

DISK EDITORASM™

COLOR COMPUTER
DISK EDITOR ASSEMBLER WITH ZBUG

CUSTOM MANUFACTURED
IN USA BY RADIO SHACK
A DIVISION OF TANDY CORPORATION

TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK COMPUTER EQUIPMENT AND SOFTWARE
PURCHASED FROM A RADIO SHACK COMPANY-OWNED COMPUTER CENTER, RETAIL STORE OR FROM A
RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

- A. CUSTOMER assumes full responsibility that this Radio Shack computer hardware purchased (the "Equipment"), and any copies of Radio Shack software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.
- B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE

- A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. THIS WARRANTY IS ONLY APPLICABLE TO PURCHASES OF RADIO SHACK EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.
- B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.
- C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.
- D. Except as provided herein, **RADIO SHACK MAKES NO WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.**
- E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

- A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE". IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE".

NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.

- B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.
- C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.
- D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK Software on **one** computer, subject to the following provisions:

- A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.
- B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.
- C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.
- D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.
- E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.
- F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.
- G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

- A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party for lease to CUSTOMER.
- B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

DISK EDITORASM™

COLOR COMPUTER
DISK EDITOR ASSEMBLER WITH ZBUG

CUSTOM MANUFACTURED
IN USA BY RADIO SHACK
A DIVISION OF TANDY CORPORATION

Disk EDTASM Software: Copyright 1983, Microsoft. All Rights Reserved. Licensed to Tandy Corporation.

Disk EDTASM Manual: Copyright 1983, Tandy Corporation.
All Rights Reserved.

Reproduction or use without express written permission from Tandy Corporation, of any portion of this manual is prohibited. While reasonable efforts have been taken in the preparation of this manual to assure its accuracy, Tandy Corporation assumes no liability resulting from any errors or omissions in this manual, or from the use of the information contained herein.

To Our Customers . . .

The heart of the Color Computer is a 6809E "processor." It controls all other parts of the Color Computer.

The processor understands only a code of 0s and 1s, not at all intelligible to the human mind. This code is called "6809 machine code."

When you run a BASIC program, a system called the "BASIC Interpreter" translates each statement, one at a time, into 6809 machine code. This is an easy way to program, but inefficient.

The Disk EDTASM lets you program using an intelligible representation of 6809 machine code, called "assembly language," that talks directly to the processor. You then assemble the entire program into 6809 machine code before running it.

Programming with the Disk EDTASM gives you these benefits:

- You have direct and complete control of the Color Computer. You can use its features — such as high resolution graphics — in ways that are impossible with BASIC.
- Your program runs faster. This is because it is already translated into 6809 machine code when you run it.

To Use the Disk EDTASM You Need . . .

A Color Computer Disk System that has at least 16K of RAM, preferably 32K. (A 16K System will leave you little room for programs.)

The Disk EDTASM Contains:

- EDTASM/BIN, a system for creating 6809 programs. EDTASM contains:

An editor, for writing and editing 6809 assembly-language programs.

An assembler, for assembling the programs into 6809 machine code.

ZBUG, for examining and debugging 6809 machine-code programs.

You must have 32K to run EDTASM. If you have 16K, run EDTASMOV (described next).

- EDTASMOV/BIN, a memory-efficient version of EDTASM consisting of overlays. EDTASMOV contains the editor and assembler, but not ZBUG.
- ZBUG/BIN, a stand-alone version of ZBUG, primarily for use with EDTASMOV.
- DOS/BIN, a disk operating system. DOS contains disk access routines that you can call from an assembly language program. (You cannot call BASIC's disk access routines with any program other than BASIC.)

EDTASM/BIN, EDTASMOV/BIN, and ZBUG/BIN all use DOS routines and must be run with DOS.

The Disk EDTASM also contains:

- DOS/BAS. A BASIC program that loads DOS/BIN.
- ZBUG/BAS. A BASIC program that loads ZBUG/BIN.

How to Use this Manual

This manual is organized for both beginning and advanced assembly language programmers. Sections I-IV are tutorials; Section V is reference.

Beginning Programmers:

Read Section I first. It shows how the entire system works and explains enough about assembly language to get you started.

Then, read Sections II, III, and IV in any order you want. Use Section V, "Reference," as a summary.

This manual does not try to teach you 6809 mnemonics. To learn this, read:

Radio Shack Catalog #62-2077
by William Barden Jr.

6809 Assembly Language Programming
by Lance A. Leventhal

Nor does it teach you disk programming concepts. To learn these, read:

Color Computer Disk System Manual
(Radio Shack Catalog #26-3022)

Advanced Programmers:

First, read Chapters 1 and 2 to get started and see how the entire system works. Then, read Section V, "Reference."

You can use the DOS program listing to obtain information on routines and addresses not explained in this manual. Please note the following:

Radio Shack supports only these DOS routines: OPEN, CLOSE, READ, and WRITE. Additional DOS routines are listed in Reference H. However, Radio Shack does not promise to support them.

Even more DOS routines and addresses can be found in the program listing. However, Radio Shack does not promise to support them nor even provide them in the future.

For technical information on the Color Computer Disk System and 6809, refer to 6809 Assembly Language Programming and Color Computer Disk System Manual, listed above.

This manual uses these terms and notations:

KEY	To denote a key you must press.
<i>Italics</i>	To denote a value you must supply.
<i>filespec</i>	To denote a DOS file specification. A DOS filespec is in one of these formats: <i>filename/ext:drive</i> <i>filename.ext:drive</i>
	<i>filename</i> has one to eight characters. <i>extension</i> has one to three characters.
	<i>drive</i> is the drive number. If the drive number is omitted, DOS uses the first available drive.
\$	To denote a hexadecimal (Base 16) number. For example, \$0F represents hexadecimal 0F, which is equal to 15 in decimal (Base 10) notation.

Contents

Section I/ Getting Started

Chapter 1/ Preparing Diskettes.....	3
Chapter 2/ Running a Sample Program	5
Chapter 3/ Overview.....	9

Section II/ Commands

Chapter 4/ Using the DOS Menu (DOS Commands).....	15
Chapter 5/ Examining Memory (ZBUG Commands — Part I)	17
Chapter 6/ Editing the Source Program (Editor Commands).....	21
Chapter 7/ Assembling the Program (Assembler Commands).....	25
Chapter 8/ Debugging the Program (ZBUG Commands — Part II).....	31
Chapter 9/ Using the ZBUG Calculator (ZBUG Commands — Part III)	35

Section III/ Assembly Language

Chapter 10/ Writing the Program	41
Chapter 11/ Using Pseudo Ops	47
Chapter 12/ Using Macros.....	51

Section IV/ ROM and DOS Routines

Chapter 13/ Using the Keyboard and Video Display (ROM Routines).....	57
Chapter 14/ Opening and Closing a Disk File (DOS Routines — Part I).....	61
Chapter 15/ Reading and Writing a Disk File (DOS Routines — Part II)	65

Section V/ Reference

A/ Editor Commands	71
B/ Assembler Commands and Switches	75
C/ ZBUG Commands.....	77
D/ EDTASM Error Messages	81
E/ Assembler Pseudo Ops	85
F/ ROM Routines	89
G/ DOS Data Control Block (DCB)	91
H/ DOS Routines	95
I/ DOS Error Codes	101
J/ Memory Map	103
K/ ASCII Codes	105
L/ 6809 Mnemonics.....	109
M/ Sample Programs	125

Section VI/ Program Listing

Index



SECTION I

GETTING STARTED

SECTION I**GETTING STARTED**

This section gets you started using the Disk EDTASM and explains some concepts you need to know.

Chapter 1/ Preparing Diskettes

Before using the Disk EDTASM, you need to format blank diskettes and back up the master Disk EDTASM diskette.

Formatting Blank Diskettes

1. Power up your disk system and insert a blank diskette in Drive 0. (See the *Color Computer Disk System Manual* for help.)
2. At the OK prompt, type:

DSKINIØ **(ENTER)**

BASIC formats the diskette. When finished, it again shows the OK prompt.

Making Backups of Disk EDTASM

Single-Drive Systems

1. Insert the master Disk EDTASM diskette, your "source" diskette, in Drive 0.

2. At the BASIC OK prompt, type:

BACKUP Ø TO Ø **(ENTER)**

3. BASIC then prompts you to insert the "destination" diskette. Remove the source diskette and insert a formatted diskette. Press **(ENTER)**
4. BASIC prompts you to alternatively insert the source, then destination diskettes. When the backup is finished, the OK prompt appears.

The destination diskette is now a duplicate of the master Disk EDTASM diskette.

Multi-Drive Systems

1. Insert the master Disk EDTASM diskette in Drive 0.
2. Insert a formatted diskette in Drive 1.
3. At BASIC's OK prompt, type:

BACKUP Ø TO 1 **(ENTER)**

BASIC makes the backup. When the backup is finished, the OK prompt appears.

The diskette in Drive 1 is now a duplicate of the master Disk EDTASM diskette.

Chapter 2/ Running a Sample Program

This "sample session" gets you started writing programs and shows how to use the Disk EDTASM. The next chapters explain why the program works the way it does.

1. Load and Run DOS

Insert the Disk EDTASM diskette in Drive 0. At the OK prompt, type:

RUN "DOS" **ENTER**

DOS then loads and puts you in its "command mode." The screen shows the DOS command menu:

1. Exit to BASIC
2. Exec a Program
3. Start Clock Display
4. Disk Allocation Map
5. Copy Files
6. Directory

DOS consists of many disk input and output routines which EDTASM uses. You must load DOS before loading EDTASM.

2. Load and Run EDTASM

At the DOS Menu, press **2** to select "Execute a Program." The screen asks for the name of a program file.

If your system has 32K or more, use EDTASM. If it has only a 16K system, use EDTASMOV.

Loading EDTASM:

Type EDTASM. The screen shows:

EXECUTE A PROGRAM
PROGRAM NAME: [EDTASM]/BIN

If you make a typing error, use the **←** to reposition the cursor at the beginning of the line, then correct the mistake. Replace any trailing characters with blank spaces.

Press **ENTER**. EDTASM loads and shows its startup message.

Loading EDTASMOV:

Type EDTASMOV. The screen shows:

EXECUTE A PROGRAM
PROGRAM NAME: [EDTASMOV]/BIN

If you make a mistake, use the **←** to reposition the cursor, then correct the mistake.

EDTASMOV loads and shows its startup message.

Always keep EDTASMOV in Drive 0. It contains overlays which it loads into memory as required. It always looks for these overlays in Drive 0.

3. Type the Source Program

Notice the asterisk (*) prompt. This means you are in the editor program of EDTASM or EDTASMOV. The editor lets you type and edit an assembly language "source" program.

At the * prompt, type:

I **ENTER**

This puts you in the editor's insert mode. The editor responds with line number 00100. Type:

START **→** LDA **→** #\\$F9 **ENTER**

The right arrow tabs to the next column. **ENTER** inserts the line in the editor's "edit buffer." The \$ means that F9 is a hexadecimal (Base 16) number.

Your screen should show:

```
00100 START LDA    #$F9
00110
```

meaning that you inserted line 100 and can now insert line 110.

If you make a mistake, press **BREAK**. Then, at the * prompt, delete Line 100 by typing:

D100 [ENTER]

Now, insert Line 100 correctly in the same manner described above.

Insert the entire assembly language program listed below.

Note that line 150 uses brackets. Do not substitute parentheses for the brackets. To produce the left bracket, press **SHIFT** and **[** at the same time. To produce the right bracket, press **SHIFT** and **]** at the same time.

```
00100 START LDA    #$F9
00110      LDX    #$400
00120 SCREEN STA    ,X+
00130      CMPX   #$600
00140      BNE    SCREEN
00150 WAIT   JSR    [$A000]
00160      BEQ    WAIT
00170      CLR    $71
00180      JMP    [$FFFE]
00190 DONE   EQU    *
00200      END
```

If you make a mistake, press **BREAK**. Then, at the * prompt, delete the program by typing:

D# : *

Now, insert the program correctly.

When finished, press **BREAK**. The program you have inserted is an assembly language "source" program, which we'll explain in the next chapter.

4. Assemble the Source Program in Memory

At the * prompt, type:

A/I/M/WE [ENTER]

which loads the assembler program. The assembler then assembles your source program into 6809 machine code

into the memory area just above the EDTASM or EDTASMOV program. To let you know what it has done, it prints this listing:

```
4B28 86 F9      00100 START
        LDA    #$F9
4B2A 8E 0400    00110
        LDX    #$400
4B2D A7 B0      00120 SCREEN
        STA    ,X+
4B2F 8C 0600    00130
        CMPX   #$600
4B32 26 F9      00140
        BNE    SCREEN
4B34 AD 9F A000 00150 WAIT
        JSR    [$A000]
4B38 27 FA      00160
        BEQ    WAIT
4B3A 0F 71      00170
        CLR    $71
4B3C 6E 9F FFFE 00180
        JMP    [$FFFE]
4B40          *      00190 DONE
EQU
        0000    00200
END
00000 TOTAL ERRORS
DONE     4B40
SCREEN  4B2D
START   4B28
WAIT    4B34
```

(If using EDTASMOV, the numbers will be different.)

If the assembler does not print this entire listing, but stops and shows an error message instead, you have an error in the source program. Repeat Steps 3 and 4.

The assembler listing is explained in *Figure 1* of Chapter 7.

5. Prepare the Program for DOS

Before saving the program, you need to prepare it so that you can load and run it from DOS.

First, you must give it an "origination address" for DOS to use in loading the program back into memory. (We recommend you use Address \$1200, the first address

available after the DOS system.) To do so, type:

I50 **ENTER**

and insert this line:

50 ORG \$1200

Next, you need to add two lines to your program to tell DOS how long the program is. Insert these lines:

60 BEGIN JMP START
70 FDB DONE-BEGIN

When finished, press **BREAK**. To see the entire program, type:

P# : * **ENTER**

It should look like this:

00050	ORG	\$1200
00060	BEGIN	JMP START
00070		FDB DONE-BEGIN
00100	START	LDA ##F9
00110		LDX ##400
00120	SCREEN	STA ,X+
00130		CMPX ##600
00140		BNE SCREEN
00150	WAIT	JSR [\$A000]
00160		BEQ WAIT
00170		CLR \$71
00180		JMP [\$FFFE]
00190	DONE	EQU *
00200		END

If you make a mistake, delete the line with the error and insert it again.

6. Save the Source Program on Disk

To save the source program, type (at the * prompt):

WD SAMPLE **ENTER**

This saves the source program on disk as SAMPLE/ASM.

7. Save the Assembled Program on Disk

At the * prompt, type:

AD SAMPLE /SR **ENTER**

Be sure you have a blank space between SAMPLE and /SR. This causes the assembler to again assemble the source program into 6809 code. This time, the Assembler saves the assembled program on disk as SAMPLE/BIN.

(You must use the /SR "switch" to assemble any program that you want to load and run from DOS.)

8. Run the Assembled Program from DOS

To run the assembled program, you need to be in the DOS command mode. At the * prompt, type:

K **ENTER**

which causes the Editor to return you to the DOS command menu. Press **2** to execute a program. Then type SAMPLE, the name of the assembled program. (The assembler assumes you mean SAMPLE/BIN.) The screen shows:

EXECUTE A PROGRAM
PROGRAM NAME: [SAMPLE]/BIN

Press **ENTER**. The SAMPLE program executes, filling your entire screen with a graphics checkerboard.

Press any key to exit the program. The program returns to BASIC startup message.

9. Debug the Program (if necessary)

ZBUG lets you to look at memory. How you load ZBUG depends on whether you are using EDTASM or EDTASMOV.

EDTASM Users:

You can load ZBUG from EDTASM. Load DOS and EDTASM again (Steps 1 and 2). Then, at the * prompt, type:

Z **ENTER**

EDTASM loads its ZBUG program and displays ZBUG's # prompt. You can now examine any memory address. Type:

4000/

and ZBUG shows you what is in memory at this address. Press **↓** a few times to look at more memory addresses. When finished, press **BREAK**.

In *Chapter 8*, we'll show you how to use ZBUG to examine and test your program. To return to EDTASM's editor, type:

E **ENTER**

EDTASMOV Users:

You must use the Stand-Alone ZBUG. Load DOS again (Step 1). At the DOS Menu, press **2**, "Execute a Program," and run the ZBUG program. After typing ZBUG, the screen shows:

```
EXECUTE A PROGRAM  
PROGRAM NAME: [ZBUG] /BIN
```

DOS loads the stand-alone ZBUG and displays ZBUG's # prompt. You can now examine any memory address. Type:

3800/

and ZBUG shows you what is in memory at this address. Press **↓** a few times to look at more memory addresses. When finished, press **BREAK**.

