits but bit 3
4750  1000
4752  045B    RT                         ** Go back

** Subroutine to convert ASCII # in the VDP PAB to integer returned **
* in VDP PAB to integer returned in R1 and R5                       **

4754  C90B    MOV  R11,@>FF8A(R4)        ** Save return address
4756  FF8A
4758  04C1    CLR  R1                    ** Clear out temporary sum
475A  04CB    CLR  R11                   ** Set # times loop executed to 0
475C  1003    JMP  $+>08          >4764
475E  D1AF    MOVB @>FBFE(R15),R6        ** Get another char from the PAB
4760  FBFE
4762  0600    DEC  R0                    ** Subtract 1 from option length
4764  C1C6    MOV  R6,R7                 ** Put char byte in R7
4766  0987    SRL  R7,8                  ** Put it in low byte
4768  0227    AI   R7,-48                ** Subtract ascii offset
476A  FFD0
476C  110C    JLT  $+>1A          >4786  ** Less than 0? then jump
476E  0287    CI   R7,>0009              ** Bigger than 9?
4770  0009
4772  1B09    JH   $+>14          >4786  ** Yes?, then jump
4774  058B    INC  R11                   ** Add 1 to # times through loop
4776  3860    MPY  @>4796,R1             ** Multiply Temporary sum by 10
4778  4796
477A  C041    MOV  R1,R1                 ** Is number bigger than 16 bits?
477C  1606    JNE  $+>0E          >478A  ** Yes?, return error
477E  A087    A    R7,R2                 ** Add new value to old value
4780  C042    MOV  R2,R1                 ** Move it to the temporary sum
4782  C000    MOV  R0,R0                 ** Hit end of PAB yet?
4784  16EC    JNE  $->26          >475E  ** No?, go do it again
4786  C2CB    MOV  R11,R11               ** Have we got a valid number?
4788  1602    JNE  $+>06          >478E  ** Yes?, then go exit
```

```
478A  0460     B    @>444A                ** Return Error from card
478C  444A
478E  C141     MOV  R1,R5                 ** Copy the value into R5
4790  C2E4     MOV  @>FF8A(R4),R11        ** Restore return address
4792  FF8A
4794  045B     RT                         ** Go back!
4796  000A     DATA 10

** Options search subroutine **

4798  C17B     MOV  *R11+,R5              ** Get char we're looking for
479A  9185     CB   R5,R6                 ** Is it there?
479C  1307     JEQ  $+>10          >47AC  ** Yes?, then jump
479E  D1AF     MOVB @>FBFE(R15),R6        ** Read next byte from PAB
47A0  FBFE
47A2  0600     DEC  R0                    ** Subtract 1 from option length
47A4  9185     CB   R5,R6                 ** Is it the char we're looking
                                          ** for?
47A6  1302     JEQ  $+>06          >47AC  ** Yes?, then jump
47A8  C000     MOV  R0,R0                 ** Are we at the end of the PAB?
47AA  16F9     JNE  $->0C          >479E  ** No?, then do it again
47AC  C000     MOV  R0,R0                 ** Are we at the end of the PAB?
47AE  1307     JEQ  $+>10          >47BE  ** Yes?, then jump
47B0  04C6     CLR  R6                    ** Clear out return word
47B2  D1AF     MOVB @>FBFE(R15),R6        ** Read the next byte in PAB
47B4  FBFE
47B6  0600     DEC  R0                    ** Subtract one from the option
                                          ** length
47B8  0286     CI   R6,>2000              ** Is it a space character?
47BA  2000
47BC  13F7     JEQ  $->10          >47AC  ** Yes?, then go get another
47BE  045B     RT                         ** Go back

** CRC calculation routine **

47C0  C046     MOV  R6,R1                 ** Put char in R1
47C2  0241     ANDI R1,>FF00              ** Mask off lower byte
47C4  FF00
47C6  2A41     XOR  R1,R9                 ** XOR new byte with CRC
47C8  C049     MOV  R9,R1                 ** Move to scratch register
47CA  0941     SRL  R1,4                  ** Shift scratch right 4
47CC  2849     XOR  R9,R1                 ** XOR CRC with scratch reg
47CE  0241     ANDI R1,>FF00              ** Mask off lower byte
47D0  FF00
47D2  0941     SRL  R1,4                  ** Shift scratch right 4
47D4  2A41     XOR  R1,R9                 ** XOR scratch with CRC
47D6  0B71     SRC  R1,7                  ** Rotate scratch right 7
47D8  2A41     XOR  R1,R9                 ** XOR scratch with CRC
47DA  06C9     SWPB R9                    ** Reverse bytes in CRC
47DC  045B     RT                         ** Go back
47DE  D1AF     MOVB @>FBFE(R15),R6        ** Get byte from PAB data buffer
47E0  FBFE
47E2  1001     JMP  $+>04          >47E6
47E4  C1BB     MOV  *R11+,R6              ** Put DATA in R6
47E6  C90B     MOV  R11,@>FF8C(R4)        ** Save return address
47E8  FF8C
47EA  C0C3     MOV  R3,R3                 ** Check for PIO
47EC  160D     JNE  $+>1C          >4808  ** If PIO then jump
47EE  1D10     SBO  >10                   ** Turn on request to send
47F0  1F1B     TB   >1B                   ** Test Data Set Ready
```

```
47F2  16Ø2    JNE   $+>Ø6           >47F8  ** Nope?, then jump
47F4  1F16    TB    >16                    ** Transmitter buffer empty?
47F6  13Ø3    JEQ   $+>Ø8           >47FE  ** Yes?, jump
47F8  Ø6AØ    BL    @>488Ø                 ** Check for Function 4 key
47FA  488Ø
47FC  1ØF6    JMP   $->12           >47EA  ** Nope!, so do loop again
47FE  32Ø6    LDCR  R6,8                   ** Write the character to RS232
48ØØ  1E1Ø    SBZ   >1Ø                    ** Clear the request to send
48Ø2  C2E4    MOV   @>FF8C(R4),R11         ** Restore the return address
48Ø4  FF8C                                 ** Go back
48Ø6  Ø45B    RT
48Ø8  1EØ1    SBZ   >Ø1                    ** Set data strobe out
48ØA  1FØ2    TB    >Ø2                    ** Check if busy
48ØC  13F5    JEQ   $->14           >47F8  ** Yes?, then go test again
48ØE  D8Ø6    MOVB  R6,@>5ØØØ              ** Write char to PIO
481Ø  5ØØØ
4812  1EØ2    SBZ   >Ø2                    ** We sent it!
4814  1FØ2    TB    >Ø2                    ** Did the port get it?
4816  13Ø3    JEQ   $+>Ø8           >481E  ** Yes?, then jump
4818  Ø6AØ    BL    @>488Ø                 ** Check for Function 4 key
481A  488Ø
481C  1ØFB    JMP   $->Ø8           >4814
481E  1DØ2    SBO   >Ø2                    ** Tell port we're done
4820  1ØFØ    JMP   $->1E           >48Ø2
4822  CØC3    MOV   R3,R3                  ** Is it RS232?
4824  13Ø3    JEQ   $+>Ø8           >482C  ** Yes?, then jump
4826  1DØ2    SBO   >Ø2                    ** Reset PIO
4828  1EØ1    SBZ   >Ø1                    **   "    "
482A  Ø45B    RT                           ** Go back

** Set options and baud rate **

482C  1D1F    SBO   >1F                    ** Reset RS232
482E  3224    LDCR  @>FFFA(R4),8           ** Load control register
4832  1EØD    SBZ   >ØD                    ** Disable LDIR
4834  3324    LDCR  @>FFFE(R4),12          ** Load receive data rate registe
4838  DØ64    MOVB  @>FF7D(R4),R1
483A  FF7D
483C  13Ø1    JEQ   $+>Ø4           >4840
483E  1D12    SBO   >12                    ** Enable RBRL Interrupts
4840  Ø45B    RT                           ** Go back
4842  CØ64    MOV   @>FF76(R4),R1          ** Put device pointer in R1
4844  FF76
4846  6Ø64    S     @>FF74(R4),R1          ** Subtract the device name lengt
4848  FF74
484A  Ø221    AI    R1,>FFF6               ** Subtract 1Ø( point to start of
484C  FFF6                                 ** PAB)
484E  A07B    A     *R11+,R1               ** Add the DATA statement offset
4850  D7E4    MOVB  @>ØØØ3(R4),*R15        ** Move LSByte R1 into VDPWA
4852  ØØØ3
4854  1ØØØ    NOP                          ** Wait
4856  D7C1    MOVB  R1,*R15                ** Move MSByte R1 into VDPWA
4858  Ø45B    RT                           ** Go back
485A  Ø2Ø1    LI    R1,>2Ø2Ø               ** Put space char in R1
485C  2Ø2Ø
485E  B064    AB    @>FF72(R4),R1          ** Add PAB screen offset
4860  FF72
4862  Ø2Ø2    LI    R2,>ØØØE               ** 15 spaces to write to get
4864  ØØØE                                 ** to middle of screen
4866  DBC1    MOVB  R1,@>FFFE(R15)         ** Write it to screen
```

```
4868  FFFE
486A  Ø6Ø2     DEC   R2                      ** Done yet?
486C  16FC     JNE   $->Ø6          >4866     ** nope?, do it again
486E  Ø45B     RT                            ** go back
4870  CØC3     MOV   R3,R3                   ** Check to see if PIO,RS232
4872  16Ø4     JNE   $+>ØA          >487C     ** PIO?, then jump
4874  1F1B     TB    >1B                     ** Test Data Set Ready
4876  16Ø1     JNE   $+>Ø4          >487A     ** Not ready?, then jump
4878  1F15     TB    >15                     ** Receive buffer loaded?
487A  Ø45B     RT                            ** Go back
487C  1FØ2     TB    >Ø2                     ** Check Data Strobe
487E  Ø45B     RT                            ** Go back

** Test Function 4 routine **

4880  C04C     MOV   R12,R1                  ** Save CRU port address
4882  Ø20C     LI    R12,>ØØ24               ** Set to Keyboard select
4884  ØØ24
4886  30EØ     LDCR  @>4073,3                ** Load it with Ø
488A  1FF5     TB    ->ØB                    ** Test function key
488C  1304     JEQ   $+>ØA          >4896     ** Not pressed?, then jump
488E  30EØ     LDCR  @>4074,3                ** Load KB select with 3
4892  1FF5     TB    ->ØB                    ** Test for 4 key
4894  16Ø2     JNE   $+>Ø6          >489A     ** Pressed?, then jump
4896  C301     MOV   R1,R12                  ** Restore CRU address
4898  Ø45B     RT                            ** Return
489A  C301     MOV   R1,R12                  ** Restore CRU address
489C  Ø46Ø     B     @>445C                  ** Go report error
489E  445C
48AØ  ABCD     DATA  >ABCD                   ** Not used??
48A2  ØB8Ø     SRC   RØ,8                    ** Delay routine
48A4  ØB8Ø     SRC   RØ,8
48A6  ØB8Ø     SRC   RØ,8
48A8  ØB8Ø     SRC   RØ,8
48AA  ØB8Ø     SRC   RØ,8
48AC  ØB8Ø     SRC   RØ,8
48AE  Ø45B     RT
```

```
** This is actual code which sets RS232/1 to 1200 baud and then writes   **
** the command "ATD2220000" to the port.  This is the command to dial    **
** phone number 2220000 using a Hayes compatable smart modem.            **

          DEF  START
          REF  VSBW
CRUBAS    EQU  >1300          ** CRU Base Address for RS232 card
PORT1     EQU  >1340          ** CRU Address for RS232/1
PORT2     EQU  >1380          ** CRU Address for RS232/2
PORTNM    DATA PORT1          ** This word selects which port to set
*                             ** up, write and read from
OPBYTE    BYTE >00            ** Byte used to set options for the
*                             ** port
BDBIT     BYTE >00            ** Control bit for baud rates
COMAND    BYTE >0B            ** 11 bytes to write
          TEXT 'ATD2220000'   ** Have the modem call phone number
          BYTE >0D            ** Carriage return
          EVEN

START     LI   R12,CRUBAS     ** Set the CRU Base to the RS232 card
          SBO  >00            ** Turn on the card
          SBO  >07            ** Turn on the LED
          LI   R5,1200        ** Set baud rate to 1200 baud
          LI   R1,>40A6       ** Point to valid baud rate table
          CLR  R2             ** Clear table offset
BDCK      MOV  *R1+,R11       ** Get table baud rate
          C    R5,R11         ** Is baud equal to table baud?
          JEQ  FNDBD          ** Yes?, then jump
          INCT R2             ** Add 2 to the offset
          JMP  BDCK           ** Go check next baud
FNDBD     MOVB @>000C,R11     ** Get clock frequency
          SRL  R11,8          ** Put it in 16 bits
          LI   R1,>409C       ** Point to frequency table
TMRCHK    MOV  *R1+,R5        ** Get frequency
          C    R5,R11         ** Are they equal?
          JEQ  FNDTMR         ** Yes?, then jump
          INCT R1             ** Point to next frequency
          JMP  TMRCHK         ** Go check another
FNDTMR    A    *R1,R2         ** Add offset pointer to R2
          MOV  *R2,R1         ** Get baud rate
          JGT  NOBIT          ** MSbit set?, then jump
          SOCB @>4072,@BDBIT  ** Set bit for option byte
          ANDI R1,>7FFF       ** Mask off MSBit
          JMP  BOVER
NOBIT     SZCB @>4072,@BDBIT  ** Clear bit for option byte
*                             ** return it in R5
BOVER     MOV  @PORTNM,R12    ** Put the correct Port CRU Base in R12

          LI   R9,OPBYTE      ** R9 points to options byte

          SZCB @>45A2,*R9     ** Clear the byte
*                             ** No Parity, default
          SOCB @>4074,*R9     ** 8 Data Bits
*                             ** 1 Stop Bit, default

          SOCB @BDBIT,*R9     ** Set bit for baud rates
          SBO  >1F
          LDCR @OPBYTE,8      ** Load options in control register
```

```
          SBZ   >ØD
          LDCR R1,12              ** Load Baud rate in register

          LI    R12,CRUBAS        ** Load CRU Base for RS232 card
          SBZ   >Ø7               ** Turn off LED
          SBZ   >ØØ               ** Turn off card

          LI RØ,COMAND            ** Point to number of bytes to write
          MOVB *RØ+,R1            ** Put it in R1
          SRL R1,8                ** Put it in 16 bits
WLOOP1    MOVB *RØ+,R5            ** Read byte of command
          BL @WRTCHR              ** Write it to the port
          DEC R1                  ** Done yet?
          JNE WLOOP1              ** No?, then do it again
LOOPIT    JMP LOOPIT              ** Infinite loop


** Write Character Subroutine **
** Pass in character to write in MSByte of R5 **

WRTCHR    LI    R12,CRUBAS        ** Load CRU Base for RS232 card
          SBO   >ØØ               ** Turn on card
          SBO   >Ø7               ** Turn on LED
          MOV   @PORTNM,R12       ** CRU base for correct port
LOOP1     SBO   >1Ø               ** Request to send
          TB    >1B               ** Data Set Ready?
          JNE   LOOP1             ** No?, then jump
          TB    >16               ** Transmitter Buffer Empty?
          JNE   LOOP1             ** No?, then jump
          LDCR R5,8               ** Write character to port
          SBZ   >1Ø               ** Clear Request to Send
          LI    R12,CRUBAS        ** Re-Load CRU Base for RS232 card
          SBZ   >Ø7               ** Turn off LED
          SBZ   >ØØ               ** Turn off card
          RT                      ** Return

          END
```

```
** This is code which can be used to initialize an RS232 port, write **
** to it, and read from it.                                           **

CRUBAS    EQU  >1300          ** CRU Base Address for RS232 card
PORT1     EQU  >1340          ** CRU Address for RS232/1
PORT2     EQU  >1380          ** CRU Address for RS232/2
                              ** baud rate search routine
PORTNM    DATA PORT1          ** This word selects which port to set
                              ** up, write and read from
OPBYTE    BYTE >00            ** Byte used to set options for the
                              ** port
BDBIT     BYTE >00            ** Control bit for baud rate
          EVEN

** This routine sets up the RS232 port
SETUP     LI   R12,CRUBAS     ** Set the CRU Base to the RS232 card
          SBO  >00            ** Turn on the card
          SBO  >07            ** Turn on the LED
          LI   R5,1200        ** Put the baud rate you want to use
                              ** in R5, Either 110,300,600,1200,4800,
                              ** or 9600 baud.  This example uses
                              ** 1200 baud.
** Find the correct baud rate data

          LI   R1,>40A6       ** Point to valid baud rate table
          CLR  R2             ** Clear table offset
BDCK      MOV  *R1+,R11       ** Get table baud rate
          C    R5,R11         ** Is baud equal to table baud?
          JEQ  FNDBD          ** Yes?, then jump
          INCT R2             ** Add 2 to the offset
          JMP  BDCK           ** Go check next baud
FNDBD     MOVB @>000C,R11     ** Get clock frequency
          SRL  R11,8          ** Put it in 16 bits
          LI   R1,>409C       ** Point to frequency table
TMRCHK    MOV  *R1+,R5        ** Get frequency
          C    R5,R11         ** Are they equal?
          JEQ  FNDTMR         ** Yes?, then jump
          INCT R1             ** Point to next frequency
          JMP  TMRCHK         ** Go check another
FNDTMR    A    *R1,R2         ** Add offset pointer to R2
          MOV  *R2,R1         ** Get baud rate
          JGT  NOBIT          ** MSbit set?, then jump
          SOCB @>4072,@BDBIT  ** Set bit for option byte
          ANDI R1,>7FFF       ** Mask off MSBit
          JMP  BOVER
NOBIT     SZCB @>4072,@BDBIT  ** Clear bit for option byte
                              ** return it in R5
BOVER     MOV  @PORTNM,R12    ** Put the correct Port CRU Base in R12

          MOV  @PORTNM,R12    ** Put the correct Port CRU Base in R12

          LI   R9,OPBYTE      ** R9 points to options byte

          SZCB @>45A2,*R9     ** Clear the byte

                              ** The following instructions should be
```

```
                                ** selected to set the options you want

                                ** No Parity, no instruction

          SOCB @>40A1,*R9       ** Odd Parity

          SOCB @>422C,*R9       ** Even Parity

          SOCB @>4074,*R9       ** 8 Data Bits

          SOCB @>4074,*R9       ** 7 Data Bits (both instructions)
          SZCB @>45F9,*R9

                                ** 1 Stop Bit, no instruction

          SOCB @>4004,*R9       ** 2 Stop Bits

          SOCB @BDBIT,*R9       ** Set control bit for baud rate

          SBO  >1F
          LDCR @OPBYTE,8        ** Load options in control register
          SBZ  >0D
          LDCR R5,12            ** Load Baud rate in register

          LI   R12,CRUBAS       ** Load CRU Base for RS232 card
          SBZ  >07              ** Turn off LED
          SBZ  >00              ** Turn off card


** Write Character Subroutine **
** Pass in character in MSByte of R5 **

WRTCHR    LI   R12,CRUBAS       ** Load CRU Base for RS232 card
          SBO  >00              ** Turn on card
          SBO  >07              ** Turn on LED
          MOV  @PORTNM,R12      ** CRU base for correct port
LOOP1     SBO  >10              ** Request to send
          TB   >1B              ** Data Set Ready?
          JNE  LOOP1            ** No?, then jump
          TB   >16              ** Transmitter Buffer Empty?
          JNE  LOOP1            ** No?, then jump
          LDCR R5,8             ** Write character to port
          SBZ  >10              ** Clear Request to Send
          LI   R12,CRUBAS       ** Re-Load CRU Base for RS232 card
          SBZ  >07              ** Turn off LED
          SBZ  >00              ** Turn off card
          RT                    ** Return

** Read Character Subroutine **
** Character returned in MSByte of R5 **

RDCHR     LI   R12,CRUBAS       ** Load CRU Base for RS232 card
          SBO  >00              ** Turn on card
          SBO  >07              ** Turn on LED
          MOV  @PORTNUM,R12     ** CRU base for correct port
LOOP2     TB   >1B              ** Data Set Ready?
          JNE  LOOP2            ** No?, then jump
          TB   >15              ** Receive buffer loaded?
          JNE  LOOP2            ** No?, then jump
          STCR R5,8             ** Get the character
```

```
SBZ   >12               ** Reset Receive Buffer Reg
LI    R12,CRUBAS        ** Re-Load CRU Base for RS232 card
SBZ   >07               ** Turn off LED
SBZ   >00               ** Turn off card
RT                      ** Return
```

```
** This program shows how to write to PIO/1 directly. **

        DEF START

PORT    EQU   >5000              ** Address of CRU data register
CRUBAS  EQU   >1300              ** CRU base for RS232 card
CARLNF  DATA  >0D0A              ** Data for carriage return and line feed

MESSG   TEXT 'This is a test'
MESSG2  TEXT ' Notice this is right next to it!'
        EVEN

START   MOV   R11,R10            ** Save return address
        LI    R1,MESSG           ** First message
        LI    R2,14              ** 14 bytes long
        BL    @WRTPIO
        LI    R1,MESSG2          ** Second message
        LI    R2,33              ** 33 bytes long
        BL    @WRTPIO
        LI    R1,CARLNF          ** Send a carriage return and line feed
        LI    R2,2               ** 2 bytes long
        BL    @WRTPIO
        MOV   R10,R11            ** Restore return address
        RT


** Subroutine WRTPIO **
** Place address of data to print to PIO in R1, Length of data in R2 **
** Access with a BL @WRTPIO


WRTPIO  MOV   R12,R5             ** Save current CRU offset
        LI    R12,CRUBAS         ** Load R12 with the CRU base address
        SBO   >00                ** Turn on card
        SBO   >07                ** Turn on LED
        SBO   >02                ** Reset PIO
        SBZ   >01
LP2     SBZ   >01                ** Request to send
        TB    >02                ** Busy?
        JEQ   LP2                ** Yes?, check again
        MOVB  *R1+,@PORT         ** Write the byte
        SBZ   >02                ** We sent it
LP1     TB    >02                ** Busy?
        JNE   LP1                ** Yes?, Check again
        SBO   >02                ** Acknowledge receipt
        DEC   R2                 ** Done yet?
        JNE   LP2                ** No?, then jump
        SBO   >02                ** Reset PIO
        SBZ   >01
        SBZ   >07                ** Turn off LED
        SBZ   >00                ** Turn off the card!
        MOV   R5,R12             ** Restore CRU offset
        RT                       ** Return

        END
```

The Texas Instruments Disk Controller card is probably the most important peripheral to the TI system. It allows storage and retrieval of data from up to three 5-1/4 inch double sided disk drives. This is understandably the longest section of code in the book do to the many functions it performs.

Data can be written in many file types with varying lengths and records within these blocks must be able to be accessed separately. All of this causes a large amount of overhead in calculations and buffer space. Could you imagine if you couldn't just access records in BASIC by using INPUT statements but had to tell exactly where on the disk you wanted data read from!

Although this code is large and overbearing when glancing over it, I find it the most interesting. By studying it you can learn about the exact structure of the TI file system.

You will notice that throughout the code I will use the abbreviation AU which stands for Allocation Unit. This is just a synonym for sector and can be used interchangably.

For more information on this peripheral consult the TI disk memory manuals or the technical data sheets on the Western Digital 1771 Floppy Disk Controller chip contained within it.

NOTE: Ryte Data now markets a TI disk controller EPROM set which allows the use of 80 track (96tpi) disk drives to achieve 1440 sectors per disk. This retains all features of the original controller with the added features of reading and writing to 80 track disks. The existing PROMs must be removed and the two new 2732 EPROMs are installed in their place. Disk Manager II will allow you to format 80 track disks on two sides. Both 40 track and 80 track drives can be used in the same system with certain restrictions. Write or call for more information.

VOLUME INFORMATION BLOCK

The Volume Information Block is located at sector Ø and contains data required by the DSR. This data is shown in the following table.

Address	Contents
>ØØØØ - >ØØØ9	Disk name - 1Ø characters padded with spaces on the right.
>ØØØA - >ØØØB	Total number of sectors on the disk SS = >Ø168, DS = >Ø2DØ
>ØØØC	Number of sectors per track, >Ø9
>ØØØD - >ØØØF	>44534B = 'DSK'
>ØØ1Ø	Protection byte ' ' = >2Ø = Not protected 'P' = >5Ø = Protected
>ØØ11	Number of tracks per side
>ØØ12	Number of formatted sides, >Ø1 or >Ø2
>ØØ13	Density byte, >Ø1 = Single Density
>ØØ38 - >ØØ64	Sector bit map for side 1
>ØØ65 - >ØØ91	Sector bit map for side 2

The bit map is the way which the disk controller keeps track of what sectors are used on the disk. Each bit in the 45 bytes stands for one sector on that side of the disk. If a bit is equal to 1 then that sector is used. If it is equal to Ø then it is available. The first byte is for sectors Ø - 7, the second byte for sectors 8 - 15, and so on. Within each byte the sectors are equated to bits from right to left, so the smallest numbered sector for that byte is equated to the least significant bit.
Example: Byte >ØØ38 equal to >8F; Sectors Ø,1,2,3, and 7 are used, sectors 4,5, and 6 are available. During initialization bits corresponding to bad sectors and sectors reserved for controller use are set to 1.

DIRECTORY LINK

The Directory Link is located at sector 1. This is used to keep track of the locations of the File Descriptor Records on the disk. Each word in this sector contains the sector number of the File

Descriptor for an allocated file. These words are alphabetically sorted in order of the filenames. The list is terminated by a word containing >0000. Thus, the maximum number of files a disk can contain is 127. Since the list is in alphabetical order the binary search method used by the disk controller will take a MAXIMUM of 7 tries to find a file.

FILE DESCRIPTOR RECORDS (FDR's)

Each file on a disk has a File Descriptor Record which contains information such as the name, type, location and size of the file. A detailed description of the FDR is contained in the following table.

Address	Contents
>0000 - >0009	File name - 10 characters padded to the right with spaces.
>000A - >000B	Reserved >0000
>000C	File Status Flags Bit # Description 0 0 = Data file 1 = Program file (Memory Image) 1 0 = Display 1 = Internal 2 Reserved 3 0 = Unprotected 1 = Write protected 4 - 6 Reserved 7 0 = Fixed length records 1 = Variable length records
>000D	Number of records per sector
>000E - >000F	Number of sectors (including FDR sector)
>0010	End Of File offset
>0011	Logical record size or maximum record size for variable length records
>0012 - >0013	Number of logical records or number of sectors actually used for variable length records (Bytes in reverse order)
>0014 - >001B	Reserved

>ØØ1C - >ØØFF Sector Block Link

The sector block link is used to keep track of data sectors allocated for a file. It is split into seventy six groups of three bytes identifying clusters of sectors allocated for a file. Twelve bits of each block hold the sector number of the first AU in the cluster. These twelve bits are the least significant four bits of byte two and the eight bits of byte one. The other twelve bits hold the highest logical record offset in that cluster. The bits are the eight bits of byte three joined with the most significant four bits of byte two. I.E. If a three byte block contained >54,>3Ø,>ØØ then the rearranged twelve bit numbers would be >Ø54 and >ØØ3. Each cluster holds at least 1 sector. Thus, even if every cluster was used the file would still have 76 sectors available to it and could hold approximately 19K of data.

When allocating FDR's the DSR will first try to place it in a sector between >Ø2 and sector >21. If none of these are available it will then try to place it in a sector higher than number >21.

VDP Memory use by the TI Disk Controller:

```
+-------------------------------------------------------------------+
|>37D8          DISK BUFFER AREA , default of 3 buffers      5 bytes |
|                                                                   |
|       >37D8 Validation code for the disk controller DSR ( >AA )   |
|       >37D9 Points to TOP of VDP memory ( >3FFF )                |
|       >37DB CRU base identifier                                   |
|       >37DC Maximum number of file buffers ( >03 default )  >37DC |
|    ---------------------------------------------------------      |
|         File Control Block for first open file   518 bytes        |
|                                                                   |
|       >37DD Current AU offset                                     |
|       >37DF Sector number of File Descriptor Record               |
|       >37E1 Logical Record Offset(used only with variable files)  |
|       >37E2 Drive number (If the most significant bit is set,     |
|             it indicates an updated data buffer.)                 |
|      File Descriptor Record (256 bytes)                           |
|       >37E3 File name                                             |
|       (If the most significant bit of the first character is set   |
|        it indicates updated file descriptor record.)              |
|       >37ED Reserved ( >0000 )                                    |
|       >37EF File status flags                                     |
|       >37F0 Max number of records per Allocation Unit(AU)         |
|       >37F1 Number of sectors currently allocated (256 byte blocks)|
|       >37F3 End of File offset within the last sector             |
|       >37F4 Logical record length                                 |
|       >37F5 Number of fixed length records or number of bytes for |
|             variable length records                               |
|       >37F7 Reserved, was intended for date and time, 4 words >0000|
|       >37FF Pointer blocks                                        |
|       >38E3 Data Buffer area (256 bytes)                  >39E2   |
|    ---------------------------------------------------------      |
|       >39E3 File Control Block for second open file   518 bytes   |
|                                                                   |
|       >39E9 File Descriptor Record (256 bytes)                    |
|       >3AE9 Data Buffer area (256 bytes)                  >3BE8   |
|    ---------------------------------------------------------      |
|       >3BE9 File Control Block for third open file    518 bytes   |
|                                                                   |
|       >3BEF File Descriptor Record (256 bytes)                    |
|       >3CEF Data Buffer area (256 bytes)                          |
|                                                           >3DEE   |
+-------------------------------------------------------------------+
|>3DEF           VDP STACK AREA                          252 bytes  |
|      Used for register and return address storage.                |
|                                                           >3EEA   |
+-------------------------------------------------------------------+
|>3EEB          DISK DRIVE INFO                            4 bytes  |
|                                                                   |
|       >3EEB Last drive number accessed                            |
|       >3EEC Last track access on drive #1                         |
|       >3EED Last track access on drive #2                         |
|       >3EEE Last track access on drive #3                         |
|                                                           >3EEE   |
+-------------------------------------------------------------------+

+-------------------------------------------------------------------+
|>3EEF       Not used but reserved for possible software    5 bytes  |
```

	compatability with older DSR versions. >3EF3
>3EF4	Drive number of Volume Information Block If most significant bit is set then the VIB has been updated.
>3EF5	VOLUME INFORMATION BLOCK 256 bytes A copy of sector zero from the current disk being accessed. >3FF4
>3FF5	FILE NAME COMPARE BUFFER 11 bytes Used to compare drive number, and ten character file and disk names. >3FFF

Built into the TI Disk Controller are seven low level routines. These allow functions which can't be done using the standard high level access routines described in the Editor/Assembler manual. The seven routines are as follows:

Routine	Function
>10	Read/Write Individual Sector
>11	Format Disk
>12	Modify File Protection
>13	Rename File
>14	Transfer Direct Input File
>15	Transfer Direct Output File
>16	Buffer Allocation

The parameters for these routines are transferred through the FAC block in CPU RAM, starting at >834A. Error codes for all routines are returned in memory location >8350.

Routine >10, Read/Write Individual Sector

Parameter Locations

>834A->834B	Sector # <Returned>
>834C	Drive #
>834D	Read/Write Flag
>834E->834F	VDP Buffer Address
>8350->8351	Sector # to read/write

To use this routine the drive # (1,2, or 3) is placed in the byte at >834C. The read/write flag, >00 for a write, a non-zero value for a read, is placed in the byte at >834D. A pointer to a block of VDP memory 256 bytes long is placed in the word at >834E. The sector # to read or write is placed in the word at >8350. The sector # will be returned in the word at >834A. Error codes will be returned in >8350.

Example:

```
        DEF READ
        REF DSRLNK,VMBW
PAB     DATA >0110          ** Name length of 1, routine >10
DATBUF  EQU  >1000          ** Pointer to data buffer in VDP RAM
PABPTR  EQU  >F80           ** Pointer to PAB in VDP RAM
READ    LI R0,>01FF         ** Drive 1, Read Flag
        MOV R0,@>834C       ** Put it in parameter block
        LI R0,DATBUF        ** Put buffer pointer in R0
        MOV R0,@>834E       ** Put it in parameter block
```

```
        CLR RØ                  ** Going to read sector # Ø
        MOV RØ,@>835Ø           ** Put it in parameter block
        LI RØ,PABPTR            ** Time to put PAB in >F8Ø of VDP RAM
        LI R1,PAB
        LI R2,2                 ** 2 bytes to write
        BLWP @VMBW              ** Write PAB into VDP
        MOV RØ,@>8356           ** Put pointer to PAB in >8356
        BLWP @DSRLNK            ** Read the sector
        DATA >A                 ** Low level data statement
        RT
        END
```

Routine >11, Format Disk

Parameter Locations

>834A->834B	# of sectors/disk <Returned>
>834C	MSNybble DSR Version, LSNYBBLE Drive #
>834D	# of tracks
>834E->834F	VDP Buffer Address
>835Ø	Density
>8351	# Sides

This routine will format the disk in the drive specified unless the disk is hardware write protected. The TI disk controller needs 33ØØ bytes of VDP buffer space for formatting the disk, since it writes one track at a time to the disk from VDP RAM. To use the routine the DSR version must be placed in the MSNybble and the drive # in the LSNybble of the byte at >834C. The DSR version is either Ø,1, or 2. Ø means that the format can be done on any version of the DSR ROM. 1 means that the format will use either double sided formatting or a number of tracks other than 35 or 4Ø. Thus the routine requires version 2 of the DSR ROM. 2 means that the format requires options not available on the TI disk controller and will return an error. The number of tracks must be placed in the byte at >834D. Only 35 or 4Ø may be placed here in the 1st version of the DSR ROM. The VDP Buffer address to be used must be placed in the word at >834E. The density byte at >835Ø is ignored by the TI DSR since it is not supported. The number of sides must be placed in the byte at >8351. (>Ø2 for double sided, >Ø1 for single sided) The total number of sectors on the disk is returned in the word at >834A. It must be remembered that this is a low level format routine. Directory information is not placed on the disk so all sectors contain blocks of 256 >E5 bytes.

Example:

```
        DEF FORMAT
        REF DSRLNK,VMBW
PAB     DATA >Ø111              ** Name length of 1, routine >11
DATBUF  EQU  >1ØØØ              ** Pointer to data buffer in VDP RAM
PABPTR  EQU  >F8Ø               ** Pointer to PAB in VDP RAM
FORMAT  LI RØ,>Ø128             ** DSR VRS Ø, Drive 1, 4Ø tracks
        MOV RØ,@>834C           ** Put it in parameter block
        LI RØ,DATBUF            ** Put buffer pointer in RØ
        MOV RØ,@>834E           ** Put it in parameter block
        LI RØ,>Ø2ØØ             ** Two sides
        MOVB RØ,@>8351          ** Put it in parameter block
```

```
        LI RØ,PABPTR            ** Time to put PAB in >F8Ø of VDP RAM
        LI R1,PAB
        LI R2,2                 ** 2 bytes to write
        BLWP @VMBW              ** Write PAB into VDP
        MOV RØ,@>8356           ** Put pointer to PAB in >8356
        BLWP @DSRLNK            ** Format the disk
        DATA >A                 ** Low level data statement
        RT
        END
```

Routine >12, Modify File Protection
==========

Parameter Locations

>834C	Drive #
>834D	Protect Code
>834E->834F	Pointer to File Name in VDP RAM

This routine will modify the write protect status of a file currently on the disk. The drive # is placed in the byte at >834C. The protect code is placed in the byte at >834D. To write protect the file, the MSBit of the LSNybble must be set. I.E. to write protect the file place the byte >Ø8 in >834D. To unprotect the file place the byte >ØØ in >834D. A pointer to the file name to be accessed must be placed in the word at >834E. This file name must be in VDP RAM and must be 1Ø bytes long. If the name is less than 1Ø bytes long it must be padded at the end with space characters to make it 1Ø bytes long.

Example:

```
        DEF PROTCT
        REF DSRLNK,VMBW
PAB     DATA >Ø112              ** Name length of 1, routine >12
NAME    TEXT 'TESTFILE  '       ** Name of file, 8 bytes +2 spaces
DATBUF  EQU  >1ØØØ              ** Put name here in VDP RAM
PABPTR  EQU  >F8Ø                ** Pointer to PAB in VDP RAM

PROTCT  LI RØ,DATBUF            ** Pointer to name in VDP
        LI R1,NAME
        LI R2,1Ø                ** 1Ø bytes to write to VDP
        BLWP @ VMBW             ** Put the file name in VDP RAM
        MOV RØ,@>834E           ** Put pointer to name in parameter block
        LI RØ,>Ø1Ø8             ** Drive 1, Protect the file code
        MOV RØ,@>834C           ** Put it in parameter block
        LI RØ,PABPTR            ** Time to put PAB in >F8Ø of VDP RAM
        LI R1,PAB
        LI R2,2                 ** 2 bytes to write
        BLWP @VMBW              ** Write PAB into VDP
        MOV RØ,@>8356           ** Put pointer to PAB in >8356
        BLWP @DSRLNK            ** Protect the file
        DATA >A                 ** Low level data statement
        RT
        END
```

Routine >13, Rename File

Parameter Locations

>834C	Drive #
>834E->834F	Pointer to new name in VDP RAM
>8350->8351	Pointer to current name in VDP RAM

This routine will change the name of a file currently on the disk. The drive # is placed in the byte at >834C. The new name for the file and the current name of the file must be in VDP RAM. Both names must be 10 characters long. If either name is less than 10 characters it must be padded to the right with space characters. The pointers to the new and current names in VDP RAM are placed in the words at >834E and >8350 respectively.

Example:

```
        DEF RENAME
        REF DSRLNK,VMBW
CURNAM  TEXT 'TESTFILE  '      ** Current name of the file
NEWNAM  TEXT 'MYFILE    '      ** New name for the file
PAB     DATA >0113             ** Name length of 1, routine >13
NAMBF1  EQU  >1000             ** VDP location for current name
NAMBF2  EQU  >1010             ** VDP location for new name
PABPTR  EQU  >F80              ** Pointer to PAB in VDP RAM
RENAME  LI R0,NAMBF1           ** Pointer to current name in VDP RAM
        MOV R0,@>8350          ** Put it in parameter block
        LI R1,CURNAM
        LI R2,10               ** 10 bytes to write to VDP
        BLWP @ VMBW            ** Put the current name in VDP RAM
        LI R0,NAMBF2           ** Pointer to new name in VDP RAM
        MOV R0,@>834E          ** Put it in parameter block
        LI R1,NEWNAM
        LI R2,10               ** 10 bytes to write to VDP
        BLWP @ VMBW            ** Put the new name in VDP RAM
        LI R0,>0100            ** Drive # 1
        MOV R0,@>834C          ** Put it in parameter block
        LI R0,PABPTR           ** Time to put PAB in >F80 of VDP RAM
        LI R1,PAB
        LI R2,2                ** 2 bytes to write
        BLWP @VMBW             ** Write PAB into VDP
        MOV R0,@>8356          ** Put pointer to PAB in >8356
        BLWP @DSRLNK           ** Rename the file
        DATA >A                ** Low level data statement
        RT
        END
```

Routine >14, Transfer Direct Input File

Parameter Locations

>834C	Drive #
>834D	Subroutine code
>834E->834F	Pointer to file name in VDP RAM
>8350	Pointer to extra parameters in CPU RAM

Extra Parameter Block

Starting point + offset

>0000	Pointer to VDP buffer
>0002	First AU #
>0004	Status Flags
>0005	# records/AU
>0006	End of File offset
>0007	Logical record size
>0008	# of records allocated

This routine allows reading in the contents, or information and status, of a file. It can be used in conjunction with routine >15 to copy files. To use the routine, the drive # is placed in the byte at >834C. The pointer to the name of the file in VDP RAM is placed in the word at >834E. This routine has 2 different subroutines built into it and these are distinguished between each other by the subroutine code byte at >834D. One subroutine is to read the files status. Information on the file is then returned in the extra parameter block. The total # of AU's allocated to the file will be returned in the word of the extra parameter block labeled as "First AU #". To access this subroutine place a >00 in the subroutine code byte at >834D. If the subroutine code is not equal to >00 then the second subroutine is executed. This subroutine transfers the number of AU's given in the subroutine code byte to the VDP buffer. Transfer starts at the AU given in the extra parameter block. After reading the AU's, the total number actually read is placed in this word. If all the AU's were read by the routine this word contains >0000. The drive # is placed in the byte at >834C. The pointer to the name of the file in VDP RAM is placed at >834E. The extra parameter block is a ten byte block in CPU RAM pointed to by the byte at >8350. I.E. the starting address of this block is >8300 + the byte in >8350. The first word of this block is a pointer to the VDP buffer. This buffer must be big enough to hold the number of AU's passed in by the routine. The second word is the number of the first AU to transfer. If the subroutine is the pass parameter routine then the total number of AU's allocated to that file is returned in this word. The last three words correspond to the bytes in the File Control Block of a file.

Example:

```
       DEF READIT
       REF DSRLNK,VMBW
FILNAM TEXT 'TESTFILE  '      ** Name of the file
PAB    DATA >0114             ** Name length of 1, routine >14
BUFFER EQU  >1000             ** VDP location for current name
PABPTR EQU  >F80              ** Pointer to PAB in VDP RAM
READIT LI R0,BUFFER           ** Pointer to current name in VDP RAM
       MOV R0,@>834E          ** Put it in parameter block
       LI R1,FILNAM
       LI R2,10               ** 10 bytes to write to VDP
       BLWP @ VMBW            ** Put the filename in VDP RAM
       LI R0,>0100            ** Drive # 1, Read file parameters
                              ** subroutine code >00
       MOV R0,@>834C          ** Put it in parameter block
       CLR R0                 ** Put parameters in block at >8300
       MOVB R0,@>8350         ** Put >00 in pointer
       LI R0,PABPTR           ** Time to put PAB in >F80 of VDP RAM
```

```
        LI R1,PAB
        LI R2,2                 ** 2 bytes to write
        BLWP @VMBW              ** Write PAB into VDP
        MOV RØ,@>8356           ** Put pointer to PAB in >8356
        BLWP @DSRLNK            ** Read the file status and parameters
        DATA >A                 ** Low level data statement
        RT
        END
```

Routine >15, Transfer Direct Output File

Parameter Locations

>834C	Drive #
>834D	Subroutine code
>834E->834F	Pointer to file name in VDP RAM
>835Ø	Pointer to extra parameters in CPU RAM

Extra Parameter Block

Starting point + offset

>ØØØØ	Pointer to VDP buffer
>ØØØ2	First AU #
>ØØØ4	Status flags
>ØØØ5	# records/AU
>ØØØ6	End of File offset
>ØØØ7	Logical record size
>ØØØ8	# of records allocated

This routine allows creation of, or writing a given # of AU's to, a file. It is used with routine >14 to copy files. Copying of files would go as follows. First, the status and total # of AU's in a file are read using >14. Second, a file with the same status and # of AU's is created using >15. Next the AU's of the first file are read into VDP RAM using >14. Finally the same AU's are written back to the new file using >15. To use routine >15 the subroutine code must be placed in the byte at >834D. If a >ØØ is placed in this byte, a new file is created on the drive # placed in >834C. The status and type of file are determined by the information in the extra parameter block. If the subroutine code is not >ØØ then it indicates the number of AU's to write to a file. Writing starts at the AU # found in the extra parameter block. A pointer to the file name in VDP RAM must be placed in the word at >834E. The extra parameter block is a ten byte block in CPU RAM pointed to by the byte at >835Ø. The first word of this block is a pointer to the VDP buffer to write from. The second word is the number of the first AU to write. If the access code is >ØØ then this word should hold the total # of AU's to be allocated to the file. The last three words correspond to the bytes found in the FCB of a file. Again, error codes are returned in location >835Ø.

Routine >16, Buffer Allocation

Parameter Locations

>834C	# of file buffers to allocate

This routine allocates buffers so that a given number of files may be open simultaneously. This routine works in much the same way that the CALL FILES(N) statement of BASIC works. The only parameter needed for this routine is the number of buffers to allocate. This byte must be greater than >00, no larger than >10, and placed in the CPU RAM location >834C. The software checks to make sure at least >800 bytes of VDP memory are left besides the area set aside for buffers. A non-zero byte returned in >8350 indicates an error.

The instruction BLWP @>ØØ3A(R9) followed by a data statement occurs a good amount of time throughout the disk controller card. This is an important instruction to the card so here is an explanation of exactly what it does.

The BLWP instruction of course, causes a context switch and jumps to a new execution point. (If you are not familiar with this instruction refer to page 109 of the Editor/Assembler manual.) The BLWP needs a 2 word place in memory to get its new workspace pointer and program counter values from. In this case these two locations are >835A and >835C. Upon entry to routines in the controller card these two locations are set to the values >834E and >469Ø respectively. The instruction upon execution sets the new workspace pointer to the 32 byte space at >834E of scratch pad RAM and then branches to >469Ø of the controller ROM. There are four routines located in the ROM which are selected by the data statement following the BLWP instruction. These four routines are:

Routine Ø: Save registers on the VDP stack

Routine 1: Restore registers from the VDP stack

Routine 2: Set VDPWA for a read operation from a selected register

Routine 3: Set VDPWA for a write operation from a selected register

The DATA statement following the BLWP instruction is divided into 4 nybbles with the nybbles having different meaning depending on the routine which is to be used. The least significant nybble always determines which routine is to be accessed.

For example:

DATA >ØØØ2 would select routine 2

DATA >Ø1Ø3 would select routine 3

DATA >Ø3Ø1 would select routine 1

DATA >1ØØØ would select routine Ø

The first routines Ø and 1 allow you to save any combination of

registers Ø through 11 on the VDP stack. The data statement is set up for these routines with each bit of the 3 most significant nybbles representing a register. A 1 in the corresponding bit means that the register is to be stored or restored from the stack. The bits are set up as follows:

```
          Bit Ø 1 2 3 4 5 6 7 8 9 10 11|12 13 14 15
                                       |
     Register Ø 1 2 3 4 5 6 7 8 9 10 11
```

For routine Ø the registers are stored on the stack from highest register number to the lowest register number. When routine 1 is used the registers are restored in opposite order. Here are a few examples to clarify this:

DATA >7ØØØ would store registers 3,2, and 1 on the stack

DATA >ØØ1Ø would store register 11 on the stack

DATA >7ØØ1 would restore registers 1,2, and 3 from the stack

DATA >8ØØ1 would restore register Ø from the stack

For routines 2 and 3 the 3 most significant nybbles have a completely different meaning. In this case the 3 most significant nybbles hold the value of which register times two, to set the VDPWA from. In other words if you want to set the VDPWA from register 2 you would place >ØØ4 in the 3 nybbles. Again here are some examples to clear this up:

DATA >ØØØ2 would set the VDPWA for a read from register zero

DATA >Ø1Ø2 would set the VDPWA for a read from register eight

DATA >ØØ23 would set the VDPWA for a write from register one

DATA >0103 would set the VDPWA for a write from register eight

The beauty of these routines is that when I refer to registers in the above discussion what is meant is the registers of the calling workspace! In other words the values used are for the old workspace yet they are not altered in any way!

Another pair of instructions found throughout the software which have a very large importance are the instructions BL @>4658 followed by a data statement and B @>4676.

The statement BL @>4658 works much the same as a BL instruction except for 2 small differences. First the return address of the BL statement is saved onto the VDP stack and then the address in the DATA statement is the address of the routine to be executed. The B @>4676 instruction jumps to a routine which pops an address off the VDP stack and then jumps to that address. Here is a small example of the two instructions:

```
BL   @>4658
DATA >4ED2
```

This pair of instructions would save the address immediately following the DATA statement on the stack and then execute the routine at >4ED2. At the end of of the routine at >4ED2 would be a B @>4676 which would then pop the address off the stack and return to the address following the DATA statement and resume execution there.

```
*****************************************************************
*                                                               *
*       Source code for Texas Instruments Disk Controller Card  *
*                                                               *
*                  Disassembled and commented 6/20/86           *
*                                                               *
*                           by Monty Schmidt                    *
*                                                               *
*****************************************************************

4000    AA02    BYTE >AA                ** Identifier byte
                BYTE >02                ** Version 2
4002    0000    DATA >0000
4004    4044    DATA >4044              ** Power up link
4006    0000    DATA >0000
4008    404A    DATA >404A              ** DSRLNK high level
400A    4010    DATA >4010              ** DSRLNK low level
400C    0000    DATA >0000
400E    0000    DATA >0000              ** No INTLNK
4010    4016    DATA >4016              ** Pointer to next low level
4012    5B38    DATA >5B38              ** Routine >10 at >5B38
4014    0110    DATA >0110              ** .R/W Sector.
4016    401C    DATA >401C              ** Pointer to next low level
4018    5B48    DATA >5B48              ** Routine >11 at >5B48
401A    0111    DATA >0111              ** .Initialize disk.
401C    4022    DATA >4022              ** Pointer to next low level
401E    5B52    DATA >5B52              ** Routine >12 at >5B52
4020    0112    DATA >0112              ** .Modify file protection.
4022    4028    DATA >4028              ** Pointer to next low level
4024    5BAC    DATA >5BAC              ** Routine >13 at >5BAC
4026    0113    DATA >0113              ** .Rename file.
4028    402E    DATA >402E              ** Pointer to next low level
402A    5C78    DATA >5C78              ** Routine >14 at >5C78
402C    0114    DATA >0114              ** .Access direct input file.
402E    4034    DATA >4032              ** Pointer to next low level
4030    5CE8    DATA >5CE8              ** Routine >15 at >5CE8
4032    0115    DATA >0115              ** .Access direct output file.
4034    403A    DATA >4038              ** Pointer to next low level
4036    5DAE    DATA >5DAE              ** Routine >16 at >5DAE
4038    0116    DATA >0116              ** .Buffer allocation.
403A    0000    DATA >0000              ** No more low levels!
403C    5D5A    DATA >5D5A              ** CALL FILES at >5D5A
403E    0546    BYTE 5                  ** 5 Bytes in name
4040    494C    TEXT 'FILES'            ** The name
4042    4553
4044    0000    DATA >0000              ** No more basic commands
4046    4070    DATA >4070              ** Power up at >4070
4048    0000    DATA >0000              ** Only one power up
404A    4052    DATA >4052              ** Pointer to next high level
404C    504E    DATA >504E              ** Routine 'DSK' starts at >504E
404E    0344    BYTE >03                ** 3 bytes in name
4050    534B    TEXT 'DSK'              ** The name
4052    405C    DATA >405C              ** Pointer to next high level
4054    505C    DATA >505C              ** Routine 'DSK1' starts at >505C
4056    0444    BYTE 4                  ** 4 Bytes in name
4058    534B    TEXT 'DSK1'             ** The name
405A    3100    EVEN
405C    4066    DATA >4066              ** Pointer to next high level
405E    5062    DATA >5062              ** Routine 'DSK2' starts at >5062
4060    0444    BYTE 4                  ** 4 Bytes in name
```

```
4062   534B   TEXT 'DSK2'                  ** The name
4064   3200   EVEN
4066   0000   DATA >0000                   ** No more high levels
4068   5068   DATA >5068                   ** Routine 'DSK3' starts at >5068
406A   0444   BYTE 4                       ** 4 bytes in name
406C   534B   TEXT 'DSK3'                  ** The name
406E   3300   EVEN

** Start of Power up routine **

4070   C1CB   MOV  R11,R7                  ** Save the return address
4072   02A9   STWP R9                      ** Put the GPLWSP in R9
4074   0229   AI   R9,>FF20                ** Make it point to >8300
4076   FF20
4078   C029   MOV  @>0070(R9),R0           ** Get highest available address of
407A   0070                                ** VDP RAM into R0
407C   C080   MOV  R0,R2                   ** Save it in R2
407E   0220   AI   R0,-2088                ** Subtract 2088 bytes for default
                                           ** file
4080   F7D8                                ** area in VDP RAM (3 open files)
4082   CA40   MOV  R0,@>0070(R9)           ** Put new address back in the CPU
4084   0070                                ** RAM Pointer
4086   0580   INC  R0                      ** Point to beginning of Disk VDP
                                           ** Area
4088   06A0   BL   @>40DA                  ** Set the VDPWA to it
408A   40DA
408C   0201   LI   R1,2088                ** Going to clear out the Disk VDP
408E   0828                                ** Area
4090   04EF   CLR  @>FFFE(R15)             ** Clear the byte
4092   FFFE
4094   0601   DEC  R1                      ** Done yet?
4096   16FC   JNE  >4090                   ** No?, then go clear another
4098   06A0   BL   @>40DA                  ** Set VDPWA back to beginning of
409A   40DA                                ** DVA
409C   0203   LI   R3,>AA03                ** Validation code and # of files
409E   AA03                                ** open into R3
40A0   DBC3   MOVB R3,@>FFFE(R15)          ** Write the validation code to the
40A2   FFFE                                ** the DVA
40A4   1000   NOP                          ** Wait for VDP
40A6   DBC2   MOVB R2,@>FFFE(R15)          ** Put pointer to Top of VDP into
40A8   FFFE                                ** DVA area
40AA   06C2   SWPB R2
40AC   DBC2   MOVB R2,@>FFFE(R15)
40AE   FFFE
40B0   1000   NOP
40B2   DBCC   MOVB R12,@>FFFE(R15)         ** Move first byte of CRU Base to
40B4   FFFE                                ** DVA
40B6   06C3   SWPB R3                      ** Put # of files into MSByte
40B8   DBC3   MOVB R3,@>FFFE(R15)          ** Put it into the DVA
40BA   FFFE
40BC   06A0   BL   @>4726                  ** Init sequence
40BE   4726
40C0   06A0   BL   @>4658                  ** Zero out last track and drive
40C2   4658                                ** numbers accessed
40C4   4554   DATA >4554
40C6   04E9   CLR  @>0054(R9)
40C8   0054
40CA   0429   BLWP @>005A(R9)              ** Get return address from stack
40CC   005A                                ** into R11
40CE   0011   DATA >0011
```

```
40D0    CA60    MOV  @>40D8,@>006C(R9)    ** Put >0404 at >836C
40D2    40D8
40D4    006C
40D6    045B    RT                       ** Return!
40D8    0404    DATA >0404

** Routine to set VDPWA to val in R0

40DA    0260    ORI  R0,>4000            ** Set bit for write
40DC    4000
40DE    06C0    SWPB R0                  ** Low byte first!
40E0    D7C0    MOVB R0,*R15             ** Write it to VDPWA
40E2    06C0    SWPB R0                  ** Now high byte
40E4    D7C0    MOVB R0,*R15             ** Write it to VDPWA
40E6    045B    RT

** Routine >10 continued

40E8    0204    LI   R4,10               ** 10 tries before error
40EA    000A
40EC    DA60    MOVB @>4630,@>0050(R9)   ** Clear out error byte
40EE    4630
40F0    0050
40F2    06A0    BL   @>4496              ** Change drive number
40F4    4496
40F6    06A0    BL   @>45F0              ** Point to last track accessed
40F8    45F0
40FA    04C0    CLR  R0
40FC    D02F    MOVB @>FBFE(R15),R0      ** Get the last track accessed
40FE    FBFE
4100    0280    CI   R0,>D700            ** Bigger than 40?(inverted)
4102    D700
4104    1B03    JH   >410C               ** Yes?, then jump
4106    06A0    BL   @>4524              ** Restore drive
4108    4524
410A    0700    SETO R0                  ** Inverted >0000
410C    D800    MOVB R0,@>5FFA           ** Write >00 to track register
410E    5FFA
4110    C069    MOV  @>004A(R9),R1       ** Get Sector # from CPU RAM
4112    004A
4114    1E07    SBZ  >0007               ** Set to side 1
4116    04C7    CLR  R7                  ** Byte >00 for side 1
4118    0281    CI   R1,>02D0            ** Is sector # bigger than 719?
411A    02D0
411C    144C    JHE  >41B6               ** Yes?, then jump to error
411E    0281    CI   R1,>0168            ** Is is less than 360?
4120    0168
4122    1A0D    JL   >413E               ** Yes?, then jump
4124    0221    AI   R1,-719             ** Subtract 719 (Tracks go opposite
4126    FD31                             ** direction on side 2)
4128    0741    ABS  R1                  ** Absolute it
412A    04C0    CLR  R0                  ** Set up for division
412C    3C20    DIV  @>4632,R0           ** Divide Track # by 9 sectors per
412E    4632                             ** track
4130    0221    AI   R1,>FFF8            ** Subtract 8 (Track # of sector)
4132    FFF8
4134    0741    ABS  R1                  ** Absolute it.
4136    1D07    SBO  >0007               ** Set to side 2
4138    0207    LI   R7,>0100            ** Byte >01 for side 2
413A    0100
```

```
413C  1008  JMP  >414E
413E  0281  CI   R1,>0001      ** Sector 1?
4140  0001
4142  1B02  JH   >4148        ** Greater? then jump
4144  06A0  BL   @>4524       ** Restore drive
4146  4524
4148  04C0  CLR  R0
414A  3C20  DIV  @>4632,R0    ** Divide track by 9
414C  4632
414E  06C0  SWPB R0           ** Put it in high byte
4150  0540  INV  R0           ** Invert it for controller
4152  06A0  BL   @>4614       ** Set VDPWA to last track accessed
4154  4614
4156  DBC0  MOVB R0,@>FFFE(R15) ** Write new track number there
4158  FFFE
415A  D800  MOVB R0,@>5FFE    ** Write it to data register
415C  5FFE
415E  06C1  SWPB R1           ** Put sector # in high byte
4160  0541  INV  R1           ** Invert it for controller
4162  D801  MOVB R1,@>5FFC    ** Write it to sector register
4164  5FFC
4166  9800  CB   R0,@>5FF2    ** Same track as last time?
4168  5FF2
416A  1307  JEQ  >417A        ** Yes?, then jump
416C  06A0  BL   @>45CA       ** Seek the sector
416E  45CA
4170  E100  DATA >E100
4172  06A0  BL   @>4482       ** Check for drive busy
4174  4482
4176  0AD0  SLA  R0,13        ** Seek error?
4178  181B  JOC  >41B0        ** Yes?, then jump
417A  06A0  BL   @>45CA       ** Write read address command to
417C  45CA                    ** drive
417E  3F00  DATA >3F00
4180  1D02  SBO  >0002        ** Ready line
4182  D020  MOVB @>5FF6,R0    ** Get track address from
4184  5FF6                    ** controller
4186  0206  LI   R6,4         ** 4 bytes left to read
4188  0004
418A  D160  MOVB @>5FF6,R5    ** Get side # from controller
418C  5FF6
418E  0545  INV  R5           ** Invert it( inverted bus)
4190  D020  MOVB @>5FF6,R0    ** Get rid of null bytes
4192  5FF6
4194  0606  DEC  R6           ** Done yet?
4196  16FC  JNE  >4190        ** No?, then jump
4198  06A0  BL   @>4480       ** Check if controller busy and
419A  4480                    ** read status
419C  0AD0  SLA  R0,13        ** ID not found error?
419E  180E  JOC  >41BC        ** Yes?, then jump
41A0  1110  JLT  >41C2        ** CRC error?, then jump
41A2  0A20  SLA  R0,2         ** Lost data error?
41A4  1811  JOC  >41C8        ** Yes?, then jump
41A6  9147  CB   R7,R5        ** Compare side #'s
41A8  1312  JEQ  >41CE        ** Same?, then jump
41AA  06A0  BL   @>45AC       ** Report error
41AC  45AC
41AE  0600  DATA >0600        ** Device error
41B0  06A0  BL   @>4590       ** Error handling routine
41B2  4590
```

```
41B4   1100   DATA >1100
41B6   06A0   BL   @>45AC          ** Report error
41B8   45AC
41BA   0700   DATA >0700           ** File error
41BC   06A0   BL   @>4590          ** Error handling routine
41BE   4590
41C0   2100   DATA >2100
41C2   06A0   BL   @>4590          ** Error handling routine
41C4   4590
41C6   2200   DATA >2200
41C8   06A0   BL   @>4590          ** Error handling routine
41CA   4590
41CC   2300   DATA >2300
41CE   D801   MOVB R1,@>5FFC       ** Put sector # in sector register
41D0   5FFC
41D2   C0A9   MOV  @>004E(R9),R2   ** Get address of VDP buffer in R2
41D4   004E
41D6   D029   MOVB @>004D(R9),R0   ** Get read/write flag
41D8   004D
41DA   1340   JEQ  >425C           ** Write command?, then jump

** Read sector routine

41DC   06A0   BL   @>4614          ** Set VDPWA to VDP buffer
41DE   4614
41E0   06A0   BL   @>45CA          ** Read command to controller
41E2   45CA
41E4   7700   DATA >7700
41E6   0206   LI   R6,256          ** 256 bytes to read
41E8   0100
41EA   0705   SETO R5
41EC   1D02   SBO  >0002           ** Turn on auto timing
41EE   D029   MOVB @>004D(R9),R0   ** Get flag for read/write
41F0   004D
41F2   1610   JNE  >4214           ** Read?, then jump

** Verify a written sector routine

41F4   04C0   CLR  R0
41F6   D020   MOVB @>5FF6,R0       ** Get byte from disk
41F8   5FF6                        ** (Inverted)
41FA   B02F   AB   @>FBFE(R15),R0  ** add from VDP buffer
41FC   FBFE
41FE   0280   CI   R0,>FF00        ** Bytes equal?
4200   FF00
4202   1615   JNE  >422E          ** No?, then jump
4204   5020   SZCB @>5FF6,R0       ** Compare another byte from disk
4206   5FF6
4208   702F   SB   @>FBFE(R15),R0  ** with byte from the buffer
420A   FBFE
420C   1610   JNE  >422E          ** Not the same?, then jump
420E   0646   DECT R6              ** Done yet?
4210   16F2   JNE  >41F6          ** No?, then jump
4212   100C   JMP  >422C
4214   D020   MOVB @>5FF6,R0       ** Read byte from drive
4216   5FF6
4218   0540   INV  R0              ** Invert it
421A   DBC0   MOVB R0,@>FFFE(R15)  ** Put it in VDP buffer
421C   FFFE
421E   D020   MOVB @>5FF6,R0       ** Read byte from drive
```

```
4220   5FF6
4222   0540   INV  R0                ** Invert it
4224   DBC0   MOVB R0,@>FFFE(R15)    ** Put it in VDP buffer
4226   FFFE
4228   0646   DECT R6                ** Done yet?
422A   16F4   JNE  >4214             ** No?, go read more bytes
422C   04C5   CLR  R5
422E   06A0   BL   @>4480            ** Check if drive busy, read status
4230   4480
4232   0AD0   SLA  R0,13             ** Record not found?
4234   1807   JOC  >4244             ** Yes?, then jump
4236   1109   JLT  >424A             ** CRC error?, then jump
4238   C145   MOV  R5,R5             ** Read 256 bytes?
423A   160D   JNE  >4256             ** No?, then jump
423C   0A20   SLA  R0,2              ** Lost data error?
423E   1808   JOC  >4250             ** Yes?, then jump
4240   0460   B    @>4676            ** Return
4242   4676
4244   06A0   BL   @>4590            ** Error Handling
4246   4590
4248   2100   DATA >2100
424A   06A0   BL   @>4590            ** Error Handling
424C   4590
424E   2200   DATA >2200
4250   06A0   BL   @>4590            ** Error Handling
4252   4590
4254   2300   DATA >2300
4256   06A0   BL   @>4590            ** Error Handling
4258   4590
425A   2800   DATA >2800

** Write sector routine

425C   06A0   BL   @>461E            ** Set VDPWA to VDP buffer
425E   461E
4260   06A0   BL   @>45CA            ** Send write command to controller
4262   45CA
4264   5700   DATA >5700
4266   0206   LI   R6,>0100          ** 256 bytes to write
4268   0100
426A   1D02   SBO  >0002             ** Turn on auto timing
426C   D02F   MOVB @>FBFE(R15),R0    ** Read byte from VDP buffer
426E   FBFE
4270   0540   INV  R0                ** Invert it for card
4272   D800   MOVB R0,@>5FFE         ** Write it to drive
4274   5FFE
4276   D02F   MOVB @>FBFE(R15),R0    ** Read byte from VDP buffer
4278   FBFE
427A   0540   INV  R0                ** Invert it for card
427C   D800   MOVB R0,@>5FFE         ** Write it to drive
427E   5FFE
4280   0646   DECT R6                ** Done yet?
4282   16F4   JNE  >426C             ** No?, go write some more
4284   06A0   BL   @>4480            ** Check if drive busy, read status
4286   4480
4288   0AB0   SLA  R0,11             ** Write protect error?
428A   1807   JOC  >429A             ** Yes?, then jump
428C   0A20   SLA  R0,2              ** Record not found error?
428E   1808   JOC  >42A0             ** Yes?, then jump
4290   0A20   SLA  R0,2              ** Lost data error?
```

```
4292   1809   JOC  >42A6                ** Yes?, then jump
4294   06A0   BL   @>461E                ** Set VDPWA to VDP buffer
4296   461E
4298   10A3   JMP  >41E0                ** Go read the data back  verify
429A   06A0   BL   @>45AC                ** Report error
429C   45AC
429E   3400   DATA >3400
42A0   06A0   BL   @>4590                ** Error handling
42A2   4590
42A4   3100   DATA >3100
42A6   06A0   BL   @>4590                ** Error handling
42A8   4590
42AA   3300   DATA >3300