In *Chapter 8*, we'll show you how to use ZBUG to examine and test your program. To return to DOS, type:

K **ENTER**

Chapter 3/ Overview

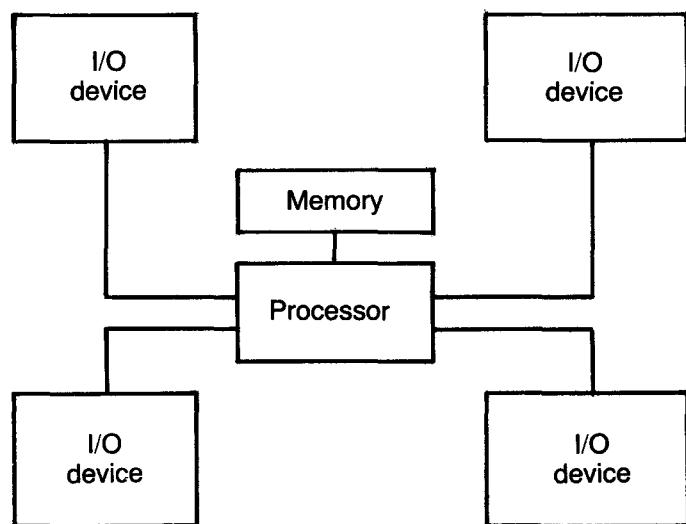
This chapter is for beginning assembly language programmers. It explains some concepts you need. If you're not a beginner, use this chapter as a refresher or skip it.

The Color Computer Hardware

The Color Computer consists of:

- The 6809E Processor
- Memory
- Input/Output Devices

This shows how they relate to each other:



The Processor

The processor processes all data going to each memory address and device. It contains:

- Registers — for temporarily storing 1- or 2-byte values.

- Buses — for transferring data to or from the processor.

All instructions to the processor must be in 6809 machine code: a code of 0s and 1s containing "opcodes" and data. "OpCodes" are instructions that tell the processor to manipulate data in some way.

For example, the machine-code instruction "10000110 11111001" contains:

- The opcode "10000110" (decimal 134 or hexadecimal 86)
- The data "11111001" (decimal 249 or hexadecimal F9)

This instruction tells the processor to load Register A with 11111001.

Memory

Memory is a storage area for programs and data. There are two kinds of memory:

- Random access memory (RAM) — for temporary storage of programs or data. When you load a program from disk, you load it into RAM. Many opcodes store data in RAM temporarily.
- Read only memory (ROM) — for permanent storage of programs. BASIC, as well as any program pack you use, is stored in ROM. The Color Computer contains several "ROM routines" that you can use to access the keyboard, screen, or tape recorder.

When writing an assembly language program, you must constantly be aware of what's happening in memory. For this reason, this manual provides a memory map. (See Reference J.)

Devices

All other parts of the hardware are called devices. A device expects the processor to input or output data to it in a certain format. To input or output data in this format, you can use these pre-programmed subroutines:

- Routines stored in ROM (ROM routines) — for inputting or outputting to the keyboard, screen, printer, or tape recorder.
- Routines stored in DOS (DOS routines) — for inputting or outputting to disk.

The Disk EDTASM Assembler

The Disk EDTASM looks for three fields in your instructions: label, command, and operand. For example, in this instruction:

```
BEGIN      JMP      START
```

BEGIN is the label. JMP is the command. START is the operand.

In the label field, it looks for:

- Symbols (symbolic names)

In the command field, it looks for:

- Mnemonics
- Pseudo Ops

In the operand field, it looks for:

- Symbols
- Operators
- Addressing-Mode Characters
- Data

Symbols

A symbol is similar to a variable. It can represent a value or a location. BEGIN (in the sample session) is a symbol that represents the location of the instruction JMP START. START is also a symbol that represents the location of LDA #\$F9.

Mnemonics

A mnemonic is a symbolic representation of an opcode. It is a command to the processor. "LDA" is a mnemonic. Depending on which "addressing-mode character" you use, LDA represents one of these opcodes:

```
10000110
10010110
10110110
10100110
```

(Addressing-mode characters are discussed below.)

Mnemonics are specific to a particular processor. For example, Radio Shack's Model 4 uses the Z80 processor, which understands Z80 mnemonics, rather than the 6809 mnemonics.

Pseudo Ops

A pseudo op is a command to the assembler. END (in the sample session) is a pseudo op. It tells the assembler to quit assembling the program.

Data

Data is numbers or characters. Many of the mnemonics and pseudo ops call for data. Unless you use an operator (described next), the assembler interprets your data as a decimal (Base 10) number.

Operators

An operator tells the assembler to perform a certain operation on the data. In the value \$1200, the \$ sign is an operator. It tells the assembler that 1200 is a hexadecimal (Base 16) number, rather than a decimal (Base 10) number.

The more commonly used operators are arithmetic and relational. Addition (+) and equation (=) are examples of these operators.

Addressing-Mode Characters

An addressing mode character tells the assembler how it should interpret the mnemonic. The assembler then assembles the mnemonic into the appropriate opcode.

The sample session uses the # character with the LDA mnemonic to denote the "immediate" addressing mode. This causes the assembler to assemble LDA into the opcode 10000110.

The immediate mode means that the number following the mnemonic (in this case, \$F9) is data rather than an address where the data is stored.

Pseudo ops, symbols, operators, and addressing-mode characters vary from one assembler to another. Section III explains them in detail.

Sample Program

This is how each line in the sample program works:

```
50          ORG    $1200
```

ORG is a pseudo op for "originate." It tells the assembler to begin loading the program at Location \$1200 (Hexadecimal 1200). This means that when you load and run the program from DOS, the program starts at Memory Address \$1200.

```
60          BEGIN   JMP    START
```

BEGIN is a symbol. It equals the location where the JMP START instruction is stored.

JMP is a mnemonic for "jump to an address." It causes the processor to jump to the location of the program labeled by the symbol START, which is the LDA #\$F9 instruction. You must use JMP or LBRA as the first instruction in a DOS program.

```
70          FDB    DONE-BEGIN
```

FDB is a pseudo op for "store a 2-byte value in memory." It stores the value of DONE-BEGIN (the length of the program) in the next two bytes of memory. You must store this value at the beginning of the program to tell DOS how much of the program to load.

```
00100      START   LDA    #$F9
```

START is a symbol. It equals the location where LDA #\$F9 is stored.

LDA is a mnemonic for "load Register A." It loads Register A with \$F9, which is the hexadecimal ASCII code for a graphics character. The ASCII characters are listed in Reference K.

```
00110      LDX    #$400
```

LDX is a mnemonic for "load Register X." It loads Register X with \$400, the first address of video memory. Reference J shows where video memory begins and ends.

```
00120      SCREEN  STA    ,X+
```

SCREEN is a symbol. It equals the location where STA ,X+ is stored.

STA is a mnemonic for "store Register A." It stores the contents of Register A (\$F9) in the address contained in Register X (\$400). This puts the \$F9 graphics character at the upper left corner of your screen.

The "," and "+" are addressing-mode characters. The + causes the processor to store \$F9 in the address con-

tained in Register X. The + causes the processor to then increment the contents of Register X to \$401.

```
00130      CMPX   #$600
```

CMPX is a mnemonic for "compare Register X." It compares the contents of Register X with \$600. If Register X contains \$600, the processor sets the "Z" bit in the Register CC to 1.

```
00140      BNE    SCREEN
```

BNE is a mnemonic for "branch if not equal." It tells the processor return to SCREEN (the STA,X+ instruction) until the Z bit is set.

The BNE SCREEN instruction creates a loop. The program branches back to SCREEN, filling all video memory addresses with \$F9, until it fills Address \$600. At that time, Register X contains \$600, Bit Z is set, and program control continues to the next instruction.

```
00150      WAIT   JSR    [$A000]
```

JSR is a mnemonic for "jump to a subroutine." \$A000 is a memory address that stores the address of a ROM routine called POLCAT. (See Reference F.)

POLCAT scans the keyboard to see if you press a key. When you do, it clears the Z bit.

The "["]" are addressing-mode characters. They tell the processor to use an address contained in an address, rather than the address itself. Always use the "[]" signs when calling ROM routines.

```
00160      BEQ    WAIT
```

BEQ is a mnemonic for "branch if equal." It branches to the JSR [\$A000] instruction until the Z bit is clear. This causes the program to loop until you press a key, at which time POLCAT clears the Z bit.

```
00170      CLR    $71
00180      JMP    [$FFFE]
```

CLR is a mnemonic for "clear," and JMP is a mnemonic for "jump to memory address." These two instructions end the program and return to BASIC's startup message.

(CLR inserts a zero in Address \$71; this signals that the system is at its original "uninitialized" condition. JMP goes to the address contained in Address \$FFFE; this is where BASIC initialization begins.)

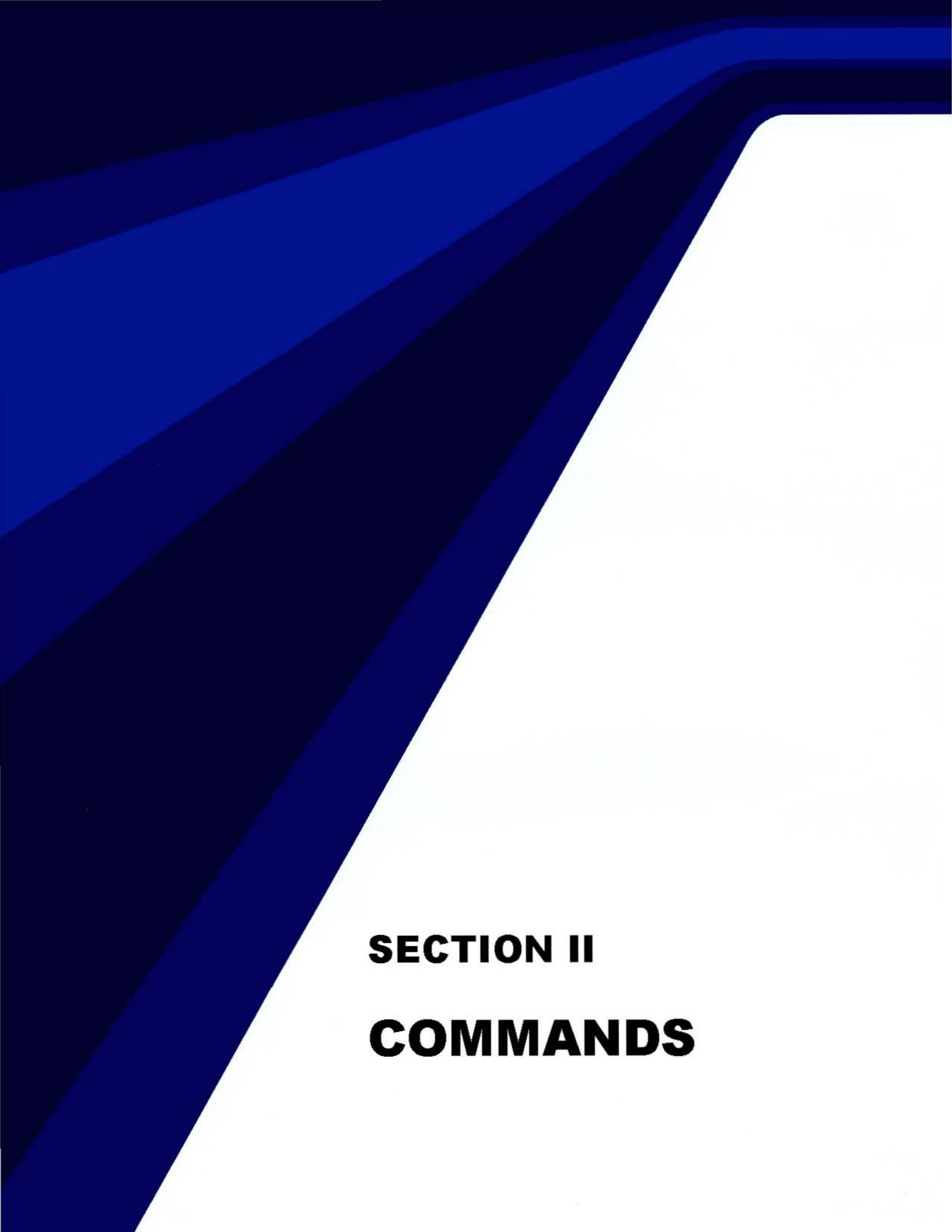
```
00180      DONE   EQU    *
```

EQU is a pseudo op. It equates the symbol DONE with an asterisk (*), which represents the last line in the program.

00190

END

END is a pseudo op. It tells the assembler to quit
assembling the program.



SECTION II

COMMANDS

Section II**COMMANDS**

This section shows how to use the many Disk EDTASM commands. Knowing these commands will help you edit and test your program.

Chapter 4/ Using the DOS Menu (DOS Commands)

When you first enter DOS, a menu of six DOS commands appear on the screen. Chapter 2 shows how to use the first two DOS commands. This chapter shows how to use the remaining commands:

- Start Clock Display
- Disk Allocation Map
- Copy Files
- Directory

To use the examples in this chapter, you need to have the SAMPLE disk files, which you created in *Chapter 2*, on the diskette in Drive 0.

Directory

The DOS "directory" command lets you select the directory entries you want to see, using three fields: filename, extension, and drive number.

To select the directory entries, press **6** at the DOS Menu. Then, press the **↑** to move the cursor left or **↓** to move right.

Type this line to select all directory entries that have the filename SAMPLE.

```
[SAMPLE**] [***] :[0] <FILE SPEC
```

Use the **SPACEBAR** to erase characters. Press **ENTER** when finished. Then, press any key to return to the DOS menu, and press **6** to return to the directory.

Type this line to select all directory entries with the extension /BIN:

```
[*****] [BIN] :[0] <FILE SPEC
```

Press **ENTER** when finished. Return to the main menu.

To see all directory entries on the disk in Drive 0, simply press **ENTER** without specifying a filename or extension:

```
[*****] [***] :[0] <FILE SPEC
```

Disk Allocation Map

The "disk allocation map" command tells you how much free space you have on your diskettes. To see the map, press **4** at the DOS menu.

DOS shows a map of the diskettes that are in each drive. The map shows how each of the diskette's 68 granules is allocated:

- A period (.) means the granule is free.
- An X means all the sectors in the granule are currently allocated to a file.
- A number indicates how many sectors in the granule are currently allocated to a file.

Press any key to return to the DOS menu.

Copy Files

The "Copy Files" command makes a duplicate of a disk file. To use it, press **5** at the DOS menu. DOS then prompts you for the names of the files.

Single-Drive Copy

The first example copies SAMPLE/ASM to another file named COPY/ASM. Use the **↑** and **↓** to position the cursor. Answer the prompts as shown:

Source File Name	[SAMPLE]
Extension	[ASM]
Drive	[0]
Destination File Name	[COPY]
Extension	[ASM]
Drive	[0]
If Drives are the same are you	
using different diskettes?	
(Y or N)? [N]	

When finished, press **ENTER**. DOS copies SAMPLE/ASM to a new file named COPY/ASM and then returns to the DOS menu. Check the directory (by pressing **6**) and you'll see that both SAMPLE/ASM and COPY/ASM are on your diskette.

The next example copies SAMPLE/ASM to another diskette. Answer the prompts as shown:

```
Source File Name      [SAMPLE ]  
Extension            [ASM]  
Drive                [0]
```

```
Destination File Name [COPY     ]  
Extension            [ASM]  
Drive                [0]
```

If Drives are the same are you
using different diskettes?
(Y or N)? [Y]

Press **ENTER**. DOS then prompts you to insert the source diskette. Press **ENTER** again.

DOS then prompts you for a destination diskette. Insert the destination diskette and press **ENTER**. After copying the file, DOS prompts you for a system diskette. If you press **ENTER** without inserting a system diskette, you will get a SYSTEM FAILURE error.

When finished, it returns to the DOS menu.

Multi-Drive Copy

This example copies SAMPLE/ASM in Drive 0 to SAMPLE/ASM in Drive 1. Answer the prompts as shown:

```
Source File Name      [SAMPLE ]  
Extension            [ASM]  
Drive                [0]
```

```
Destination File Name [SAMPLE ]  
Extension            [ASM]  
Drive                [1]
```

If Drives are the same are you
using different diskettes?
(Y or N)? [N]

Start Clock Display

The Color Computer has a clock that runs on 60-cycle interrupts. Since the clock skips a second or more when the computer accesses tape or disk, we recommend that you not use it while executing a program.

To use the clock, press **3**, "Start Clock Display." Six digits appear at the upper right corner of your screen. The first two are hours, the next are minutes, and the next are seconds. This clock counts the time until you exit DOS.

Chapter 5/ Examining Memory ZBUG Commands — Part I

To use the Disk EDTASM, you must understand the Color Computer's memory. You need to know about memory to write the program, assemble it, debug it, and execute it.

In this chapter, we'll explore memory and see some of the many ways you can get the information you want. To do this, we'll use ZBUG.

If you are not "in" ZBUG, with the ZBUG # prompt displayed, you need to get in it now.

EDTASM: Load and run DOS, then execute the EDTASM program. At the editor's * prompt, type

Z **ENTER**

EDTASMOV: Load and run DOS, then execute the ZBUG program.

You should now have a # prompt on your screen. This means you are in ZBUG and you may enter a ZBUG command. All ZBUG commands must be entered at this command level. You can return to the command level by pressing **BREAK** or **ENTER**.

Examining a Memory Location

The 6809 can address 65,536 one-byte memory addresses, numbered 0-65535 (\$0000-\$FFFF). We'll examine Address \$A000. At the # prompt, type:

B **ENTER**

to get into the "byte mode." Then type:

A000/

and ZBUG shows the contents of Address \$A000. To see the contents of the next bytes, press **↓**. Use **↑** to scroll to the preceding address.

Continue pressing **↓** or **↑**. Notice that as you use the **↓** the screen continues to scroll down. The smaller addresses are on the lower part of the screen.

All the numbers you see are hexadecimal (Base 16). You see not only the 10 numeric digits, but also the 6 alpha characters needed for Base 16 (A-F). Unless you specify another base (which we do in Chapter 9), ZBUG assumes you want to see Base 16 numbers.

Notice that a zero precedes all the hexadecimal numbers that begin with an alphabetic character. This is done to avoid any confusion between hexadecimal numbers and registers.

Examination Modes

To help you interpret the contents of memory, ZBUG offers four ways of examining it:

- Byte Mode
- Word Mode
- ASCII Mode
- Mnemonic Mode

Byte Mode

Until now, you've been using the byte mode. Typing B **ENTER**, at the # prompt got you into this mode.

The byte mode displays every byte of memory as a number, whether it is part of a machine-language program or data.

In this examination mode, the **↓** increments the address by one. The **↑** decrements the address by one.

Word Mode

Type **(ENTER)** to get back to the # prompt. To enter the word mode, type:

W (ENTER)

Look at the same memory address again. Press the **(↓)** key a few times. In this mode, the **(↓)** increments the address by two. The numbers contained in each address are the same, but you are seeing them two bytes or one word at a time.

Press the **(↑)** a few times. The **(↑)** always decrements the address by one, regardless of the examination mode.

Look at Address \$A000 again by typing:

A000 /

Note the contents of this address "word." This is the address where POLCAT, a ROM routine, is stored.

Examine the POLCAT routine. For example, if \$A000 contains A1C1, type:

A1C1 /

and you'll see the contents of the first two bytes in the POLCAT routine. We'll examine this routine later in this chapter using the "mnemonic mode."

ASCII Mode

Return to the command level. To enter the ASCII mode, type:

A (ENTER)

ZBUG now assumes the content of each memory address is an ASCII code. If the "code" is between \$21 and \$7F, ZBUG displays the character it represents. Otherwise, it displays meaningless characters or "garbage."

Here, the **(↓)** increments the address by one.

Mnemonic Mode

This is the default mode. Unless you ask for some other mode, you will be in the default mode.

Return to the # prompt. To enter the mnemonic mode from another mode, type:

M (ENTER)

Look at the addresses where the POLCAT routine is

stored. For example, if you found that POLCAT is at address \$A1C1, type:

A1C1 /

Press the **(↓)** a few times. In the mnemonic mode, ZBUG assumes you're examining an assembly language program. The **(↓)** increments memory one to five bytes at a time by "disassembling" the numbers into the mnemonics they represent.

For example, assume the first two addresses in POLCAT contain \$3454. \$3454 is an opcode for the PSHS U,X,B mnemonic. Therefore, ZBUG disassembles \$3454 into PSHS U,X,B.

Begin the disassembly at a different byte. Press **(BREAK)** and then examine the address of POLCAT plus one. For example, if POLCAT starts at address \$A1C1, type:

A1C2 /

You now see a different disassembly. The contents of memory have not changed. ZBUG has, however, interpreted them differently.

For example, assume \$A1C2 contains a \$54. This is the opcode for the LSRB mnemonic. Therefore, ZBUG disassembles \$54 into LSRB.

To see the program correctly, you must be sure you are beginning at the correct byte. Sometimes, several bytes will contain the symbol "?". This means ZBUG can't figure out which instruction is in that byte and is possibly disassembling from the wrong point. The only way of knowing you're on the right byte is to know where the program starts.

Changing Memory

As you look at the contents of memory addresses, notice that the cursor is to the right. This allows you to change the contents of that address. After typing the new contents, press **(ENTER)** or **(↓)**; the change will be made.

To show how to change memory, we'll open an address in video memory. Get into the byte mode and open Address \$015A by typing:

**(BREAK) B (ENTER)
015A /**

Note that the cursor is to the right. To put a 1 in that address, type:

1 (ENTER)

If you want to change the contents of more than one address, type:

015A/

Then type:

DD ↴

This changes the contents to DD and lets you change the next address. (Press the ↴ to see that the change has been made.)

The size of the changes you make depends on the examination mode you are in. In the byte mode, you will change one byte only and can type one or two digits.

In the word mode, you will change one word at a time. Any 1-, 2-, 3-, or 4-digit number you type will be the new value of the word.

If you type a hexadecimal number that is also the name of a 6809 registers (A,B,D,CC,DP,X,Y,U,S,PC), ZBUG assumes it's a register and gives you an "EXPRESSION ERROR." To avoid this confusion, include a leading zero (0A,0B, etc.)

To change memory in the ASCII mode, use an apostrophe before the new letter. For example, here's how to write the letter C in memory at Address \$015A. To get into the ASCII examination mode, type:

A [ENTER]

To open Address \$015A, type:

015A/

To change its contents to a C, type:

'C ↴

Pressing the ↴ will assure you that the address contains the letter C.

If you are in mnemonic mode, you must change one to five bytes of memory depending on the length of the opcode. Changing memory is complex in mnemonic mode because you must type the opcodes rather than the mnemonic.

For example, get into the mnemonic mode and open Address \$015A. Type:

**M [ENTER]
015A/**

To change this instruction, type:

B6 [ENTER]

Now Address \$015A contains the opcode for the LDA mnemonic. Open location 015B:

015B/

and insert \$06, the operand:

06 [ENTER]

Upon examining Address \$015A again, you'll see it now contains an LDA #6 instruction.

Exploring the Computer's Memory

You are now invited to examine each section of memory using ZBUG commands to change examination modes. Use the Memory Map in Reference J.

Don't hesitate to try commands or change memory. You can restore anything you alter simply by removing the diskette and turning the computer off and then on again.

Chapter 6/ Editing the Program Editor Commands

The editor has many commands to help you edit your source program. *Chapter 2* shows how to enter a source program. This chapter shows how to edit it.

To use the edit commands you must return to the editor from ZBUG:

EDTASM: From EDTASM ZBUG, return to the editor by typing E **(ENTER)**

EDTASMOV: From Stand-Alone ZBUG, return to the DOS menu by typing K **(ENTER)**. Then, execute the EDTASMOV program.

The screen now shows the editor's * prompt. While in the editor, you can return to the * prompt at any time by pressing **BREAK**.

This chapter uses SAMPLE/ASM from *Chapter 2* as an example. To load SAMPLE/ASM into the editor, type:

L SAMPLE/ASM **(ENTER)**

Print Command

Prange

To print a line of the program on the screen, type:

P 100 **(ENTER)**

To print more than one line, type:

P 100 : 130 **(ENTER)**

You will often refer to the first line, last line, and current line (the last line you printed or inserted). To make this easier, you can refer to each with a single character:

- # first line
- * last line
- current line (the last line you printed or inserted.)

To print the current line, type:

P . **(ENTER)**

To print the entire text of the sample program, type:

P # : * **(ENTER)**

This is the same as P050:200 **(ENTER)**.

The colon separates the beginning and ending lines in a range of lines. Another way to specify a range of lines is with !. Type:

P # ! 5 **(ENTER)**

and five lines of your program, beginning with the first one, are printed on the screen.

To stop the listing while it is scrolling, quickly type:

SHIFT @

To continue, press any key.

Printer Commands

Hrange

Trange

If you have a printer, you can print your program with the H and T commands. The H command prints the editor-supplied line numbers. The T command does not.

To print every line of the edit buffer to the printer, type:

H # : * **(ENTER)**

You are prompted with:

PRINTER READY

Respond with **(ENTER)** when ready.

The next example prints six lines, beginning with line 100, but without the editor-supplied line numbers. Type:

T 100 ! 6 **(ENTER)**

Edit Command

Eline

You can edit lines in the same way you edit Extended

COLOR BASIC lines. For example, to edit line 100, type:

E 100 **ENTER**

The new line 100 is displayed below the old line 100 and is ready to be changed.

Press the **SPACEBAR** to position the cursor just after START. Type this insert subcommand:

I ED **ENTER**

which inserts ED in the line.

The edit subcommands are listed in *Reference A*.

Delete Command

Drange

If you are using the sample program, be sure you have written it on disk before you experiment with this command. Type:

D 110 : 140 **ENTER**

Lines 110 through 140 are gone.

Insert Command

Istartline, increment

Type:

I 152 , 2 **ENTER**

You may now insert lines (up to 127 characters long) beginning with line 152. Each line is incremented by two. (The editor does not allow you to accidentally overwrite an existing line. When you get to line 160, it gives you an error message.)

Press **BREAK** to return to the command level. Then type:

I 200 **ENTER**

This lets you begin inserting lines at the end of the program. Each line is incremented by two, the last increment you used.

Type:

BREAK I **ENTER**

The editor begins inserting at the current line.

On startup, the editor sets the current line to 100 and the increment to 10. You may use any line numbers between 0 and 63999.

Renumber Command

Nstartline,increment

Another command that helps with inserting lines between the lines is N (for renumber). From the command level, type:

N 100 , 50 **ENTER**

The first line is now Line 100 and each line is incremented by 50. This allows much more room for inserting between lines.

Type:

N **ENTER**

The current line is now the first line number.

Renumber now so you will be ready for the next instruction. Type:

N 100 , 10 **ENTER**

Replace Command

Rstartline,increment

The replace command is a variation of the insert command. Type:

R 100 , 3 **ENTER**

You may now replace line 100 with a new line and begin inserting lines using an increment of three.

Copy Command

Cstartline,range,increment

The copy command saves typing by duplicating any part of your program to another location in the program.

To copy lines, type:

C 500 , 100 : 150 , 10 **ENTER**

This copies lines 100 to 150 to a new location beginning at Line 500, with an increment of 10. An attempt to copy lines over each other will fail.

ZBUG Command

The EDTASM system contains a copy of the stand-alone ZBUG program. This allows you to enter ZBUG while your program is still in memory.

EDTASMOV Users: You need to use the Stand-Alone ZBUG program, as shown in *Chapter 2*.

To enter ZBUG, type:

Z [ENTER]

The # prompt tells you that you are now in ZBUG.

To re-enter the editor from ZBUG, type the ZBUG command:

E [ENTER]

If you print your program, you'll see that entering and exiting ZBUG did not change it.

BASIC Command

To enter BASIC from the editor, type:

Q [ENTER]

If you want to enter DOS from the editor, type:

K [ENTER]

Entering DOS or BASIC empties your edit buffer. Re-entering the editor empties your BASIC buffer.

Write Command

WD filespec

This command is the same one you used in Chapter 2 to write the source program to disk. It saves the program in a disk file named *filespec*. *Filespec* can be in one of these forms:

filename/ext:drive
filename.ext:drive

The *filename* can be one to eight characters. It is required.

The *extension* can be one to three characters. It is optional. If the extension is omitted, the editor assigns the file the extension /ASM.

The *drive* can be a number from 0 to 4. It is also optional. If the drive number is omitted, the editor uses the first available drive.

Examples:

WD TEST [ENTER]

saves source file currently in memory as TEST/ASM.

WD TEST/PR1

saves the source file currently in memory as TEST/PR1.

Load Command

LD filespec

LDA filespec

This command loads a source *filespec* from disk into the edit buffer. If the source *filespec* you specify does not have an extension, the editor uses /ASM.

If you don't specify the A option, the editor empties the edit buffer before loading the file.

If you specify the A option, the editor appends the file to the current contents of the edit buffer.

Appending files can be useful for chaining long programs. When the second file is loaded, simply renumber the file with the renumber command.

Examples:

LD SAMPLE:1

empties the edit buffer, then loads a file named SAMPLE/ASM from Drive 1.

LDA SAMPLE/PRO

loads a file named SAMPLE/PRO from the first available drive, then appends to the current contents of the edit buffer.

The editor has several other commands. These are listed in Reference A.

Hints on Writing Your Program

- Copy short programs from any legal source available to you. Then modify them one step at a time to learn how different commands and addressing modes work. Try to make the program relocatable by using indexed, relative, and indirect addressing (described in Section III).
- Try to write a long program as a series of short routines that use the same symbols. They will be easier to understand and debug. They can later be combined into longer routines.

Note: You can use the editor to edit your BASIC programs, as well as assembly language programs. You might find this very useful since the EDTASM editor is much more powerful than the BASIC editor. You need to first save the BASIC program in ASCII format:

SAVE filespec, A

Then, load the program into the editor.

Chapter 7/ Assembling the Program (Assembler Commands)

To load the assembler program and assemble the source program into 6809 machine code, EDTASM (or EDTASMOV) has an "assembly command." Depending on how you enter the command, the assembler:

- Shows an "assembly listing" giving information on how the assembler is assembling the program.
- Stores the assembled program in memory.
- Stores the assembled program on disk.
- Stores the assembled program on tape.

This chapter shows the different ways you can control the assembly listing, the in-memory assembly, and the disk assembly. Knowing this will help you understand and debug a program.

The Assembly Command

The command to assemble your source program into 6809 machine code is:

Assembling in memory:

A /IM /switch2/switch3/ . . .

The /IM (in memory) switch is required.

Assembling to disk:

A filespec /switch1/switch2/ . . .

The assembled program is stored on disk as *filespec*. If *filespec* does not include an extension, the assembler uses /BIN.

Assembling to tape:

A filename /switch1/switch2/ . . .

The assembled program is stored on tape as *filename*.

The switch options are as follows:

/AO	Absolute origin
/IM	Assemble into memory
/LP	Assembler listing on the line printer
/MO	Manual origin
/NL	No listing
/NO	No object code in memory or disk
/NS	No symbol table in the listing
/SR	Single record
/SS	Short screen listing
/WE	Wait on assembly errors
/WS	With symbols

You may use any combination of the switch options. Be sure to include a blank space before the first switch. If you omit *filespec*, you must use the in-memory switch (/IM).

Examples:

A /IM /WE

assembles the source program in memory (/IM) and stops at each error (/WE).

A TEST /LP

assembles the source program and saves it on disk as TEST/BIN. The listing is printed on the printer (/LP). Note that there must be a space between the *filespec* and the switch.

A TEST /PRO

assembles the source program and saves it on disk as TEST/PRO.

The figure shows two screens of assembly code from a computer interface. The top screen displays assembly code with annotations:

- Annotation 1:** A box around the first instruction (ORG \$1200) with a line pointing to the right margin.
- Annotation 2:** A box around the LDA #\$F9 instruction with a line pointing to the right margin.
- Annotation 3:** A box around the symbol START with a line pointing to the right margin.