** Routine >11 continued

42AC   04E9   CLR  @>004A(R9)            ** Set flag for Single sided
42AE   004A
42B0   D229   MOVB @>004C(R9),R8         ** Get DSR Version and drive #
42B2   004C
42B4   09C8   SRL  R8,12                 ** Put DSR version in LSByte
42B6   1306   JEQ  >42C4                ** Version 0?, then jump
42B8   8808   C    R8,@>4630            ** Version 1?
42BA   4630
42BC   1303   JEQ  >42C4                ** Yes?, then jump
42BE   06A0   BL   @>45AC                ** Report error!
42C0   45AC
42C2   0700   DATA >0700
42C4   5A60   SZCB @>4638,@>004C(R9)     ** Clear out DSR version leave
42C6   4638                              ** drive number
42C8   004C
42CA   9829   CB   @>0051(R9),@>4657     ** Double sided?
42CC   0051
42CE   4657
42D0   1602   JNE  >42D6                ** No?, then jump
42D2   0729   SETO @>004A(R9)           ** Set flag for Double sided
42D4   004A
42D6   DA60   MOVB @>4630,@>0050(R9)     ** Clear out density byte(not
42D8   4630                              ** supported)
42DA   0050
42DC   06A0   BL   @>4496                ** Change drive number
42DE   4496
42E0   06A0   BL   @>4524                ** Restore drive etc..
42E2   4524
42E4   04C3   CLR  R3                    ** Clear out track count
42E6   CA69   MOV  @>004A(R9),@>004A(R9)** Single sided?
42E8   004A
42EA   004A
42EC   1305   JEQ  >42F8                ** Yes?, then jump
42EE   1D07   SBO  >0007                 ** Set to side two of drive
42F0   0207   LI   R7,>0100             ** Identifier byte for side 2 (>01)
42F2   0100
42F4   06A0   BL   @>43AA                ** Write an initialized track
42F6   43AA
42F8   1E07   SBZ  >0007                 ** Set to side one of drive
42FA   04C7   CLR  R7                    ** Identifier byte for side 1 (>00)
42FC   06A0   BL   @>43AA                ** Write an initialized track
42FE   43AA
4300   06A0   BL   @>45CA                ** Step in head
4302   45CA
```

```
4304   A500   DATA >A500
4306   06A0   BL   @>4482                ** Check busy drive
4308   4482
430A   0223   AI   R3,>0100              ** Add 1 to track count
430C   0100
430E   9A43   CB   R3,@>004D(R9)         ** Done with # of tracks we want?
4310   004D
4312   16E9   JNE  >42E6                ** No?, then go write another!
4314   CA69   MOV  @>004A(R9),@>004A(R9)** Single sided?
4316   004A
4318   004A
431A   132F   JEQ  >437A                ** Yes?, then jump
431C   1D07   SBO  >0007                ** Set drive to side 2
431E   0204   LI   R4,>000A
4320   000A
4322   06A0   BL   @>4524                ** Restore head etc...
4324   4524
4326   C0A9   MOV  @>004E(R9),R2         ** Get address of buffer
4328   004E
432A   06A0   BL   @>4614                ** Set VDPWA to it
432C   4614
432E   06A0   BL   @>45CA                ** Read address command
4330   45CA
4332   3F00   DATA >3F00
4334   0206   LI   R6,>0006              ** Read 3 words
4336   0006
4338   1D02   SBO  >0002                ** Turn on auto timing
433A   D020   MOVB @>5FF6,R0             ** Read the address
433C   5FF6
433E   0540   INV  R0                   ** Invert it
4340   DBC0   MOVB R0,@>FFFE(R15)       ** Put it in VDP buffer
4342   FFFE
4344   D020   MOVB @>5FF6,R0             ** Get next byte
4346   5FF6
4348   0540   INV  R0                   ** Invert it
434A   DBC0   MOVB R0,@>FFFE(R15)       ** Put it in VDP buffer
434C   FFFE
434E   0646   DECT R6                    ** Done yet?
4350   16F4   JNE  >433A                ** No?, get more bytes
4352   06A0   BL   @>4480               ** Check busy drive, read status
4354   4480
4356   0AD0   SLA  R0,13                ** ID not found error?
4358   181F   JOC  >4398                ** Yes?, then jump
435A   1121   JLT  >439E                ** CRC error?, then jump
435C   0A20   SLA  R0,2                 ** Lost data error?
435E   1822   JOC  >43A4                ** Yes?, then jump
4360   C0A9   MOV  @>004E(R9),R2        ** Put VDP buffer pointer in R2
4362   004E
4364   0582   INC  R2                   ** Point to side #
4366   06A0   BL   @>461E               ** Set VDPWA to it
4368   461E
436A   04C0   CLR  R0
436C   D02F   MOVB @>FBFE(R15),R0       ** Get the side # from VDP buffer
436E   FBFE
4370   1304   JEQ  >437A                ** Side #1?, then jump
4372   D029   MOVB @>004D(R9),R0        ** Get # tracks into R0
4374   004D
4376   0A10   SLA  R0,1                 ** Multiply by 2 (2 sides times
                                        ** number of tracks per side)
4378   1005   JMP  >4384
```

```
437A   DA60   MOVB @>4631,@>0051(R9)    ** Put single sided (>01) at >8351
437C   4631
437E   0051
4380   D029   MOVB @>004D(R9),R0        ** Put # of tracks in R0
4382   004D
4384   0980   SRL  R0,8                 ** Put it in 16 bits
4386   3820   MPY  @>4632,R0           ** Multiply by 9(sectors per track)
4388   4632
438A   CA41   MOV  R1,@>004A(R9)        ** Move total # sectors to >834A
438C   004A
438E   DA60   MOVB @>4633,@>004D(R9)    ** Put 9 at >834D (Sectors per
4390   4633                            ** track)
4392   004D
4394   0460   B    @>4676               ** Return!
4396   4676
4398   06A0   BL   @>4590               ** Error handling
439A   4590
439C   2101   DATA >2101
439E   06A0   BL   @>4590               ** Error handling
43A0   4590
43A2   2201   DATA >2201
43A4   06A0   BL   @>4590               ** Error handling
43A6   4590
43A8   2301   DATA >2301

** Subroutine to set up the VDP buffer
** and write data to initialize a track

43AA   C20B   MOV  R11,R8               ** Save the return address
43AC   C0A9   MOV  @>004E(R9),R2        ** Put address of buffer in R2
43AE   004E
43B0   06A0   BL   @>4614               ** Set VDPWA to it
43B2   4614
43B4   0206   LI   R6,22                ** Write 22 >00's (Start of track)
43B6   0016
43B8   04C2   CLR  R2
43BA   1002   JMP  >43C0
43BC   0206   LI   R6,6                 ** Write 6 >00's (Sync between
43BE   0006                            ** sectors)
43C0   DBE0   MOVB @>4630,@>FFFE(R15)   ** Write a >00 to the buffer
43C2   4630
43C4   FFFE
43C6   0606   DEC  R6                   ** Done yet?
43C8   16FB   JNE  >43C0                ** No?, write another
43CA   DBE0   MOVB @>4639,@>FFFE(R15)   ** Write a >FE (ID mark single
43CC   4639                            ** density)
43CE   FFFE
43D0   1000   NOP                      ** Wait
43D2   DBC3   MOVB R3,@>FFFE(R15)       ** Write track # to buffer
43D4   FFFE
43D6   1000   NOP                      ** Wait
43D8   DBC7   MOVB R7,@>FFFE(R15)       ** Write Side # byte (>00 or >01)
43DA   FFFE
43DC   D003   MOVB R3,R0                ** Move track # to R0
43DE   0980   SRL  R0,8                 ** Put it in 16 bits
43E0   06C7   SWPB R7                   ** Put side # in 16 bits
43E2   3827   MPY  @>4635(R7),R0
43E4   4635
43E6   06C7   SWPB R7                   ** Put side # back in MSByte
43E8   A042   A    R2,R1
```

```
43EA   3C20   DIV  @>4632,R0            ** Divide by 9
43EC   4632
43EE   DBE1   MOVB @>464F(R1),@>FFFE(R15)** Write sector address to buffer
43F0   464F
43F2   FFFE

** The next 4 lines of code put the following in the buffer
** >01        Sector length
** >F7        2 CRC's
** 11,>FF's   Data seperator
** 6,>00's    Sync between sectors
** >FB        Data address mark

43F4   0206   LI   R6,-20                ** 20 bytes to write
43F6   FFEC
43F8   DBE6   MOVB @>464E(R6),@>FFFE(R15)** Move it to buffer
43FA   464E
43FC   FFFE
43FE   0586   INC  R6                    ** Increment the pointer
4400   16FB   JNE  >43F8                ** Not done?, then jump
4402   0200   LI   R0,>E5E5                  ** Init byte for data block
4404   E5E5                                  ** (>E5)
4406   06A0   BL   @>4474                    ** Write 256 >E5 Bytes
4408   4474                                  ** to buffer
440A   0100   DATA 256
440C   DBE0   MOVB @>464E,@>FFFE(R15)        ** Put in a >F7 (2 CRC's)
440E   464E
4410   FFFE
4412   0700   SETO R0                       ** Data seperator byte (>FF)
4414   06A0   BL   @>4474                    ** Write 45 >FF bytes to buff.
4416   4474
4418   002D   DATA 45
441A   0582   INC  R2                       ** Add 1 to # of sectors
441C   0282   CI   R2,9                     ** 9 Sectors yet?
441E   0009
4420   16CD   JNE  >43BC                    ** No?, go do another
4422   06A0   BL   @>4474                    ** Write 231 >FF bytes to
4424   4474                                  ** buffer (End of track fill)
4426   00E7   DATA 231
4428   0204   LI   R4,>0003                 ** 3 tries at writing before
442A   0003                                  ** error
442C   D0A9   MOVB @>004E(R9),R2            ** Address of buffer into R2
442E   004E
4430   06A0   BL   @>461E                    ** Set VDPWA to buffer
4432   461E
4434   06A0   BL   @>45CA                    ** Send a write track command
4436   45CA                                  ** to controller
4438   0B00   DATA >0B00
443A   0206   LI   R6,>0CA3                 ** 3235 bytes to write
443C   0CA3
443E   1D02   SBO  >0002                    ** Turn on auto timing
4440   D02F   MOVB @>FBFE(R15),R0           ** Read byte from VDP Buffer
4442   FBFE
4444   0540   INV  R0                       ** Invert it for controller
4446   D800   MOVB R0,@>5FFE                ** Write it to controller
4448   5FFE
444A   D02F   MOVB @>FBFE(R15),R0           ** Read byte from VDP Buffer
444C   FBFE
444E   0540   INV  R0                       ** Invert it for controller
4450   D800   MOVB R0,@>5FFE                ** Write it to controller
```

```
4452   5FFE
4454   0646   DECT R6                        ** Done yet?
4456   15F4   JGT  >4440                     ** No, do it again!
4458   06A0   BL   @>4480                    ** Get status byte from
445A   4480                                  ** controller
445C   0AB0   SLA  R0,11                     ** Write protected?
445E   1702   JNC  >4464                     ** No?, then jump
4460   0460   B    @>429A                    ** Go return error!
4462   429A
4464   0A40   SLA  R0,4                      ** Lost data?
4466   1705   JNC  >4472                    ** No?, then jump
4468   0604   DEC  R4                       ** Tried 3 times?
446A   16E0   JNE  >442C                     ** No?, then try writing again
446C   06A0   BL   @>45AC                    ** Return error
446E   45AC
4470   3300   DATA >3300
4472   0458   B    *R8                      ** Return!
4474   C1BB   MOV  *R11+,R6                  ** Get # times to write byte
4476   DBC0   MOVB R0,@>FFFE(R15)            ** Write byte to buffer
4478   FFFE
447A   0606   DEC  R6                       ** Done yet?
447C   16FC   JNE  >4476                     ** No?, then write another
447E   045B   B    *R11                     ** Return

** Routine to check if drive busy and to read status of controller

4480   1E02   SBZ  >0002                     ** Turn off auto timing
4482   D020   MOVB @>5FF0,R0                 ** Read the status of the
4484   5FF0                                  ** controller
4486   0540   INV  R0                       ** Invert it
4488   1103   JLT  >4490                     ** Ready? (Disk there?),
                                              ** No?, then jump
448A   0B90   SRC  R0,9                      ** Drive busy?
448C   18FA   JOC  >4482                     ** Yes?, then jump
448E   045B   B    *R11                     ** Return
4490   06A0   BL   @>45AC                    ** Return error
4492   45AC
4494   0600   DATA >0600                     ** Device error

** Routine to change Drive #

4496   C1CB   MOV  R11,R7                    ** Save Return address
4498   C0A9   MOV  @>0058(R9),R2             ** Pointer to Volume Info
449A   0058                                  ** Block
449C   0222   AI   R2,-10                    ** Point to last drv. accessed
449E   FFF6
44A0   06A0   BL   @>461E                    ** Set VDPWA to it
44A2   461E
44A4   D02F   MOVB @>FBFE(R15),R0            ** Get last drive accessed
44A6   FBFE
44A8   04C5   CLR  R5                        ** Set flag not to change
                                              ** drive #
44AA   9A40   CB   R0,@>004C(R9)             ** Is it same as drive we want
44AC   004C                                  ** to access?
44AE   1301   JEQ  >44B2                     ** Yes?, then jump
44B0   0705   SETO R5                        ** Set flag to change drive #
44B2   04C0   CLR  R0
44B4   D029   MOVB @>004C(R9),R0             ** Get drive # we want into R0
44B6   004C
44B8   1332   JEQ  >451E                     ** Drive 0?, then jump to err.
```

```
44BA   06A0   BL   @>4614                  ** Set VDPWA to last drive
44BC   4614                                ** accessed
44BE   DBC0   MOVB R0,@>FFFE(R15)          ** Write the new one there
44C0   FFFE
44C2   06C0   SWPB R0                      ** Put it in low byte
44C4   0280   CI   R0,3                    ** Check if bigger than drv. 3
44C6   0003
44C8   1B2A   JH   >451E                   ** Yes?, then jump to file err
44CA   0202   LI   R2,>0080                ** Control byte
44CC   0080
44CE   0A02   SLA  R2,0                    ** Shift it # of times
                                           ** in R0 to turn on right drv.
44D0   022C   AI   R12,8                   ** Add 8 to CRU offset
44D2   0008
44D4   C145   MOV  R5,R5                   ** Check to see if we need to
                                           ** change drive #
44D6   1317   JEQ  >4506                   ** No?, then jump
44D8   30E0   LDCR @>4505,3                ** Set drives so none on
44DA   4505
44DC   022C   AI   R12,-6                  ** Offset to find what drives
44DE   FFFA                                ** available
44E0   34C0   STCR R0,3                    ** Get the bits
44E2   022C   AI   R12,6                   ** Put offset back on
44E4   0006
44E6   2402   CZC  R2,R0                   ** Is the drive we want there?
44E8   130E   JEQ  >4506                   ** Yes?, then jump
44EA   04C0   CLR  R0                      ** Set to no drive accessed
44EC   C0A9   MOV  @>0058(R9),R2           ** Get pointer to VIB
44EE   0058
44F0   0222   AI   R2,-10                  ** Point to last drive
44F2   FFF6                                ** accessed
44F4   06A0   BL   @>4614                  ** Set VDPWA to it
44F6   4614
44F8   DBC0   MOVB R0,@>FFFE(R15)          ** Put a zero there!
44FA   FFFE
44FC   022C   AI   R12,-8                  ** Restore CRU base
44FE   FFF8
4500   06A0   BL   @>45AC                  ** Report error
4502   45AC
4504   0600   DATA >0600                   ** Device error
4506   30C2   LDCR R2,3                    ** Select the drive!
4508   022C   AI   R12,-8                  ** Resore CRU base
450A   FFF8
450C   C145   MOV  R5,R5                   ** Did we change drive #?
450E   1306   JEQ  >451C                   ** No?, then jump
4510   0200   LI   R0,3000                 ** Delay loop!
4512   0BB8
4514   0B45   SRC  R5,4                    ** Waste time
4516   0B45   SRC  R5,4
4518   0600   DEC  R0                     ** Done delaying?
451A   16FC   JNE  >4514                   ** No?, jump and delay more
451C   0457   B    *R7                    ** Return

** File Error

451E   06A0   BL   @>45AC                  ** Report error
4520   45AC
4522   0700   DATA >0700                   ** File error
4524   C20B   MOV  R11,R8                  ** Save return address
4526   06A0   BL   @>45CA                  ** Write restore command
```

```
4528   45CA
452A   F500    DATA >F500
452C   06A0    BL   @>4482                 ** Wait for drive not busy
452E   4482
4530   06A0    BL   @>4544                 ** Check for track 0
4532   4544
4534   06A0    BL   @>45F0                 ** Get pointer to last track
4536   45F0                                ** accessed
4538   06A0    BL   @>4614                 ** Set VDPWA for a write to it
453A   4614
453C   DBE0    MOVB @>4640,@>FFFE(R15)     ** Put a >FF there
453E   4640
4540   FFFE
4542   0458    B    *R8                    ** Return

** Routine to check for track 0

4544   D020    MOVB @>5FF0,R0              ** Read status of controller
4546   5FF0
4548   0540    INV  R0                     ** Invert it
454A   0A60    SLA  R0,6                   ** On track 0?
454C   1802    JOC  >4552                  ** Yes?, then jump
454E   0460    B    @>4490                 ** Report device error!
4550   4490
4552   045B    B    *R11                   ** Return
4554   022C    AI   R12,8                  ** Point to drive selection
4556   0008
4558   3120    LDCR @>4630,4               ** Unselect all drives
455A   4630                                ** (Set side to 1?)
455C   022C    AI   R12,-8                 ** Take off the offset
455E   FFF8
4560   1E01    SBZ  >0001                  ** Turn on drive motor
4562   1D01    SBO  >0001
4564   D820    MOVB @>45BC,@>5FF8          ** Force interrupt to drive
4566   45BC
4568   5FF8
456A   C0A9    MOV  @>0058(R9),R2          ** Point to Volume info block
456C   0058
456E   0222    AI   R2,-10                 ** Point to disk drive area
4570   FFF6
4572   06A0    BL   @>4614                 ** Set VDPWA to it
4574   4614
4576   0200    LI   R0,>0004               ** Going to zero out 4 bytes
4578   0004                                ** (Last drive and track
                                           ** accessed  on all drives)
457A   DBC0    MOVB R0,@>FFFE(R15)         ** Write a zero to VDP
457C   FFFE
457E   0600    DEC  R0                     ** Done yet?
4580   16FC    JNE  >457A                  ** No?, do again
4582   06A0    BL   @>4480                 ** Read status, check for
4584   4480                                ** drive busy
4586   DA60    MOVB @>4630,@>0050(R9)      ** Clear the error byte
4588   4630
458A   0050
458C   0460    B    @>4676                 ** Return
458E   4676

** Error handling subroutine, # of times left to try again in R4, Error
code is in word after calling instruction
```

```
4590   0604   DEC  R4                    ** Too many tries?
4592   130C   JEQ  >45AC                 ** Yes?, then jump
4594   C2DB   MOV  *R11,R11               ** Get DATA statement
4596   0B1B   SRC  R11,1                  ** Routine >11?
4598   1702   JNC  >459E                  ** No?, then jump
459A   0460   B    @>4322                 ** Go retry routine >11
459C   4322
459E   0284   CI   R4,>0005               ** Greater than 5 tries left?
45A0   0005
45A2   1B02   JH   >45A8                  ** Yes?, then jump
45A4   06A0   BL   @>4524                 ** Restore the drive
45A6   4524
45A8   0460   B    @>40EC                 ** Go retry routine >10
45AA   40EC

** Routine to return error codes

45AC   C01B   MOV  *R11,R0                ** Get error code into R0
45AE   DA40   MOVB R0,@>0050(R9)          ** Put it into error byte
45B0   0050                               ** in CPU RAM
45B2   0280   CI   R0,>0600               ** Is it Device error?
45B4   0600
45B6   1607   JNE  >45C6                  ** Yes?, then jump
45B8   06A0   BL   @>45CA                 ** Send interrupt command to
45BA   45CA                               ** drive
45BC   2F00   DATA >2F00
45BE   D020   MOVB @>5FF0,R0              ** Read status
45C0   5FF0
45C2   0B90   SRC  R0,9                   ** Is controller busy?
45C4   17FC   JNC  >45BE                  ** Yes?, then jump and try to
                                          ** interrupt it again
45C6   0460   B    @>4676                 ** Return
45C8   4676

** Subroutine to write a command to the controller card

45CA   C03B   MOV  *R11+,R0               ** Get command into R0
45CC   D1A0   MOVB @>5FF0,R6              ** Read the status byte
45CE   5FF0
45D0   0A16   SLA  R6,1                   ** Check to see if drive ready
45D2   1E01   SBZ  >0001                  ** Turn on drive motor
45D4   1D01   SBO  >0001
45D6   1806   JOC  >45E4                  ** It's ready?, then jump
45D8   0206   LI   R6,30000               ** Load # times to delay
45DA   7530
45DC   0B45   SRC  R5,4                   ** Waste time
45DE   0B45   SRC  R5,4
45E0   0606   DEC  R6                    ** Done yet?
45E2   16FC   JNE  >45DC                  ** No?, go waste some more
45E4   D800   MOVB R0,@>5FF8              ** Write command to controller
45E6   5FF8
45E8   1D03   SBO  >0003                  ** Load the head
45EA   0B85   SRC  R5,8                   ** Waste time
45EC   0B85   SRC  R5,8
45EE   045B   B    *R11                   ** Return

** Subroutine to set VDPWA to last track accessed on drive # in >834C

45F0   04C0   CLR  R0
45F2   D029   MOVB @>004C(R9),R0          ** Put drive # in R0
```

```
45F4   004C
45F6   130B   JEQ  >460E                  ** Is it zero?, then jump
45F8   06C0   SWPB R0                     ** Put it in 16 bits
45FA   0280   CI   R0,3                   ** Is it greater than 3?
45FC   0003
45FE   1B07   JH   >460E                  ** Yes?, then jump
4600   C0A9   MOV  @>0058(R9),R2          ** Get pointer to VIB
4602   0058
4604   0222   AI   R2,-10                 ** Point to last drive number
4606   FFF6                              ** accessed
4608   A080   A    R0,R2                 ** Add drive # to point to
                                          ** last track accessed
460A   0460   B    @>461E
460C   461E
460E   06A0   BL   @>45AC                 ** Report error
4610   45AC
4612   0700   DATA >0700                 ** File error
4614   0262   ORI  R2,>4000              ** Set VDPWA for a write
4616   4000                              ** from R2
4618   0242   ANDI R2,>7FFF              ** Mask off MSBit
461A   7FFF
461C   1002   JMP >4622
461E   0242   ANDI R2,>3FFF              ** Set VDPWA for a read
4620   3FFF                              ** from R2
4622   06C2   SWPB R2                    ** LSByte first!
4624   D7C2   MOVB R2,*R15               ** Write it to VDPWA
4626   06C2   SWPB R2                    ** Now MSByte
4628   D7C2   MOVB R2,*R15               ** Write it to VDPWA
462A   0242   ANDI R2,>3FFF              ** Mask it off again
462C   3FFF
462E   045B   B    *R11                   ** Return!

** Data for formatting the disk

4630   0001   DATA >0001
4632   0009   DATA >0009
4634   0006   DATA >0006
4636   0003   DATA >0003
4638   F0FE   DATA >F0FE
463A   01F7   DATA >01F7
463C   FFFF   DATA >FFFF
463E   FFFF   DATA >FFFF
4640   FFFF   DATA >FFFF
4642   FFFF   DATA >FFFF
4644   FFFF   DATA >FFFF
4646   FF00   DATA >FF00
4648   0000   DATA >0000
464A   0000   DATA >0000
464C   00FB   DATA >00FB
464E   F700   DATA >F700
4650   0705   DATA >0705
4652   0301   DATA >0301
4654   0806   DATA >0806
4656   0402   DATA >0402

** Routine to execute a subroutine in a data statement after pushing the
** return address onto the VDP stack

4658   0669   DECT @>0066(R9)              ** Decrement stack pointer
465A   0066
```

```
465C   C2A9   MOV  @>0066(R9),R10          ** Move stack pointer to R10
465E   0066
4660   0429   BLWP @>005A(R9)              ** Set VDPWA to top of stack
4662   005A
4664   0143   DATA >0143
4666   C2BB   MOV  *R11+,R10               ** Get the subroutine address
                                           ** to jump to
4668   06CB   SWPB R11                     ** Push return address on the
466A   DBCB   MOVB R11,@>FFFE(R15)         ** stack
466C   FFFE
466E   06CB   SWPB R11
4670   DBCB   MOVB R11,@>FFFE(R15)
4672   FFFE
4674   045A   B    *R10                   ** Jump to routine to execute

** This routine pops an address off the stack and returns to that address

4676   C2E9   MOV  @>0066(R9),R11          ** Get the pointer to top of
4678   0066                                ** stack
467A   0429   BLWP @>005A(R9)              ** Set VDPWA to it
467C   005A
467E   0162   DATA >0162
4680   D2EF   MOVB @>FBFE(R15),R11         ** Get the address off the
4682   FBFE                                ** stack
4684   06CB   SWPB R11
4686   D2EF   MOVB @>FBFE(R15),R11
4688   FBFE
468A   05E9   INCT @>0066(R9)              ** Increment the stack pointer
468C   0066
468E   045B   B    *R11                   ** Return to address

** BLWP Routines
** Used to manipulate the VDPWA and to store and restore registers from
** the VDP stack area


4690   C28D   MOV  R13,R10                 ** Put old workspace pointer
                                           ** in R10
4692   C23E   MOV  *R14+,R8                ** Get the DATA into R8
4694   C248   MOV  R8,R9                   ** Duplicate it in R9
4696   0949   SRL  R9,4                    ** Get rid of the routine #
4698   0248   ANDI R8,>0003                ** Mask out all but routine #
469A   0003
469C   A208   A    R8,R8                  ** Multiply by 2 for table
                                           ** offset
469E   C2ED   MOV  @>001E(R13),R11         ** Put old R15 into R11(VDPWA)
46A0   001E
46A2   C228   MOV  @>46A8(R8),R8           ** Get address of routine from
46A4   46A8                                ** the table
46A6   0458   B    *R8                     ** Jump to the routine

** Table for subroutines

46A8   46B0   DATA >46B0                   ** Save regs on stack
46AA   46DC   DATA >46DC                   ** Restore regs from stack
46AC   4712   DATA >4712                   ** Set VDPWA for a read
46AE   4708   DATA >4708                   ** Set VDPWA for a write
                                           ** Routine 0
46B0   022A   AI   R10,>0016               ** Point to old workspace R11
46B2   0016
```

```
46B4   0919   SRL  R9,1              ** Check if we should save it
46B6   1710   JNC  >46D8             ** No?, then jump
46B8   064C   DECT R12               ** Subtract 2 from stack pntr.
46BA   C21A   MOV  *R10,R8           ** Get value from register
46BC   06CC   SWPB R12               ** Set VDPWA for a write
46BE   D6CC   MOVB R12,*R11
46C0   06CC   SWPB R12
46C2   026C   ORI  R12,>4000
46C4   4000
46C6   D6CC   MOVB R12,*R11
46C8   06C8   SWPB R8                ** Switch bytes
46CA   DAC8   MOVB R8,@>FFFE(R11)    ** Write LSByte first to stack
46CC   FFFE
46CE   06C8   SWPB R8                ** Switch it again
46D0   DAC8   MOVB R8,@>FFFE(R11)    ** Write MSByte to stack
46D2   FFFE
46D4   C249   MOV  R9,R9             ** Done saving yet?
46D6   1317   JEQ  >4706             ** Yes?, then jump
46D8   064A   DECT R10               ** Point to next reg of old
46DA   10EC   JMP  >46B4             ** old WSP
                                     ** Routine 1
46DC   0A49   SLA  R9,4              ** Move data into 3 MSNybbles
46DE   0A19   SLA  R9,1              ** Restore register?
46E0   1710   JNC  >4702             ** No?, then jump
46E2   06CC   SWPB R12               ** Set up VDPWA to read word
46E4   D6CC   MOVB R12,*R11
46E6   06CC   SWPB R12
46E8   024C   ANDI R12,>3FFF         ** This is a read!!
46EA   3FFF
46EC   D6CC   MOVB R12,*R11
46EE   1000   NOP
46F0   D22B   MOVB @>FBFE(R11),R8    ** Get byte from stack
46F2   FBFE
46F4   06C8   SWPB R8                ** Swap em
46F6   D22B   MOVB @>FBFE(R11),R8    ** Get next byte from stack
46F8   FBFE
46FA   C688   MOV  R8,*R10           ** Put it in reg. of old WSP
46FC   05CC   INCT R12               ** Add 2 to stack pointer
46FE   C249   MOV  R9,R9             ** Done restoring registers?
4700   1302   JEQ  >4706             ** Yes?, then jump
4702   05CA   INCT R10               ** Point to next reg of old
4704   10EC   JMP  >46DE             ** old WSP
4706   0380   RTWP                  ** Return from BLWP inst.
                                     ** Routine 2
4708   A24D   A    R13,R9           ** Add register offset into
                                     ** old WSP
470A   C219   MOV  *R9,R8           ** Get value from register
470C   0268   ORI  R8,>4000         ** Set bit for a write
470E   4000
4710   1004   JMP  >471A
                                     ** Routine 3
4712   A24D   A    R13,R9           ** Add register offset into
                                     ** old WSP
4714   C219   MOV  *R9,R8           ** Get value from register
4716   0248   ANDI R8,>3FFF         ** Clear bit for a read
4718   3FFF
471A   06C8   SWPB R8                ** Switch bytes
471C   D6C8   MOVB R8,*R11          ** Write it to VDPWA
471E   06C8   SWPB R8                ** Switch bytes
4720   D6C8   MOVB R8,*R11          ** Write it to VDPWA
```

```
4722   0380   RTWP                    ** Return from BLWP inst.

** Init sequence for entry into card.  Initializes BLWP instruction and
** registers used by the disk controller routines.