The bottom screen displays assembly code with annotations:

- Annotation 4:** A box around the error message "0000 TOTAL ERRORS" with a line pointing to the right margin.
- Annotation 5:** A box around symbols and their memory locations: BEGIN (1200), DONE (121D), SCREEN (120A), START (1205), and WAIT (1211).

```

    ORG      $1200
1200 7E  1205    00060 BEGIN
JMP     START
1203 001D    00070
FDB     DONE-BEGIN
1205 [B6] F9    00100 START
LDA     #$F9
1207 8E  0400  00110
LDX     #$400
120A A7  80     00120 SCREEN
STA     ,X+
120C BC  0600  00130
CMPX   #$600
120F 26  F9     00140
BNE     SCREEN
1211 AD  9F A000 00150 WAIT
JSR     [$A000]
1215 27  FA     00160
BEQ     WAIT
1217 0F  71     00170

```

```

CLR      $71
1219 6E  9F FFFE 00180
JMP     [$FFFF]
        121D    00190 DONE
EQU     *
        0000    00200
END

0000 TOTAL ERRORS
BEGIN 1200
DONE   121D
SCREEN 120A
START  1205
WAIT   1211

```

1. The location in memory where the assembled code will be stored. In this example, the assembled code for LDA#\$F9 will be stored at hexadecimal location #1200.
2. The assembled code for the program line. \$86F9 is the assembled code for LDA #\$F9.
3. The program line.
4. The number of errors. If you have errors, you will want to assemble the program again with the /WE switch.
5. The symbols you used in your program and the memory locations they refer to.

Figure 1. Assembly Display Listing

Controlling the Assembly Listing

The assembler normally displays an assembly listing similar to the one in *Figure 1*. You can alter this listing with one of these switches:

- | | |
|-----|--------------------------------|
| /SS | Short screen listing |
| /NS | No symbol table in the listing |
| /NL | No listing |
| /LP | Listing printed on the printer |

For example:

A SAMPLE /NS

assembles SAMPLE and shows a listing without the symbol table.

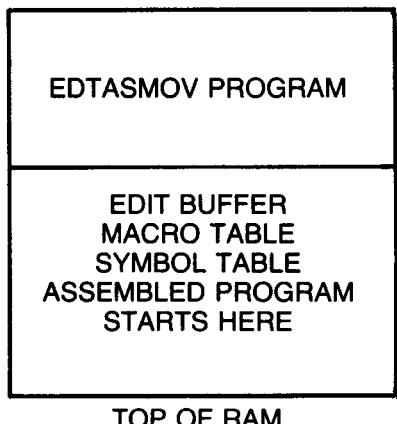
If you are printing the listing on the printer, you might want to set different parameters. You can do this with the editor's "set line printer parameters" command:

To use this command, type (at the * prompt):

S **ENTER**

The editor shows you the current values for:

- LINCNT — the number of lines printed on each page. ("line count")
- PAGLEN — the number of lines on a page. ("page length")
- PAGWID — the number of columns on a page. ("page width")
- FLDFLG — the "fold flag" (This flag should contain 1 if your printer does not "wrap around." Otherwise, the flag should contain 0.)



It then prompts you for different values. Check your printer manual for the appropriate parameters. If you want the value to remain the same, simply press **ENTER**. For example:

```
LINCNT=58
PAGLEN=66
PAGWID=80
FLDFLG=0
```

sets the number of lines to 58, the page length to 66, and the page width to 80 columns. You can then assemble the program with the /LP switch:

A SAMPLE /LP

and the assembler prints the listing on the line printer using the parameters just set.

In-Memory Assembly The /IM Switch

The /IM switch causes the program to be assembled in memory, not on disk or tape. This is a good way to find errors in a program.

Where in memory? This depends on whether you use the /IM switch alone or accompany it with an ORG instruction, an /AO switch, or an /MO switch.

Using the /IM Switch Alone

This is the most efficient use of memory. The assembler stores your program at the first available address after the EDTASM (or EDTASMOV) program, the edit buffer, and the symbol table:

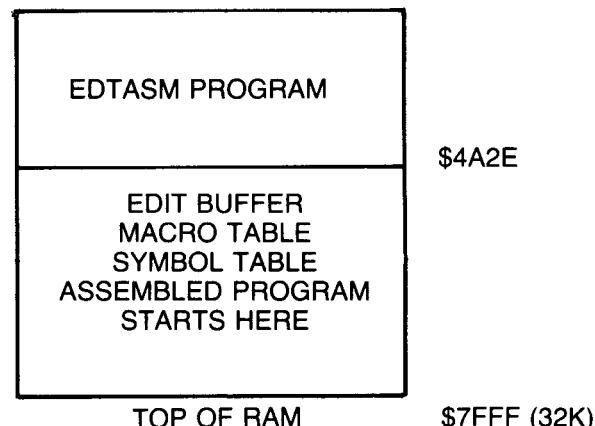


Figure 2. In-Memory Assembly

The EDTASM program ends at Address \$4A2D. The EDTASMOV program ends at \$36D5.

The edit buffer contains the source program. It begins at Address \$4A2E or \$36D6 and varies in size depending on your program's length.

The macro table references all the macro symbols in your program and their corresponding values. (Macros are described in Chapter 12.) Its size varies depending on how many macros your program contains.

The symbol table references all your program's symbols and their corresponding values. Its size varies depending on how many symbols your program contains.

Example:

Load the SAMPLE/ASM back into the edit buffer. At the * prompt, type:

L SAMPLE/ASM **ENTER**

Delete the ORG line. At the * prompt, type:

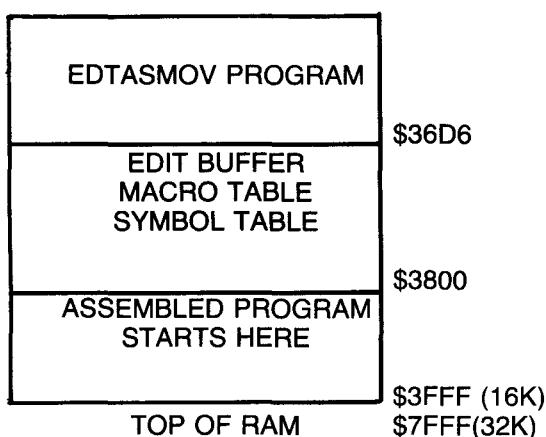
D50 **ENTER**

Then assemble the program in memory by typing:

A / IM **ENTER**

(If you want another look, type A/IM again. You can pause the display by pressing **SHIFT** @ and continue by pressing any key.)

Since this sample program uses START to label the beginning of the program, you can find its originating address from the assembler listing. If you are using EDTASM, it should begin at Address \$4B1E. If you are using EDTASMOV, it should begin at \$37C6.



Using ORG with /IM for Origination Offset

If you have an ORG instruction in your program and do not use the AO switch, the assembler stores your program at:

the first available address + the value of ORG

Example:

Insert this line at the beginning of the sample program:

EDTASM Systems:

0050 ORG \$6000

EDTASMOV Systems:

0050 ORG \$3800

Then, at the * prompt, type:

A / IM **ENTER**

The START address is now the first available address + \$6000 or \$3800. This means that if you have less than 32K (with EDTASM) or less than 16K (with EDTASMOV), the program extends past the top of RAM and you will get a BAD MEMORY error.

Using IM with /AO for Absolute Origin

The AO switch causes the assembler to store your program "absolutely" at the address specified by ORG.

With the ORG instruction inserted, type (at the * prompt):

A / IM / AO **ENTER**

Your program now starts at address \$6000 or \$3800:

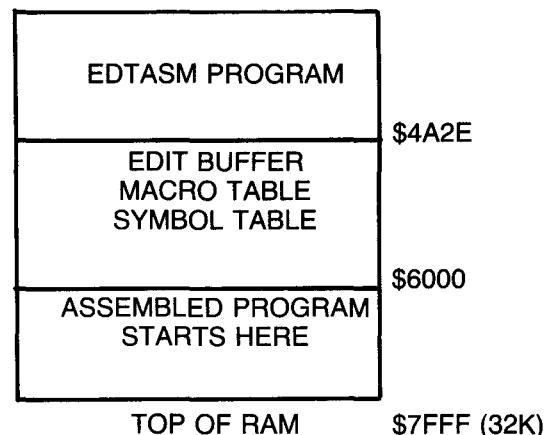


Figure 3. /AO In-Memory Assembly.

As you can see, the AO switch set the location of the assembled program only. It did not set the location of the edit buffer or the symbol table.

If your ORG instruction does not allow enough memory for your program, you will get a BAD MEMORY error. The assembler cannot store your program beyond the top of RAM.

Using /MO with /IM for Manual Origin

The /MO switch causes your program to be assembled at the address set by USRORG (plus the value set in your ORG instruction, if you use one). To set USRORG, use the editor's "origin" command.

Before setting USRORG, remove the ORG instruction from your program. Then, at the * prompt, type:

ENTER

The editor shows you the current values for:

- FIRST — the first hexadecimal address available

- LAST — the last hexadecimal address available
- USRORG — the current hexadecimal value of USRORG. (On startup, USRORG is set to the top of RAM.)

It then prompts you for a new value for USRORG. If you want USRORG to remain the same, press **ENTER**.

If you want to enter a new value, it must be between the FIRST address and LAST address. Otherwise, you will get a BAD MEMORY error.

EDTASM Systems: Set USRORG to \$6050:

USRORG=6050 **ENTER**

EDTASMOV Systems: Set USRORG to \$3800:

USRORG=3800 **ENTER**

After setting USRORG, you can assemble the program at the USRORG address. Type:

A/IM/MO **ENTER**

Your assembled program now starts at Address \$6050 or \$3800:

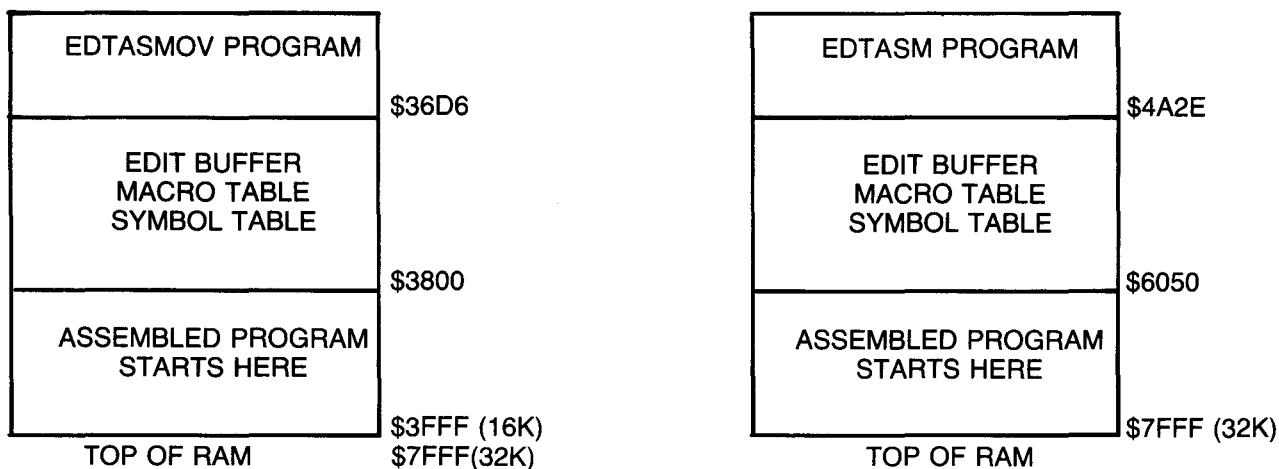


Figure 4. /MO In-Memory Assembly.

Disk Assembly

When you specify a filespec in the assembler command, the assembler saves the assembled program on disk. You can then load the program from one of these systems:

- DOS (to run as a stand-alone program)
- ZBUG (to debug with the stand-alone ZBUG program)
- BASIC (to call from a BASIC program)

The program originates at the address you specify in the ORG instruction.

What address you should use as the originating address depends upon which of the three systems you will be loading it into.

Assembling for DOS

Reference J shows the memory map that is in effect when DOS is loaded. As you can see, DOS consumes all the memory up to Address \$1200. This means you must originate the program after \$1200 or you will overwrite DOS.

In the sample program, reinsert the ORG \$1200 instruction:

```
50      ORG  $1200
```

and assemble it to disk by typing:

```
A SAMPLE /SR [ENTER]
```

Note the /SR switch. You must use /SR when assembling to disk a program that you plan to load back into DOS. This puts the program in the format expected by DOS.

The assembler saves SAMPLE/BIN to disk with a starting address of \$1200. You can now load and execute SAMPLE/BIN from the DOS menu.

Assembling for Stand-Alone ZBUG (EDTASMOV Users)

If you plan to use the stand-alone ZBUG for debugging

your program, you need to save the program on disk so that you can load it into ZBUG.

Reference J also shows the memory map that is in effect when ZBUG is loaded. As you can see, you must use an originating address of at least \$3800 or you will overwrite ZBUG. Change the ORG instruction to:

```
50      ORG  $3800
```

So that you can test this from ZBUG, without the program returning to BASIC, you need to change the ending of it. First, delete the CLR instruction in Line 170:

```
D170 [ENTER]
```

Then, change the JMP instruction in Line 180 to this:

```
180  SWI
```

After making the changes to the program, assemble it to disk by typing:

```
A SAMPLE/BUG /WS [ENTER]
```

The assembler saves SAMPLE/BUG on disk with a starting address of \$3800. The /WS switch causes the assembler to save the symbol table also.

Hints On Assembly

- Use a symbol to label the beginning of your program.
- When doing an in-memory assembly on a program with an ORG instruction, you may want to use the /AO switch. Otherwise, the assembler will not use ORG as the program's originating address. It will use it to offset (add-to) the loading address.
- The /WE switch is an excellent debugging tool. Use it to detect assembly errors before debugging the program.
- If you would like to examine the edit buffer and symbol table after an in-memory assembly, use ZBUG to examine the appropriate memory locations.

Chapter 8/ Debugging the Program (ZBUG Commands — Part II)

ZBUG has some powerful tools for a trial run of your assembled program. You can use them to look at each register, every flag, and every memory address during every step of running the program.

Before reading any further, you might want to review the ZBUG commands you learned in *Chapter 5*. We will be using these commands here.

Preparing the Program for ZBUG

In this chapter, we'll use the sample program from *Chapter 2* to show how to test a program. How you load the program into ZBUG depends on whether you are using EDTASM's ZBUG program or the stand-alone ZBUG program.

EDTASM ZBUG:

If you are using EDTASM, you can use EDTASM's ZBUG program.

1. Load SAMPLE/ASM into EDTASM (if it's not already loaded).
2. So that your program will be in the same area of memory as ours, change the ORG instruction to:

50 ORG \$5800

3. So that you can test the program properly from ZBUG (without the program returning to BASIC), you need to change the program's ending. First, delete the CLR instruction in Line 170:

D170 (**ENTER**)

Then, change the JMP instruction in Line 180 to this:

180 SWI

4. Assemble the program in memory using the /IM and /AO switches. At the * prompt, type:

A / IM / AO (**ENTER**)

5. Enter ZBUG. At the * prompt, type:

Z (**ENTER**)

When the # prompt appears, you're in ZBUG and can test the sample program.

Stand-Alone ZBUG:

If you are using EDTASMOV, you should use the Stand-Alone ZBUG.

1. Assemble SAMPLE/BUG to disk as instructed in the last chapter ("Assembling for Stand-Alone ZBUG").
2. Return to DOS and execute the stand-alone ZBUG program:

EXECUTE A PROGRAM
PROGRAM NAME [ZBUG]]/BIN

ZBUG loads and displays its # prompt.

3. Load SAMPLE/BUG, along with its symbol table, into ZBUG. Type:

LDS SAMPLE/BUG (**ENTER**)

When the # prompt appears, you're ready to test the sample program with ZBUG.

Display Modes

In *Chapter 5*, we discussed four examination modes. ZBUG also has three display modes.

We'll examine each of these display modes from the mnemonic examination mode. If you're not in this mode, type M (**ENTER**) to get into it.

Numeric Mode

Type:

N [ENTER]

and examine the memory addresses that contain your program: \$5800-\$5817 for EDTASM's ZBUG or \$3800-\$3817 for Stand-Alone ZBUG.

In the numeric mode, you do not see any of the symbols in your program (BEGIN, START, SCREEN, WAIT, and DONE). All you see are numbers. For example, with EDTASM's ZBUG, Address \$580F shows the instruction BNE 580A rather than BNE SCREEN.

Symbolic Mode

From the command level, type:

S [ENTER]

and examine your program again. ZBUG displays your entire program in terms of its symbols (BEGIN, START, SCREEN, WAIT, and DONE). Examine the memory address containing the BNE SCREEN instruction and type:

;

The semicolon causes ZBUG to display the operand (SCREEN) as a number (580A or 380A).

Half-Symbolic Mode

From the command level, type:

H [ENTER]

and examine the program. Now all the memory addresses (on the left) are shown as symbols, but the operands (on the right) are shown as numbers.

Using Symbols to Examine Memory

Since ZBUG understands symbols, you can use them in your commands. For example, with EDTASM's ZBUG, both these commands open the same memory address no matter which display mode you are in:

**BEGIN/
5800/**

Both of these commands get ZBUG to display your entire program:

**T BEGIN DONE
T 5800 5817**

You can print this same listing on your printer by substituting TH for T.

Executing the Program

You can run your program from ZBUG using the G (Go) command followed by the program's start address:

EDTASM ZBUG: Type either of the following:

**GBEGIN [ENTER]
G5800 [ENTER]**

Stand-Alone ZBUG: Type either of the following:

**GBEGIN [ENTER]
G3800 [ENTER]**

The program executes, filling all of your screen with a pattern made up of F9 graphics characters. If you don't get this pattern, the program probably has a "bug." The rest of the chapter discusses program bugs.

After executing the program, ZBUG displays 8 BRK @ 5817, 8 BRK @ 3817, or 8 BRK @ DONE. This tells you the program stopped executing at the SWI instruction located at Address DONE. ZBUG interprets your closing SWI instruction as the eighth or final "breakpoint" (discussed below).

Setting Breakpoints

If your program doesn't work properly, you might find it easier to debug it if you break it up into small units and run each unit separately. From the command level, type X followed by the address where you want execution to break.

We'll set a breakpoint at the first address that contains the symbol SCREEN: \$580A for EDTASM's ZBUG or 380A for Stand-Alone ZBUG.

EDTASM ZBUG: Type either of the following:

**XSCREEN [ENTER]
X580A [ENTER]**

Stand-Alone ZBUG: Type either of the following:

XSCREEN [ENTER]
X380A [ENTER]

Now type GBEGIN [ENTER] to execute the program. Each time execution breaks, type:

C [ENTER]

to continue. A graphics character appears on the screen each time ZBUG executes the SCREEN loop. (The characters appear to be in different positions because of scrolling. You will not see the first 32 characters because they scroll off the screen.)

Type:

D [ENTER]

to display all the breakpoints you have set. (You may set up to eight breakpoints numbered 0 through 7.)

Type:

C10 [ENTER]

and the tenth time ZBUG encounters that breakpoint, it halts execution.

Type:

Y [ENTER]

This is the command to "yank" (delete) all breakpoints. You can also delete a specific breakpoint. For example:

Y0 [ENTER]

This deletes the first breakpoint (Breakpoint 0).

You may not set a breakpoint in a ROM routine. If you set a breakpoint at the point where you are calling a ROM routine, the C command will not let you continue.

Examining Registers and Flags

Type:

R [ENTER]

What you see are the contents of every register during this stage of program execution. (See Chapter 10 for definition of all the 6809 registers and flags.)

Look at Register CC (the Condition Code). Notice the letters to the right of it. These are the flags that are set in Register CC. The E, for example, means the E flag is set.

Type:

X /

and ZBUG displays only the contents of Register X. You can change this in the same way you change the contents of memory. Type:

Ø [ENTER]

and the Register X now contains a zero.

Stepping Through the Program

Type:

BEGIN, Note the comma!

LDA #\$F9 is the next instruction to be executed. The first instruction, JMP START, has just been executed. To see the next instruction, type:

, Simply a comma

Now, LDA #\$F9 has been executed and LDX #\$500 is the next. Type:

R [ENTER]

and you'll see this instruction has loaded Register A with \$F9.

Use the comma and R command to continue single-stepping through the program examining the registers at will. If you manage to reach the JSR [\$A000] instruction, ZBUG prints:

CAN'T CONTINUE

ZBUG cannot single-step through a ROM routine or through some of the DOS routines.

Transferring a Block of Memory

EDTASM ZBUG: Type:

U 5800 5000 6 [ENTER]

Stand-Alone ZBUG: Type:

U 3800 3850 6 [ENTER]

Now the first six bytes of your program have been copied to memory addresses beginning at 5000 or 3850.

Saving Memory to Disk

To save a block of memory from ZBUG, including the symbol table, type:

**EDTASM ZBUG: PS TEST/BUG 5800
5817 5800 [ENTER]**

**Stand-Alone ZBUG: PS TEST/BUG 3800
3817 3800 [ENTER]**

This saves your program on disk, beginning at Address 5800 (or 3800) and ending at Address 5817 (or 3817). The last address is where your program begins execution when you load it back into memory. In this case, this

address is the same as the start address.

To load TEST/BUG and its symbol table back into ZBUG, type:

LDS TEST/BUG [ENTER]

Hints on Debugging

- Don't expect your first program to work the first time. Have patience. Most new programs have bugs. Debugging is a fact of life for all programmers, not just beginners.
- Be sure to make a copy of what you have in the edit buffer before executing the program. The edit buffer is not protected from machine language programs.

Chapter 9/ Using the ZBUG Calculator (ZBUG Commands — Part III)

ZBUG has a built-in calculator that performs arithmetic, relational, and logical operations. Also, it lets you use three different numbering systems, ASCII characters, and symbols.

This chapter contains many examples of how to use the calculator. Some of these examples use the same assembled program that we used in the last chapter.

Stand-Alone ZBUG: Some of the memory addresses we use in the examples are too high for your system. Subtract \$1000 from all the hexadecimal addresses and 4096 from all the decimal numbers.

Numbering System Modes

ZBUG recognizes numbers in three numbering systems: hexadecimal (Base 16), decimal (Base 10), and octal (Base 8).

Output Mode

The output mode determines which numbering system ZBUG uses to output (display) numbers. From the ZBUG command level, type:

010 **ENTER**

Examine memory. The T at the end of each number stands for Base 10. Type:

08 **ENTER**

Examine memory. The Q at the end of each number stands for Base 8. Type:

016 **ENTER**

You're now back in Base 16, the default output mode.

Input Mode

You can change input modes in the same way you change output modes. For example, type:

I10 **ENTER**

Now, ZBUG interprets any number you input as a Base 10 number. For example, if you are in this mode and type:

T 49152 49162 **ENTER**

ZBUG shows you memory addresses 49152 (Base 10) through 49162 (Base 10). Note that what is printed on the screen is determined by the output mode, not the input mode.

You can use these special characters to "override" your input mode:

BASE	BEFORE NUMBER	AFTER NUMBER
Base 10	&	T
Base 16	\$	H
Base 8	@	Q

Table 1. Special Input Mode Characters

For example, while still in the I10 mode, type:

T 49152 \$C010 **ENTER**

The "\$" overrides the I10 mode. ZBUG, therefore, interprets C010 as a hexadecimal number. As another example, get into the I16 mode and type:

T 49152T C010 **ENTER**

Here, the "T" overrides the I16 mode. ZBUG interprets 49152 as decimal.

Operations

ZBUG performs many kinds of operations for you. For example, type:

`C000+25T/`

and ZBUG goes to memory address C019 (Base 16), the sum of C000 (Base 16) and 25 (Base 10). If you simply want ZBUG to print the results of this calculation, type:

`C000+25T=`

On the following pages, we'll use the terms "operands," "operators," and "operation." An operation is any calculation you want ZBUG to solve. In this operation:

`1 + 2 =`

"1" and "2" are the operands. "+" is the operator.

Operands

You may use any of these as operands:

1. ASCII characters
2. Symbols
3. Numbers (in either Base 8, 10, or 16) — Please note that ZBUG recognizes integers (whole numbers) only

Examples (Get into the 016 mode):

`'A=`

prints 41, the ASCII hexadecimal code for "A".

`START=`

prints the START address of the sample program. (It will print UNDEFINED SYMBOL if you don't have the sample program assembled in memory.)

`15Q=`

prints the hexadecimal equivalent of octal 15.

If you want your results printed in a different numbering

system, use a different output mode. For example, get into the O10 mode and try the above examples again.

Operators

You may use arithmetic, relational, or logical operators. (Get into the O16 mode for the following examples.)

Arithmetic Operators

Addition	+
Subtraction	-
Multiplication	*
Division	.DIV.
Modulus	.MOD.
Positive	+
Negative	-

Examples:

`DONE-START=`

prints the length of the sample program (not including the SWI at the end).

`9.DIV.2=`

prints 4. (ZBUG can divide integers only.)

`9.MOD.2=`

prints 1, the remainder of 9 divided by 2.

`1-2=`

prints OFFFF,65535T, or 177777Q, depending on which output mode you are in. ZBUG does not use negative numbers. Instead, it uses a "number circle" which operates on modulus 10000 (hexadecimal):

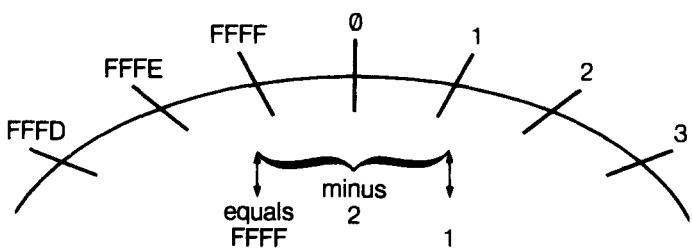


Figure 5. Number Circle Illustration of Memory.

To understand this number circle, you can use the clock as an analogy. A clock operates on modulus 12 in the same way the ZBUG operates on modulus 10000. Therefore, on a clock, 1:00 minus 2 equals 11:00:

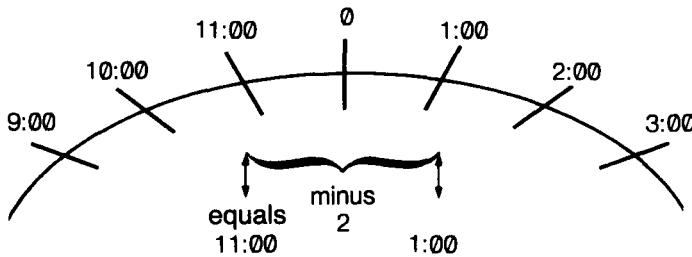


Figure 6. Number Circle Illustration of Clock.

Relational Operators

Equal to	.EQU.
Not Equal to	.NEQ.

These operators determine whether a relationship is true or false.

Examples:

5.EQU.5=

prints 0FFFF, since the relationship is true. (ZBUG prints 65535T in the O10 mode or 177777Q in the O8 mode.)

5.NEQ.5=

prints 0, since the relationship is false.

Logical Operators

Shift	<
LogicalAND	.AND.
InclusiveOR	.OR.
ExclusiveOR	.XOR.
Complement	.NOT.

Logical operators perform bit manipulation on binary numbers. To understand bit manipulation, see the

6809 assembly language book we referred to in the introduction.

Examples:

10<2=

shifts 10 two bits to the left to equal 40. The 6809 SL instruction also performs this operation.

10<-2=

shifts 10 two bits to the right to equal 4. The 6809 ASR instruction also performs this operation.

6.XOR.5=

prints 3, the exclusive or of 6 and 5. The 6809 EOR instruction also performs this operation.

Complex Operations

ZBUG calculates complex operations in this order:

+	.DIV.	,MOD.	<
		,AND.	
	,OR.	.XOR	
+ -			
,EQU.	,NEQ.		

You may use parentheses to change this order.

Examples:

4+4.DIV.2=

The division is performed first.

(4+4).DIV.2=

The addition is performed first.

4*4.DIV.4=

The multiplication is performed first.



SECTION III

ASSEMBLY

LANGUAGE

SECTION III

ASSEMBLY LANGUAGE

This section gives details on the Disk EDTASM assembly language. It does not explain the 6809 mnemonics, however, since there are many books available on the 6809.

To learn about 6809 mnemonics, read one of the books listed in "About This Manual." If you need more technical information on the 6809, read:

MC6809-MC6809E

**8-Bit Microprocessor Programming
Manual
Motorola, Inc.**

Chapter 10/ Writing the Program

Chapter 3 gives a general description of assembly language instructions. This chapter describes them in detail.

The 6809 Registers

The 6809 contains nine temporary storage areas that you may use in your program:

REGISTER	SIZE	DESCRIPTION
A	1 byte	Accumulator
B	1 byte	Accumulator
D	2 bytes	Accumulator (a combination of A and B)
DP	1 byte	Direct Page
CC	1 byte	Condition Code
PC	2 bytes	Program Counter
X	2 bytes	Index
Y	2 bytes	Index
U	2 bytes	Stack Pointer
S	2 bytes	Stack Pointer

Table 2. 6809 Registers

Registers A and B can manipulate data and perform arithmetic calculations. They each hold one byte of data. If you like, you can address them as D, a single 2-byte register.

Register DP is for direct addressing. It stores the most significant byte of an address. This lets the processor directly access an address with the single, least significant byte.

Registers X and Y can each hold two bytes of data. They are mainly for indexed addressing.

Register PC stores the address of the next instruction to be executed.

Registers U and S each hold a 2-byte address that points to an entire "stack" of memory. This address is the top of the stack + 1. For example, if Register U contains 0155, the stack begins with Address 154 and continues downwards.

The processor automatically points Register S to a stack of memory during subroutine calls and interrupts. Register U is solely for your own use. You can access either stack with the PSH and PUL mnemonics or with indexed addressing.

Register CC is for testing conditions and setting interrupts. It consists of eight "flags." Many mnemonics "set" or "clear" one or more of these flags. Others test to see if a certain flag is set or clear.

This is the meaning of each flag, if set:

C (Carry), Bit 0 — an 8-bit arithmetic operation caused a carry or borrow from the most significant bit.

V (Overflow), Bit 1 — an arithmetic operation caused a signed overflow.

Z (Zero), Bit 2 — the result of the previous operation is zero.

N (Negative), Bit 3 — the result of the previous operation is a negative number.

I (Interrupt Request Mask), Bit 4 — any requests for interrupts are disabled.

H (Half Carry), Bit 5 — an 8-bit addition operation caused a carry from Bit 3.

F (Fast Interrupt Request Mask), Bit 6 — any requests for fast interrupts are disabled.

E (Entire Flag), Bit 7 — all the registers were stacked during the last interrupt stacking operation. (If not set, only Registers PC and CC were stacked.)

Assembly Language Fields

You may use four fields in an assembly language instruction: label, command, operand, comment. In this instruction:

```
START    LDA    #$F9      GETS CHAR
```

START is the label. LDA is the command. #\$F9 + 1 is the operand. GETS CHAR is the comment.

The comment is solely for your convenience. The assembler ignores it.

The Label

You can use a symbol in the label field to define a memory address or data. The above instruction uses START to define its memory address.

Once the address is defined, you can use START as an operand in other instructions. For example:

```
BNE      START
```

branches to the memory address defined by START.

The assembler stores all the symbols, with the addresses or data they define, in a "symbol table," rather than as part of the "executable program." The symbol can be up to six characters.

The Command

The command can be either a pseudo op or a mnemonic.

Pseudo ops are commands to the assembler. The assembler does not translate them into opcodes and does not store them with the executable program. For example:

```
NAME    EQU    $43
```

defines the symbol NAME as \$43. The assembler stores this in its symbol table.

```
ORG    $3000
```

tells the assembler to begin the executable program at Address \$3000.

```
SYMBOL   FCB    $6
```

stores 6 in the current memory address and labels this address SYMBOL. The assembler stores this information in its symbol table.

Mnemonics are commands to the processor. The

assembler translates them into opcodes and stores them with the executable program. For example:

```
CLRA
```

tells the processor to clear Register A. The assembler assembles this into opcode number \$4F and stores it with the executable program.

The next chapter shows how to use pseudo ops. Reference L lists the 6809 mnemonics.

The Operand

The operand is either a memory address or data. For example:

```
LDD      #3000+COUNT
```

loads Register D with \$3000 plus the value of COUNT. The operand, #\$3000 + COUNT, specifies a data constant.

The assembler stores the operand with its opcode. Both are stored with the executable program.

Operators

The plus sign (+) in the above operand (#3000 + COUNT) is called an operator.

You can use any of the operators described in Chapter 9, "Using the ZBUG Calculator," as part of the operand.

Addressing Modes

The above example uses the # sign to tell the assembler and the processor that \$3000 is data. When you omit the # sign, they interpret \$3000 in a different "addressing mode."