4724   05C7   INCT R7                 ** Increment return addr. from
                                      ** the disk controller card
4726   C28B   MOV  R11,R10            ** Save return addr. of subrtn
4728   02A9   STWP R9                 ** Put WS pointer in R9
472A   0229   AI   R9,>FF20           ** Set R9 to base of >8300
472C   FF20
472E   0200   LI   R0,>4690           ** Entry point for BLWP routn.
4730   4690
4732   CA40   MOV  R0,@>005C(R9)      ** Put it at >835C
4734   005C
4736   C009   MOV  R9,R0              ** Put >8300 in R0
4738   0220   AI   R0,>004E           ** Put WSP for BLWP routine
473A   004E                           ** in R0, address >834E
473C   CA40   MOV  R0,@>005A(R9)      ** Put it at >835A
473E   005A
4740   C229   MOV  @>0070(R9),R8      ** Get highest available
4742   0070                           ** address in VDP memory
4744   05C8   INCT R8                 ** Point to pointer to top of
                                      ** VDP in DVA area
4746   06A0   BL   @>4B76             ** Read it from VDP into R0
4748   4B76
474A   C088   MOV  R8,R2              ** Put pointer into R2
474C   C200   MOV  R0,R8              ** Put top of VDP address, R0
474E   D06F   MOVB @>FBFE(R15),R1     ** Get CRU base identifier
4750   FBFE                           ** from DVA
4752   904C   CB   R12,R1            ** Does it equal CRU base, R12
4754   16F7   JNE  >4744             ** No?, then jump
4756   0228   AI   R8,-266           ** Point to Volume Info Block
4758   FEF6
475A   CA48   MOV  R8,@>0058(R9)      ** Put address to VIB in >8358
475C   0058
475E   0228   AI   R8,-10            ** Point to Disk Drive Info
4760   FFF6                           ** area
4762   CA48   MOV  R8,@>0066(R9)      ** Put address to DDI at >8366
4764   0066
4766   0429   BLWP @>005A(R9)         ** Save R7 on the stack
4768   005A
476A   0100   DATA >0100
476C   C1E9   MOV  @>0056(R9),R7      ** Pointer to byte past name
476E   0056                           ** length
4770   C0C7   MOV  R7,R3              ** Put it in R3
4772   61E9   S    @>0054(R9),R7      ** Subtract the device length
4774   0054
4776   CA42   MOV  R2,@>0056(R9)      ** Move pointer to DVA
4778   0056                           ** to >8356
477A   0607   DEC  R7                 ** Point to device length
477C   04C2   CLR  R2
477E   0429   BLWP @>005A(R9)         ** Set VDPWA to device length
4780   005A
4782   00E2   DATA >00E2
4784   D0AF   MOVB @>FBFE(R15),R2     ** Read it from VDP PAB
4786   FBFE
4788   06C2   SWPB R2
478A   60A9   S    @>0054(R9),R2      ** Make R2 hold name length +1
478C   0054                           ** for the period (.TEST=5)
```

```
478E   0227   AI   R7,-9                  ** Point to opcode of PAB
4790   FFF7
4792   CA47   MOV  R7,@>0054(R9)           ** Put address at >8354
4794   0054
4796   045A   B    *R10                   ** Return
4798   06A0   BL   @>4658                 ** See if file is already on
479A   4658                               ** disk
479C   4E02   DATA >4E02

** Subroutine to create a file on the disk.

479E   C104   MOV  R4,R4                  ** Is the file on the disk?
47A0   1604   JNE  >47AA                  ** No?, then jump
47A2   06A0   BL   @>4658                 ** Yes?, delete it first!
47A4   4658
47A6   48DE   DATA >48DE
47A8   1045
47AA   06A0   BL   @>4B0A                 ** Open up a spot in the
47AC   4B0A                               ** directory link
47AE   0701   SETO R1                     ** Flag for a file descriptor
                                          ** sector
47B0   06A0   BL   @>4658                 ** Get correct VIB, find
47B2   4658                               ** first available sector on
47B4   4EF6   DATA >4EF6                  ** the disk
47B6   C000   MOV  R0,R0                  ** Found available sector?
47B8   1603   JNE  >47C0                  ** Yes?, then jump
47BA   06A0   BL   @>4C72                 ** Report error
47BC   4C72
47BE   8000   DATA >8000                  ** Disk full
47C0   0429   BLWP @>005A(R9)             ** Set VDPWA to word in the
47C2   005A                               ** Directory link
47C4   0103   DATA >0103
47C6   DBC0   MOVB R0,@>FFFE(R15)         ** Write MSByte of sector #
47C8   FFFE
47CA   06C0   SWPB R0
47CC   DBC0   MOVB R0,@>FFFE(R15)         ** Write LSByte of sector #
47CE   FFFE
47D0   06C0   SWPB R0
47D2   C069   MOV  @>0056(R9),R1          ** Put FCB pointer in R1
47D4   0056
47D6   0221   AI   R1,-4                  ** Point to sector # of File
47D8   FFFC                               ** Descriptor in FCB
47DA   0429   BLWP @>005A(R9)             ** Set VDPWA to it
47DC   005A
47DE   0023   DATA >0023
47E0   DBC0   MOVB R0,@>FFFE(R15)         ** Write MSByte of sector #
47E2   FFFE
47E4   06C0   SWPB R0
47E6   DBC0   MOVB R0,@>FFFE(R15)         ** Write LSByte of sector #
47E8   FFFE
47EA   0221   AI   R1,3                   ** Point to drive # of FCB
47EC   0003
47EE   0429   BLWP @>005A(R9)             ** Set VDPWA to it
47F0   005A
47F2   0023   DATA >0023
47F4   DBC6   MOVB R6,@>FFFE(R15)         ** Write the drive #
47F6   FFFE
47F8   04C2   CLR  R2
47FA   0581   INC  R1                     ** Point to start of
                                          ** Descriptor
```

```
47FC   06A0   BL   @>4658                  ** Write the file descriptor
47FE   4658                                 ** to the disk
4800   4D36   DATA >4D36
4802   C169   MOV  @>0058(R9),R5            ** Pointer to Volume Info
4804   0058                                 ** Block
4806   04C4   CLR  R4                      ** Sector # 0
4808   06A0   BL   @>4658                  ** Write the VIB to the disk
480A   4658
480C   4D4E   DATA >4D4E
480E   C0C5   MOV  R5,R3                   ** Put pointer in R3
4810   0223   AI   R3,257                  ** Point to name in Compare
4812   0101                                ** buffer
4814   0202   LI   R2,10                   ** 10 bytes in name
4816   000A
4818   0429   BLWP @>005A(R9)              ** Set VDPWA to it
481A   005A
481C   0062   DATA >0062
481E   D02F   MOVB @>FBFE(R15),R0          ** Read the byte
4820   FBFE
4822   0583   INC  R3                      ** Update pointer
4824   0429   BLWP @>005A(R9)              ** Set VDPWA to val in R1
4826   005A
4828   0023   DATA >0023
482A   DBC0   MOVB R0,@>FFFE(R15)          ** Write the byte
482C   FFFE
482E   0581   INC  R1                      ** Update pointer
4830   0602   DEC  R2                      ** Done yet?
4832   16F2   JNE  >4818                   ** No?, then jump
4834   C069   MOV  @>0056(R9),R1           ** Put FCB pointer in R1
4836   0056
4838   0221   AI   R1,10                   ** Point to FCB after name
483A   000A
483C   0202   LI   R2,246                  ** 246 bytes to clear
483E   00F6
4840   0429   BLWP @>005A(R9)              ** Set VDPWA to it
4842   005A
4844   0023   DATA >0023
4846   DBC2   MOVB R2,@>FFFE(R15)          ** Clear the byte(Write >00)
4848   FFFE
484A   0602   DEC  R2                      ** Done yet?
484C   16FC   JNE  >4846                   ** No?, then jump
484E   1010   JMP  >4870

** Subroutine to check if File Descriptor needs to be written to disk
** and if so writes it.

4850   C169   MOV  @>0056(R9),R5           ** Put FCB pointer in R5
4852   0056
4854   0429   BLWP @>005A(R9)              ** Set VDPWA to name in FCB
4856   005A
4858   00A2   DATA >00A2
485A   D12F   MOVB @>FBFE(R15),R4          ** Read the byte
485C   FBFE
485E   1101   JLT  >4862                   ** Updated file descriptor
                                            ** record?
                                            ** Yes?, then jump
4860   100A   JMP  >4876
4862   0244   ANDI R4,>7F00                ** Mask off the update bit
4864   7F00
4866   0429   BLWP @>005A(R9)              ** Set VDPWA back to name
```

```
4868   005A
486A   00A3   DATA >00A3
486C   DBC4   MOVB R4,@>FFFE(R15)          ** Write the byte
486E   FFFE
4870   04C2   CLR  R2                      ** Write flag
4872   0460   B    @>4D40                  ** Go write the sector and
4874   4D40                                ** return
4876   0460   B    @>4676                  ** Return
4878   4676

**

487A   06A0   BL   @>4658                  ** Check to see if Data Buffer
487C   4658                                ** Area needs to be written
487E   489A   DATA >489A
4880   06A0   BL   @>4658                  ** Check if Update File
4882   4658                                ** Descriptor, and write it
4884   4850   DATA >4850
4886   C169   MOV  @>0056(R9),R5           ** Put FCB pointer in R5
4888   0056
488A   04C6   CLR  R6
488C   0429   BLWP @>005A(R9)              ** Set VDPWA to it
488E   005A
4890   00A3   DATA >00A3
4892   DBC6   MOVB R6,@>FFFE(R15)          ** Clear byte in name
4894   FFFE                                ** (FCB available)
4896   0460   B    @>4CD2                  ** Go see if VIB needs update
4898   4CD2

** Subroutine to write AU in data buffer to disk etc..,
** if updated buffer

489A   C129   MOV  @>0056(R9),R4           ** Put FCB pointer in R4
489C   0056
489E   0604   DEC  R4                      ** Point to the drive #
48A0   0429   BLWP @>005A(R9)              ** Set VDPWA to it
48A2   005A
48A4   0082   DATA >0082
48A6   D06F   MOVB @>FBFE(R15),R1          ** Read the byte
48A8   FBFE
48AA   1101   JLT  >48AE                   ** Updated buffer area of FCB?
                                           ** then jump
48AC   1016   JMP  >48DA
48AE   0241   ANDI R1,>7F00                ** Mask off update bit
48B0   7F00
48B2   0429   BLWP @>005A(R9)              ** Set VDPWA back to drive #
48B4   005A
48B6   0083   DATA >0083
48B8   DBC1   MOVB R1,@>FFFE(R15)          ** Write the byte
48BA   FFFE
48BC   0224   AI   R4,-5                   ** Point to current AU #
48BE   FFFB
48C0   0429   BLWP @>005A(R9)              ** Set VDPWA to it
48C2   005A
48C4   0082   DATA >0082
48C6   D0EF   MOVB @>FBFE(R15),R3          ** Read it
48C8   FBFE
48CA   06C3   SWPB R3
48CC   D0EF   MOVB @>FBFE(R15),R3
48CE   FBFE
```

```
48DØ   Ø6C3   SWPB R3                    ** Adjust it
48D2   C1C4   MOV  R4,R7                  ** Move pointer to R7
48D4   Ø227   AI   R7,262                 ** Point to data buffer of FCB
48D6   Ø1Ø6
48D8   1Ø3A   JMP  >494E
48DA   Ø46Ø   B    @>4676                 ** Return
48DC   4676

** Subroutine to delete data sectors in a file

48DE   CØA9   MOV  @>ØØ56(R9),R2          ** Put FCB pointer in R2
48EØ   ØØ56
48E2   Ø222   AI   R2,12                  ** Point to File status flags
48E4   ØØØC
48E6   Ø429   BLWP @>ØØ5A(R9)             ** Set VDPWA to it
48E8   ØØ5A
48EA   ØØ42   DATA >ØØ42
48EC   DØ2F   MOVB @>FBFE(R15),RØ         ** Read the byte
48EE   FBFE
48FØ   Ø24Ø   ANDI RØ,>Ø8ØØ               ** Write protected?
48F2   Ø8ØØ
48F4   13Ø3   JEQ  >48FC                  ** No?, then jump
48F6   Ø6AØ   BL   @>4C72                 ** Report error
48F8   4C72
48FA   2ØØØ   DATA >2ØØØ                  ** Device is write protected
48FC   C229   MOV  @>ØØ56(R9),R8          ** Put FCB pointer in R8
48FE   ØØ56
49ØØ   Ø228   AI   R8,28                  ** Point to pointer blocks
49Ø2   ØØ1C
49Ø4   Ø7Ø4   SETO R4
49Ø6   Ø6AØ   BL   @>4EB6                 ** Read 3 bytes from pointer
49Ø8   4EB6                               ** block
49ØA   Ø228   AI   R8,3                   ** Point to next 3 bytes of
49ØC   ØØØ3                               ** block
49ØE   CØ41   MOV  R1,R1                  ** End of pointer blocks?
491Ø   13ØC   JEQ  >492A                  ** Yes?, then jump
4912   CØØ2   MOV  R2,RØ                  ** Put sector count in RØ
4914   6Ø84   S    R4,R2                  ** Subtract # in previous
                                          ** blocks
                                          ** (Calculate # AU's in
                                          ** this one)
4916   C1ØØ   MOV  RØ,R4                  ** Put Max AU # so far in R4
4918   Ø6AØ   BL   @>4658                 ** Delete the sectors
491A   4658
491C   4FC2   DATA >4FC2
491E   CØ48   MOV  R8,R1                  ** Put the pointer in R1
492Ø   6Ø69   S    @>ØØ56(R9),R1          ** Subtract the FCB offset
4922   ØØ56
4924   Ø281   CI   R1,256                 ** At end of pointer blocks?
4926   Ø1ØØ
4928   16EE   JNE  >49Ø6                  ** No?, then jump
492A   Ø46Ø   B    @>4676                 ** Return
492C   4676

** Subroutine to read an AU  put it in the buffer

492E   Ø6AØ   BL   @>4B7Ø                 ** Get # sectors currently
493Ø   4B7Ø                               ** allocated(+1 for FDR)
4932   ØØØE   DATA >ØØØE
4934   8ØCØ   C    RØ,R3                  ** Check if # we want > max #
```

```
                                        ** of sectors
4936   1B03   JH   >493E                ** No?, then jump
4938   06A0   BL   @>4C72                ** Report error
493A   4C72
493C   A000   DATA >A000
493E   06A0   BL   @>4658                ** Get Sector cluster of AU
4940   4658                             ** we want
4942   4A4E   DATA >4A4E
4944   A101   A    R1,R4                 ** Add offset
4946   0702   SETO R2                    ** Flag for read sector
4948   C147   MOV  R7,R5                 ** Put Data buffer ptr. in R5
494A   0460   B    @>4D4E                ** Go to low level sector rtn.
494C   4D4E

** Write a new AU for a file

494E   0429   BLWP @>005A(R9)            ** Save R0,R7 on the stack
4950   005A
4952   8100   DATA >8100
4954   06A0   BL   @>4658                ** Calculate sector to write
4956   4658                             ** data to, update pointer
4958   4964   DATA >4964                ** blocks
495A   0429   BLWP @>005A(R9)            ** Restore R7,R0 from stack
495C   005A
495E   8101   DATA >8101
4960   04C2   CLR  R2                    ** Flag for write sector

** Subroutine to calculate sector # and update pointer blocks...

4962   10F2   JMP  >4948                ** Go write sector and return
4964   06A0   BL   @>4B70                ** Get # sectors currently
4966   4B70                             ** allocated
4968   000E   DATA >000E
496A   80C0   C    R0,R3                 ** Compare to # we need to
                                        ** access
496C   1B47   JH   >49FC                ** Don't need to add AU to
                                        ** file?, then jump
496E   C000   MOV  R0,R0                ** 0 sectors in file?
4970   1301   JEQ  >4974                ** Yes?, then jump
4972   0600   DEC  R0                   ** Subtract 1, (For file dir?)
4974   C143   MOV  R3,R5                 ** Put # we need in R5
4976   C0C0   MOV  R0,R3
4978   06A0   BL   @>4658                ** Find Sector Cluster of AU
497A   4658
497C   4A4E   DATA >4A4E
497E   0429   BLWP @>005A(R9)            ** Save R1 on the stack
4980   005A
4982   4000   DATA >4000
4984   A044   A    R4,R1                 ** Add sector offset
4986   C105   MOV  R5,R4
4988   06A0   BL   @>4658                ** Check if VIB needs to be
498A   4658                             ** changed, and find first
498C   4EF6   DATA >4EF6                ** available sector
498E   C000   MOV  R0,R0                ** Sector available?
4990   1604   JNE  >499A                ** Yes?, then jump
4992   C142   MOV  R2,R5
4994   06A0   BL   @>4A08                ** Write 3 bytes to pointer
4996   4A08                             ** blocks
4998   101B   JMP  >49D0                ** Report error
499A   C041   MOV  R1,R1                ** 1st data sector?
```

```
499C   1320   JEQ   >49DE                    ** Yes?, then jump
499E   0581   INC   R1                       ** Add 1 to count
49A0   8040   C     R0,R1
49A2   131C   JEQ   >49DC
49A4   0429   BLWP  @>005A(R9)               ** Restore R1 from the stack
49A6   005A
49A8   4001   DATA  >4001
49AA   0429   BLWP  @>005A(R9)               ** Save R0,R2 on the stack
49AC   005A
49AE   A000   DATA  >A000
49B0   06A0   BL    @>4ED2                   ** Write 3 bytes to the
49B2   4ED2                                  ** pointer block
49B4   0429   BLWP  @>005A(R9)               ** Restore R0,R2 from stack
49B6   005A
49B8   A001   DATA  >A001
49BA   0228   AI    R8,3                     ** Point to next 3 bytes of
49BC   0003                                  ** pointer block
49BE   C048   MOV   R8,R1                    ** Put it in R1
49C0   6069   S     @>0056(R9),R1            ** Subtract the FCB offset
49C2   0056
49C4   0281   CI    R1,256                   ** End of pointer block?
49C6   0100
49C8   1605   JNE   >49D4                    ** No?, then jump
49CA   06A0   BL    @>4658                   ** Delete sector # in R0
49CC   4658
49CE   4FBC   DATA  >4FBC
49D0   0460   B     @>4E28                   ** Report error
49D2   4E28
49D4   C040   MOV   R0,R1
49D6   0429   BLWP  @>005A(R9)               ** Save R1 on the stack
49D8   005A
49DA   4000   DATA  >4000
49DC   1007   JMP   >49EC
49DE   0429   BLWP  @>005A(R9)               ** Restore R1 from the stack
49E0   005A
49E2   4001   DATA  >4001
49E4   0429   BLWP  @>005A(R9)               ** Save R0 on the stack
49E6   005A
49E8   8000   DATA  >8000
49EA   0702   SETO  R2                       ** When Inc'd becomes 0
49EC   C040   MOV   R0,R1
49EE   0582   INC   R2
49F0   8102   C     R2,R4
49F2   16CA   JNE   >4988                    ** Check another sector
49F4   C144   MOV   R4,R5                    ** Move count for subroutine
49F6   06A0   BL    @>4A08                   ** Write 3 bytes to pointer
49F8   4A08                                  ** blocks
49FA   1004   JMP   >4A04
49FC   06A0   BL    @>4658                   ** Find sector cluster of AU
49FE   4658                                  ** wanted
4A00   4A4E   DATA  >4A4E
4A02   A101   A     R1,R4                    ** Add offset??
4A04   0460   B     @>4676                   ** Return
4A06   4676

** Subroutine to write 3 bytes to pointer block

4A08   C28B   MOV   R11,R10                  ** Save return address
4A0A   C101   MOV   R1,R4
4A0C   0429   BLWP  @>005A(R9)               ** Restore R1 from the stack
```

```
4A0E   005A
4A10   4001    DATA >4001
4A12   C041    MOV  R1,R1                 ** New sectors added?
4A14   131B    JEQ  >4A4C                 ** No?, then jump
4A16   06A0    BL   @>4ED2                ** Write 3 bytes to pointer
4A18   4ED2                               ** block
4A1A   C0A9    MOV  @>0056(R9),R2         ** Put FCB pointer in R2
4A1C   0056
4A1E   0429    BLWP @>005A(R9)            ** Set VDPWA to it
4A20   005A
4A22   0042    DATA >0042
4A24   D06F    MOVB @>FBFE(R15),R1        ** Read the byte
4A26   FBFE
4A28   0261    ORI  R1,>8000              ** Set Updated file descriptor
4A2A   8000                               ** bit
4A2C   0429    BLWP @>005A(R9)            ** Set VDPWA back to it
4A2E   005A
4A30   0043    DATA >0043
4A32   DBC1    MOVB R1,@>FFFE(R15)        ** Write it back
4A34   FFFE
4A36   0222    AI   R2,14                 ** Point to number sectors
4A38   000E                               ** currently allocated
4A3A   0585    INC  R5                    ** Add 1 to the count
4A3C   0429    BLWP @>005A(R9)            ** Set VDPWA to it
4A3E   005A
4A40   0043    DATA >0043
4A42   DBC5    MOVB R5,@>FFFE(R15)        ** Write the word
4A44   FFFE
4A46   06C5    SWPB R5
4A48   DBC5    MOVB R5,@>FFFE(R15)
4A4A   FFFE
4A4C   045A    B    *R10                  ** Return

** Find sector cluster of AU we want

4A4E   C229    MOV  @>0056(R9),R8         ** Put FCB pointer in R8
4A50   0056
4A52   0228    AI   R8,>001C              ** Point to Pointer blocks
4A54   001C
4A56   C103    MOV  R3,R4                 ** Put AU # we want in R4
4A58   06A0    BL   @>4EB6                ** Get data from pointer block
4A5A   4EB6
4A5C   80C2    C    R2,R3                 ** Compare highest AU with
                                          ** one we want
4A5E   1101    JLT  >4A62                 ** Not in this cluster?, jump
4A60   1006    JMP  >4A6E
4A62   0228    AI   R8,>0003              ** Point to next 3 bytes of
4A64   0003                               ** block
4A66   C103    MOV  R3,R4                 ** Put AU we want in R4
4A68   6102    S    R2,R4                 ** Subtract AU # in cluster
4A6A   0604    DEC  R4                    ** Subtract 1 more
4A6C   10F5    JMP  >4A58                 ** Go check next cluster
4A6E   0460    B    @>4676                ** Return
4A70   4676

** Delete opcode **

4A72   06A0    BL   @>4658                ** Find correct FCB
4A74   4658
4A76   4DA4    DATA >4DA4
```

```
4A78   C104   MOV  R4,R4                  ** Already open?
4A7A   1307   JEQ  >4A8A                  ** Yes?, then jump
4A7C   06A0   BL   @>4658                 ** Find the file on the disk
4A7E   4658
4A80   4E0C   DATA >4E0C
4A82   C104   MOV  R4,R4                  ** Found the file?
4A84   1313   JEQ  >4AAC                  ** Yes?, then jump
4A86   0460   B    @>4676                 ** Return
4A88   4676
4A8A   0581   INC  R1                    ** Point to File Descriptor
                                          ** in DVA
4A8C   CA41   MOV  R1,@>0056(R9)          ** Put it in the FCB pointer
4A8E   0056
4A90   06A0   BL   @>4658                 ** Read the File Descriptor
4A92   4658                               ** into the FCB
4A94   4D34   DATA >4D34
4A96   06A0   BL   @>4B70                 ** Read Sector # of file
4A98   4B70                               ** descriptor record from FCB
4A9A   FFFC   DATA -4
4A9C   C0C0   MOV  R0,R3                  ** Put it in R3
4A9E   C205   MOV  R5,R8                  ** Put FCB pointer in R8
4AA0   06A0   BL   @>4B76                 ** Read a word from the
4AA2   4B76                               ** Directory Link sector
4AA4   05C8   INCT R8                     ** Point to next word of the
                                          ** Directory Link
4AA6   80C0   C    R0,R3                  ** Equal to sector # of the
                                          ** one to delete?
4AA8   16FB   JNE  >4AA0                  ** No?, then jump
4AAA   0648   DECT R8                     ** Point to word we need to
                                          ** remove
4AAC   0429   BLWP @>005A(R9)             ** Save R8 on the stack
4AAE   005A
4AB0   0080   DATA >0080
4AB2   06A0   BL   @>4658                 ** Delete the data sectors
4AB4   4658                               ** in the file
4AB6   48DE   DATA >48DE
4AB8   0429   BLWP @>005A(R9)             ** Restore R8 from the stack
4ABA   005A
4ABC   0081   DATA >0081
4ABE   06A0   BL   @>4ADA                 ** Move the directory words to
4AC0   4ADA                               ** delete the Directory Link
                                          ** word for this file
4AC2   04C2   CLR  R2                     ** Flag for write sector
4AC4   06A0   BL   @>4658                 ** Write the new Directory
4AC6   4658                               ** Link to the disk
4AC8   4D36   DATA >4D36
4ACA   06A0   BL   @>4B70                 ** Get sector # of the file
4ACC   4B70                               ** descriptor record
4ACE   FFFC   DATA -4
4AD0   06A0   BL   @>4658                 ** Delete the sector
4AD2   4658
4AD4   4FBC   DATA >4FBC
4AD6   0460   B    @>4886                 ** Go close the file and
4AD8   4886                               ** return

** Move bytes in VDP RAM pointed to by R8+2 to data buffer of FCB

4ADA   C169   MOV  @>0056(R9),R5          ** Put FCB pointer in R5
4ADC   0056
4ADE   0225   AI   R5,256                 ** Point to Data buffer of FCB
```

```
4AE0   0100
4AE2   C0C5   MOV  R5,R3                    ** Put it in R3
4AE4   0223   AI   R3,256                   ** Point to 1 past data
4AE6   0100                                 ** buffer area
4AE8   C088   MOV  R8,R2                    ** Get where to move it from
4AEA   05C2   INCT R2                       ** Add 2
4AEC   0429   BLWP @>005A(R9)               ** Set VDPWA to it
4AEE   005A
4AF0   0042   DATA >0042
4AF2   D02F   MOVB @>FBFE(R15),R0           ** Read the byte
4AF4   FBFE
4AF6   0582   INC  R2                       ** Update pointer
4AF8   0429   BLWP @>005A(R9)               ** Set VDPWA to address in FCB
4AFA   005A
4AFC   0103   DATA >0103
4AFE   DBC0   MOVB R0,@>FFFE(R15)           ** Write the byte
4B00   FFFE
4B02   0588   INC  R8                       ** Update pointer
4B04   80C2   C    R2,R3                    ** Done yet?
4B06   16F2   JNE  >4AEC                    ** No?, then jump
4B08   045B   B    *R11                     ** Return

** Subroutine to open up a word in the directory link, R8 points to word

4B0A   C28B   MOV  R11,R10                  ** Save return address
4B0C   C048   MOV  R8,R1                    ** Save pointer in R1
4B0E   06A0   BL   @>4B70                   ** Read last available word
4B10   4B70                                 ** in the directory link
4B12   01FC   DATA >01FC
4B14   C000   MOV  R0,R0                    ** Is there an available word?
4B16   1302   JEQ  >4B1C                    ** Yes?, then jump
4B18   0460   B    @>4E28                   ** Error out of space
4B1A   4E28

** This series of instructions moves words up one word in memory until
** the word for the new file is open

4B1C   0608   DEC  R8                       ** Update pointer
4B1E   0429   BLWP @>005A(R9)               ** Set VDPWA to it
4B20   005A
4B22   0102   DATA >0102
4B24   D02F   MOVB @>FBFE(R15),R0           ** Read the byte
4B26   FBFE
4B28   05C8   INCT R8                       ** Update pointer
4B2A   0429   BLWP @>005A(R9)               ** Set VDPWA to it
4B2C   005A
4B2E   0103   DATA >0103
4B30   DBC0   MOVB R0,@>FFFE(R15)           ** Write the byte
4B32   FFFE
4B34   0648   DECT R8                       ** Update pointer
4B36   8048   C    R8,R1                    ** Done yet?
4B38   16F1   JNE  >4B1C                    ** No?, then jump
4B3A   045A   B    *R10                     ** Return

** Subroutine to check if name pointed to by R1 is the same as the name in
** the compare buffer

4B3C   C0A9   MOV  @>0058(R9),R2            ** Volume info block pointer
4B3E   0058
4B40   0222   AI   R2,256                   ** Point to compare buffer
```

```
4B42   0100
4B44   0429   BLWP @>005A(R9)                  ** Set VDPWA to value in R1
4B46   005A
4B48   0022   DATA >0022
4B4A   D02F   MOVB @>FBFE(R15),R0              ** Read the byte
4B4C   FBFE
4B4E   0240   ANDI R0,>7FFF                   ** Mask off MSBit
4B50   7FFF
4B52   0429   BLWP @>005A(R9)                  ** Set VDPWA to compare buffer
4B54   005A
4B56   0042   DATA >0042
4B58   D0EF   MOVB @>FBFE(R15),R3              ** Read the byte
4B5A   FBFE
4B5C   90C0   CB   R0,R3                      ** Are they equal?
4B5E   1604   JNE  >4B68                      ** No?, then jump
4B60   0581   INC  R1                         ** Update pointers
4B62   0582   INC  R2
4B64   0604   DEC  R4                         ** Done yet?
4B66   16EE   JNE  >4B44                      ** No?, then jump
4B68   045B   B    *R11                       ** Return

** Subroutine to read a word from the PAB **

4B6A   C229   MOV  @>0054(R9),R8              ** Get pointer to start of PAB
4B6C   0054
4B6E   1002   JMP  >4B74

** Subroutine to read a word from the File Control Block

4B70   C229   MOV  @>0056(R9),R8              ** Get pointer to start of FCB
4B72   0056
4B74   A23B   A    *R11+,R8                   ** Add the offset from DATA
4B76   0429   BLWP @>005A(R9)                 ** Set the VDPWA to it
4B78   005A
4B7A   0102   DATA >0102
4B7C   D02F   MOVB @>FBFE(R15),R0             ** Read the word
4B7E   FBFE
4B80   06C0   SWPB R0
4B82   D02F   MOVB @>FBFE(R15),R0
4B84   FBFE
4B86   0B80   SRC  R0,8                       ** Adjust it
4B88   045B   B    *R11                       ** Return

** Subroutine to find correct drive for diskname access

4B8A   C069   MOV  @>0058(R9),R1              ** Volume Info Block pointer
4B8C   0058
4B8E   0221   AI   R1,256                     ** Point to File compare
4B90   0100                                   ** buffer
4B92   06A0   BL   @>4C14                     ** Put disk name in compare
4B94   4C14                                   ** buffer
4B96   0429   BLWP @>005A(R9)                 ** Save R2,R3 on the stack
4B98   005A
4B9A   3000   DATA >3000
4B9C   C000   MOV  R0,R0                      ** MSBit set?
4B9E   1101   JLT  >4BA2                      ** Yes?, then jump
4BA0   1005   JMP  >4BAC
4BA2   C180   MOV  R0,R6                      ** Put the char in R6
4BA4   0246   ANDI R6,>0300                   ** Mask out all but the number
4BA6   0300
```

```
4BA8   1337   JEQ  >4C18                    ** Ø?, then jump
4BAA   102F   JMP  >4C0A
4BAC   04C6   CLR  R6                      ** Clear the drive #
4BAE   06A0   BL   @>4658                  ** Check if VIB needs to be
4BB0   4658                                ** changed
4BB2   4CD2   DATA >4CD2
4BB4   0226   AI   R6,>0100                ** Add 1 to the drive #
4BB6   0100
4BB8   0707   SETO R7                      ** Read flag
4BBA   0702   SETO R2                      ** Read flag
4BBC   04C4   CLR  R4                      ** Sector # Ø
4BBE   06A0   BL   @>4658                  ** Read the sector
4BC0   4658
4BC2   4D50   DATA >4D50
4BC4   C1C7   MOV  R7,R7                   ** Disk there?
4BC6   161D   JNE  >4C02                   ** No?, then jump
4BC8   0204   LI   R4,10                   ** 10 bytes to compare
4BCA   000A
4BCC   C069   MOV  @>0058(R9),R1           ** Volume Info Block pointer
4BCE   0058
4BD0   0429   BLWP @>005A(R9)              ** Save R1,R2,R3 on the stack
4BD2   005A
4BD4   7000   DATA >7000
4BD6   06A0   BL   @>4B3C                  ** Check if disk name same as
4BD8   4B3C                                ** in the compare buffer
4BDA   0429   BLWP @>005A(R9)              ** Restore R3,R2,R1 from stack
4BDC   005A
4BDE   7001   DATA >7001
4BE0   C104   MOV  R4,R4                   ** Are they the same?
4BE2   160F   JNE  >4C02                   ** No?, then jump
4BE4   C006   MOV  R6,R0                   ** Put Drive # in R0
4BE6   0260   ORI  R0,>8000                ** Set MSBit
4BE8   8000
4BEA   0429   BLWP @>005A(R9)              ** Set VDPWA to val in R3
4BEC   005A
4BEE   0063   DATA >0063
4BF0   DBC0   MOVB R0,@>FFFE(R15)          ** Write drive num with MSBit
4BF2   FFFE                                ** set
4BF4   0605   DEC  R5                      ** Back off R5
4BF6   0429   BLWP @>005A(R9)              ** Set VDPWA to it
4BF8   005A
4BFA   00A3   DATA >00A3
4BFC   DBC6   MOVB R6,@>FFFE(R15)          ** Write drive num
4BFE   FFFE
4C00   1004   JMP  >4C0A
4C02   0286   CI   R6,>0300                ** Are we at highest drive #?
4C04   0300
4C06   16D6   JNE  >4BB4                   ** No?, then jump
4C08   1007   JMP  >4C18                   ** Go report error
4C0A   0429   BLWP @>005A(R9)              ** Restore R3,R2 from stack
4C0C   005A
4C0E   3001   DATA >3001
4C10   0460   B    @>4676                  ** Return
4C12   4676

** Subroutine to put disk or file name in compare buffer **

4C14   0602   DEC  R2                      ** Name there?
4C16   1B03   JH   >4C1E                   ** Yes?, then jump
4C18   06A0   BL   @>4C9E                  ** Report error
```

```
4C1A  4C9E
4C1C  E000    DATA >E000                    ** File error
4C1E  0200    LI   R0,>2000                 ** Space character
4C20  2000
4C22  0429    BLWP @>005A(R9)               ** Set VDPWA to compare buffer
4C24  005A
4C26  0023    DATA >0023
4C28  0208    LI   R8,10                    ** 10 bytes to clear in name
4C2A  000A
4C2C  DBC0    MOVB R0,@>FFFE(R15)           ** Write space character
4C2E  FFFE
4C30  0608    DEC  R8                       ** Done yet?
4C32  16FC    JNE  >4C2C                    ** No?, then jump
4C34  0208    LI   R8,11                    ** 11 bytes (Filename + '.')
4C36  000B
4C38  0583    INC  R3                       ** Pointer to name in PAB
4C3A  0429    BLWP @>005A(R9)               ** Set VDPWA to it
4C3C  005A
4C3E  0062    DATA >0062
4C40  D02F    MOVB @>FBFE(R15),R0           ** Read the byte
4C42  FBFE
4C44  13E9    JEQ  >4C18                    ** Is it a >00?, yes? - jump
4C46  1110    JLT  >4C68                    ** MSBit set?, then jump
4C48  0280    CI   R0,>2E00                 ** Is it a '.' character?
4C4A  2E00
4C4C  130D    JEQ  >4C68                    ** Yes?, then jump
4C4E  0608    DEC  R8                       ** Checked all chars?
4C50  13E3    JEQ  >4C18                    ** Yes?, then jump
4C52  0280    CI   R0,>2000                 ** Is it a space characer?
4C54  2000
4C56  13E0    JEQ  >4C18                    ** Yes?, then jump
4C58  0429    BLWP @>005A(R9)               ** Set VDPWA to compare buffer
4C5A  005A
4C5C  0023    DATA >0023
4C5E  DBC0    MOVB R0,@>FFFE(R15)           ** Write the byte
4C60  FFFE
4C62  0581    INC  R1                       ** Update pointer
4C64  0602    DEC  R2                       ** Done yet?
4C66  16E8    JNE  >4C38                    ** No?, then jump
4C68  0288    CI   R8,>000B                 ** First character checked?
4C6A  000B
4C6C  13D5    JEQ  >4C18                    ** Yes?, then jump
4C6E  045B    B    *R11                     ** Return
4C70  AA00    DATA >AA00

** Error reporting routine

4C72  C00B    MOV R11,R0                    ** Save return address
4C74  C0E9    MOV  @>0056(R9),R3            ** Get DVA pointer
4C76  0056
4C78  0603    DEC  R3                       ** Point to disk controller ID
4C7A  0429    BLWP @>005A(R9)               ** Set VDPWA to it
4C7C  005A
4C7E  0062    DATA >0062
4C80  D0AF    MOVB @>FBFE(R15),R2           ** Read it from VDP
4C82  FBFE
4C84  9802    CB   R2,@>4C70                ** Is it equal to >AA?
4C86  4C70
4C88  1309    JEQ  >4C9C                    ** Yes?, then jump
4C8A  0429    BLWP @>005A(R9)               ** Save R0 on the stack
```

```
4C8C   005A
4C8E   8000   DATA >8000
4C90   06A0   BL   @>4658                    ** Close the file
4C92   4658
4C94   487A   DATA >487A
4C96   0429   BLWP @>005A(R9)                ** Restore R0 from the stack
4C98   005A
4C9A   8001   DATA >8001
4C9C   C2C0   MOV  R0,R11                    ** Put return address in R11
4C9E   C069   MOV  @>0054(R9),R1             ** Get the PAB pointer
4CA0   0054
4CA2   130D   JEQ  >4CBE                     ** 0 in the pointer?, jump
4CA4   0581   INC  R1                       ** Point to status byte
4CA6   0429   BLWP @>005A(R9)                ** Set VDPWA to it
4CA8   005A
4CAA   0022   DATA >0022
4CAC   D0AF   MOVB @>FBFE(R15),R2            ** Read the byte
4CAE   FBFE
4CB0   E0BB   SOC  *R11+,R2                  ** Set the error bits
4CB2   0429   BLWP @>005A(R9)                ** Set VDPWA to it
4CB4   005A
4CB6   0023   DATA >0023
4CB8   DBC2   MOVB R2,@>FFFE(R15)            ** Write it back
4CBA   FFFE
4CBC   1002   JMP  >4CC2
4CBE   CA7B   MOV  *R11+,@>0050(R9)          ** Put the error code at >8350
4CC0   0050
4CC2   C229   MOV  @>0058(R9),R8             ** Get VIB pointer
4CC4   0058
4CC6   0228   AI   R8,-12                    ** Point to top of VDP Stack
4CC8   FFF4
4CCA   CA48   MOV  R8,@>0066(R9)             ** Put it in >8366
4CCC   0066
4CCE   0460   B    @>4676                    ** Return
4CD0   4676

** Subroutine to check if the current Volume Info Block needs to be changed

4CD2   0429   BLWP @>005A(R9)                ** Save R2,R3 on the stack
4CD4   005A
4CD6   3000   DATA >3000
4CD8   C169   MOV  @>0058(R9),R5             ** Volume info block pointer
4CDA   0058
4CDC   0605   DEC  R5                        ** Point to VIB status byte
4CDE   0429   BLWP @>005A(R9)                ** Set VDPWA to it
4CE0   005A
4CE2   00A2   DATA >00A2
4CE4   D0AF   MOVB @>FBFE(R15),R2            ** Read the byte
4CE6   FBFE
4CE8   C0C2   MOV  R2,R3                     ** Put it in R3
4CEA   04C4   CLR  R4                        ** Sector #0
4CEC   0242   ANDI R2,>0300                  ** Mask off Update bit
4CEE   0300
4CF0   9182   CB   R2,R6                     ** Same drive #?
4CF2   131A   JEQ  >4D28                     ** Yes?, then jump
4CF4   C0C3   MOV  R3,R3                     ** Updated Volume Info Block?
4CF6   1101   JLT  >4CFA                     ** Yes?, then jump
4CF8   100A   JMP  >4D0E
4CFA   0429   BLWP @>005A(R9)                ** Set VDPWA to VIB status
4CFC   005A                                  ** byte
```

```
4CFE  00A3    DATA >00A3
4D00  DBC4    MOVB R4,@>FFFE(R15)      ** Write a >00 there
4D02  FFFE
4D04  0585    INC  R5                  ** Buffer pointer
4D06  06A0    BL   @>4658              ** Write the Volume Info
4D08  4658                             ** sector to the disk
4D0A  4D52    DATA >4D52
4D0C  0605    DEC  R5                  ** Point to VIB status byte
4D0E  0702    SETO R2                  ** Flag for read sector
4D10  D086    MOVB R6,R2               ** Drive #
4D12  1305    JEQ  >4D1E               ** 0 for drive #?, then jump
4D14  0585    INC  R5                  ** Buffer pointer
4D16  06A0    BL   @>4658              ** Read the sector of the new
4D18  4658                             ** VIB
4D1A  4D50    DATA >4D50
4D1C  0605    DEC  R5                  ** Point to VIB status byte
4D1E  0429    BLWP @>005A(R9)          ** Set VDPWA to it
4D20  005A
4D22  00A3    DATA >00A3
4D24  DBC6    MOVB R6,@>FFFE(R15)      ** Write the drive #
4D26  FFFE
4D28  0585    INC  R5                  ** Point to volume info block
4D2A  0429    BLWP @>005A(R9)          ** Restore R3,R2 from stack
4D2C  005A
4D2E  3001    DATA >3001
4D30  0460    B    @>4676              ** Return
4D32  4676


4D34  0702    SETO R2                  ** Set flag for read sector
4D36  0204    LI   R4,>0001            ** Sector #1
4D38  0001
4D3A  0205    LI   R5,>0100            ** Point to data buffer of FCB
4D3C  0100
4D3E  1005    JMP  >4D4A
4D40  06A0    BL   @>4B70              ** Get sector # of File
4D42  4B70                             ** Descriptor record
4D44  FFFC    DATA >FFFC
4D46  C100    MOV  R0,R4               ** Put it in R4
4D48  04C5    CLR  R5                  ** Point to File Descriptor
                                       ** area of FCB
4D4A  A169    A    @>0056(R9),R5       ** Add the FCB offset
4D4C  0056
4D4E  04C7    CLR  R7
4D50  D086    MOVB R6,R2               ** Put drive # in R2
4D52  0429    BLWP @>005A(R9)          ** Save R0,R1,R2,R3,R4,R5,R6,
4D54  005A                             ** R7,R8 on the stack
4D56  FF80    DATA >FF80
4D58  CA44    MOV  R4,@>004A(R9)       ** Sector #
4D5A  004A
4D5C  CA42    MOV  R2,@>004C(R9)       ** Drive # and Read/Write flag
4D5E  004C
4D60  CA45    MOV  R5,@>004E(R9)       ** Data buffer pointer
4D62  004E
4D64  06A0    BL   @>4658              ** Read/Write the sector
4D66  4658
4D68  40E8    DATA >40E8
4D6A  0429    BLWP @>005A(R9)          ** Restore R8,R7,R6,R5,R4,R3,
4D6C  005A                             ** R2,R1,R0 from the stack
4D6E  FF81    DATA >FF81
```

```
4D7Ø   D1E9   MOVB @>ØØ5Ø(R9),R7          ** Get the error byte
4D72   ØØ5Ø
4D74   Ø6C7   SWPB R7                     ** Put it in low byte
4D76   1312   JEQ  >4D9C                  ** No error?, then jump
4D78   C1C7   MOV  R7,R7
4D7A   1501   JGT  >4D7E                  ** Was it a read operation?,
4D7C   100B   JMP  >4D94                  ** then jump
4D7E   Ø247   ANDI R7,>ØØFF               ** Clear out the operation
4D8Ø   ØØFF
4D82   Ø287   CI   R7,>ØØ34               ** Error >34?
4D84   ØØ34
4D86   1603   JNE  >4D8E                  ** No?, then jump
4D88   Ø6AØ   BL   @>4C72                 ** Report error
4D8A   4C72
4D8C   2ØØØ   DATA >2ØØØ                  ** Write protected
4D8E   Ø6AØ   BL   @>4C72                 ** Report error
4D9Ø   4C72
4D92   CØØØ   DATA >CØØØ                  ** Device error
4D94   DA69   MOVB @>ØØ4D(R9),@>ØØ4D(R9)  ** Write opcode?
4D96   ØØ4D
4D98   ØØ4D
4D9A   13F1   JEQ  >4D7E                  ** Yes?, then jump
4D9C   Ø247   ANDI R7,>ØØFF               ** Mask out operation byte
4D9E   ØØFF
4DAØ   Ø46Ø   B    @>4676                 ** Return
4DA2   4676

** Subroutine to put Drive # and filename in Filename Compare Buffer

4DA4   CØ69   MOV  @>ØØ58(R9),R1          ** Point to Volume Info Buffer
4DA6   ØØ58
4DA8   Ø221   AI   R1,256                 ** Point to Compare buffer
4DAA   Ø1ØØ
4DAC   Ø429   BLWP @>ØØ5A(R9)             ** Set VDPWA to it
4DAE   ØØ5A
4DBØ   ØØ23   DATA >ØØ23
4DB2   DBC6   MOVB R6,@>FFFE(R15)         ** Put the drive # there
4DB4   FFFE
4DB6   Ø581   INC  R1                     ** Add 1 to the pointer
4DB8   Ø6AØ   BL   @>4C14                 ** Put the filename there
4DBA   4C14
4DBC   CØ82   MOV  R2,R2                  ** Any left over bytes?
4DBE   13Ø2   JEQ  >4DC4                  ** No?, then jump
4DCØ   Ø46Ø   B    @>4C18                 ** Report file error
4DC2   4C18

** Find correct File Control Block routine

4DC4   CØ69   MOV  @>ØØ56(R9),R1          ** Put DVA pointer in R1
4DC6   ØØ56
4DC8   Ø221   AI   R1,3                   ** Point to Max # file blocks
4DCA   ØØØ3
4DCC   Ø429   BLWP @>ØØ5A(R9)             ** Set VDPWA to it
4DCE   ØØ5A
4DDØ   ØØ22   DATA >ØØ22
4DD2   DØAF   MOVB @>FBFE(R15),R2         ** Read it
4DD4   FBFE
4DD6   Ø982   SRL  R2,8                   ** Put it in 16 bits
4DD8   CØC2   MOV  R2,R3                  ** Save it in R3
4DDA   Ø221   AI   R1,6                   ** Point to drive # of 1st FCB
```

```
4DDC   0006
4DDE   0204   LI   R4,11                    ** 11 bytes to compare
4DE0   000B
4DE2   0429   BLWP @>005A(R9)                ** Save R1,R2,R3 on stack
4DE4   005A
4DE6   7000   DATA >7000
4DE8   06A0   BL   @>4B3C                    ** Check if filename in FCB
4DEA   4B3C                                  ** same as in Compare Buffer
4DEC   0429   BLWP @>005A(R9)                ** Restore R3,R2,R1
4DEE   005A                                  ** from the stack
4DF0   7001   DATA >7001
4DF2   C104   MOV  R4,R4                     ** Name in FCB same as Compare
                                             ** Buffer?
4DF4   1304   JEQ  >4DFE                     ** Yes?, then jump
4DF6   0221   AI   R1,518                    ** Point to next FCB
4DF8   0206
4DFA   0602   DEC  R2                       ** More FCB's?
4DFC   16F0   JNE  >4DDE                     ** Yes?, then jump
4DFE   0460   B    @>4676                  ** Return
4E00   4676

** Put File Descriptor record in first available File Control Block **

4E02   06A0   BL   @>4658                    ** Find correct FCB
4E04   4658
4E06   4DA4   DATA >4DA4
4E08   C104   MOV R4,R4                     ** Found it? (already open?)
4E0A   13DA   JEQ  >4DC0                     ** Yes?, then jump
4E0C   C169   MOV  @>0056(R9),R5             ** Put DVA pointer in R5
4E0E   0056
4E10   0225   AI   R5,10                     ** Point to filename of
4E12   000A                                  ** first FCB
4E14   0429   BLWP @>005A(R9)                ** Set VDPWA to 1st byte of
4E16   005A                                  ** of name
4E18   00A2   DATA >00A2
4E1A   D0AF   MOVB @>FBFE(R15),R2            ** Read it
4E1C   FBFE
4E1E   1307   JEQ  >4E2E                     ** >00?,(FCB available?),
                                             ** Yes?, then jump
4E20   0225   AI   R5,518                    ** Point to next FCB
4E22   0206
4E24   0603   DEC  R3                       ** Out of FCB's yet?
4E26   16F6   JNE  >4E14                     ** No?, then jump
4E28   06A0   BL   @>4C72                    ** Report error
4E2A   4C72
4E2C   8000   DATA >8000                     ** Out of buffer space
4E2E   CA45   MOV  R5,@>0056(R9)             ** Move pointer to name of
4E30   0056                                  ** FCB to >8356
4E32   06A0   BL   @>4658                    ** Get directory link sector
4E34   4658                                  ** from the disk
4E36   4D34   DATA >4D34
4E38   C229   MOV  @>0056(R9),R8             ** Get FCB pointer
4E3A   0056
4E3C   0228   AI   R8,>017E                  ** Point to file #>7E (63rd)
4E3E   017E
4E40   0202   LI   R2,64                     ** Initial subtraction offset
4E42   0040                                  ** for file search algorithm
4E44   0704   SETO R4
4E46   06A0   BL   @>4B76                    ** Get the word from directory
4E48   4B76
```

```
4E4A   C000   MOV  R0,R0                  ** File there?
4E4C   132E   JEQ  >4EAA                  ** No?, then jump
4E4E   0429   BLWP @>005A(R9)             ** Save R2 on stack (offset)
4E50   005A
4E52   2000   DATA >2000
4E54   0702   SETO R2                     ** Flag for read sector
4E56   06A0   BL   @>4658                 ** Read the File descriptor
4E58   4658                               ** from the disk
4E5A   4D46   DATA >4D46
4E5C   C045   MOV  R5,R1                  ** Move FCB pointer to R1
4E5E   0225   AI   R5,-4                  ** Point to Sector # location
4E60   FFFC                               ** of File control block
4E62   0429   BLWP @>005A(R9)             ** Set VDPWA to it
4E64   005A
4E66   00A3   DATA >00A3
4E68   DBC4   MOVB R4,@>FFFE(R15)         ** Write the sector # we just
4E6A   FFFE                               ** read
4E6C   06C4   SWPB R4
4E6E   DBC4   MOVB R4,@>FFFE(R15)
4E70   FFFE
4E72   C141   MOV  R1,R5                  ** Restore FCB pointer in R5
4E74   0601   DEC  R1                     ** Point to drive # of FCB
4E76   0429   BLWP @>005A(R9)             ** Set VDPWA to it
4E78   005A
4E7A   0023   DATA >0023
4E7C   DBC6   MOVB R6,@>FFFE(R15)         ** Write current drive # there
4E7E   FFFE
4E80   0204   LI   R4,11                  ** 11 bytes to compare
4E82   000B
4E84   06A0   BL   @>4B3C                 ** Compare file name of file
4E86   4B3C                               ** to the Filename Compare
                                          ** Buffer
4E88   0429   BLWP @>005A(R9)             ** Restore R2 from the stack
4E8A   005A                               ** (Algorithm offset)
4E8C   2001   DATA >2001
4E8E   130B   JEQ  >4EA6                  ** Found file?, then jump
4E90   0429   BLWP @>005A(R9)             ** Set VDPWA to Filename
4E92   005A                               ** of FCB
4E94   00A3   DATA >00A3
4E96   DBC4   MOVB R4,@>FFFE(R15)         ** Write a >00 to first byte
4E98   FFFE                               ** (Block available)
4E9A   80C0   C    R0,R3                  ** Is the name we want lower
                                          ** in alphabetical order?
4E9C   1B06   JH   >4EAA                  ** Yes?, then jump
4E9E   A202   A    R2,R8                  ** Add offset
4EA0   C082   MOV  R2,R2                  ** Out of files?
4EA2   1606   JNE  >4EB0                  ** No?, then jump
4EA4   05C8   INCT R8                     ** Update pointer
4EA6   0460   B    @>4676                 ** Return
4EA8   4676
4EAA   6202   S    R2,R8                  ** Subtract offset
4EAC   C082   MOV  R2,R2                  ** Out of files?
4EAE   13FB   JEQ  >4EA6                  ** Yes?, then jump
4EB0   0922   SRL  R2,2                   ** Calculate new offset by
4EB2   A082   A    R2,R2                  ** dividing by two (kinda)
4EB4   10C7   JMP  >4E44                  ** Go check another

** Routine to get 3 bytes from the pointer block, adjust them so Sector #
** in R1, Record(AU) # in R2
```

```
4EB6   C28B   MOV  R11,R10                  ** Save the return address
4EB8   06A0   BL   @>4B76                   ** Get word from block link
4EBA   4B76
4EBC   06C0   SWPB R0
4EBE   D0AF   MOVB @>FBFE(R15),R2           ** Get 3rd byte of block link
4EC0   FBFE
4EC2   C040   MOV  R0,R1                    ** Duplicate in R1
4EC4   0241   ANDI R1,>0FFF                 ** Mask off Nybble of Record #
4EC6   0FFF                                 ** R1 holds starting Sector
                                            ** number of cluster
4EC8   4001   SZC  R1,R0                    ** Clear sector nybbles in R0
4ECA   0982   SRL  R2,8                     ** Put MSNybble of high
                                            ** record # in LSByte of R2
4ECC   E080   SOC  R0,R2                    ** Put LSNybble Record # in
                                            ** MSNybble of R2
4ECE   0BC2   SRC  R2,12                    ** Adjust it!
4ED0   045A   B    *R10                    ** Return

** Routine to write 3 bytes to the pointer block (VDPWA in R8), sector #
** in R1, record(AU)# in R2

4ED2   0B42   SRC  R2,4                     ** Put 2 MSNybbles of rec # in
                                            ** LSByte, LSNybble of rec #
                                            ** in MSNybble R2
4ED4   C002   MOV  R2,R0                    ** Put it in R0
4ED6   0240   ANDI R0,>F000                 ** Mask out all but LSByte of
4ED8   F000                                 ** Rec #
4EDA   E040   SOC  R0,R1                    ** LSNybble Rec # in
                                            ** MSNybble R1
4EDC   0429   BLWP @>005A(R9)               ** Set VDPWA to value in R8
4EDE   005A
4EE0   0103   DATA >0103
4EE2   06C1   SWPB R1                       ** Write the 3 bytes to the
4EE4   DBC1   MOVB R1,@>FFFE(R15)           ** pointer block
4EE6   FFFE
4EE8   06C1   SWPB R1
4EEA   DBC1   MOVB R1,@>FFFE(R15)
4EEC   FFFE
4EEE   06C2   SWPB R2
4EF0   DBC2   MOVB R2,@>FFFE(R15)
4EF2   FFFE
4EF4   045B   B    *R11                    ** Return
4EF6   0429   BLWP @>005A(R9)               ** Save R1,R2,R3,R4 on stack
4EF8   005A
4EFA   7800   DATA >7800
4EFC   06A0   BL @>4658                     ** Check to see if VIB needs
4EFE   4658                                 ** to be changed
4F00   4CD2   DATA >4CD2

** Subroutine to find the first available sector on the disk

4F02   C041   MOV  R1,R1                    ** File Descriptor Sector?
4F04   1602   JNE  >4F0A                    ** Yes?, then jump
4F06   0201   LI   R1,>0021                 ** Add offset past File
4F08   0021                                 ** Descriptor sectors
4F0A   0581   INC  R1                       ** Sector #
4F0C   C001   MOV  R1,R0                    ** Put it in R0
4F0E   0931   SRL  R1,3                     ** Divide by 8(Point to byte
                                            ** of sector in the Bit Map)
4F10   0202   LI   R2,>00FF                 ** Check byte
```

```
4F12  00FF
4F14  0240  ANDI R0,>0007      ** Mask off all bits but 3
4F16  0007                     ** LSBits of the sector #
4F18  1301  JEQ  >4F1C         ** No offset into byte?, jump
4F1A  0A02  SLA  R2,0          ** Shift the check byte # bits
                               ** corresponding to the offset
4F1C  C0C1  MOV  R1,R3         ** Put byte offset in R3
4F1E  A0C5  A    R5,R3         ** Add VIB pointer offset
4F20  0223  AI   R3,>0038      ** Add Bit Map offset
4F22  0038
4F24  0281  CI   R1,>00C8      ** Past end of VIB?
4F26  00C8
4F28  1101  JLT  >4F2C         ** No?, then jump
4F2A  04C1  CLR  R1
4F2C  A045  A    R5,R1         ** Add VIB offset
4F2E  0221  AI   R1,>0038      ** Add Bit Map offset
4F30  0038
4F32  0429  BLWP @>005A(R9)    ** Set VDPWA to byte of Bit
4F34  005A                     ** Map
4F36  0022  DATA >0022
4F38  0700  SETO R0            ** For compare
4F3A  D02F  MOVB @>FBFE(R15),R0 ** Read the byte
4F3C  FBFE
4F3E  C100  MOV  R0,R4         ** Put it in R4
4F40  E002  SOC  R2,R0         ** Set bits according to
                               ** shifted check byte
4F42  04C2  CLR  R2
4F44  0580  INC  R0            ** Offset to check if sectors
                               ** in this block greater than
                               ** the current # are full
4F46  1610  JNE  >4F68         ** Not full?, then jump
4F48  0581  INC  R1            ** Point to next byte of
                               ** Bit Map
4F4A  C001  MOV  R1,R0         ** Put it in R0
4F4C  0220  AI   R0,-256       ** Subtract offset
4F4E  FF00
4F50  8140  C    R0,R5         ** At end of Bit Map?
4F52  1605  JNE  >4F5E         ** No?, then jump
4F54  0221  AI   R1,>FF38      ** Point to start of Bit Map
4F56  FF38
4F58  0429  BLWP @>005A(R9)    ** Set VDPWA to it
4F5A  005A
4F5C  0022  DATA >0022
4F5E  80C1  C    R1,R3         ** At start of Bit Map?
4F60  16EB  JNE  >4F38         ** No?, then jump
4F62  C004  MOV  R4,R0         ** Put byte in R0
4F64  0580  INC  R0            ** All sectors full?
4F66  1325  JEQ  >4FB2         ** Yes?, go return
4F68  0600  DEC  R0            ** Back off the offset
4F6A  C080  MOV  R0,R2         ** Put it in R2
4F6C  04C0  CLR  R0            ** Clear the count of avlbl.
                               ** sectors within this byte
4F6E  06C2  SWPB R2            ** Put Bit Map byte in LSByte
4F70  0580  INC  R0            ** Add 1 to the count
4F72  0B12  SRC  R2,1          ** Sector available?
4F74  18FD  JOC  >4F70         ** No?, then jump
4F76  0202  LI   R2,>0080      ** Set Bit(Sector now used)
4F78  0080
4F7A  0A02  SLA  R2,0          ** Shift it to point to bit of
                               ** sector to be used
```

```
4F7C   0600   DEC  R0                  ** Subtract 1 from the count
4F7E   E102   SOC  R2,R4               ** Set the bit
4F80   0429   BLWP @>005A(R9)          ** Set VDPWA to byte of
4F82   005A                           ** Bit Map
4F84   0023   DATA >0023
4F86   DBC4   MOVB R4,@>FFFE(R15)     ** Write the byte back
4F88   FFFE
4F8A   0221   AI   R1,>FFC8           ** Get rid of Bit Map offset
4F8C   FFC8
4F8E   6045   S    R5,R1              ** Get rid of VIB offset
4F90   0A31   SLA  R1,3               ** Multiply by 8 for sector #
4F92   E040   SOC  R0,R1              ** Add count within the byte
4F94   C001   MOV  R1,R0              ** Save Sector # in R0
4F96   0605   DEC  R5                 ** Point to VIB status byte
4F98   0429   BLWP @>005A(R9)         ** Set VDPWA to it
4F9A   005A
4F9C   00A2   DATA >00A2
4F9E   D06F   MOVB @>FBFE(R15),R1     ** Read the byte
4FA0   FBFE
4FA2   0261   ORI  R1,>8000           ** Set the updated VIB
4FA4   8000                           ** status bit
4FA6   0429   BLWP @>005A(R9)         ** Set VDPWA back to it
4FA8   005A
4FAA   00A3   DATA >00A3
4FAC   DBC1   MOVB R1,@>FFFE(R15)     ** Write the byte back
4FAE   FFFE
4FB0   0585   INC  R5                 ** Point to Volume Info Block
4FB2   0429   BLWP @>005A(R9)         ** Restore R4,R3,R2,R1
4FB4   005A                           ** from stack
4FB6   7801   DATA >7801
4FB8   0460   B  @>4676               ** Return
4FBA   4676

4FBC   C040   MOV  R0,R1              ** Put sector # in R1
4FBE   0202   LI   R2,1               ** Only 1 sector to delete
4FC0   0001

** Subroutine to delete a block of sectors on the disk, starting sector #
** passed in R1, Number of sectors to delete passed in R2

4FC2   0429   BLWP @>005A(R9)         ** Save R1,R2,R3,R4 on stack
4FC4   005A
4FC6   7800   DATA >7800
4FC8   06A0   BL @>4658               ** Update the VIB
4FCA   4658
4FCC   4CD2   DATA >4CD2
4FCE   C001   MOV  R1,R0              ** Put sector # in R0
4FD0   0240   ANDI R0,>0007           ** Mask off the byte pointer
4FD2   0007
4FD4   0931   SRL  R1,3               ** Divide sector # by 8
                                      ** (byte pointer)
4FD6   A045   A    R5,R1              ** Add FCB offset
4FD8   0221   AI   R1,>0038           ** Add bit map offset
4FDA   0038
4FDC   C0C0   MOV  R0,R3              ** Put bit offset in R3
4FDE   0500   NEG  R0                 ** Neg it
4FE0   0220   AI   R0,8               ** Add 8 (Calculate # bits in
4FE2   0008                           ** this byte >= to starting
                                      ** bit)
4FE4   0204   LI   R4,>00FF           ** Put the mask byte in R4
```

```
4FE6   00FF
4FE8   8002   C     R2,R0          ** Do we need more than one
                                    ** byte in the bit map?
4FEA   1101   JLT   >4FEE         ** No?, then jump
4FEC   100B   JMP   >5004
4FEE   0200   LI    R0,8
4FF0   0008
4FF2   6002   S     R2,R0          ** Calculate # bits to shift
4FF4   0B04   SRC   R4,0           ** Adjust the bits
4FF6   C003   MOV   R3,R0          ** Bit 0?
4FF8   1301   JEQ   >4FFC         ** Yes?, then jump
4FFA   0A04   SLA   R4,0           ** Shift for correct bit
4FFC   1001   JMP   >5000
4FFE   0939   SRL   R9,3           ** Not used?? Error in
                                    ** programming??
5000   06C4   SWPB  R4             ** Put mask byte in MSByte
5002   1019   JMP   >5036
5004   0B04   SRC   R4,0           ** Shift to clear sectors
5006   6080   S     R0,R2          ** Subtract # we're going to
                                    ** clear
5008   0429   BLWP  @>005A(R9)     ** Set VDPWA to byte in
500A   005A                        ** bit map
500C   0022   DATA  >0022
500E   D02F   MOVB  @>FBFE(R15),R0 ** Read the byte
5010   FBFE
5012   4004   SZC   R4,R0          ** Clear the correct bits
5014   0429   BLWP  @>005A(R9)     ** Set VDPWA back to the byte
5016   005A
5018   0023   DATA  >0023
501A   DBC0   MOVB  R0,@>FFFE(R15) ** Write it
501C   FFFE
501E   0204   LI    R4,>FF00       ** Put mask byte in R4
5020   FF00
5022   0581   INC   R1             ** Point to next byte in
                                    ** bit map
5024   0200   LI    R0,8           ** 8 sectors per byte
5026   0008
5028   8002   C     R2,R0          ** More than 1 byte left?
502A   1101   JLT   >502E         ** No?, then jump
502C   10EC   JMP   >5006
502E   C002   MOV   R2,R0          ** Put # left in R0
5030   0204   LI    R4,>00FF       ** Put mask byte in R4
5032   00FF
5034   0A04   SLA   R4,0           ** Shift to clear # bits
5036   0429   BLWP  @>005A(R9)     ** Set VDPWA to byte in
5038   005A                        ** bit map
503A   0022   DATA  >0022
503C   D02F   MOVB  @>FBFE(R15),R0 ** Read the byte
503E   FBFE
5040   4004   SZC   R4,R0          ** Clear correct bits
5042   0429   BLWP  @>005A(R9)     ** Set VDPWA back to it
5044   005A
5046   0023   DATA  >0023
5048   DBC0   MOVB  R0,@>FFFE(R15) ** Write it back
504A   FFFE
504C   10A4   JMP   >4F96          ** Go update the VIB status
                                    ** bit and return

** Start of "DSK." routine **
```

```
504E    C1CB    MOV  R11,R7          ** Save return address
5050    06A0    BL   @>4724          ** Init routine
5052    4724
5054    06A0    BL   @>4658          ** Find Drive # for disk name
5056    4658
5058    4B8A    DATA >4B8A
505A    100B    JMP  >5072

** Start of "DSK1." routine **

505C    0206    LI   R6,>0100        ** Drive #1
505E    0100
5060    1005    JMP  >506C

** Start of "DSK2." routine **

5062    0206    LI   R6,>0200        ** Drive #2
5064    0200
5066    1002    JMP  >506C

** Start of "DSK3." routine **

5068    0206    LI   R6,>0300        ** Drive #3
506A    0300
506C    C1CB    MOV  R11,R7          ** Save return address
506E    06A0    BL   @>4724          ** Init routine
5070    4724
5072    C029    MOV  @>0054(R9),R0    ** Put pointer to VDP PAB - R0
5074    0054
5076    0429    BLWP @>005A(R9)       ** Set VDPWA to VDP PAB
5078    005A
507A    0002    DATA >0002
507C    D06F    MOVB @>FBFE(R15),R1   ** Get opcode into R1
507E    FBFE
5080    0981    SRL  R1,8             ** Put it in low byte
5082    0281    CI   R1,>0009         ** Is opcode greater than 9?
5084    0009
5086    1B19    JH   >50BA            ** Yes?, then jump to err rtn.
5088    0282    CI   R2,>0001         ** Is there a file name?
508A    0001
508C    1605    JNE  >5098            ** No?, then jump
508E    0221    AI   R1,10            ** Add 10 for disk directory
5090    000A                          ** routines
5092    0281    CI   R1,>000C         ** Bigger than largest
5094    000C                          ** routine #?
5096    1B11    JH   >50BA            ** Yes?, then jump to err rtn.
5098    A041    A    R1,R1            ** Multiply by 2 for table
                                      ** offset
509A    C061    MOV  @>50A0(R1),R1    ** Get routine address from
509C    50A0                          ** table
509E    0451    B    *R1              ** Jump to it!