Example:

```
LDD      $3000
```

tells the assembler and processor that \$3000 is an address. The processor loads D with the data contained in Address \$3000 and \$3001.

Each of the 6809 mnemonics lets you use one to six addressing modes. These addressing modes tell you:

- If the processor requires an operand to execute the opcode
- How the assembler and processor will interpret the operand

1. Inherent Addressing

There is no operand, since the instruction doesn't require one. For example:

SWI

interrupts software. No operand is required.

CLRA

clears Register A. Again, no operand is required. Register A is part of the instruction.

2. Immediate Addressing

The operand is *data*. You must use the # sign to specify this mode. For example:

ADDA #\$30

adds the value \$30 to the contents of Register A.

DATA EQU \$8004
LDX #DATA

loads the value \$8004 into Register X.

CMPX #\$1234

compares the contents of Register X with the value 1234.

3. Extended Addressing

The operand is an *address*. This is the default mode of all operands.

(Exception: If the first byte of the operand is identical to the direct page, which is 00 on startup, it is directly addressed. This is an automatic function of the assembler and the processor. You need not be concerned with it if you're a beginner.)

For example:

JSR #\$1234

jumps to Address \$1234.

SPOT EQU \$1234
STA SPOT

stores the contents of Register A in Address \$1234.

If the instruction calls for data, the operand contains the address where the data is stored.

LDA \$1234

does not load Register A with \$1234. The processor loads A with whatever data is in Address \$1234. If \$06 is

stored in Address \$1234, Register A is loaded with \$06.

ADDA \$1234

adds whatever data is stored in Address \$1234 to the contents of Register A.

LDD \$1234

loads D, a 2-byte register, with the data stored in memory addresses \$1234 and \$1235.

You can use the > sign, which is the sign for extended addressing, to force this mode. (See "Direct Addressing.")

Extended Indirect Addressing.

The operand is the *address of an address*. This is a variation of the extended addressing mode. The [] signs specify it. (Use **SHIFT** ↴ to produce the [sign and **SHIFT** ↵ to produce the] sign.)

In understanding this mode, think of a treasure hunt game. The first instruction is "Look in the clock." The clock contains the second instruction, "Look in the refrigerator."

Examples:

JSR [\$1234]

jumps to the address contained in Addresses \$1234 and \$1235. If \$1234 contains \$06 and \$1235 contains \$11, the effective address is \$0611. The program jumps to \$0611.

SPOT EQU \$1234
STA [SPOT]

stores the contents of Register A in the address contained in Addresses \$1234 and \$1235.

LDD [\$1234]

loads D with the data stored in the address that is stored in Addresses \$1234 and \$1235.

This is a good mode of addressing to use when calling ROM routines. For example, the entry address of the POLCAT routine is contained in Address \$A000. Therefore, you can call it with these instructions:

POLCAT EQU \$A000
JSR [POLCAT]

If a new version of ROM puts the entry point in a different address, your program still works without changes.

4. Indexed Addressing

The operand is an *index register* which points to an

address. The *index register* can be any of the 2-byte registers, including PC. You can augment it with:

- A constant or register offset
- An auto-increment or auto-decrement of 1 or 2

The comma (,) indicates indexed addressing.

As an example, load X, a 2-byte register, with \$1234:

```
LDX      #$1234
```

You can now access Address \$1234 through indexed addressing. This instruction:

```
STA      ,X
```

stores the contents of A in Address \$1234

```
STA      3,X
```

stores the contents of A in Address \$1237, which is \$1234 + 3. (The number 3 is a constant offset.)

```
SYMBOL    EQU      $4
          STA      SYMBOL,X
```

stores the contents of A in Address \$1238, which is \$1234 + SYMBOL. (SYMBOL is a constant offset.)

```
LDB      #$5
          STA      B,X
```

stores the contents of A in Address \$1239 which is \$1234 + the contents of B. (B is a register offset. You can use either of the accumulator registers as a register offset.)

```
STA      ,X+
```

This instruction does two tasks: (1) stores A's contents in Address \$1234 (the contents of X) and then (2) increments X's contents by one, so that X contains \$1235.

```
STA      ,X++
```

(1) stores A's contents in Address \$1235 (the current contents of X) and then (2) increments X's contents by two to equal \$1237.

```
STA      ,--X
```

(1) decrements the current contents of X by two to equal \$1235 (\$1237 - 2) and then (2) stores A's contents in Address \$1235.

As we said above, you can use PC as an index register. In this form of addressing, called program counter relative, the offset is interpreted differently. For example:

```
SYMBOL    FCB      0
          LDA      SYMBOL,PCR
```

While assembling the program, the assembler subtracts the contents of Register PC from the offset:

```
LDA      SYMBOL-PC,PCR
```

While running the program, the processor adds the contents of Register PC to the offset. This causes A to be loaded with SYMBOL.

This seems to be the same as extended addressing. But, by using program counter relative addressing, you can relocate the program without having to reassemble it.

Indexed Indirect Addressing.

The operand is an *index register* which points to the address of an address. This is a variation of indexed addressing.

For example, assume that :

- Register X contains \$1234
- Address \$1234 contains \$11
- Address \$1235 contains \$23
- Address \$1123 contains \$64

This instruction:

```
LDA      [ ,X ]
```

loads A with 64. (Register X points to the addresses of the address. This address is storing 6, the required data.)

```
STA      [ ,X ]
```

stores the contents of A in Address \$1123. (Register X points to the addresses, \$1234 and \$1235, of the effective address, \$1123.)

5. Relative Addressing

The assembler interprets the operand as a *relative address*. There is no sign to indicate this mode. The assembler automatically uses it for all branching instructions.

For example, if this instruction is located at Address \$0580:

```
BRA      $0585
```

The assembler converts \$0585 to a relative branch of +3 (0585-0582).

This mode is invisible to you unless you get a BYTE OVERFLOW error, which we discuss below. Because the processor uses this mode, you can relocate your

program in memory without changing any of the branching instructions.

The BYTE OVERFLOW error means that the relative branch is outside the range of -128 to +127. You must use a long branching instruction instead. For example:

```
LBRA      $0600
```

allows a relative branching range of -32768 to +32767.

6. Direct Addressing

In this mode, the operand is *half of an address*. The other half of the address is in Register DP:

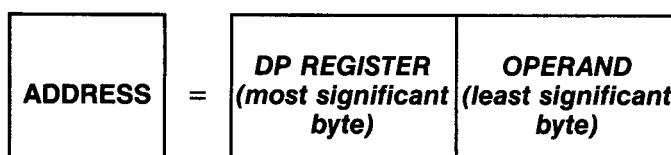


Figure 7. Direct Addressing

The assembler and the processor use this mode automatically whenever they approach an operand whose first byte is what they assume to be a "direct page" (the contents of Register DP). Until you change the direct page, the assembler and the processor assume it is 00.

For example, both of these instructions:

```
JSR      $0015
JSR      $15
```

cause a jump to Address \$0015. In both cases, the assembler uses only 15 as the operand, not 00. When the processor executes them, it gets the 00 portion from Register DP and combines it with \$15. (On startup, DP contains 0, as do all the other registers.)

Because of direct addressing, all operands beginning with 00, the direct page, consume less room in memory

and run quicker. If most of your operands begin with \$12, you might want to make \$12 the direct page.

To do this, you first need to tell the assembler what you are doing, by putting a SETDP pseudo-operation in your program:

```
SETDP    $12
```

This tells the assembler to drop the \$12 from all operands that begin with \$12. That is, the assembler assembles the operand "1234" as simply "34".

Then, you must load Register DP with \$12. Since you can use LD only with the accumulator registers, you have to load DP in a round-about manner:

```
LDB      #$12
TFR      B,DP
```

Now the direct page is \$12, rather than 00. The processor executes all operands that begin with \$12 (rather than 00) in an efficient, direct manner.

The assembler uses direct addressing on all operands whose first byte is the same as the direct page. You can denote direct addressing with the < sign if you want to document or be sure that direct addressing is being used.

For example, if the direct page is \$12:

```
JSR      <$15
```

jumps to Address \$1215. This instruction documents that the processor uses direct addressing.

Similarly, you might want to use the > sign to force extended addressing. For example:

```
JSR      >$1215
```

jumps to Address \$1215. The assembler and processor use both bytes of the operand.

To learn more about 6809 addressing modes, read one of the books listed at the beginning of this manual.

Chapter 11/ Using Pseudo Ops

As discussed earlier, pseudo ops direct the assembler. You can use them to:

- Control where the program is assembled
- Define symbols
- Insert data into the program
- Change the assembly listing
- Do a “conditional” assembly
- Include another source file in your program

Pseudo ops are unique to the assembler you are using. Other 6809 assemblers may not recognize the Disk EDTASM pseudo ops.

The Disk EDTASM pseudo ops make it easier for you to program. This chapter shows how to use pseudo ops.

Controlling Where the Program is Assembled

The Disk EDTASM has two pseudo ops that control where the program is assembled:

- ORG, sets the first location
- END, ends the assembly

ORG

ORG expression

Tells the assembler to begin assembling the program at *expression*. Example:

```
ORG      $1800
```

tells the assembler to start assembling the program at Address \$1800.

You can put more than one ORG command in a pro-

gram. When the assembler arrives at the new ORG, it begins assembling at the new *expression*.

END

END expression

Tells the assembler to quit assembling the program. The *expression* option lets you store the program's start address. Use END as the last instruction in all your assembly language programs.

Example:

DATA	ORG	\$1800
	FCC	'This is some data'
START	LDA	DATA
	.	.
	END	START

The END pseudo op quits the assembly and stores the program's entry address (the value of START) on disk. When you load the program, the processor knows to start executing at START (the LDA instruction) rather than at DATA (the FCC instruction).

FCC is a pseudo op explained later in this chapter.

Defining Symbols

Symbols make it easy to write a program and also make the program easy to read and revise. The Disk EDTASM has two pseudo ops for defining symbols:

- EQU, for defining a constant value
- SET, for defining a variable value

EQU

symbol EQU expression

Equates *symbol* to *expression*. Examples:

CHAR EQU \$F9

equates CHAR to \$F9.

SCREEN EQU \$500
LDX #SCREEN

equates SCREEN to \$500. The next instruction loads X with \$500.

EQU helps set the values of constants. You can use it anywhere in your program.

SET

symbol SET expression

Sets *symbol* equal to *expression*. You can use SET to reset the symbol elsewhere in the program. Example:

SYMBOL SET 25

sets SYMBOL equal to 25. Later in the program, you can reset SYMBOL.

SYMBOL SET SYMBOL+COUNT

now SYMBOL equals 25 + COUNT.

Inserting Data into Your Program

The Disk EDTASM has four pseudo ops that make it simple for you to reserve memory and insert data in your program:

- RMB, for reserving areas of memory for data
- FCB, for inserting one byte of data in memory
- FDB, for inserting two bytes of data in memory
- FCC, for inserting a string of data in memory

Remember that the processor cannot “execute” a block of data in your program. If you use these pseudo ops:

- Use them at the end of your program (just before the END instruction), or
- Precede them with an instruction that jumps or branches to the next “executable” instruction.

RMB

symbol RMB expression

Reserves *expression* bytes of memory for data. Example:

BUFFER RMB 256

reserves 256 bytes for data, starting at Address BUFFER.

DATA RMB 6+SYMBOL

reserves 6 + SYMBOL bytes for data beginning at Address DATA.

FCB

symbol FCB expression

Stores a 1-byte *expression* in memory at the current address. The *symbol* is optional.

Examples:

DATA FCB \$33

stores \$33 in Address DATA.

FACTOR FCB LDA NUM/2
FACTORY

stores NUM/2 in Address FACTOR, then, loads NUM/2 into Register A.

FDB

symbol FDB expression

Stores a 2-byte *expression* in memory starting at the current address. The *symbol* is optional. Example:

DATA FDB \$3322

stores \$3322 in Address DATA and DATA + 1.

FCC

symbol FCC delimiter string delimiter

Stores an ASCII string in memory, beginning at the current address. The *symbol* is optional. The *delimiter* can be any character.

Examples:

TABLE FCC /THIS IS A STRING/

stores the ASCII codes for THIS IS A STRING in memory locations, beginning with TABLE.

NAME	FCC	'Dylan'	MD	List macro definitions (default)
	FCB	\$0D	NOMD	Do not list macro definitions
	LDB	#NAME	MEX	List macro expansions
INIT	LDA	NAME	NOMEX	Do not list macro expansions (default)
	.		L	Turn on the listing (default)
	.		NOL	Turn off the listing
	INCB			
	CMPA	NAME		
	BNE	INIT		

The first instruction stores "Dylan" in the five memory addresses beginning with NAME. The next instructions process this data.

Changing the Assembly Listing

You can use three pseudo ops to change the listing the assembler prints for you:

- TITLE, inserts a title at the top of each listing page
- PAGE, ejects the listing to the next page
- OPT, turns on or off the switches that determine how the assembler lists "macros" (Macros are discussed in the next chapter.)

TITLE string

Tells the assembler to print the first 32 characters of the *string* at the top of each assembly listing page. Example:

```
TITLE      Budget Program
```

causes the assembler to print Budget Program as the title of each page in the assembly listing.

PAGE

Starts a new page if the assembly listing is being printed on the line printer. Example:

```
PAGE
```

tells the assembler to eject the listing to the next page.

OPT

OPT switch, switch, ...

Causes the assembler to use the specified *switches* when printing its listing. You can specify these *switches* with OPT:

MC	List macro calls (default)
NOMC	Do not list macro calls

MD	List macro definitions (default)
NOMD	Do not list macro definitions
MEX	List macro expansions
NOMEX	Do not list macro expansions (default)
L	Turn on the listing (default)
NOL	Turn off the listing

Example:

```
OPT      MEX
```

Causes the assembler to list the macro expansions in its listing. (Macros are discussed in the next chapter.)

Conditional Assembly

You may want to execute a certain section of your program only if a certain condition is true. The Disk EDTASM lets you set up a "conditional" section of your program, using these two pseudo ops:

COND

COND condition expression

Assembles the following instructions only if the *expression* is true (non-zero). If not true (zero), the assembler goes to the instruction that immediately follows the ENDC instruction.

Only these operators are recognized in a condition expression: +,-,/,*. See ENDC below for an example.

ENDC

ENDC

Ends a conditional assembly, initiated by COND.

Examples:

```
COND SYMBOL
```

```
.
```

```
ENDC
```

assembles the lines between COND SYMBOL and ENDC only if SYMBOL is not equal to zero.

```
COND VALUE2-VALUE1
```

```
.
```

```
ENDC
```

assembles the lines between VALUE2-VALUE1 only if VALUE2-VALUE1 are not equal (which causes the result to be a non-zero value).

Including Other Source Files

To let you load another source file and include it in your program, the Disk EDTASM offers an INCLUDE pseudo op.

INCLUDE

INCLUDE *filespec*

Inserts *filespec*, a file of source assembly language instructions, at the point where INCLUDE appears in the

program. The assembler assembles the entire included file before assembling the next instruction.

Example:

INCLUDE ROUTINE/SRC

inserts and assembles ROUTINE/SRC, a source file, before assembling the next instruction.

INCLUDE SUB1/SRC

INCLUDE SUB2/SRC

inserts and assembles SUB1, then inserts and assembles SUB2, then proceeds with the next instruction.

Chapter 12/ Using Macros

A macro is like a subroutine. It lets you call an entire group of instructions with a single program line. This helps when you want to use the same group of instructions many times in the program.

This chapter first tells how to use a macro. It then gives guidelines on the format of a macro.

How to Use a Macro

To use a macro, you must first define it. For example, you could define the entire sample program (from Chapter 2) as a macro named GRAPH.

After defining the macro, you can use its name the same way you use a mnemonic. Whenever the assembler encounters the macro's name, it expands it into the defined instructions.

Defining a Macro

To define a macro, you need to:

- Use MACRO (a pseudo op) to begin the macro definition and assign it a name.
- Use source instructions to define the macro.
- Use ENDM (a pseudo op) to end the macro definition.

This is an example of the sample program converted into a macro definition:

00030	GRAPH	MACRO	
00100	LDA	#\$F9	
00110	LDX	#\$400	
00120	STA	,X+	
00130	CMPX	#\$600	
00140	BNE	\.A	
00150	JSR	[\$A000]	
00160	BEQ	\.B	
00180	ENDM		

Line 30 names the macro as GRAPH, lines 50-160 define the macro, and line 180 ends the macro definition.