** Routine table for opcodes **

50A0    50C0    DATA >50C0            ** Open
50A2    52D2    DATA >52D2            ** Close
50A4    52DC    DATA >52DC            ** Read
50A6    53C6    DATA >53C6            ** Write
50A8    567A    DATA >567A            ** Restore/Rewind
50AA    56CE    DATA >56CE            ** Load
```

```
50AC   5770   DATA >5770                    ** Save
50AE   4A72   DATA >4A72                    ** Delete
50B0   4C9E   DATA >4C9E                    ** Scratch Record
50B2   57F4   DATA >57F4                    ** Status
50B4   58B4   DATA >58B4                    ** Open Disk Directory
50B6   5912   DATA >5912                    ** Close Disk Directory
50B8   5928   DATA >5928                    ** Read Disk Directory
50BA   06A0   BL   @>4C72                   ** Report error
50BC   4C72
50BE   6000   DATA >6000                    ** Illegal operation

** Start of Open opcode routine

50C0   04C0   CLR  R0
50C2   D02F   MOVB @>FBFE(R15),R0           ** Get Flag/Status byte in R0
50C4   FBFE
50C6   0429   BLWP @>005A(R9)               ** Save R0 on the stack
50C8   005A
50CA   8000   DATA >8000
50CC   0240   ANDI R0,>1600                 ** Mask bits except VAR/FIX
50CE   1600                                 ** and Mode
50D0   0280   CI   R0,>0600                 ** Append, fixed length recs?
50D2   0600
50D4   1603   JNE  >50DC                    ** No?, then jump
50D6   06A0   BL   @>4C72                   ** Report error
50D8   4C72
50DA   4000   DATA >4000                    ** Bad Open Attribute
50DC   120A   JLE  >50F2                    ** Fixed length recs?, jump
50DE   C040   MOV  R0,R1                    ** Put status byte in R1
50E0   06A0   BL   @>4B6A                   ** Logical Rec length in R0
50E2   4B6A
50E4   0004   DATA >0004
50E6   0280   CI   R0,>FF00                 ** Is it 254 bytes or less?
50E8   FF00
50EA   14F5   JHE  >50D6                    ** No?, then jump  report err
50EC   C001   MOV  R1,R0                    ** Move status byte back to R0
50EE   0240   ANDI R0,>0600                 ** Mask out bits except mode
50F0   0600
50F2   0280   CI   R0,>0200                 ** Is it output mode?
50F4   0200
50F6   1657   JNE  >51A6                    ** No?, then jump
50F8   06A0   BL   @>4658
50FA   4658
50FC   4798   DATA >4798
50FE   06A0   BL   @>5280                   ** Set up for File status flag
5100   5280
5102   0429   BLWP @>005A(R9)              ** Set VDPWA to F/S flag
5104   005A                                ** in DVA
5106   0083   DATA >0083
5108   DBC2   MOVB R2,@>FFFE(R15)          ** Write it to VDP
510A   FFFE
510C   C0E9   MOV  @>0054(R9),R3           ** Get pointer to start
510E   0054                                ** of VDP PAB
5110   0223   AI   R3,>0004                 ** Point to record length
5112   0004
5114   04C5   CLR  R5
5116   0429   BLWP @>005A(R9)              ** Set VDPWA to it
5118   005A
511A   0062   DATA >0062
511C   D16F   MOVB @>FBFE(R15),R5          ** Get it into R5
```

```
511E   FBFE
5120   1607   JNE   >5130                    ** Not zero?, then jump

** Set up for default length of 80 characters

5122   0205   LI    R5,>5000                 ** Put 80 in MSByte
5124   5000
5126   0429   BLWP  @>005A(R9)               ** Set VDPWA to rec length
5128   005A
512A   0063   DATA  >0063
512C   DBC5   MOVB  R5,@>FFFE(R15)           ** Write 80 there
512E   FFFE
5130   0224   AI    R4,>0005                 ** Point to rec length of DVA
5132   0005
5134   0429   BLWP  @>005A(R9)               ** Set VDPWA to it
5136   005A
5138   0083   DATA  >0083
513A   DBC5   MOVB  R5,@>FFFE(R15)           ** Write 80 to it
513C   FFFE

** Calculate # of records per sector

513E   0201   LI    R1,256
5140   0100
5142   C082   MOV   R2,R2                    ** Variable length records?
5144   1101   JLT   >5148                    ** Yes?, then jump
5146   1002   JMP   >514C
5148   A141   A     R1,R5                    ** Add 1 to rec length for the
                                             ** length byte used by VAR
                                             ** VAR recs
514A   0601   DEC   R1                       ** Subtract 1 (254 bytes per
                                             ** sector for VAR recs)
514C   06C5   SWPB  R5                       ** Put rec length in 16 bits
514E   04C0   CLR   R0                       ** Set up for divide
5150   3C05   DIV   R5,R0                    ** Divide bytes/sector by
                                             ** bytes/record
5152   0224   AI    R4,-4                    ** Point to Max # recs/sector
5154   FFFC                                  ** in FCB
5156   C040   MOV   R0,R1                    ** Put # recs/sector in R1
5158   06C0   SWPB  R0                       ** Put it in MSByte
515A   0429   BLWP  @>005A(R9)               ** Set VDPWA to it
515C   005A
515E   0083   DATA  >0083
5160   DBC0   MOVB  R0,@>FFFE(R15)           ** Write it to FCB
5162   FFFE
5164   C229   MOV   @>0056(R9),R8            ** Get pointer to FCB
5166   0056
5168   0429   BLWP  @>005A(R9)               ** Set VDPWA to it
516A   005A
516C   0102   DATA  >0102
516E   D02F   MOVB  @>FBFE(R15),R0           ** Get 1st byte of filename
5170   FBFE                                  ** in FCB
5172   0260   ORI   R0,>8000                 ** Set Updated File
5174   8000                                  ** Descriptor bit
5176   0429   BLWP  @>005A(R9)               ** Set VDPWA to it again
5178   005A
517A   0103   DATA  >0103
517C   DBC0   MOVB  R0,@>FFFE(R15)           ** Write it back
517E   FFFE
5180   0429   BLWP  @>005A(R9)               ** Restore R0 from stack
```

```
5182  005A
5184  8001    DATA >8001
5186  06A0    BL   @>4B6A                ** Get current # of records
5188  4B6A                               ** from PAB (size of the file)
518A  0006    DATA >0006
518C  C100    MOV  R0,R4                 ** Put it in R4
518E  1309    JEQ  >51A2                 ** Is it 0? then jump
5190  11A2    JLT  >50D6                 ** MSBit set?, jump to error
5192  A101    A    R1,R4                 ** Calculate total # sectors *
5194  0604    DEC  R4                    ** needed                    *
5196  04C3    CLR  R3                    ** Clear if for division
5198  3CC1    DIV  R1,R3
519A  0603    DEC  R3
519C  06A0    BL   @>4658                ** Calculate sector #, update
519E  4658                               ** pointer blocks
51A0  4964    DATA >4964
51A2  0460    B    @>56A8                ** Go set Rec offset to 0
51A4  56A8                               ** and return


51A6  0429    BLWP @>005A(R9)            ** Save R0 on the stack
51A8  005A
51AA  8000    DATA >8000
51AC  06A0    BL   @>4658                ** Check if file on the disk,
51AE  4658                               ** get the file descriptor
51B0  4E02    DATA >4E02
51B2  0429    BLWP @>005A(R9)            ** Restore R0 from the stack
51B4  005A
51B6  8001    DATA >8001
51B8  C104    MOV  R4,R4
51BA  1309    JEQ  >51CE
51BC  0280    CI   R0,>0400              ** Input mode?
51BE  0400
51C0  1304    JEQ  >51CA                 ** Yes?, then jump
51C2  06A0    BL   @>4658                ** Create the file
51C4  4658
51C6  47AA    DATA >47AA
51C8  109A    JMP  >50FE
51CA  0460    B    @>50D6                ** Report error
51CC  50D6
51CE  C1C0    MOV  R0,R7
51D0  06A0    BL   @>5280                ** Convert the PAB status byte
51D2  5280                               ** to FCB format
51D4  0429    BLWP @>005A(R9)            ** Set VDPWA to status byte
51D6  005A                               ** in FCB
51D8  0082    DATA >0082
51DA  D02F    MOVB @>FBFE(R15),R0        ** Get the byte from the FCB
51DC  FBFE
51DE  C0C0    MOV  R0,R3                 ** Put it in R3
51E0  0243    ANDI R3,>0800              ** Write protected?
51E2  0800
51E4  1306    JEQ  >51F2                 ** No?, then jump
51E6  0287    CI   R7,>0400              ** Input mode?
51E8  0400
51EA  1303    JEQ  >51F2                 ** Yes?, then jump
51EC  06A0    BL   @>4C72                ** Report error
51EE  4C72
51F0  2000    DATA >2000                 ** Device write protected
51F2  0240    ANDI R0,>8300              ** Mask bits except file type
51F4  8300
```

```
51F6   2802   XOR  R2,R0                  ** Check if status byte of
                                           ** file same as one in the PAB
51F8   16E8   JNE  >51CA                  ** No?, then jump
51FA   C0E9   MOV  @>0054(R9),R3          ** Put PAB pointer in R3
51FC   0054
51FE   0223   AI   R3,>0004               ** Point to Logical Rec Length
5200   0004
5202   0224   AI   R4,>0005               ** Point to rec length in FCB
5204   0005
5206   0429   BLWP @>005A(R9)             ** Set VDPWA to FCB rec length
5208   005A
520A   0082   DATA >0082
520C   D02F   MOVB @>FBFE(R15),R0         ** Read the byte
520E   FBFE
5210   0429   BLWP @>005A(R9)             ** Set VDPWA to PAB rec length
5212   005A
5214   0062   DATA >0062
5216   D0AF   MOVB @>FBFE(R15),R2         ** Read the byte
5218   FBFE
521A   1302   JEQ  >5220                  ** Is it 0?, then jump
521C   9080   CB   R0,R2                  ** Check to see if equal
521E   16D5   JNE  >51CA                  ** No?, then jump to error
5220   0429   BLWP @>005A(R9)             ** Set VDPWA to PAB rec length
5222   005A
5224   0063   DATA >0063
5226   DBC0   MOVB R0,@>FFFE(R15)         ** Write length from FCB
5228   FFFE
522A   0429   BLWP @>005A(R9)             ** Restore R0 from the stack
522C   005A
522E   8001   DATA >8001
5230   0240   ANDI R0,>0600              ** Mask out all but mode bits
5232   0600
5234   04C2   CLR  R2
5236   0703   SETO R3
5238   0280   CI   R0,>0600              ** Append mode?
523A   0600
523C   161D   JNE  >5278                 ** No?, then jump
523E   C129   MOV  @>0056(R9),R4          ** Get pointer to FCB
5240   0056
5242   C1C4   MOV  R4,R7                 ** Save it in R7
5244   0224   AI   R4,>000E              ** Point to # of sectors
5246   000E                              ** currently allocated
5248   0429   BLWP @>005A(R9)             ** Set VDPWA to it
524A   005A
524C   0082   DATA >0082
524E   D0EF   MOVB @>FBFE(R15),R3         ** Get the MSByte
5250   FBFE
5252   06C3   SWPB R3
5254   D0EF   MOVB @>FBFE(R15),R3         ** Get the LSByte
5256   FBFE
5258   06C3   SWPB R3
525A   D0AF   MOVB @>FBFE(R15),R2         ** Get end of file offset
525C   FBFE
525E   0603   DEC  R3                     ** Subtract 1 (linkage sector)
5260   110B   JLT  >5278                 ** No data sectors?, then jump
5262   0429   BLWP @>005A(R9)             ** Save R2,R3 on stack
5264   005A
5266   3000   DATA >3000
5268   0227   AI   R7,>0100              ** Point to data buffer of FCB
526A   0100
```

```
526C   06A0   BL   @>4658                    ** Read the AU into data
526E   4658                                  ** buffer
5270   492E   DATA >492E
5272   0429   BLWP @>005A(R9)                 ** Restore R2,R3 from stack
5274   005A
5276   3001   DATA >3001
5278   06A0   BL   @>52AA                     ** Write AU and logical record
527A   52AA                                  ** offsets
527C   0460   B    @>4676                     ** Return
527E   4676

** Subroutine to put filetype specified in the PAB in the same format as
** the status byte in the FCB

5280   0429   BLWP @>005A(R9)                 ** Restore R0 from stack
5282   005A
5284   8001   DATA >8001
5286   0429   BLWP @>005A(R9)                 ** Save it back on the stack
5288   005A
528A   8000   DATA >8000
528C   0202   LI   R2,>0002                   ** INT/FIX file type
528E   0002
5290   C129   MOV  @>0056(R9),R4              ** Put pointer to FCB in R4
5292   0056
5294   0A40   SLA  R0,4                       ** Check for fixed file type
5296   1703   JNC  >529E                     ** Yes?, then jump
5298   0202   LI   R2,>0082                   ** INT/VAR file type
529A   0082
529C   C000   MOV  R0,R0                      ** Is it display file type?
529E   1101   JLT  >52A2                     ** No?, then jump
52A0   0642   DECT R2                        ** Change to DIS rather
                                             ** than INT
52A2   0224   AI   R4,>000C                   ** Point to status byte of FCB
52A4   000C
52A6   06C2   SWPB R2
52A8   045B   B    *R11                      ** Return
52AA   C129   MOV  @>0056(R9),R4              ** Put pointer to FCB in R4

** Subroutine to write AUoffset and logical record offset

52AC   0056
52AE   0224   AI   R4,-6                     ** Point to AU offset
52B0   FFFA
52B2   0429   BLWP @>005A(R9)                 ** Set VDPWA to it
52B4   005A
52B6   0083   DATA >0083
52B8   DBC3   MOVB R3,@>FFFE(R15)            ** Write first byte
52BA   FFFE
52BC   06C3   SWPB R3
52BE   DBC3   MOVB R3,@>FFFE(R15)            ** Write second byte
52C0   FFFE
52C2   0224   AI   R4,>0004                  ** Point to logical rec offset
52C4   0004
52C6   0429   BLWP @>005A(R9)                ** Set VDPWA to it
52C8   005A
52CA   0083   DATA >0083
52CC   DBC2   MOVB R2,@>FFFE(R15)            ** Write byte
52CE   FFFE
52D0   045B   B    *R11                      ** Return
```

```
** Close opcode

52D2   06A0   BL   @>4658          ** Find File Control Block,
52D4   4658                        ** get status byte
52D6   54D0   DATA >54D0
52D8   0460   B    @>487A
52DA   487A

** Read opcode

52DC   06A0   BL   @>4658          ** Find the FCB, get status
52DE   4658                        ** byte
52E0   54D0   DATA >54D0
52E2   0240   ANDI R0,>0200       ** Is file type OUTPUT?
52E4   0200
52E6   1302   JEQ  >52EC           ** No?, then jump
52E8   0460   B    @>50BA         ** Report error, Illegal
52EA   50BA                        ** operation
52EC   06A0   BL   @>54FC         ** Get status flag from FCB
52EE   54FC
52F0   110A   JLT  >5306          ** Variable length recs?, jump

** Fixed Records

52F2   06A0   BL   @>5510          ** Get the rec # and compare,
52F4   5510                        ** get AU#, and sector offset
52F6   1A03   JL   >52FE           ** Rec # less than max rec #?,
                                   ** jump
52F8   06A0   BL   @>4C72          ** Report error
52FA   4C72
52FC   A000   DATA >A000
52FE   06A0   BL   @>4658          ** Calculate and read correct
5300   4658                        ** sector for the record, R1
5302   5576   DATA >5576          ** points to Data buffer
                                   ** address, R2 points to
                                   ** current rec in FCB buffer
5304   1011   JMP  >5328          ** Go return the data

** Variable records

5306   06A0   BL   @>4658          ** Get pointers to data,
5308   4658                        ** # bytes to return, and
530A   5362   DATA >5362          ** pointer to PAB buffer
530C   10F5
530E   0582   INC  R2              ** Add 1 to pointer to data
                                   ** buffer of FCB (skip length
                                   ** byte)
5310   A004   A    R4,R0          ** Add length of record just
                                   ** read, to logical rec offset
5312   0580   INC  R0             ** Add 1 to point to beginning
                                   ** of next record
5314   C169   MOV  @>0056(R9),R5  ** Get pointer to FCB
5316   0056
5318   0645   DECT R5             ** Point to logical rec offset
531A   06C0   SWPB R0
531C   0429   BLWP @>005A(R9)     ** Set VDPWA to it
531E   005A
5320   00A3   DATA >00A3
5322   DBC0   MOVB R0,@>FFFE(R15) ** Write the byte
5324   FFFE
```

```
5326   C004   MOV  R4,R0                    ** Put # bytes to read in R0

** Return data

5328   C129   MOV  @>0054(R9),R4            ** Get pointer to PAB
532A   0054
532C   0224   AI   R4,5                     ** Point to Max # bytes
532E   0005                                 ** actually read
5330   06C0   SWPB R0                       ** Put # bytes in MSByte
5332   0429   BLWP @>005A(R9)               ** Set VDPWA to it
5334   005A
5336   0083   DATA >0083
5338   DBC0   MOVB R0,@>FFFE(R15)           ** Write it
533A   FFFE
533C   06C0   SWPB R0                       ** Put # bytes back in 16 bits
533E   C000   MOV  R0,R0                    ** 0 bytes to put in buffer?
5340   130E   JEQ  >535E                    ** Yes?, then jump
5342   0429   BLWP @>005A(R9)               ** Set VDPWA to data buffer
5344   005A                                 ** of FCB
5346   0042   DATA >0042
5348   D0EF   MOVB @>FBFE(R15),R3           ** Read the byte
534A   FBFE
534C   0582   INC  R2                       ** Update pointer
534E   0429   BLWP @>005A(R9)               ** Set VDPWA to PAB Buffer
5350   005A
5352   0023   DATA >0023
5354   DBC3   MOVB R3,@>FFFE(R15)           ** Write the byte there
5356   FFFE
5358   0581   INC  R1                       ** Update pointer
535A   0600   DEC  R0                       ** Done yet?
535C   16F2   JNE  >5342                    ** No?, then jump
535E   0460   B    @>4676                   ** Return
5360   4676

** Subroutine to get pointers to data, # bytes to return, and pointer
** to PAB buffer for variable length records

5362   06A0   BL   @>5650                   ** Get current log rec offset
5364   5650                                 ** in R0, AU offset in R3,
                                            ** # of AU's in R2
5366   C0C3   MOV  R3,R3                    ** Current AU offset
5368   1101   JLT  >536C                    ** MSBit set?, then jump
536A   1012   JMP  >5390
536C   C003   MOV  R3,R0                    ** Current AU offset in R0
536E   0580   INC  R0                       ** Add 1 to it
5370   8080   C    R0,R2                    ** Equal to Max # of sectors?
5372   13F5   JEQ  >535E                    ** Yes?, then jump
5374   06A0   BL   @>4658                   ** Check if Updated buffer
5376   4658                                 ** area  if so, write it!
5378   489A   DATA >489A
537A   C0C0   MOV  R0,R3                    ** Put AU offset in R3
537C   04C5   CLR  R5                       ** 0 logical rec offset
537E   06A0   BL   @>55EE                   ** Write it to FCB
5380   55EE
5382   0227   AI   R7,256                   ** Point to data buffer
5384   0100
5386   06A0   BL   @>4658                   ** Read the AU
5388   4658
538A   492E   DATA >492E
538C   04C0   CLR  R0                       ** Set offset into sector to 0
```

```
538E   10Ø4   JMP  >5398
5390   C000   MOV  R0,R0          ** Ø logical rec offset?
5392   1602   JNE  >5398          ** No?, then jump
5394   8080   C    R0,R2          ** Ø AU's in file?
5396   13E3   JEQ  >535E          ** Yes?, then jump
5398   C080   MOV  R0,R2          ** Put offset in sector in R2
539A   06A0   BL   @>4658        ** Get pointer to data in FCB,
539C   4658                      ** Pointer to data buffer of
539E   55CA   DATA >55CA          ** PAB
53A0   0429   BLWP @>005A(R9)     ** Set VDPWA to data
53A2   005A
53A4   0042   DATA >0042
53A6   D12F   MOVB @>FBFE(R15),R4 ** Read the byte (length)
53A8   FBFE
53AA   0984   SRL  R4,8           ** Put in 16 bits
53AC   C000   MOV  R0,R0          ** Ø offset into sector?
53AE   1306   JEQ  >53BC          ** Yes?, then jump
53B0   0284   CI   R4,>00FF       ** End of data in sector?
53B2   00FF
53B4   1603   JNE  >53BC          ** No?, then jump
53B6   06A0   BL   @>5650        ** Set up for reading next AU
53B8   5650
53BA   10D8   JMP  >536C          ** Go read it!
53BC   0429   BLWP @>005A(R9)     ** Restore return address
53BE   005A
53C0   0011   DATA >0011
53C2   05CB   INCT R11           ** Add to two skip DATA stmnt.
53C4   045B   B    *R11          ** And...Return!!!

** WRITE opcode

53C6   06A0   BL   @>4658        ** Find the FCB, read status
53C8   4658                      ** byte
53CA   54D0   DATA >54D0
53CC   0240   ANDI R0,>0600      ** Mask all but mode bits
53CE   0600
53D0   0280   CI   R0,>0400      ** Is it INPUT mode?
53D2   0400
53D4   1389   JEQ  >52E8          ** Yes?, then jump report err.
53D6   06A0   BL   @>54FC        ** Read status flag from FCB
53D8   54FC
53DA   1113   JLT  >5402          ** Variable recs?, then jump
53DC   06A0   BL   @>5510        ** Compare rec # to current #
53DE   5510                      ** of records
53E0   1A0C   JL   >53FA          ** Do we need to add to end of
                                 ** of the file?, No?, jump
53E2   0429   BLWP @>005A(R9)     ** Store R0,R1,R3,R4 on stack
53E4   005A
53E6   D800   DATA >D800
53E8   C0C0   MOV  R0,R3          ** Put AU # need to access in
53EA   06A0   BL   @>4658        ** in R3
53EC   4658
53EE   4964   DATA >4964
53F0   0429   BLWP @>005A(R9)     ** Restore R4,R3,R1,R0 from
53F2   005A                      ** stack
53F4   D801   DATA >D801
53F6   06A0   BL   @>561A        ** Set Updated descriptor bit,
53F8   561A                      ** write # fixed length recs
                                 ** in FCB
53FA   06A0   BL   @>4658        ** Get AU, Rec length, PAB
```

```
53FC   4658                          ** buffer address, pointer to
53FE   5576   DATA >5576             ** data
5400   1053   JMP  >54A8

** Variable Records

5402   06A0   BL   @>5650            ** Get Logical Rec offset, AU
5404   5650                          ** offset, total AU's
5406   C0C3   MOV  R3,R3             ** Update data buffer?
5408   1101   JLT  >540C             ** Yes?, then jump
540A   100B   JMP  >5422
540C   0429   BLWP @>005A(R9)        ** Save R2,R3 on the stack
540E   005A
5410   3000   DATA >3000
5412   06A0   BL   @>4658            ** Write AU in data buffer to
5414   4658                          ** the disk
5416   489A   DATA >489A
5418   0429   BLWP @>005A(R9)        ** Restore R3,R3 from stack
541A   005A
541C   3001   DATA >3001
541E   0583   INC  R3                ** Add it to AU offset
5420   04C0   CLR  R0                ** Clear logical offset
5422   8083   C    R3,R2             ** At maximum sector?
5424   1609   JNE  >5438             ** No?, then jump
5426   0429   BLWP @>005A(R9)        ** Save R0,R3 on the stack
5428   005A
542A   9000   DATA >9000
542C   06A0   BL   @>4658            ** Calculate sector #, update
542E   4658                          ** pointer blocks
5430   4964   DATA >4964
5432   0429   BLWP @>005A(R9)        ** Restore R3,R0 from stack
5434   005A
5436   9001   DATA >9001
5438   C169   MOV  @>0054(R9),R5     ** Put PAB pointer in R5
543A   0054
543C   0225   AI   R5,5              ** Point to # bytes to write
543E   0005
5440   0429   BLWP @>005A(R9)        ** Set VDPWA to it
5442   005A
5444   00A2   DATA >00A2
5446   D12F   MOVB @>FBFE(R15),R4    ** Read the byte
5448   FBFE
544A   0984   SRL  R4,8              ** Put it in 16 bits
544C   C144   MOV  R4,R5             ** Put it in R5
544E   A140   A    R0,R5             ** Add pointer to data in rec
5450   0585   INC  R5                ** Add 1 to pointer into data
                                     ** buffer
5452   0285   CI   R5,>00FF          ** 255 bytes?
5454   00FF
5456   1BDA   JH   >540C             ** Greater?, then jump
5458   0702   SETO R2                ** >FF, end of data marker
545A   C069   MOV  @>0056(R9),R1     ** Get FCB pointer
545C   0056
545E   A045   A    R5,R1             ** Pointer into data buffer
5460   0221   AI   R1,256            ** Add 256 to point to data
5462   0100                          ** buffer
5464   0429   BLWP @>005A(R9)        ** Set VDPWA to it
5466   005A
5468   0023   DATA >0023
546A   DBC2   MOVB R2,@>FFFE(R15)    ** Write end of data marker
```

```
546C   FFFE
546E   0429   BLWP @>005A(R9)              ** Save R0 on the stack
5470   005A
5472   8000   DATA >8000
5474   C003   MOV  R3,R0                  ** Put AU # in R0
5476   06A0   BL   @>55EE                 ** Update AU# and logical
5478   55EE                               ** record offset
547A   0221   AI   R1,>0012               ** Point to length byte
547C   0012
547E   0429   BLWP @>005A(R9)              ** Set VDPWA to it
5480   005A
5482   0023   DATA >0023
5484   DBC5   MOVB R5,@>FFFE(R15)         ** Write the length
5486   FFFE
5488   06A0   BL   @>561A                 ** Update File Descriptor bit
548A   561A
548C   0429   BLWP @>005A(R9)              ** Restore R2 from the stack
548E   005A
5490   2001   DATA >2001
5492   C004   MOV  R4,R0                  ** Number of bytes to write
5494   06A0   BL   @>4658
5496   4658
5498   55CA   DATA >55CA
549A   06C4   SWPB R4                     ** Put # bytes in MSB R4
549C   0429   BLWP @>005A(R9)
549E   005A
54A0   0043   DATA >0043
54A2   DBC4   MOVB R4,@>FFFE(R15)         ** Write the # bytes
54A4   FFFE
54A6   0582   INC  R2                     ** Add 1 to pointer to data
                                          ** in the record
54A8   C0C2   MOV  R2,R3                  ** Put pointer to data in R3
54AA   C081   MOV  R1,R2                  ** Pab buffer address in R2
54AC   C043   MOV  R3,R1                  ** Put pointer to data in R1
54AE   C129   MOV  @>0056(R9),R4          ** Get the FCB pointer
54B0   0056
54B2   0604   DEC  R4                     ** Point to Drive #
54B4   0429   BLWP @>005A(R9)              ** Set VDPWA to it
54B6   005A
54B8   0082   DATA >0082
54BA   D16F   MOVB @>FBFE(R15),R5         ** Read the drive #
54BC   FBFE
54BE   0265   ORI  R5,>8000               ** Set Updated Data Buffer bit
54C0   8000
54C2   0429   BLWP @>005A(R9)              ** Set VDPWA back to it
54C4   005A
54C6   0083   DATA >0083
54C8   DBC5   MOVB R5,@>FFFE(R15)         ** Write it back
54CA   FFFE
54CC   0460   B    @>533E                 ** Go finish  return
54CE   533E

** Subroutine to find the correct File Control Block and read the status
** byte from the PAB returned in MSByte of R0

54D0   06A0   BL   @>4658                 ** Find FCB of the file
54D2   4658
54D4   4DA4   DATA >4DA4
54D6   C104   MOV  R4,R4                  ** Found the FCB?
54D8   1303   JEQ  >54E0                  ** Yes?, then jump
```

```
54DA   06A0   BL   @>4C72                  ** Report error
54DC   4C72
54DE   E000   DATA >E000                  ** File not open
54E0   0581   INC  R1                     ** Point to Filename of FCB
54E2   CA41   MOV  R1,@>0056(R9)          ** Put the pointer in >8356
54E4   0056
54E6   C129   MOV  @>0054(R9),R4          ** Put PAB pointer in R4
54E8   0054
54EA   0584   INC  R4                     ** Point to status byte
54EC   04C0   CLR  R0
54EE   0429   BLWP @>005A(R9)             ** Set VDPWA to it
54F0   005A
54F2   0082   DATA >0082
54F4   D02F   MOVB @>FBFE(R15),R0         ** Read the byte
54F6   FBFE
54F8   0460   B    @>4676                 ** Return
54FA   4676

** Subroutine to read the Status byte from the Descriptor Record and
** return it in the MSByte of R0

54FC   C129   MOV  @>0056(R9),R4          ** Put the FCB pointer in R4
54FE   0056
5500   0224   AI   R4,12                  ** Point to the status byte
5502   000C
5504   0429   BLWP @>005A(R9)             ** Set the VDPWA to it
5506   005A
5508   0082   DATA >0082
550A   D02F   MOVB @>FBFE(R15),R0         ** Read the byte
550C   FBFE
550E   045B   B    *R11                   ** Return

** Subroutine to get current record # (add 1 for next access) and compare
** the record # we want with the maximum record # [Fixed records routine]
** Returns AU #,in R0, offset into buffer in R1

5510   D16F   MOVB @>FBFE(R15),R5         ** Read maximum # of records
5512   FBFE                               ** per AU
5514   0985   SRL  R5,8                   ** Put it in 16 bits
5516   1602   JNE  >551C                  ** Not = 0? then jump
5518   0205   LI   R5,256
551A   0100
551C   C0E9   MOV  @>0054(R9),R3          ** Put PAB pointer in R3
551E   0054
5520   0223   AI   R3,6                   ** Point to record #
5522   0006
5524   0429   BLWP @>005A(R9)             ** Set VDPWA to it
5526   005A
5528   0062   DATA >0062
552A   D06F   MOVB @>FBFE(R15),R1         ** Get the record # into R1
552C   FBFE
552E   06C1   SWPB R1
5530   D06F   MOVB @>FBFE(R15),R1
5532   FBFE
5534   06C1   SWPB R1                     ** Adjust it
5536   C001   MOV  R1,R0                  ** Put it in R0
5538   1101   JLT  >553C                  ** MSBit set?, jump to error
                                          ** (too big!)
553A   1003   JMP  >5542
553C   06A0   BL   @>4C72                  ** Report error
```

```
5540  8000  DATA >8000                    ** Illegal record #
5542  0580  INC  R0                      ** Add 1 to the record #
5544  0429  BLWP @>005A(R9)              ** Set VDPWA back to the
5546  005A                               ** PAB offset
5548  0063  DATA >0063
554A  DBC0  MOVB R0,@>FFFE(R15)          ** Write the record # back
554C  FFFE
554E  06C0  SWPB R0
5550  DBC0  MOVB R0,@>FFFE(R15)
5552  FFFE
5554  04C0  CLR  R0                      ** Set up for divide
5556  C0C1  MOV  R1,R3                   ** Put Record # in R3
5558  3C05  DIV  R5,R0                   ** Divide Rec # by # rec's
                                         ** per AU
555A  C0A9  MOV  @>0056(R9),R2           ** Put FCB pointer in R2
555C  0056
555E  0222  AI   R2,18                   ** Point to # fixed length
                                         ** records
5560  0012                               ** # AU's for var length recs
5562  0429  BLWP @>005A(R9)              ** Set VDPWA to it
5564  005A
5566  0042  DATA >0042
5568  D0AF  MOVB @>FBFE(R15),R2          ** Read it
556A  FBFE
556C  06C2  SWPB R2
556E  D0AF  MOVB @>FBFE(R15),R2
5570  FBFE
5572  8083  C    R3,R2                   ** Compare # we want to # of
                                         ** fixed length records or
                                         ** # AU's for var length recs
5574  045B  B    *R11                    ** Return

** Subroutine to put correct AU for a record into the data buffer of the
** FCB Returned>> R0= Rec length R1= PAB buffer address, R2= pointer to
** data in the record in the FCB

5576  0429  BLWP @>005A(R9)              ** Save R1 on the stack
5578  005A
557A  4000  DATA >4000
557C  0224  AI   R4,-18                  ** Current AU Offset pointer
557E  FFEE
5580  0429  BLWP @>005A(R9)              ** Set VDPWA to it
5582  005A
5584  0082  DATA >0082
5586  D16F  MOVB @>FBFE(R15),R5          ** Read it
5588  FBFE
558A  06C5  SWPB R5
558C  D16F  MOVB @>FBFE(R15),R5
558E  FBFE
5590  0B85  SRC  R5,8                    ** Adjust it
5592  1105  JLT  >559E                   ** MSBit set?, then jump
5594  8005  C    R5,R0                   ** Compare AU # we want to
                                         ** current
5596  130B  JEQ  >55AE                   ** Are they the same?,
                                         ** Yes?, jump
5598  06A0  BL   @>4658                  ** Check if updated buffer
559A  4658                               ** area
559C  489A  DATA >489A
559E  C0C0  MOV  R0,R3                   ** Put AU # we want in R3
```

```
55A0   06A0   BL   @>55EE                  ** Put it in the FCB
55A2   55EE
55A4   0227   AI   R7,256                  ** For data buffer pointer
55A6   0100
55A8   06A0   BL   @>4658                  ** Read the AU for this record
55AA   4658
55AC   492E   DATA >492E
55AE   0429   BLWP @>005A(R9)              ** Restore R1 from the stack
55B0   005A
55B2   4001   DATA >4001
55B4   C0E9   MOV  @>0056(R9),R3           ** Put the FCB pointer in R3
55B6   0056
55B8   0223   AI   R3,17                   ** Point to logical rec length
55BA   0011
55BC   0429   BLWP @>005A(R9)              ** Set VDPWA to it
55BE   005A
55C0   0062   DATA >0062
55C2   D02F   MOVB @>FBFE(R15),R0          ** Read it
55C4   FBFE
55C6   0980   SRL  R0,8                    ** Put it in 16 bits
55C8   3840   MPY  R0,R1                   ** Multiply by offset into
                                           ** sector
55CA   A0A9   A    @>0056(R9),R2           ** Add the FCB offset
55CC   0056
55CE   0222   AI   R2,256                  ** Offset into data buffer
55D0   0100
55D2   C0E9   MOV  @>0054(R9),R3           ** Get the PAB pointer
55D4   0054
55D6   05C3   INCT R3                      ** Point to PAB buff address
55D8   0429   BLWP @>005A(R9)              ** Set VDPWA to it
55DA   005A
55DC   0062   DATA >0062
55DE   D06F   MOVB @>FBFE(R15),R1          ** Put PAB Buff address in R1
55E0   FBFE
55E2   06C1   SWPB R1
55E4   D06F   MOVB @>FBFE(R15),R1
55E6   FBFE
55E8   06C1   SWPB R1
55EA   0460   B    @>4676                  ** Return
55EC   4676

** Subroutine to put AU #, and Logical Record Offset for Variable
** files, in the FCB. Passed in from R0, R5 respectively.