Notice the names of the symbols within the macro definition: \.A and \.B. If you do not use this format for naming symbols, you'll get a MULTIPLY DEFINED SYMBOL error when you call the macro more than once. (More on this later.)

Insert the above program using **SHIFT CLEAR** to generate the backslash character (\). Save the program on disk as MACRO1 and then delete it.

```
WD MACRO1 ENTER  
D# : * ENTER
```

Calling a Macro

To call a macro, simply use the macro name as if it were a mnemonic. For example, this sample program calls GRAPH and then ends:

00110	ORG	\$1200
00120	BEGIN	JMP START
00130	FDB	DONE-BEGIN
00140	START	*
00150	INCLUDE	MACRO1/ASM
00160	GRAPH	
00170	CLR	\$71
00180	JMP	[\$FFFF]
00190	DONE	*
00200	END	

Line 150 loads MACRO1, the file containing the definition of GRAPH, and includes it in the source program. Line 160 calls the GRAPH macro.

To see how the assembler expands the GRAPHIC macro, insert this line:

```
00135 OPT MEX
```

and assemble the program. The assembler listing shows how the assembler expands GRAPH into its defined instructions.

Note that the assembler has replaced \.A with A0000 and \.B with B0000. The zeroes indicate that this is the first expansion of the symbols in GRAPH. (In this case, this is the only expansion.)

Passing Values to a Macro

A convenient way to use a macro is to pass values to it. You can use a macro many times in your program, passing different values to it each time.

This is a definition of the GRAPH macro, slightly modified so that you can pass two values to it. Insert this program, save it as MACRO2 and then delete it.

```

00030  GRAPH2  MACRO
00100          LDA      \0
00110          LDX      \1
00120  \.A      STA      ,X+
00130          CMPX     #$B00
00140          BNE      \.A
00150  \.B      JSR      [$A000]
00160          BEQ      \+B
00190          ENDM

```

The \0 and \1 are dummy values. The assembler replaces these numbers with the values you specify when you call GRAPH.

The following program calls GRAPH2 three times. Each time it passes two different sets of values:

```

00100          ORG      $1200
00110  BEGIN    JMP      START
00120          FDB      DONE-BEGIN
00130  START    *
00140          OPT      MEX
00150          INCLUDE MACRO2/ASM
00160          GRAPH2  #$F9,$$400
00170          GRAPH2  #$F8,$$450
00180          GRAPH2  #$F7,$$500
00190          CLR      $71
00200          JMP      [$FFE]
00210  DONE    *
00220          END

```

When the assembler expands the macro, it replaces the dummy values with the values passed by the macro call. For example, the second time GRAPH2 is called, the assembler replaces \0 with #\$F8 and replaces \1 with #\$450.

Assemble the above program. Note that each time the assembler expands GRAPH2, it replaces the \.A and \.B symbols with different symbol names: First A0000 and B0000, then A0001 and B0001, and finally A0002 and B0002.

If the assembler used the same symbol names in each expansion, it would be forced to assign different value to the symbols in each expansion. You would get a MULTIPLY DEFINED SYMBOL error.

Also, note the assembler has inserted an additional symbol, NARG, in the symbol table. NARG is always set to the number of values passed in the most recent macro call.

In the sample program, the symbol table shows that NARG is set to "2" at the end of the assembly. This shows that there were two values passed to GRAPH2 the last time it was called.

You might want to use NARG as a variable in your program. For example, you could conditionally assemble parts of a macro definition based on the current value of NARG.

To see the program run, assemble it to disk, press a key three times to see different graphics and then end the program.

Format of Macros

The remainder of this chapter gives details on the format to use in a macro definition and macro call.

Macro Definition

Beginning the Definition

Use this format for beginning the macro definition and assigning it a name:

symbol MACRO

symbol is the name of the macro. It is, of course, required.

Using Symbols in the Definition

Use this format to name any symbols you use within a macro definition:

\.c

c is an alpha character (A-Z). When the assembler expands the macro, it replaces \.c with:

cnnnn

NNNN is a 4-digit hexadecimal number that the assembler increments each time the assembler expands the macro.

For example, if you use the symbol \.M in the macro definition and you call the macro 10 times, the assembler replaces \.M with these symbol names:

1st expansion	M0001
2nd expansion	M0002
10th expansion	M000A

You must use this symbol-name format when calling a macro more than once. Otherwise, you get MULTIPLY DEFINED SYMBOL errors.

Using Dummy Values in the Definition

Use this format for specifying dummy values within a macro definition:

\n

n is an alphanumeric character (0-9,A-Z). The assembler replaces this dummy value with a corresponding value in the macro call line:

\0 is replaced with the 1st value
\1 is replaced with the 2nd value

\9 is replaced with the 10th value
\A is replaced with the 11th value

\Z is replaced with the 36th value

For example, this line in a macro definition:

LDA \B

specifies \B as a dummy value. The assembler replaces \B with the 12th value in the macro call line. If the macro call line is:

ADD NUM0 ,NUM1 ,NUM2 ,NUM3 ,NUM4 ,
NUM5 ,NUM6 ,NUM7 ,NUM8 ,NUM9 ,NUMA ,NUMB

the assembler replaces \B with NUMB.

You do not need to assign macro call values to dummy values in consecutive order. For example:

```
GRAPHX    $$F9,$$400,$$600
GRAPHX MACRO
LDX      \1
LDY      \2
LDA      \0
LDB      \0
ENDM
```

Here, the assembler replaces dummy value \1 with

#\$400, replaces dummy value \2 with #\$600, and, in two lines, replaces dummy value \0 with #\$F9. Note that you can pass a value to a macro more than once, as this example does with #\$F9.

If there are more dummy values than values in a macro call, a byte overflow error results.

If there are more values than dummy values in a macro call, the extra values are ignored.

Be sure not to enclose dummy values in quotes. If you do this, the assembler treats them as ordinary characters.

Ending the Macro Definition

Use this format for ending the macro definition:

ENDM

You may not use a symbol to label this line. If you do so, you get a MISSING END STATEMENT error at the end of the assembly listing.

Macro Call

Use this format when passing values to a macro in a macro call line:

macro call string1, string2, ...

macro call is the name of the macro.

string(s) is the value being passed to the macro. It can be 1 to 16 characters (any extra characters are ignored).

Each string, except the last, must be separated by a comma. The last string must be terminated by a comma, space, carriage return, or tab.

Each string may contain any characters except a carriage return. If a string contains a comma, space, tab, or left parenthesis, you must enclose it in parentheses. For example, in this macro call:

PRINT (ABC,DEF)

the assembler interprets ABC,DEF as a single string. However, in this call:

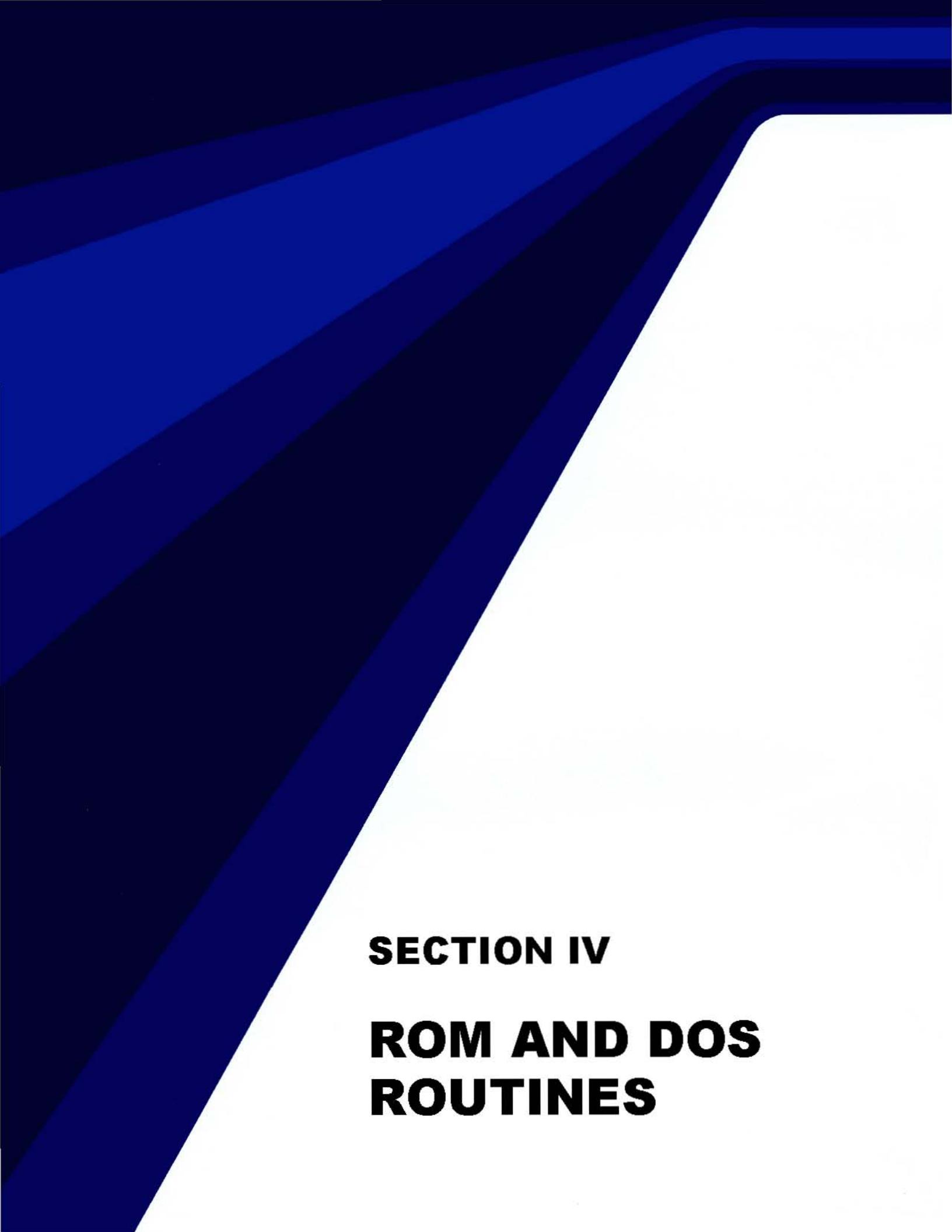
PRINT ABC,DEF

the assembler interprets ABC as one string and DEF as another.

Hints on Macros

- Remember to define a macro before calling it. If you call a macro without defining it, you get a BAD OPCODE error.

- We recommend storing all macro definitions in a file and then using INCLUDE to insert them into your main program.
- Do not use a mnemonic or pseudo op as a macro name. This causes the assembler to redefine the mnemonic or pseudo op according to the macro definition.
- If the macro definition has an error, you will not dis-
cover the error until you call the macro. The assembler waits until you call the macro before it assembles it.
- You cannot “nest” macro definitions. That is, one macro definition cannot call another.
- Using the same macro more than once uses a large amount of memory. Expand a large macro only once. When you want to use it again, call it as a subroutine.



SECTION IV

**ROM AND DOS
ROUTINES**

SECTION IV

ROM AND DOS ROUTINES

In an assembly language program, the simplest way to use the I/O devices is with ROM and DOS routines. This section shows how.

Complete lists of the ROM routines and DOS routines are in the reference section.

Chapter 13/ Using the Keyboard and Video Display (ROM Routines)

The Color Computer uses its own machine-code routines to access the screen, keyboard, and tape. These routines are built into the computer's ROM. You can use the same routines in your own program.

Appendix F lists each ROM routine and the ROM address that points to it. This chapter uses two of these routines, POLCAT and CHROUT, as samples in showing the steps for using ROM routines.

Steps for Calling ROM Routines

We recommend these steps for calling a ROM routine:

1. Equate the routine's address to its name. This lets you refer to the routine by its name rather than its address, making your program easier to read and revise.
2. Set up any entry conditions required by the routine. This lets you pass data to the routine.
3. Preserve the contents of the registers. Since many routines change the contents of the registers, you might want to store the registers' contents temporarily before jumping to the routine.
4. Call the ROM routine, using the indirect addressing mode.
5. Use any exit conditions that the routine passes back to your program.
6. Restore the contents of the registers (if you temporarily preserved them in Step 3).

Sample 1 Keyboard Input with POLCAT

POLCAT "polls" the keyboard to see if you press a key. If you do not, POLCAT sets Bit Z.

If you do press a key, POLCAT:

- (1) Clears Bit Z of Register CC and
- (2) Loads Register A with the key's ASCII code.

This short program uses POLCAT to poll the keyboard. When you press a key, the program ends:

```

ORG    $1200
BEGIN  JMP    START
        FDB    DONE-BEGIN
POLCAT EQU    $A000
START   PSHS   DP,CC,X,Y,U
WAIT    JSR    [POLCAT]
        BEQ    WAIT
        PULS   DP,CC,X,Y,U
        CLR    $71
        JMP    [$FFFE]
DONE    *
        END

```

This is how we applied the above steps in writing this program:

1. Equate POLCAT to its Address

This equates POLCAT to \$A000, the address that points to POLCAT's address:

```
POLCAT EQU    $A000
```

2. Set Up Entry Conditions

POLCAT has no entry conditions.

3. Preserve the Registers' Contents

POLCAT's "Exit Conditions" state that POLCAT modifies all registers except B and X. Assume that you want to preserve the contents of Registers DP, CC, X, Y, and U. To do this, you can "push" these values into the "hardware stack":

```
PSHS DP,CC,X,Y,U
```

(The hardware stack is an area of memory, pointed to by Register S, that the processor uses for subroutines. PSHS "preserves" the contents of certain registers by storing them in the hardware stack.)

4. Jump to POLCAT

This jumps to POLCAT using its indirect address:

```
WAIT JSR [POLCAT]
```

5. Use Exit Conditions

For now, assume you want to look only at the status of Bit Z to see if a key has been pressed:

```
BEQ WAIT
```

The above instruction branches back to WAIT (the JSR [POLCAT] instruction) unless you press a key. (Pressing a key causes POLCAT to clear Bit Z.)

6. Restore the Register's Contents

This "pulls" (inserts) the contents of the hardware stack back into the registers:

```
PULS DP,CC,X,Y,U
```

Now, the above registers are restored to the data they contained before executing the POLCAT routine.

Sample 2 Character Output with CHROUT

The CHROUT routine prints a character on either the screen or printer. On entry, it checks two places:

- Register A — to determine which character to print
- Address \$6F — to determine whether to print it on the screen or the printer

This program uses CHROUT to print "This is a Message" on the screen. It then uses POLCAT to wait for you to press a key before returning to BASIC.

```
ORG $1200
***** Equates for Routines *****
POLCAT EQU $A000
CHROUT EQU $A002
DEVNUM EQU $6F
***** Variable *****
SCREEN EQU 00
*** DOS Programming Convention ***
BEGIN JMP START
      FDB DONE-BEGIN
***** Print the Message *****
START LDB #SCREEN
      STB DEVNUM
      LDX #MSG
PRINT LDA ,X+
      JSR [CHROUT]
      CMPA #$0D
      BNE PRINT
***** Wait for a Key *****
INPUT PSHS DP,CC,X,Y,U
WAIT JSR [POLCAT]
      BEQ WAIT
      PULS DP,CC,X,Y,U
      CLR $71
      JMP [$FFFE]
***** Message *****
MSG FCC 'THIS IS A MESSAGE'
      FCB $0D
***** Memory for Stack *****
DONE *
      END
```

Most of the steps we used in writing this program are obvious. What may not be obvious is the way we set up CHROUT's entry conditions, Address \$6F and Register A.