55EE   C1E9   MOV  @>0056(R9),R7           ** Put FCB pointer in R7
55F0   0056
55F2   C047   MOV  R7,R1                   ** Put it in R1
55F4   0221   AI   R1,-6                   ** Point to Current AU #
55F6   FFFA
55F8   0429   BLWP @>005A(R9)              ** Set VDPWA to it
55FA   005A
55FC   0023   DATA >0023
55FE   DBC0   MOVB R0,@>FFFE(R15)          ** Write the word
5600   FFFE
5602   06C0   SWPB R0
5604   DBC0   MOVB R0,@>FFFE(R15)
5606   FFFE
5608   0221   AI   R1,4                    ** Point to Logical Record
560A   0004                                ** Offset byte
560C   06C5   SWPB R5                      ** Set up for writing
```

```
560E   0429   BLWP @>005A(R9)                  ** Set VDPWA to it
5610   005A
5612   0023   DATA >0023
5614   DBC5   MOVB R5,@>FFFE(R15)              ** Write the byte
5616   FFFE
5618   045B   B    *R11                        ** Return

** Subroutine to set update file descriptor bit and write # of fixed length
** records or sectors (Variable files) to the FCB

561A   C0A9   MOV  @>0056(R9),R2               ** Get the FCB pointer
561C   0056
561E   0429   BLWP @>005A(R9)                  ** Set VDPWA to it
5620   005A
5622   0042   DATA >0042
5624   D2AF   MOVB @>FBFE(R15),R10             ** Read 1st byte of filename
5626   FBFE
5628   026A   ORI  R10,>8000                   ** Set Updated File
562A   8000                                    ** Descriptor bit
562C   0429   BLWP @>005A(R9)                  ** Set VDPWA back to it
562E   005A
5630   0043   DATA >0043
5632   DBCA   MOVB R10,@>FFFE(R15)             ** Write the byte back
5634   FFFE
5636   0222   AI   R2,18                       ** Point to # of recs or
5638   0012                                    ** or sectors ,
563A   0583   INC  R3                          ** Add 1 to the count
563C   0429   BLWP @>005A(R9)                  ** Set VDPWA to it
563E   005A
5640   0043   DATA >0043
5642   06C3   SWPB R3                          ** Write the word
5644   DBC3   MOVB R3,@>FFFE(R15)              ** (LSByte,MSByte)
5646   FFFE
5648   06C3   SWPB R3
564A   DBC3   MOVB R3,@>FFFE(R15)
564C   FFFE
564E   045B   B    *R11                        ** Return

** Subroutine to read Logical Record Offset, Current AU offset, Total AU's
** in file

5650   C229   MOV  @>0056(R9),R8               ** Get FCB pointer
5652   0056
5654   C108   MOV  R8,R4                       ** Put it in R4
5656   0224   AI   R4,256                      ** Point to data buffer of FCB
5658   0100
565A   0648   DECT R8                          ** Point to logical rec offset
565C   0429   BLWP @>005A(R9)                  ** Set VDPWA to it
565E   005A
5660   0102   DATA >0102
5662   D0AF   MOVB @>FBFE(R15),R2              ** Read it
5664   FBFE
5666   0982   SRL  R2,8                       ** Put it in 16 bits
5668   C28B   MOV  R11,R10                     ** Save the return address
566A   06A0   BL   @>4B74                      ** Get current AU # offset
566C   4B74
566E   FFFC   DATA >FFFC
5670   C0C0   MOV  R0,R3                       ** Put it in R3
5672   C2CA   MOV  R10,R11                     ** Restore return address
5674   C002   MOV  R2,R0                       ** Put log rec offset in R0
```

```
5676   0460   B    @>555A
5678   555A

** Restore/Rewind Opcode

567A   06A0   BL   @>4658                ** Find the file control block
567C   4658                              ** and get the status byte
567E   54D0   DATA >54D0
5680   0429   BLWP @>005A(R9)             ** Save register 0 on stack
5682   005A
5684   8000   DATA >8000
5686   0240   ANDI R0,>0600              ** Mask all bits but the mode
5688   0600
568A   1305   JEQ  >5696                 ** Update mode?, then jump
568C   0280   CI   R0,>0400              ** Input mode?
568E   0400
5690   1302   JEQ  >5696                 ** Yes?, then jump
5692   0460   B    @>50BA                ** Report err(illegal opertn)
5694   50BA
5696   06A0   BL   @>4658                ** Mask off updated data
5698   4658                              ** buffer bit, get logical
569A   489A   DATA >489A                 ** offset in R3
569C   0429   BLWP @>005A(R9)             ** Restore R0 from the stack
569E   005A
56A0   8001   DATA >8001
56A2   0240   ANDI R0,>0100              ** Relative file type?
56A4   0100
56A6   1611   JNE  >56CA                 ** Yes?, then jump
56A8   04C2   CLR  R2                    ** Set logical rec offset to 0
                                         ** for pointer in sector
56AA   0703   SETO R3                    ** Set log rec offset to -1
56AC   06A0   BL   @>52AA                ** Set logical rec offset to
56AE   52AA                              ** value in R3
56B0   04C0   CLR  R0                    ** Pab pointer >0000
56B2   C229   MOV  @>0054(R9),R8         ** Put PAB pointer in R8
56B4   0054
56B6   0228   AI   R8,6                  ** Point to record # in PAB
56B8   0006
56BA   0429   BLWP @>005A(R9)             ** Set VDPWA to it
56BC   005A
56BE   0103   DATA >0103
56C0   DBC0   MOVB R0,@>FFFE(R15)        ** Set Record number to >0000
56C2   FFFE
56C4   1000   NOP
56C6   DBC0   MOVB R0,@>FFFE(R15)
56C8   FFFE
56CA   0460   B    @>4676                ** Return
56CC   4676

** Load opcode **

56CE   06A0   BL   @>4658                ** Go find file on the disk
56D0   4658
56D2   4E02   DATA >4E02
56D4   C104   MOV  R4,R4                 ** Found it?
56D6   1303   JEQ  >56DE                 ** Yes?, then jump
56D8   06A0   BL   @>4C72                ** Report error
56DA   4C72
56DC   E000   DATA >E000                 ** File error
56DE   06A0   BL   @>4B70                ** Read status flag from FCB
```

```
56E0  4B70
56E2  000C    DATA >000C
56E4  0240    ANDI R0,>0100          ** Is it PROGRAM file type?
56E6  0100
56E8  13F7    JEQ  >56D8             ** No?, then jump
56EA  05C8    INCT R8                ** Point to # of blocks
                                     ** in file
56EC  0429    BLWP @>005A(R9)        ** Set VDPWA to it
56EE  005A
56F0  0102    DATA >0102
56F2  D06F    MOVB @>FBFE(R15),R1    ** Read the word
56F4  FBFE
56F6  06C1    SWPB R1
56F8  D06F    MOVB @>FBFE(R15),R1
56FA  FBFE
56FC  0B81    SRC  R1,8              ** Adjust it
56FE  13EC    JEQ  >56D8             ** File empty?, then jump
5700  06A0    BL   @>57C0            ** Read data buffer address,
5702  57C0                           ** # of sectors in the
                                     ** program from PAB
5704  05C8    INCT R8                ** Point to End of File offset
                                     ** in the last used sector
5706  04C4    CLR  R4
5708  0429    BLWP @>005A(R9)        ** Set VDPWA to it
570A  005A
570C  0102    DATA >0102
570E  D12F    MOVB @>FBFE(R15),R4    ** Read the byte
5710  FBFE
5712  8081    C    R1,R2             ** Compare # sectors to read
                                     ** to actual # in the file
5714  1BE1    JH   >56D8             ** Too many?, then jump
5716  1602    JNE  >571C             ** Not the same amount?, jump
5718  8100    C    R0,R4             ** Compare # bytes left in
                                     ** buffer to EOF offset
571A  1ADE    JL   >56D8             ** Not enough room?, jump
571C  04C3    CLR  R3                ** Clear AU count
571E  06C4    SWPB R4                ** EOF offset in 16 bits
5720  0601    DEC  R1                ** Last sector to read?
5722  130D    JEQ  >573E             ** Yes?, then jump
5724  0429    BLWP @>005A(R9)        ** Save R0,R1,R3,R4,R7
5726  005A                           ** on stack
5728  D900    DATA >D900
572A  06A0    BL   @>4658            ** Find the next sector and
572C  4658                           ** read it from the disk
572E  492E    DATA >492E
5730  0429    BLWP @>005A(R9)        ** Restore R7,R4,R3,R1,R0 from
5732  005A                           ** the stack
5734  D901    DATA >D901
5736  0583    INC  R3                ** Add 1 to AU count
5738  0227    AI   R7,256            ** Update VDP buffer pointer
573A  0100
573C  10F1    JMP  >5720             ** Do it again!
573E  C104    MOV  R4,R4             ** 0 leftover bytes to read?
5740  1604    JNE  >574A             ** No?, then jump
5742  06A0    BL   @>4658            ** Find last sector and read
5744  4658
5746  492E    DATA >492E
5748  1011    JMP  >576C
574A  C147    MOV  R7,R5             ** Pointer to VDP Buff in R5
574C  C1E9    MOV  @>0056(R9),R7     ** Put FCB pointer in R7
```

```
574E  0056
5750  0227   AI   R7,256                 ** Point to data buffer of FCB
5752  0100
5754  0429   BLWP @>005A(R9)             ** Save R4,R5,R7 on the stack
5756  005A
5758  0D00   DATA >0D00
575A  06A0   BL   @>4658                 ** Find last sector and read
575C  4658
575E  492E   DATA >492E
5760  0429   BLWP @>005A(R9)             ** Restore R2,R1,R0 from stack
5762  005A
5764  E001   DATA >E001
5766  06A0   BL   @>4658                 ** Move bytes from FCB buffer
5768  4658                               ** to VDP memory buffer
576A  533E   DATA >533E
576C  0460   B    @>487A                 ** Go close the file
576E  487A

** Save opcode

5770  06A0   BL   @>4658                 ** Check if file on the disk,
5772  4658                               ** create the file
5774  4798   DATA >4798
5776  06A0   BL   @>57C0                 ** Get data buffer address and
5778  57C0                               ** number of bytes to save
577A  04C3   CLR  R3                     ** Clear sector count
577C  0429   BLWP @>005A(R9)             ** Save R0,R2,R3,R7 on stack
577E  005A
5780  B100   DATA >B100
5782  06A0   BL   @>4658                 ** Write a sector of data
5784  4658
5786  494E   DATA >494E
5788  0429   BLWP @>005A(R9)             ** Restore R7,R3,R2,R0
578A  005A                               ** from stack
578C  B101   DATA >B101
578E  0583   INC  R3                     ** Add 1 to the count
5790  0227   AI   R7,256                ** Add 256 to buffer pointer
5792  0100
5794  0602   DEC  R2                     ** Done yet?
5796  16F2   JNE  >577C                  ** No?, then jump
5798  C069   MOV  @>0056(R9),R1          ** Put FCB pointer in R1
579A  0056
579C  0221   AI   R1,12                  ** Point to the status byte
579E  000C
57A0  0202   LI   R2,>0100               ** Program file status byte
57A2  0100
57A4  0429   BLWP @>005A(R9)             ** Set VDPWA to it
57A6  005A
57A8  0023   DATA >0023
57AA  DBC2   MOVB R2,@>FFFE(R15)         ** Write the byte
57AC  FFFE
57AE  0221   AI   R1,4                   ** Point to End of file offset
57B0  0004
57B2  0429   BLWP @>005A(R9)             ** Set VDPWA to it
57B4  005A
57B6  0023   DATA >0023
57B8  DBC0   MOVB R0,@>FFFE(R15)         ** Write the byte
57BA  FFFE
57BC  0460   B    @>487A
57BE  487A
```

```
** Subroutine to get data buffer address and number of bytes to read or
** write for a save or load command.

57CØ   CØ29   MOV  @>ØØ54(R9),RØ        ** Put the PAB pointer in RØ
57C2   ØØ54
57C4   Ø5CØ   INCT RØ                   ** Point to DATA buff address
57C6   Ø429   BLWP @>ØØ5A(R9)           ** Set VDPWA to it
57C8   ØØ5A
57CA   ØØØ2   DATA >ØØØ2
57CC   D1EF   MOVB @>FBFE(R15),R7       ** Read it into R7
57CE   FBFE
57DØ   Ø6C7   SWPB R7
57D2   D1EF   MOVB @>FBFE(R15),R7
57D4   FBFE
57D6   Ø6C7   SWPB R7                   ** Adjust it
57D8   Ø22Ø   AI   RØ,4                 ** Point to Max # bytes to
57DA   ØØØ4                             ** save or load
57DC   Ø429   BLWP @>ØØ5A(R9)           ** Set VDPWA to it
57DE   ØØ5A
57EØ   ØØØ2   DATA >ØØØ2
57E2   DØAF   MOVB @>FBFE(R15),R2       ** Read # of 256 byte blocks
57E4   FBFE                             ** into R2
57E6   Ø982   SRL  R2,8                 ** Put it in 16 bits
57E8   Ø4CØ   CLR  RØ
57EA   DØ2F   MOVB @>FBFE(R15),RØ       ** Read # extra bytes we need
57EC   FBFE
57EE   13Ø1   JEQ  >57F2                ** Ø extra?, then jump
57FØ   Ø582   INC  R2                   ** Add 1 to number of blocks
57F2   Ø45B   B    *R11                 ** Return

** Status opcode **

57F4   Ø6AØ   BL   @>4658               ** Find FCB of the file
57F6   4658
57F8   4DA4   DATA >4DA4
57FA   C1Ø4   MOV  R4,R4                ** Found it? (open file)
57FC   131Ø   JEQ  >581E                ** Yes?, then jump
57FE   Ø6AØ   BL   @>4658               ** Put the file descriptor rec
58ØØ   4658                             ** in the first available FCB
58Ø2   4EØC   DATA >4EØC
58Ø4   Ø2ØØ   LI   RØ,>8ØØØ            ** Set file does not exist bit
58Ø6   8ØØØ
58Ø8   C1Ø4   MOV  R4,R4                ** Does file exist?
58ØA   1649   JNE  >589E                ** No?, then jump
58ØC   CØ69   MOV  @>ØØ56(R9),R1        ** Put FCB pointer in R1
58ØE   ØØ56
581Ø   Ø4C2   CLR  R2                   ** >ØØ (FCB Available)
5812   Ø429   BLWP @>ØØ5A(R9)           ** Set VDPWA to it
5814   ØØ5A
5816   ØØ23   DATA >ØØ23
5818   DBC2   MOVB R2,@>FFFE(R15)       ** Write a >ØØ(FCB available)
581A   FFFE
581C   1Ø36   JMP  >588A
581E   Ø6AØ   BL   @>4658               ** Read the status byte(set
582Ø   4658                             ** up CPU RAM >8356)
5822   54EØ   DATA >54EØ
5824   Ø6AØ   BL   @>54FC               ** Read the status byte
5826   54FC
5828   11Ø1   JLT  >582C                ** Relative file type?, jump
582A   1ØØ6   JMP  >5838
```

```
** Variable length records **

582C   06A0   BL   @>4658
582E   4658
5830   5362   DATA >5362
5832   1013   JMP  >585A
5834   04C2   CLR  R2                  ** Clear out open file status
5836   1029   JMP  >588A
5838   D16F   MOVB @>FBFE(R15),R5      ** Read # records/AU
583A   FBFE
583C   0985   SRL  R5,8                ** Put in 16 bits
583E   1602   JNE  >5844               ** Not zero?, then jump
5840   0205   LI   R5,256              ** 256 bytes per AU
5842   0100
5844   06A0   BL   @>4B6A              ** Read current rec # in PAB
5846   4B6A
5848   0006   DATA >0006
584A   C0C0   MOV  R0,R3               ** Put in R3
584C   111C   JLT  >5886               ** High bit set(too many
                                       ** recs?), then jump
584E   06A0   BL   @>555A              ** Compare amount to max
5850   555A
5852   04C2   CLR  R2                  ** Clear for divide
5854   1A1A   JL   >588A               ** Less than max?, then jump
5856   3C85   DIV  R5,R2               ** Divide by 256
5858   C0C2   MOV  R2,R3               ** Put sector offset in R3
585A   06A0   BL   @>4B70              ** Read # sectors allocated
585C   4B70
585E   000E   DATA >000E
5860   0583   INC  R3                  ** Add 1 to offset
5862   0202   LI   R2,>0100           ** Set EOF bit
5864   0100
5866   60C0   S    R0,R3
5868   1501   JGT  >586C
586A   100F   JMP  >588A
586C   06A0   BL   @>4658              ** Update VIB sector
586E   4658
5870   4CD2   DATA >4CD2
5872   C103   MOV  R3,R4
5874   C205   MOV  R5,R8
5876   0228   AI   R8,>000A
5878   000A
587A   06A0   BL   @>5A68              ** Get Available sectors
587C   5A68                           ** on disk into R3
587E   0202   LI   R2,>0100           ** Set at EOF bit
5880   0100
5882   8103   C    R3,R4
5884   1402   JHE  >588A
5886   0202   LI   R2,>0200           ** Set bit at physical end
5888   0200
588A   06A0   BL   @>4B70              ** Read file status flags
588C   4B70
588E   000C   DATA >000C
5890   0240   ANDI R0,>8F00           ** Variable file type?
5892   8F00
5894   1502   JGT  >589A               ** No?, then jump
5896   0260   ORI  R0,>0080           ** Set variable status bit
5898   0080
589A   0A30   SLA  R0,3                ** Adjust for return status
                                       ** byte
```

```
589C   F002   SOCB R2,R0                  ** Set bits according to R2
                                          ** for return status
589E   C069   MOV  @>0054(R9),R1          ** Pointer to PAB in R1
58A0   0054
58A2   0221   AI   R1,8                   ** Point to screen offset byte
58A4   0008
58A6   0429   BLWP @>005A(R9)             ** Set VDPWA to it
58A8   005A
58AA   0023   DATA >0023
58AC   DBC0   MOVB R0,@>FFFE(R15)         ** Write status byte to it
58AE   FFFE                               ** Return
58B0   0460   B    @>4676
58B2   4676

** Open disk directory routine **

58B4   D02F   MOVB @>FBFE(R15),R0         ** Get status byte from PAB
58B6   FBFE
58B8   0240   ANDI R0,>1E00               ** Mask out error, file type
58BA   1E00                               ** bits
58BC   0280   CI   R0,>0C00               ** Is it INPUT,INTERNAL?
58BE   0C00
58C0   1303   JEQ  >58C8                  ** Yes?, then jump
58C2   06A0   BL   @>4C9E                 ** Report error
58C4   4C9E
58C6   4000   DATA >4000                  ** Bad open attribute
58C8   06A0   BL   @>4B6A                 ** Get record length into R0
58CA   4B6A
58CC   0004   DATA >0004
58CE   0980   SRL  R0,8                   ** Put it in 16 bits
58D0   1303   JEQ  >58D8                  ** 0?, jump and set default
58D2   0280   CI   R0,38                  ** Is it 38 bytes for length?
58D4   0026
58D6   16F5   JNE  >58C2                  ** No?, then jump to error
58D8   0200   LI   R0,>2600              ** Set to 38 bytes for length
58DA   2600
58DC   0429   BLWP @>005A(R9)             ** Set VDPWA to record length
58DE   005A
58E0   0103   DATA >0103
58E2   DBC0   MOVB R0,@>FFFE(R15)         ** Write it
58E4   FFFE
58E6   04C7   CLR  R7
58E8   06A0   BL   @>5AA2                 ** Find control block
58EA   5AA2
58EC   58F4   DATA >58F4                  ** Address to return to
                                          ** if no error
58EE   06A0   BL   @>4C72                 ** Report error
58F0   4C72
58F2   E000   DATA >E000                  ** Device already open
58F4   C1C7   MOV  R7,R7                  ** Available FCB?
58F6   1603   JNE  >58FE                  ** Yes?, then jump
58F8   06A0   BL   @>4C9E                 ** Report error
58FA   4C9E
58FC   8000   DATA >8000                  ** Out of buffer space
58FE   0429   BLWP @>005A(R9)             ** Set VDPWA to drive number
5900   005A                               ** first char of filename
5902   00E3   DATA >00E3                  ** of FCB
5904   DBC3   MOVB R3,@>FFFE(R15)         ** Write drive number
5906   FFFE
5908   06C3   SWPB R3                    ** Switch it
```

```
590A   DBC3   MOVB R3,@>FFFE(R15)          ** Write a space to filename
590C   FFFE
590E   0460   B    @>4676                 ** Return
5910   4676

** Close disk directory routine **

5912   06A0   BL   @>5AA2                  ** Find FCB for disk directory
5914   5AA2                                ** If found return at >5918
5916   58EE   DATA >58EE                   ** If not found return
5918   04C0   CLR  R0                      ** at >58EE
591A   0429   BLWP @>005A(R9)              ** Set VDPWA, to 1st char
591C   005A                                ** of filename
591E   0103   DATA >0103
5920   DBC0   MOVB R0,@>FFFE(R15)          ** Write a >00 (available)
5922   FFFE
5924   0460   B    @>4676                 ** Return
5926   4676

** Read disk directory routine **

5928   06A0   BL   @>5AA2                  ** Find FCB for disk directory
592A   5AA2                                ** If found return at >592E
592C   58EE   DATA >58EE                   ** If not found return
                                           ** at >58EE
592E   0588   INC  R8                      ** Add 1 point to 2nd byte of
                                           ** filename
5930   C148   MOV  R8,R5                   ** Duplicate in R5
5932   06A0   BL   @>4B6A                  ** Get record # from PAB
5934   4B6A
5936   0006   DATA >0006
5938   C080   MOV  R0,R2                   ** Put it in R2
593A   0582   INC  R2                      ** Add 1
593C   0429   BLWP @>005A(R9)              ** Set VDPWA back to it
593E   005A
5940   0103   DATA >0103
5942   DBC2   MOVB R2,@>FFFE(R15)          ** Write it back from R2
5944   FFFE
5946   06C2   SWPB R2
5948   DBC2   MOVB R2,@>FFFE(R15)
594A   FFFE
594C   0A10   SLA  R0,1                    ** Multiply Record # by 2
594E   D000   MOVB R0,R0                   ** Less than 256?
5950   1303   JEQ  >5958                   ** Yes?, then jump
5952   06A0   BL   @>4C72                  ** Report error
5954   4C72
5956   A000   DATA >A000                   ** Attempt to read past EOF
5958   0702   SETO R2                      ** Read flag
595A   C100   MOV  R0,R4                   ** Duplicate it in R4
595C   133B   JEQ  >59D4                   ** Record #0?, then jump
595E   0204   LI   R4,>0001                ** Sector #1 (File link
5960   0001                                ** sector)
5962   06A0   BL   @>4658                  ** Sector 1 into file
5964   4658                                ** descriptor
5966   4D4E   DATA >4D4E
5968   C205   MOV  R5,R8                   ** Pointer to Record #
596A   0225   AI   R5,255                  ** Point to data buffer area
596C   00FF
596E   0640   DECT R0                      ** Subtract 2 from rec #
                                           ** (corrected)
```

```
5970   A200   A    R0,R8          ** Pointer into directory link
5972   06A0   BL   @>4B76         ** Get sector address of file
5974   4B76                       ** descriptor for this file
5976   C100   MOV  R0,R4          ** Save it in R4
5978   1328   JEQ  >59CA          ** 0?, no file there, jump
597A   06A0   BL   @>4658         ** Get the sector into data
597C   4658                       ** buffer area
597E   4D4E   DATA >4D4E
5980   06A0   BL   @>4B70         ** Get # sectors allocated to
5982   4B70                       ** file
5984   010E   DATA >010E
5986   C180   MOV  R0,R6          ** Save it in R6
5988   0586   INC  R6            ** Add 1 for file descriptor
                                  ** sector
598A   D0EF   MOVB @>FBFE(R15),R3 ** Increment VDPWA
598C   FBFE
598E   0202   LI   R2,>0A00       ** 10 bytes for filename
5990   0A00
5992   D0EF   MOVB @>FBFE(R15),R3 ** Get max rec size from file
5994   FBFE                       ** descriptor sector
5996   0983   SRL  R3,8           ** Put it in 16 bits
5998   0648   DECT R8            ** Pointer to file type byte
599A   0429   BLWP @>005A(R9)     ** Set VDPWA to it
599C   005A
599E   0102   DATA >0102
59A0   D02F   MOVB @>FBFE(R15),R0 ** Get it from file descriptor
59A2   FBFE
59A4   C1C0   MOV  R0,R7          ** Save it in R7
59A6   0240   ANDI R0,>0800       ** Mask out all but protection
59A8   0800                       ** bit
59AA   51C0   SZCB R0,R7          ** Mask off protection bit
59AC   0987   SRL  R7,8           ** Put it in 16 bits
59AE   0587   INC  R7             ** Add 1(Basic offset)
59B0   0287   CI   R7,2           ** Is it memory image type?
59B2   0002
59B4   1602   JNE  >59BA          ** No?, then jump
59B6   0227   AI   R7,3           ** Add 3, basic file type #5
59B8   0003
59BA   0287   CI   R7,8           ** Should we return positive
59BC   0008                       ** number?
59BE   1A02   JL   >59C4          ** Yes?, then jump
59C0   0227   AI   R7,>FF81
59C2   FF81
59C4   0A40   SLA  R0,4           ** Move protection bit
                                  ** to MSBit
59C6   E1C0   SOC  R0,R7          ** Set MSBit if protected
59C8   1004   JMP  >59D2          ** (negative value)
59CA   04C2   CLR  R2             ** 0 bytes for filename
59CC   04C6   CLR  R6             ** 0 for all parameters
59CE   04C3   CLR  R3
59D0   04C7   CLR  R7
59D2   100F   JMP  >59F2

** Record #0 for of directory routine **

59D4   0225   AI   R5,>00FF       ** Point to data buffer area
59D6   00FF
59D8   06A0   BL   @>4658         ** Get sector 0 of the disk
59DA   4658
59DC   4D4E   DATA >4D4E
```

```
59DE   06A0   BL   @>4B70                ** Get total # sectors on disk
59E0   4B70
59E2   010A   DATA >010A
59E4   C180   MOV  R0,R6                 ** Put it in R6 for return
59E6   0646   DECT R6                    ** Subtract 2 for Volume Info
                                         ** sector and directory link
                                         ** sector
59E8   06A0   BL   @>5A68                ** Calculate # of available
59EA   5A68                              ** sectors
59EC   04C7   CLR  R7                    ** Record type is 0, record 0
59EE   0202   LI   R2,>0A00              ** 10 bytes for disk name
59F0   0A00
59F2   06A0   BL   @>4B6A                ** Get data buffer address
59F4   4B6A
59F6   0002   DATA >0002
59F8   C200   MOV  R0,R8                 ** Put it in R8
59FA   0588   INC  R8                    ** Add 1(length is first byte)
59FC   0982   SRL  R2,8                  ** Put name length in 16 bits
59FE   1316   JEQ  >5A2C                ** 0 length of name?, jump
5A00   04C1   CLR  R1
5A02   0429   BLWP @>005A(R9)            ** Set VDPWA to name
5A04   005A
5A06   00A2   DATA >00A2
5A08   D06F   MOVB @>FBFE(R15),R1        ** Get byte of name
5A0A   FBFE
5A0C   0281   CI   R1,>2000              ** Space?
5A0E   2000
5A10   1309   JEQ  >5A24                ** Yes?, then jump
5A12   0429   BLWP @>005A(R9)            ** Set VDPWA to data buffer
5A14   005A
5A16   0103   DATA >0103
5A18   DBC1   MOVB R1,@>FFFE(R15)        ** Write the byte of name
5A1A   FFFE
5A1C   0585   INC  R5                    ** Update pointers
5A1E   0588   INC  R8
5A20   0602   DEC  R2                    ** 10 bytes?
5A22   16EF   JNE  >5A02                 ** No?, jump and read another
5A24   0502   NEG  R2                    ** Figure # of bytes in name
5A26   0222   AI   R2,10
5A28   000A
5A2A   06C2   SWPB R2                    ** Put length in MSByte
5A2C   0429   BLWP @>005A(R9)            ** Set VDPWA, start of buffer
5A2E   005A
5A30   0003   DATA >0003
5A32   DBC2   MOVB R2,@>FFFE(R15)        ** Write the length
5A34   FFFE
5A36   0429   BLWP @>005A(R9)            ** Set VDPWA to data buffer
5A38   005A                              ** after name
5A3A   0103   DATA >0103

                                         ** Return the 3 numeric values

5A3C   C047   MOV  R7,R1
5A3E   06A0   BL   @>5AE6                ** Write it to buffer
5A40   5AE6
5A42   C046   MOV  R6,R1
5A44   06A0   BL   @>5AE6                ** Write it to buffer
5A46   5AE6
5A48   C043   MOV  R3,R1
5A4A   06A0   BL   @>5AE6                ** Write it to buffer
```

```
5A4C   5AE6
5A4E   C229   MOV  @>0054(R9),R8          ** Point to PAB
5A50   0054
5A52   0228   AI   R8,5                   ** Point to char count
5A54   0005
5A56   0200   LI   R0,>2600               ** 38 bytes of data
5A58   2600
5A5A   0429   BLWP @>005A(R9)             ** Set VDPWA
5A5C   005A
5A5E   0103   DATA >0103
5A60   DBC0   MOVB R0,@>FFFE(R15)         ** Write 38 to char count
5A62   FFFE
5A64   0460   B    @>4676                 ** Return
5A66   4676

** Check for # of available sectors on a disk and return it in R3

5A68   0228   AI   R8,>002E               ** Point to Bit Map
5A6A   002E
5A6C   0202   LI   R2,200                ** 200 bytes in bit map
5A6E   00C8
5A70   04C3   CLR  R3
5A72   0429   BLWP @>005A(R9)             ** Set VDPWA to bit map
5A74   005A
5A76   0102   DATA >0102
5A78   D06F   MOVB @>FBFE(R15),R1         ** Read a byte
5A7A   FBFE
5A7C   0221   AI   R1,>0100               ** Add 1 to check if 8 sectors
5A7E   0100                               ** used
5A80   0981   SRL  R1,8                   ** Put it in low byte
5A82   130C   JEQ  >5A9C                  ** All bits used?, then jump
5A84   0601   DEC  R1                    ** Subtract off the added bit
5A86   1603   JNE  >5A8E                  ** All 8 sectors free?, no?,
                                          ** then jump
5A88   0223   AI   R3,8                   ** Add 8 to the count
5A8A   0008
5A8C   1007   JMP  >5A9C
5A8E   0200   LI   R0,8                   ** 8 bits in the block
5A90   0008
5A92   0911   SRL  R1,1                   ** Sector used?
5A94   1801   JOC  >5A98                  ** Yes?, then jump
5A96   0583   INC  R3                    ** Add 1 to the count
5A98   0600   DEC  R0                    ** Done yet?
5A9A   16FB   JNE  >5A92                  ** No?, then jump
5A9C   0602   DEC  R2                    ** Done checking all bytes?
5A9E   16EC   JNE  >5A78                  ** No?, then jump
5AA0   045B   B    *R11                   ** Return

** Subroutine to find the file control block for the disk directory.
** If it is found then it returns to the word after the DATA statement
** and continues execution. If it is not found it returns to the address
** contained in the data statement and resumes execution there.

5AA2   C2BB   MOV  *R11+,R10              ** 2nd return address in R10
5AA4   C14B   MOV  R11,R5                 ** 1st return address in R5
5AA6   C229   MOV  @>0056(R9),R8          ** Put DVA pointer in R8
5AA8   0056
5AAA   0228   AI   R8,3                   ** Point to max # files open
5AAC   0003
5AAE   0429   BLWP @>005A(R9)             ** Set VDPWA to it
```

```
5AB0   005A
5AB2   0102   DATA >0102
5AB4   D0AF   MOVB @>FBFE(R15),R2          ** Get the byte
5AB6   FBFE
5AB8   0882   SRA  R2,8                    ** Put it in 16 bits
5ABA   0228   AI   R8,6                    ** Point to drive # first FCB
5ABC   0006
5ABE   0203   LI   R3,>0020                ** Put space char in LSByte R3
5AC0   0020
5AC2   D0C6   MOVB R6,R3                   ** Put drive # want in MSByte
5AC4   06A0   BL   @>4B76                  ** Read drive # and first char
5AC6   4B76                                ** of filename from FCB
5AC8   8003   C    R3,R0                   ** Same?
5ACA   1309   JEQ  >5ADE                   ** Yes?, already open so jump
5ACC   0240   ANDI R0,>00FF                ** Is it a 0?,(FCB available?)
5ACE   00FF
5AD0   1601   JNE  >5AD4                   ** No?, then jump
5AD2   C1C8   MOV  R8,R7                   ** Move pointer to R7
5AD4   0228   AI   R8,518                  ** Point to next FCB
5AD6   0206
5AD8   0602   DEC  R2                      ** Out of File Control Blocks?
5ADA   16F4   JNE  >5AC4                   ** No?, then jump
5ADC   045A   B    *R10                    ** Return without file
                                           ** already open
5ADE   0588   INC  R8                      ** Point to file name of FCB
5AE0   CA48   MOV  R8,@>0056(R9)           ** Put it in DVA pointer
5AE2   0056
5AE4   0455   B    *R5                     ** Return with file
                                           ** already open

** Convert word to floating point routine and place in VDP buffer

5AE6   0202   LI   R2,>0800                ** 8 bytes long for number
5AE8   0800                                ** (Floating point)
5AEA   DBC2   MOVB R2,@>FFFE(R15)          ** Put length in VDP buffer
5AEC   FFFE
5AEE   C141   MOV  R1,R5                   ** Save # in R5
5AF0   0241   ANDI R1,>7FFF                ** Mask off negative bit
5AF2   7FFF
5AF4   0281   CI   R1,100                  ** Is it less than 100?
5AF6   0064
5AF8   1A07   JL   >5B08                   ** Yes?, then jump
5AFA   04C0   CLR  R0                      ** Set up for divide
5AFC   0204   LI   R4,100                  ** Divide by 100
5AFE   0064
5B00   3C04   DIV  R4,R0                   ** Divide the number
5B02   0260   ORI  R0,>4100                ** Raise it to power of 100
5B04   4100
5B06   1005   JMP  >5B12
5B08   C001   MOV  R1,R0                   ** Put the number in R0
5B0A   1302   JEQ  >5B10                   ** Number = to 0? then jump
5B0C   0260   ORI  R0,>4000                ** Number is to the 0 power
5B0E   4000
5B10   04C1   CLR  R1                      ** Less than 100 so clear
                                           ** 3rd byte
5B12   C145   MOV  R5,R5                   ** Was it a negative #?
5B14   1101   JLT  >5B18                   ** Yes?, then jump
5B16   1001   JMP  >5B1A
5B18   0500   NEG  R0                      ** It was negative so neg it
5B1A   DBC0   MOVB R0,@>FFFE(R15)          ** Write the 3 important bytes
```

```
5B1C   FFFE
5B1E   Ø6CØ    SWPB RØ
5B2Ø   DBCØ    MOVB RØ,@>FFFE(R15)
5B22   FFFE
5B24   Ø6C1    SWPB R1
5B26   DBC1    MOVB R1,@>FFFE(R15)
5B28   FFFE
5B2A   Ø2Ø2    LI   R2,5                      ** Now write 5 >ØØ's for
5B2C   ØØØ5                                   ** rest of the number
5B2E   DBC2    MOVB R2,@>FFFE(R15)
5B3Ø   FFFE
5B32   Ø6Ø2    DEC  R2                        ** Done yet?
5B34   16FC    JNE  >5B2E                     ** No?, then jump
5B36   Ø45B    B    *R11                      ** Return

** Routine >1Ø entry point (Sector Read/Write) **

5B38   C1CB    MOV  R11,R7                    ** Save return address
5B3A   Ø6AØ    BL   @>4724                    ** Init routine
5B3C   4724
5B3E   CA69    MOV  @>ØØ5Ø(R9),@>Ø04A(R9)     ** Put sector # at >834A
5B4Ø   ØØ5Ø
5B42   Ø04A
5B44   Ø46Ø    B    @>4ØE8
5B46   4ØE8

** Routine >11 entry point (Initialize disk) **

5B48   C1CB    MOV  R11,R7                    ** Save return address
5B4A   Ø6AØ    BL   @>4724                    ** Init routine
5B4C   4724
5B4E   Ø46Ø    B    @>42AC
5B5Ø   42AC

** Routine >12 entry point (Modify file protection) **

5B52   C1CB    MOV  R11,R7                    ** Save return address
5B54   Ø6AØ    BL   @>4724                    ** Init routine
5B56   4724
5B58   DØ29    MOVB @>Ø04D(R9),RØ             ** Get the protect code byte
5B5A   Ø04D
5B5C   Ø24Ø    ANDI RØ,>Ø8ØØ                  ** Mask out all but 4th bit
5B5E   Ø8ØØ
5B6Ø   Ø429    BLWP @>ØØ5A(R9)                ** Save RØ on the stack
5B62   ØØ5A
5B64   8ØØØ    DATA >8ØØØ
5B66   CØ29    MOV  @>Ø04E(R9),RØ             ** Pointer to filename in RØ
5B68   Ø04E
5B6A   Ø6AØ    BL   @>4658                    ** Check if it on the disk
5B6C   4658
5B6E   5C54    DATA >5C54
5B7Ø   Ø429    BLWP @>ØØ5A(R9)                ** Restore stack into R2
5B72   ØØ5A
5B74   2ØØ1    DATA >2ØØ1
5B76   Ø6AØ    BL   @>4B7Ø                    ** Read status flags from DVA
5B78   4B7Ø
5B7A   ØØØC    DATA >ØØØC
5B7C   Ø24Ø    ANDI RØ,>F7ØØ                  ** Mask off protection bit
5B7E   F7ØØ
5B8Ø   FØØ2    SOCB R2,RØ                     ** Set bit corresponding to
```

```
                                        ** new protect code byte
5B82   0429   BLWP @>005A(R9)           ** Set VDPWA to status flags
5B84   005A
5B86   0103   DATA >0103
5B88   DBC0   MOVB R0,@>FFFE(R15)       ** Write new status flags byte
5B8A   FFFE
5B8C   C229   MOV  @>0056(R9),R8        ** Point to first byte of
5B8E   0056                             ** filename
5B90   0429   BLWP @>005A(R9)           ** Set VDPWA to it
5B92   005A
5B94   0102   DATA >0102
5B96   D02F   MOVB @>FBFE(R15),R0       ** Read it
5B98   FBFE
5B9A   0260   ORI  R0,>8000            ** Set MSBit(updated file
5B9C   8000                             ** descriptor record)
5B9E   0429   BLWP @>005A(R9)           ** Set VDPWA back to it
5BA0   005A
5BA2   0103   DATA >0103
5BA4   DBC0   MOVB R0,@>FFFE(R15)       ** Write it back
5BA6   FFFE
5BA8   0460   B    @>487A               ** Go close the file
5BAA   487A

** Routine >13 entry point (File rename) **

5BAC   C1CB   MOV  R11,R7               ** Save return address
5BAE   06A0   BL   @>4724               ** Init routine
5BB0   4724
5BB2   C029   MOV  @>004E(R9),R0        ** Pointer to new name in R0
5BB4   004E
5BB6   0429   BLWP @>005A(R9)           ** Save R0 on the stack
5BB8   005A
5BBA   8000   DATA >8000
5BBC   C029   MOV  @>0050(R9),R0        ** Pointer to old name in R0
5BBE   0050
5BC0   06A0   BL   @>4658               ** Check if it's on the disk
5BC2   4658
5BC4   5C54   DATA >5C54
5BC6   06A0   BL   @>4ADA               ** Duplicate file descriptor
5BC8   4ADA                             ** record in data buffer area
5BCA   06A0   BL   @>4B70               ** Get the filetype byte
5BCC   4B70
5BCE   000C   DATA >000C
5BD0   0240   ANDI R0,>0800            ** Mask off all but
5BD2   0800                             ** protection bit
5BD4   1303   JEQ  >5BDC               ** Protected?, no then jump
5BD6   06A0   BL   @>4C72               ** Report error
5BD8   4C72
5BDA   2000   DATA >2000                ** Device is write protected
5BDC   06A0   BL   @>4B70               ** Get sector # of file
5BDE   4B70                             ** descriptor record
5BE0   FFFC   DATA -4
5BE2   C040   MOV  R0,R1               ** Put it in R1
5BE4   0429   BLWP @>005A(R9)           ** Restore R0 from stack
5BE6   005A
5BE8   8001   DATA >8001
5BEA   0429   BLWP @>005A(R9)           ** Save R1 on stack
5BEC   005A
5BEE   4000   DATA >4000
5BF0   06A0   BL   @>5E9C               ** Put new name in file
```

```
5BF2  5E9C
5BF4  06A0    BL   @>4658          ** compare buffer
5BF6  4658                         ** Check if its already on
5BF8  4E38    DATA >4E38           ** disk
5BFA  C104    MOV  R4,R4           ** Already on disk?
5BFC  1338    JEQ  >5C6E           ** Yes?, jump  report error
5BFE  06A0    BL   @>4B0A          ** Open spot in file directory
5C00  4B0A                         ** for it
5C02  0429    BLWP @>005A(R9)      ** Restore R4 from the stack
5C04  005A
5C06  0801    DATA >0801
5C08  0429    BLWP @>005A(R9)      ** Set VDPWA to spot in
5C0A  005A                         ** directory
5C0C  0103    DATA >0103
5C0E  DBC4    MOVB R4,@>FFFE(R15)  ** Write the sector # to
5C10  FFFE                         ** the directory
5C12  06C4    SWPB R4
5C14  DBC4    MOVB R4,@>FFFE(R15)
5C16  FFFE
5C18  06C4    SWPB R4
5C1A  0702    SETO R2              ** Flag for read sector
5C1C  04C5    CLR  R5              ** Put it in File Descriptor
                                   ** Area
5C1E  06A0    BL   @>4658          ** Read the sector
5C20  4658
5C22  4D4A    DATA >4D4A
5C24  C045    MOV  R5,R1           ** Point to File Descriptor
                                   ** Area
5C26  C029    MOV  @>0058(R9),R0   ** Get pointer to VIB
5C28  0058
5C2A  0220    AI   R0,>0101        ** Point to file name in
5C2C  0101                         ** compare buffer
5C2E  0601    DEC  R1              ** Point to Descriptor file
                                   ** name
5C30  06A0    BL   @>5EB2          ** Move name from buff to
5C32  5EB2                         ** descriptor
5C34  04C2    CLR  R2              ** Flag for write sector
5C36  06A0    BL   @>4658          ** Write file descriptor to
5C38  4658                         ** disk
5C3A  4D4E    DATA >4D4E
5C3C  06A0    BL   @>4658          ** Write VIB to disk
5C3E  4658
5C40  4D36    DATA >4D36
5C42  C069    MOV  @>0056(R9),R1   ** Pointer to name of Control
5C44  0056                         ** Block
5C46  0429    BLWP @>005A(R9)      ** Set VDPWA to it
5C48  005A
5C4A  0023    DATA >0023
5C4C  DBC4    MOVB R4,@>FFFE(R15)  ** Clear it out(FCB available)
5C4E  FFFE
5C50  0460    B    @>4676          ** Return
5C52  4676
5C54  04C6    CLR  R6
5C56  D1A9    MOVB @>004C(R9),R6   ** Put drive # in R6
5C58  004C
5C5A  06A0    BL   @>5E9C          ** Write # and name to compare
5C5C  5E9C                         ** buffer
5C5E  06A0    BL   @>4658          ** Check if file open
5C60  4658
5C62  4DC4    DATA >4DC4
```

```
5C64   06A0   BL   @>4658                  ** Find open File Control
5C66   4658                                ** Block and get descriptor
5C68   4E08   DATA >4E08                   ** sector from disk
5C6A   C104   MOV  R4,R4                   ** Found the file?
5C6C   1303   JEQ  >5C74                   ** Yes?, then jump
5C6E   06A0   BL   @>4C9E                  ** Report error
5C70   4C9E
5C72   E000   DATA >E000                   ** File not found
5C74   0460   B    @>4676                  ** Return
5C76   4676

** Routine >14 entry point (Access direct input file) **

5C78   C1CB   MOV  R11,R7                  ** Save return address
5C7A   06A0   BL   @>4724                  ** Init routine
5C7C   4724
5C7E   06A0   BL   @>5ED4                  ** Get parameters from CPU RAM
5C80   5ED4
5C82   06A0   BL   @>5EFA                  ** Find control block get
5C84   5EFA                                ** extra parameters
5C86   C082   MOV  R2,R2                   ** Ø AU's passed(Transfer file
                                           ** parameters) ?
5C88   131E   JEQ  >5CC6                   ** Yes?, then jump
5C8A   6003   S    R3,R0                   ** Subtract # of first AU
                                           ** from # AU's in the file
5C8C   1502   JGT  >5C92
5C8E   04C2   CLR  R2                      ** Ø AU's actually read
5C90   1017   JMP  >5CC0
5C92   8002   C    R2,R0                   ** Compare # AU's to be read
                                           ** to # AU's in the file
5C94   1A01   JL   >5C98                   ** Less?, then jump
5C96   C080   MOV  R0,R2                   ** Put # AU's in file in R2
5C98   0429   BLWP @>005A(R9)             ** Save R2 on the stack
5C9A   005A                                **  (# AU's to be read)
5C9C   2000   DATA >2000
5C9E   0429   BLWP @>005A(R9)             ** Save R2,R3,R11 on the stack
5CA0   005A                                ** (# AU's to be read,
5CA2   3100   DATA >3100                   ** # first AU)
5CA4   06A0   BL   @>4658                  ** Read an AU, put in buffer
5CA6   4658
5CA8   492E   DATA >492E
5CAA   0429   BLWP @>005A(R9)             ** Restore R11,R3,R2 from
5CAC   005A                                ** the stack
5CAE   3101   DATA >3101
5CB0   0583   INC  R3                     ** Add 1 to the count
5CB2   0227   AI   R7,256                 ** Add 256 to buffer pointer
5CB4   0100
5CB6   0602   DEC  R2                     ** Done yet?
5CB8   16F2   JNE  >5C9E                   ** No?, then jump
5CBA   0429   BLWP @>005A(R9)             ** Restore R2 from the stack
5CBC   005A
5CBE   2001   DATA >2001
5CC0   CA42   MOV  R2,@>004C(R9)           ** Clear out >834C
5CC2   004C
5CC4   100A   JMP  >5CDA                   ** Go close and return
5CC6   CD00   MOV  R0,*R4+                 ** Put # AU's in file in
                                           ** parameter block
5CC8   0648   DECT R8                      ** Point to file status flags
5CCA   05C2   INCT R2                      ** 2 bytes to pass
5CCC   06A0   BL   @>5F2E                  ** Write status flag  #
```

```
5CCE   5F2E                                  ** recs/AU to parameter block
5CDØ   Ø2Ø2   LI   R2,4                      ** 4 bytes to pass
5CD2   ØØØ4
5CD4   A2Ø2   A    R2,R8                     ** Add to the pointer
5CD6   Ø6AØ   BL   @>5F2E                    ** Write EOF offset, record
5CD8   5F2E                                  ** length to parameter block
5CDA   Ø6AØ   BL   @>4658                    ** Close the file
5CDC   4658
5CDE   487A   DATA >487A
5CEØ   Ø4E9   CLR  @>ØØ5Ø(R9)                ** Clear the error byte
5CE2   ØØ5Ø
5CE4   Ø46Ø   B    @>4676                    ** Return
5CE6   4676

** Routine >15 entry point (Access direct output file) **

5CE8   C1CB   MOV  R11,R7                    ** Save return address
5CEA   Ø6AØ   BL   @>4724                    ** Init routine
5CEC   4724
5CEE   Ø6AØ   BL   @>5ED4                    ** Get the parameters
5CFØ   5ED4
5CF2   1314   JEQ  >5D1C                     ** Create the file? Yes?, jump
5CF4   Ø6AØ   BL   @>5EFA                    ** Find FCB, get extra
5CF6   5EFA                                  ** parameters
5CF8   Ø429   BLWP @>ØØ5A(R9)                ** Store R2 on the stack
5CFA   ØØ5A
5CFC   2ØØØ   DATA >2ØØØ
5CFE   Ø429   BLWP @>ØØ5A(R9)                ** Store R2,R3,R4 on the stack
5DØØ   ØØ5A
5DØ2   31ØØ   DATA >31ØØ
5DØ4   Ø6AØ   BL   @>4658                    ** Write an AU to disk from
5DØ6   4658                                  ** buffer
5DØ8   494E   DATA >494E
5DØA   Ø429   BLWP @>ØØ5A(R9)                ** Restore R4,R3,R2 from stack
5DØC   ØØ5A
5DØE   3101   DATA >3101
5D1Ø   Ø583   INC  R3                        ** Add 1 to AU #
5D12   Ø227   AI   R7,256                    ** Add 256 to buffer pointer
5D14   Ø1ØØ
5D16   Ø6Ø2   DEC  R2                        ** Done yet?
5D18   16F2   JNE  >5CFE                     ** No?, then jump
5D1A   1ØCF   JMP  >5CBA                     ** Go return parameters, close
                                             ** file and return
5D1C   Ø6AØ   BL   @>4658                    ** Find the FCB
5D1E   4658
5D2Ø   4DC4   DATA >4DC4
5D22   Ø6AØ   BL   @>4658                    ** Check file already on disk
5D24   4658
5D26   4EØ8   DATA >4EØ8
5D28   Ø6AØ   BL   @>4658                    ** Create the file
5D2A   4658
5D2C   479E   DATA >479E
5D2E   Ø429   BLWP @>ØØ5A(R9)                ** Restore R4 from the stack
5D3Ø   ØØ5A
5D32   Ø8Ø1   DATA >Ø8Ø1
5D34   C229   MOV  @>ØØ56(R9),R8             ** Put FCB pointer in R8
5D36   ØØ56
5D38   Ø5C4   INCT R4
5D3A   CØF4   MOV  *R4+,R3
5D3C   Ø228   AI   R8,1Ø                     ** Point to 1 past name
```

```
5D3E   ØØØA
5D4Ø   Ø6AØ   BL   @>5F3E                    ** Write status byte, Recs/AU
5D42   5F3E                                 ** to the FCB
5D44   ØØØ2   DATA >ØØØ2
5D46   Ø6AØ   BL   @>5F3E                    ** Write # sectors allocated,
5D48   5F3E                                 ** EOF offset, rec length to
5D4A   ØØØ4   DATA >ØØØ4                    ** FCB
5D4C   Ø6Ø3   DEC  R3
5D4E   11Ø3   JLT  >5D56
5D5Ø   Ø6AØ   BL   @>4658                    ** Calculate sector #, update
5D52   4658                                 ** pointer blocks
5D54   4964   DATA >4964
5D56   Ø46Ø   B    @>5B8C                   ** Go close file, update
5D58   5B8C                                 ** status byte, and return

** BASIC CALL FILES entry point **

5D5A   C1CB   MOV  R11,R7                   ** Save return address
5D5C   Ø6AØ   BL   @>4724                   ** Init routine
5D5E   4724
5D6Ø   C229   MOV  @>ØØ2C(R9),R8            ** Token code pointer in R8
5D62   ØØ2C
5D64   Ø228   AI   R8,7                     ** Add 7 to skip 'FILES('
5D66   ØØØ7
5D68   Ø6AØ   BL   @>4B76                   ** Get 2 bytes after it
5D6A   4B76
5D6C   Ø28Ø   CI   RØ,>C8Ø1                 ** Check for unquoted string,
5D6E   C8Ø1                                 ** length of 1(tokenized code)
5D7Ø   161C   JNE  >5DAA                    ** No?, then jump
5D72   Ø5C8   INCT R8                       ** Point to ASCII #
5D74   Ø6AØ   BL   @>4B76                   ** Get it from VDP
5D76   4B76
5D78   Ø6CØ   SWPB RØ                       ** Put ASCII # in low byte,
                                            ** >B6 in high byte
5D7A   Ø22Ø   AI   RØ,>49DØ                 ** Mask out ASCII offset
5D7C   49DØ
5D7E   Ø28Ø   CI   RØ,>ØØØ9                 ** Check if Non-numeric
5D8Ø   ØØØ9
5D82   1B13   JH   >5DAA                    ** Yes?, then jump
5D84   Ø6CØ   SWPB RØ                       ** Put # of FCB's to reserve
                                            ** in MSByte
5D86   DA4Ø   MOVB RØ,@>ØØ4C(R9)            ** Put it in >834C for >16 rtn
5D88   ØØ4C
5D8A   Ø6AØ   BL   @>4658                   ** Do routine >16(reserve
5D8C   4658                                 ** buffers)
5D8E   5DB4   DATA >5DB4
5D9Ø   DA69   MOVB @>ØØ5Ø(R9),@>ØØ5Ø(R9)    ** Error?
5D92   ØØ5Ø
5D94   ØØ5Ø
5D96   1609   JNE  >5DAA                    ** Yes?, then jump
5D98   C229   MOV  @>ØØ2C(R9),R8            ** Put token pointer in R8
5D9A   ØØ2C
5D9C   Ø228   AI   R8,12                    ** Point to end of statement
5D9E   ØØØC
5DAØ   CA48   MOV  R8,@>ØØ2C(R9)            ** Put it in token pointer
5DA2   ØØ2C
5DA4   5A69   SZCB @>ØØ42(R9),@>ØØ42(R9)    ** Put >ØØ at >8342 (current
5DA6   ØØ42                                 ** token) (End of statement
5DA8   ØØ42                                 ** indicator)
5DAA   Ø46Ø   B    @>4676                   ** Return
```

```
5DAC   4676

** Routine >16 entry point (Buffer allocation)

5DAE   C1CB   MOV  R11,R7          ** Save return address
5DBØ   Ø6AØ   BL   @>4724          ** Init sequence
5DB2   4724
5DB4   Ø4CØ   CLR  RØ
5DB6   DØ29   MOVB @>ØØ4C(R9),RØ   ** Get # of files to reserve
5DB8   ØØ4C                        ** buffer space for
5DBA   136C   JEQ  >5E94           ** Ø files?, then jump
5DBC   C229   MOV  @>ØØ56(R9),R8   ** Point to Disk VDP area
5DBE   ØØ56
5DCØ   Ø228   AI   R8,3            ** Point to max # of files
5DC2   ØØØ3                        ** open
5DC4   Ø4C3   CLR  R3
5DC6   Ø429   BLWP @>ØØ5A(R9)      ** Set VDPWA to max #
5DC8   ØØ5A
5DCA   Ø1Ø2   DATA >Ø1Ø2
5DCC   DØEF   MOVB @>FBFE(R15),R3  ** Get it
5DCE   FBFE
5DDØ   Ø2Ø5   LI   R5,518          ** 518 bytes per control block
5DD2   Ø2Ø6
5DD4   9ØCØ   CB   RØ,R3           ** Compare # we want with #
                                   ** already reserved
5DD6   135B   JEQ  >5E8E           ** Same #?, then jump
5DD8   122B   JLE  >5E3Ø           ** # we want is less?, jump
5DDA   C18Ø   MOV  RØ,R6           ** Put # to reserve in R6
5DDC   Ø28Ø   CI   RØ,>1ØØØ        ** More than 16?
5DDE   1ØØØ
5DEØ   1B59   JH   >5E94           ** Yes?, then jump
5DE2   6ØØ3   S    R3,RØ          ** Figure # more FCB's we need
5DE4   Ø98Ø   SRL  RØ,8           ** Put it in 16 bits
5DE6   38Ø5   MPY  R5,RØ          ** Multiply by 518(# bytes
                                   ** per FCB)
5DE8   C1Ø1   MOV  R1,R4           ** Put result in R4
5DEA   Ø5Ø4   NEG  R4              ** Make it negative for adding
5DEC   CØA9   MOV  @>ØØ7Ø(R9),R2   ** Get highest available
5DEE   ØØ7Ø                        ** address in VDP RAM
5DFØ   CØØ2   MOV  R2,RØ           ** Put it in RØ
5DF2   6ØØ1   S    R1,RØ          ** Subtract # bytes we need
5DF4   Ø28Ø   CI   RØ,>Ø8ØØ        ** Too much VDP memory needed?
5DF6   Ø8ØØ
5DF8   114D   JLT  >5E94           ** Yes?, then jump

** Move DVA to lower VDP memory as needed by new FCB's

5DFA   CØ4Ø   MOV  RØ,R1           ** Pointer to new LFA of VDP
5DFC   Ø582   INC  R2              ** Point to first byte of
                                   ** old DVA
5DFE   Ø58Ø   INC  RØ              ** Point to first byte of
                                   ** new DVA
5EØØ   Ø429   BLWP @>ØØ5A(R9)      ** Set VDPWA to old byte
5EØ2   ØØ5A
5EØ4   ØØ42   DATA >ØØ42
5EØ6   DØEF   MOVB @>FBFE(R15),R3  ** Read it
5EØ8   FBFE
5EØA   Ø429   BLWP @>ØØ5A(R9)      ** Set VDPWA to new byte
5EØC   ØØ5A
5EØE   ØØØ3   DATA >ØØØ3
```

```
5E10  DBC3   MOVB R3,@>FFFE(R15)      ** Write it
5E12  FFFE
5E14  8202   C    R2,R8               ** Done yet?
5E16  16F2   JNE  >5DFC               ** No?, then jump
5E18  0429   BLWP @>005A(R9)          ** Set VDPWA to Max # buffers
5E1A  005A
5E1C  0003   DATA >0003
5E1E  DBC6   MOVB R6,@>FFFE(R15)      ** Write new number
5E20  FFFE
5E22  04C6   CLR  R6
5E24  6080   S    R0,R2               ** Get # bytes we need to
                                      ** clear out the new area
5E26  DBC6   MOVB R6,@>FFFE(R15)      ** Write a >00 byte to VDP
5E28  FFFE
5E2A  0602   DEC  R2                  ** Done yet?
5E2C  16FC   JNE  >5E26               ** No?, then jump
5E2E  101A   JMP  >5E64               ** Go finish
5E30  0429   BLWP @>005A(R9)          ** Set VDPWA to Max # of
5E32  005A                            ** buffers
5E34  0103   DATA >0103
5E36  DBC0   MOVB R0,@>FFFE(R15)      ** Write new # there
5E38  FFFE
5E3A  60C0   S    R0,R3               ** Figure # buffers, we need
                                      ** to remove
5E3C  0983   SRL  R3,8                ** Put it in 16 bits
5E3E  38C5   MPY  R5,R3               ** Multiply by 518
5E40  C044   MOV  R4,R1               ** Put result in R1
5E42  A048   A    R8,R1               ** Add DVA pointer(Point past
5E44  C0A9   MOV  @>0070(R9),R2       ** # buffs we need to remove)
5E46  0070
5E48  0429   BLWP @>005A(R9)          ** Set VDPWA to old location
5E4A  005A
5E4C  0102   DATA >0102
5E4E  D02F   MOVB @>FBFE(R15),R0      ** Read byte we need to move
5E50  FBFE
5E52  0429   BLWP @>005A(R9)          ** Set VDPWA to new location
5E54  005A
5E56  0023   DATA >0023
5E58  DBC0   MOVB R0,@>FFFE(R15)      ** Move it up in VDP RAM
5E5A  FFFE
5E5C  0601   DEC  R1                  ** Update pointers
5E5E  0608   DEC  R8
5E60  8088   C    R8,R2               ** Done yet?
5E62  16F2   JNE  >5E48               ** No?, then jump
5E64  CA41   MOV  R1,@>0070(R9)       ** Put new highest available
5E66  0070                            ** address in VDP RAM in >8370
5E68  C201   MOV  R1,R8               ** Move pointer to R8
5E6A  05C8   INCT R8                  ** Point to Top of VDP memory
                                      ** of DVA area
5E6C  06A0   BL   @>4B76              ** Set VDPWA to it
5E6E  4B76
5E70  D06F   MOVB @>FBFE(R15),R1      ** Read the byte
5E72  FBFE
5E74  904C   CB   R12,R1              ** Equal to CRU base?
5E76  130B   JEQ  >5E8E               ** Yes?, then jump
5E78  A004   A    R4,R0              ** Add offset
5E7A  0429   BLWP @>005A(R9)          ** Set VDPWA for a write
5E7C  005A
5E7E  0103   DATA >0103
5E80  DBC0   MOVB R0,@>FFFE(R15)      ** Write byte
```

```
5E82   FFFE
5E84   C200   MOV  R0,R8
5E86   06C0   SWPB R0
5E88   DBC0   MOVB R0,@>FFFE(R15)          ** Write byte
5E8A   FFFE
5E8C   10EE   JMP  >5E6A                   ** Keep moving
5E8E   04E9   CLR  @>0050(R9)              ** Clear out error byte
5E90   0050
5E92   1002   JMP  >5E98
5E94   0729   SETO @>0050(R9)              ** Set the error byte to >FF
5E96   0050
5E98   0460   B    @>4676                  ** Return
5E9A   4676

** Routine to write drive # and file name to compare buffer, Drive # in R6,
** Filename in VDP pointed to by R0

5E9C   04E9   CLR  @>0054(R9)              ** Clear PAB pointer
5E9E   0054
5EA0   C069   MOV  @>0058(R9),R1           ** Volume info block address
5EA2   0058
5EA4   0221   AI   R1,256                  ** Point to Drive # in
5EA6   0100                                ** Compare Buffer
5EA8   0429   BLWP @>005A(R9)              ** Set VDPWA to it
5EAA   005A
5EAC   0023   DATA >0023
5EAE   DBC6   MOVB R6,@>FFFE(R15)          ** Put the drive # there
5EB0   FFFE
5EB2   0202   LI   R2,10                   ** 10 bytes for filename
5EB4   000A
5EB6   0581   INC  R1                      ** Point to name in Compare
                                           ** Buffer
5EB8   0429   BLWP @>005A(R9)              ** Set VDPWA to filename in
5EBA   005A                                ** VDP
5EBC   0002   DATA >0002
5EBE   D0EF   MOVB @>FBFE(R15),R3          ** Read byte of name
5EC0   FBFE
5EC2   0580   INC  R0                      ** Update pointer
5EC4   0429   BLWP @>005A(R9)              ** Set VDPWA to compare buffer
5EC6   005A
5EC8   0023   DATA >0023
5ECA   DBC3   MOVB R3,@>FFFE(R15)          ** Write byte of name to
5ECC   FFFE                                ** compare buffer
5ECE   0602   DEC  R2                      ** Done yet?
5ED0   16F2   JNE  >5EB6                   ** No?, then jump
5ED2   045B   B    *R11                    ** Return

** Subroutine to get parameters for Direct file routines

5ED4   C28B   MOV  R11,R10                 ** Save return address
5ED6   04C6   CLR  R6
5ED8   D1A9   MOVB @>004C(R9),R6           ** Put drive # in R6
5EDA   004C
5EDC   C029   MOV  @>004E(R9),R0           ** Put name pointer in R0
5EDE   004E
5EE0   06A0   BL   @>5E9C                  ** Write it to compare buffer
5EE2   5E9C
5EE4   D129   MOVB @>0050(R9),R4           ** Get pointer to additional
5EE6   0050                                ** info block in CPU RAM
5EE8   0984   SRL  R4,8                    ** Put it in 16 bits
```

```
5EEA   A109   A    R9,R4                    ** Add CPU RAM offset
5EEC   D029   MOVB @>004D(R9),R0            ** Get access code
5EEE   004D
5EF0   0429   BLWP @>005A(R9)               ** Store R4 on the stack
5EF2   005A
5EF4   0800   DATA >0800
5EF6   0980   SRL  R0,8                    ** Put # AU's to be read
                                           ** in 16 bits
5EF8   045A   B    *R10                     ** Return

** Subroutine to find a control block for the direct file routines and get
** parameters from the CPU additional info block

5EFA   0429   BLWP @>005A(R9)               ** Save R0,R11 on the stack
5EFC   005A
5EFE   8010   DATA >8010
5F00   06A0   BL   @>4658                  ** Check if file already open
5F02   4658
5F04   4DC4   DATA >4DC4
5F06   06A0   BL   @>4658                  ** Find first available File
5F08   4658                                ** Control Block
5F0A   4E08   DATA >4E08
5F0C   C104   MOV  R4,R4                   ** Is one available?
5F0E   1303   JEQ  >5F16                   ** Yes?, then jump
5F10   06A0   BL   @>4C72                  ** Report error
5F12   4C72
5F14   E000   DATA >E000                   ** Out of buffer space
5F16   06A0   BL   @>4B70                  ** Get # sectors currently
5F18   4B70                                ** allocated to the file
5F1A   000E   DATA >000E
5F1C   0429   BLWP @>005A(R9)              ** Restore R11,R2 from stack
5F1E   005A
5F20   2011   DATA >2011
5F22   0429   BLWP @>005A(R9)              ** Restore R4 from the stack
5F24   005A                                ** (Pointer to CPU info block)
5F26   0801   DATA >0801
5F28   C1F4   MOV  *R4+,R7                 ** Get VDP buffer address
5F2A   C0D4   MOV  *R4,R3                  ** Get # of first AU
5F2C   045B   B    *R11                    ** Return

** Subroutine to return parameters from low level routine

5F2E   0429   BLWP @>005A(R9)              ** Set VDPWA to val in R8
5F30   005A
5F32   0102   DATA >0102
5F34   DD2F   MOVB @>FBFE(R15),*R4+        ** Write a byte from FCB to
5F36   FBFE                                ** the parameter block
5F38   0602   DEC  R2                      ** Done yet?
5F3A   16FC   JNE  >5F34                   ** No?, then jump
5F3C   045B   B    *R11                    ** Return

** Subroutine to put parameters in the FCB for low level routine

5F3E   C0BB   MOV  *R11+,R2                ** Get # bytes to write
5F40   A202   A    R2,R8                   ** Add pointer offset
5F42   0429   BLWP @>005A(R9)              ** Set VDPWA to it
5F44   005A
5F46   0103   DATA >0103
5F48   DBF4   MOVB *R4+,@>FFFE(R15)        ** Write the byte to the FCB
5F4A   FFFE
```

```
5F4C    0602    DEC  R2                 ** Done yet?
5F4E    16FC    JNE  >5F48              ** No?, then jump
5F50    045B    B    *R11               ** Return

** Memory locations 5F52 - 5FFE Do not hold any code only >FFFF.  Not
** used by the disk controller.

These locations will vary depending on the status of the controller.

5FF0    7FFF    DATA >7FFF          ** Read Status Register
5FF2    E9FF    DATA >E9FF          ** Read Track Register
5FF4    FEFF    DATA >FEFF          ** Read Sector Register
5FF6    FFFF    DATA >FFFF          ** Read Data Register
5FF8    FFFF    DATA >FFFF          ** Write Command Register
5FFA    FFFF    DATA >FFFF          ** Write Track Register
5FFC    FFFF    DATA >FFFF          ** Write Sector Register
5FFE    FF00    DATA >FF00          ** Write Data Register
```