These lines set Address \$6F to 00 (the screen):

```
DEVNUM EQU $6F
SCREEN EQU 00
START LDB #SCREEN
      STB DEVNUM
```

Setting Register A involves two steps. First, point Register X to the message:

```
MSG      FCC    'THIS IS A MESSAGE'
        FCB    $0D
        LDX    #MSG
```

and then load Register A with each character in the message:

```
PRINT   LDA    ,X+
        JSR    [CHROUT]
        CMPA   #$0D
        BNE    PRINT
```

Sample 3 POLCAT and CHROUT

This combines POLCAT with CHROUT. It prints on the screen whatever key you press. When you press  (hexadecimal 0A), the program returns to BASIC:

```
ORG    $1200
***** Equates for Routines *****
POLCAT  EQU    $A000
CHROUT  EQU    $A002
DEVNUM  EQU    $6F
```

```
***** Variable *****
SCREEN  EQU    00
*** DOS Programming Convention ***
BEGIN   JMP    MAIN
        FDB    DONE-BEGIN
***** Main Program *****
MAIN    JSR    INPUT
        CMPA   #$0A
        BEQ    FINISH
        JSR    PRINT
        BRA    MAIN
FINISH  CLR    $71
        JMP    [$FFFE]
* Input a Character from Keyboard *
INPUT   PSHS   DP,CC,X,Y,U
WAIT    JSR    [POLCAT]
        BEQ    WAIT
        PULS   DP,CC,X,Y,U
        RTS
** Print a Character on Display **
PRINT   LDB    #SCREEN
        STB    DEVNUM
        JSR    [CHROUT]
        RTS
***** Memory for Stack *****
DONE    *
        END
```


Chapter 14/ Opening and Closing a Disk File DOS Routines — Part I

Because of the organization and timing of a disk, reading it and writing to it are complex. This is why you'll want to make use of DOS routines in your disk programs.

This chapter shows how to use DOS routines to open and close a disk file. The next chapter shows how to use them to read a disk and write to it. *Reference H* contains a complete list of all the DOS routines supported by Radio Shack.

Overview

All DOS routines, like ROM routines, have their own entry and exit conditions. However, most DOS routines have more involved entry conditions than do ROM routines. They require you to set up three areas in memory: two "buffers" and a "data control block."

Buffers

Buffers are areas in memory that DOS uses for storing data to be input or output to disk. DOS requires that you reserve two buffers:

- A logical buffer — This can be any length. Your program uses this to store data for DOS to input or output to disk.
- A physical buffer — This must be 256 bytes. DOS uses this to hold data temporarily so that it can input and output the data to a disk sector in 256-byte blocks.

For example, suppose you want to output 100 10-byte records to disk. You can send each record, one at a time, to the area you reserved as the logical buffer.

DOS then transfers the records from the logical buffer to the area you reserved as the physical buffer. As soon as

there are 256 bytes in the physical buffer, DOS sends them out to a disk sector.

You need not be concerned that DOS' "physical" records are a different size from your program's "logical" records. DOS handles the "spanning" of logical records into physical records internally. Except for reserving memory for a physical buffer, you do not need to be concerned with physical records.

Data Control Block

A data control block is a 49-byte "block" of memory that DOS uses to control a disk file. You need to reserve this block of memory for each disk file you are using. If you have three disk files open at the same time, you need to reserve three 49-byte data control blocks.

Reference G shows how DOS uses each of the 49 bytes, numbered 0-48, in the data control block. As you can see, DOS divides the data control block into 21 data-control segments.

Before opening a file, you must load the proper data into four of the segments of the data control block (DCB):

DCB Segment	DCB Address	You must load with . . .
Filename (DCBFNM)	Bytes 0-7	The eight-character name of your file.
Extension (DCBEXT)	Bytes 8-10	The three character extension of your filename.
Drive Number (DCBDRV)	Byte 33	The drive containing the disk file.

Physical Buffer Address (DCBBUF)	Byte 36-37	The first address of the physical buffer you have reserved.
----------------------------------	------------	---

For example, if you want to open a file in Drive 1, you need to load "1" into the DCBDRV location, which is the 33rd byte of the data control block.

You need not be concerned with most of the remaining segments of the data control block, unless you want to use them as data in your program. They are handled internally by DOS. The exceptions to this are:

- Logical Buffer Address, Record Size, Variable Record Terminator, and Logical Record Number — You need to use these when you read and write to the file. They are discussed in the next chapter.
- *File Type and ASCII Flag* — If you want your file to be compatible with BASIC and other Radio Shack programs, you need to set these when you create the file. See the "Technical Information" chapter of your *Disk System Owners Manual and Programming Guide*.

Steps for Using DOS Routines

The steps for using DOS routines are:

1. Equate the routine's address (for ease in reading the program).
2. Reserve memory for a physical buffer, logical buffer, and the DCB.
3. Clear the DCB and the physical buffer. You need to make sure they do not have extraneous data.
4. Set up all other entry conditions. Besides setting up registers, you need to load certain segments of the DCB with data. Which segments you load depends on the DOS routine you are using.
5. Preserve the contents of the registers. DOS routines change the contents of many of the registers. To be safe, you should preserve all of them that you want to use later in your program. Be sure to preserve Registers U and DP. If DOS changes their contents, your program acts unpredictably.
6. Call the routine.
7. Restore the contents of the registers.

8. Use all exit conditions. Most DOS routines return an error code in Register A if the routine did not work properly. If there were no errors, Register A contains a zero.

Sample Session Opening and Closing a Disk File

The DOS routines for opening and closing a file are OPEN and CLOSE. Both routines check Register U for the address of DCB. They expect to find the four segments described above in this block.

OPEN also expects you to set a file mode in Register A. It creates or opens an existing file depending on the mode you set.

Both routines return a status code in Register A. *Reference I* tells the meaning of the status codes.

Figure 8 at the end of this chapter is a sample program which creates, opens, and closes a disk file named WORKFILE/TXT. After running this program, you can look at your directory to see that the program has created this file. This shows how we applied the above steps in this program.

1. Equate OPEN and CLOSE

This equates OPEN and CLOSE to \$600 and \$602, their indirect addresses:

OPEN	EQU	\$600
CLOSE	EQU	\$602

2. Reserve Memory for Buffers and DCB

The OPEN and CLOSE routines use only the physical buffer, not the logical buffer. This stores 256 bytes for the physical buffer and uses PBUF to label those bytes:

PBUF	RMB	256
------	-----	-----

This reserves memory for a 49-byte DCB and stores the filename, WORKFILE, and the extension, TXT, in the first 11 bytes:

DCB	EQU	*
FCC		'WORKFILE'
FCC		'TXT'
RMB		38

3. Clear DCB

This clears all but the first 11 bytes of DCB:

```
RCLEAR    LDX      #DCB+11
CLEAR1    CLR      ,X+
            CMPX    #DCB+48
            BNE     CLEAR1
            LDX      #PBUF
```

and this clears the physical buffer:

```
CLEAR2    CLR      ,X+
            CMPX    #PBUF+255
            BNE     CLEAR2
            RTS
```

4. Set Up Entry Conditions

On entry, OPEN and CLOSE require you to: (1) Set Register U to a DCB containing a filename, extension, drive number, and physical buffer address, and (2) Set Register A to a file mode.

Setting Register U

This sets Register U to the address of the first byte of the DCB:

```
LDU      #DCB
```

The following lines set the drive number segment to 0. They do this by storing DRVNUM (0) into DCBDRV (33) + the contents of Register U (DCB). This inserts 0 into the 33rd byte of DCB:

```
DCBDRV    EQU      33
DRVNUM    FCB      00
LDA      DRVNUM
STA      DCBDRV ,U
```

The following lines set the physical buffer address to PBUF. They do this by storing the address of PBUF into the memory address pointed to by Register U plus DCBBUF. This stores PBUF in the 36th byte of DCB:

```
DCBBUF    EQU      36
LDX      #PBUF
STX      DCBBUF ,U
```

(The filename and extension were set in Step 2.)

Setting Register A

This table shows how you should set each bit in Register

A to select one or more file modes:

MODE	BIT	DECIMAL NUMBER (IF SET)
Read	Bit 0	1
Write	Bit 1	2
Create	Bit 2	4
Extend	Bit 3	8
Work File	Bit 4	16
	(delete the file, when closed)	
FAT	Bit 5	32
	(rewrite to the FAT* only when closed)	
Shared Buffer	Bit 6	64

* The disk directory's FAT (file allocation table) is described in the "Technical Information" chapter of the *Disk System Manual*.

The sample program loads Register A with decimal $1+2+4+8+32$:

```
LDA      #1+2+4+8+32
```

This tells DOS to set the file mode to read (decimal 1), write (decimal 2), create (decimal 4), extend (decimal 8), and rewrite the FAT only when the file is closed (decimal 32).

5. Preserve Registers

This preserves the contents of Registers U and DP:

```
ROPEN    PSHS    U ,DP
```

6. Jump to the DOS Routine

These lines jump to OPEN and CLOSE:

```
JSR      [OPEN]
JSR      [CLOSE]
```

7. Restore Registers

This restores the contents of Registers U and DP:

```
PULS    U ,DP
```

8. Use Exit Conditions

The sample program branches to an error handling subroutine after each DOS routine. The subroutine tests Register A to see if it contains a non-zero value. If so, it

prints the status code on the screen and waits for you to press a key:

```

JSR      ERROR
TSTA
BEQ      RETURN
STA      $450
WAIT    JSR      [POLCAT]
        BEQ      WAIT
        RETURN   RTS
        ORG      $1200
**Equates for DOS and ROM routines **
OPEN     EQU      $600
CLOSE    EQU      $602
POLCAT   EQU      $A000
***** Equates for DCB offsets *****
DCBDRV   EQU      33
DCBBUF   EQU      36
*****DOS Programming Convention *****
BEGIN    JMP      MAIN
        FDB      DONE-BEGIN
*****Main Program *****
MAIN    JSR      RCLEAR
        JSR      ROPEN
        JSR      RCLOSE
        CLR      $71
        JMP      [$FFFE]
*****Routine to Clear the DCB *****
***** and Physical Buffer*****
RCLEAR   LDX      #DCB+11
CLEAR1   CLR      ,X+
        CMPX    #DCB+4B
        BNE     CLEAR1
        LDX      #PBUF
CLEAR2   CLR      ,X+
        CMPX    #PBUF+255
        BNE     CLEAR2
        RTS
*****Routine to Open a File *****
ROPE    PSHS   U,DP
        LDU    #DCB
        LDA    DRVNUM
        STA    DCBDRV,U
        LDX    #PBUF
        STX    DCBBUF,U
        LDA    #1+2+4+8+32
        JSR    [OPEN]
        PULS   U,DP
        JSR    ERROR
        RTS
***** Routine to Close the File *****
RCLOSE   PSHS   U,DP
        LDU    #DCB
        JSR    [CLOSE]
        PULS   U,DP
        JSR    ERROR
        RTS
*****Error Handling Routine *****
ERROR   TSTA
        BEQ    RETURN
        STA    $450
        WAIT   JSR      [POLCAT]
        BEQ    WAIT
        RETURN   RTS
*** Memory for Buffers and Stacks ***
PBUF    RMB     256
*****Memory for Variables *****
DRVNUM   FCB     00
*****Memory for DCB *****
DCB     EQU      *
        FCC     'WORKFILE'
        FCC     'TXT'
        RMB     38
***** *****
DONE    EQU      *
END

```

Chapter 15/ Reading and Writing a Disk File DOS Routines — Part 2

DOS has a WRITE routine for writing to a file and a READ routine for reading it back into memory. The way you use these routines depends on which method you are using to access the file:

- Sequential Access
- Direct Access

This chapter describes how to use these two methods in their simplest forms. You can use any variation of them that you want.

Sequential vs. Direct Access

Sequential Access (For Files with Variable-Length Records)

Sequential access lets you read and write to files with variable-length records. Using this method, you insert a terminator character at the end of each record. This character tells DOS where each record ends.

Before writing data to the file, you must load DCB with the following:

DCB Segment	DCB Address	You must load with...
Logical Buffer Address (DCBRLRB)	Bytes 39-40	The first address of the logical buffer you have reserved
Terminator Character (DCBTRM)	Byte 19	The character you select to end each record

When reading data from just one file, you need only specify the logical buffer address, not the terminator character. DOS reads the terminator character from the disk's directory into DCBTRM.

Figure 9 at the end of this chapter is a program that writes to a file using \$0D (the **ENTER** character) as a terminator character. *Figure 10* reads the same file back into memory.

Direct Access (For Files with Fixed-Length Records)

Direct access works only with files containing fixed-length records. With this method, DOS uses the record size and record number to access the record.

Before reading data from the file or writing data to it, you must set this DCB segment:

DCB Segment	DCB Address	You must load with . . .
Logical Buffer Address (DCBRLRB)	Bytes 39-40	The address of the first byte of the logical buffer you have reserved
Logical Record Size (DCBRSZ)	Bytes 17-18	The size of each record

Unless you are using the record size already in the file's directory, you must also set:

If you want to write a record which is not sequentially the next one, you must also set:

Logical Record Number (DCBLRN)	Bytes 46-47	The number of the record you want to access
-----------------------------------	-------------	---

Setting the Read/Write Option

DOS requires that you set Register A with a "read/write option" before entering the READ or WRITE routines. The read/write option lets you specify:

- Whether you want direct or sequential access
- Whether you want DOS to point to the next record after reading or writing the record

To set the read/write option, load the first two bits of Register A with one of these four values:

Read/Write Option	Bits	Decimal Number
Direct Access Point to next record	00	0
Sequential Access Point to next record	01	1
Direct Access Do not point to next record	10	2
Sequential Access Do not point to next record	11	3

For example:

```
LDA      #2
JSR      [READ]
```

tells DOS to write the record sequentially (up to the terminator character). When finished, DOS points to the next sequential record.

Figure 9. Sample Program to Write to a File

```
ORG      $1200
**Equates for DOS and ROM routines **
OPEN     EQU      $600
CLOSE    EQU      $602
WRITE    EQU      $606
POLCAT   EQU      $A000
```

```
***** Equates for DCB offsets *****
DCBTRM   EQU      19
DCBDRV   EQU      33
DCBBUF   EQU      36
DCBLRB   EQU      39
*****DOS Programming Convention *****
BEGIN    JMP      MAIN
FDB      DONE-BEGIN
*****Main Program *****
MAIN    JSR      CLEAR
        JSR      INTDCB
        JSR      SOPEN
        JSR      SPRINT
        JSR      SWRITE
        JSR      SCLOSE
        CLR      $71
        JMP      [$FFFE]

*****Routine to Clear the DCB *****
and the Physical and Logical Buffers
CLEAR   LDX      #PBUF
CLEAR1  CLR      ,X+
        CMPX   #PBUF+255
        BNE    CLEAR1
        LDX      #LBUF
CLEAR2  CLR      ,X+
        CMPX   #LBUF+24
        BNE    CLEAR2
        LDX      #DCB+11
CLEAR3  CLR      ,X+
        CMPX   #DCB+48
        BNE    CLEAR3
        RTS

***** Routine to Insert *****
Values in the DCB *****
INTDCB  LDU      #DCB
        LDA      DRVNUM
        STA      DCBDRV,U
        LDA      #$0D
        STA      DCBTRM,U
        LDX      #PBUF
        STX      DCBBUF,U
        LDX      #LBUF
        STX      DCBLRB,U
        RTS

*****Routine to Open a File *****
SOPEN   LDU      #DCB
        PSHS   U,DP
        LDA      #1+2+4+8+32
        JSR      [OPEN]
        PULS   U,DP
        JSR      ERROR
        RTS

*****Routine to Print Msg *****

```

```

SPRINT LDY     ##$500
        LDX     #MSG
CHAR   LDA     ,X+
        STA     ,Y+
        CMPA   #$3A
        BNE     CHAR
        LDX     #LBUF
        LDY     ##$525

***** Routine to Input Data *****
***** from Keyboard *****
SINPUT PSHS   U,DP,Y
WAIT1  JSR    [POLCAT]
        BEQ    WAIT1
        PULS   U,DP,Y
        STA    ,Y+
        STA    ,X+
        CMPA   ##$0D
        BEQ    ENDINP
        CMPX   #LBUF+24
        BNE    SINPUT
ENDINP RTS

***** Routine to Write Data *****
***** to File *****
SWRITE PSHS   U,DP
        LDU    #DCB
        LDA    #1
        JSR    [WRITE]
        PULS   U,DP
        JSR    ERROR
        RTS

***** Routine to Close File *****
SCLOSE PSHS   U,DP
        LDU    #DCB
        JSR    [CLOSE]
        PULS   U,DP
        JSR    ERROR
        RTS

***** Error Handling Routine *****
ERROR  TSTA
        BEQ    RETURN
        STA    $450
WAIT2  JSR    [POLCAT]
        BEQ    WAIT2
RETURN RTS

*** Memory for Buffers and Stacks ***
PBUF   RMB    256
LBUF   RMB    25
***** Memory for Variables *****
DRVNUM FCB    00
***** Memory for DCB *****
DCB    EQU    *
        FCC    'WORKFILE'
        FCC    'TXT'
        RMB    38

```

```

*****Memory for Message *****
MSG    FCC    'ENTER YOUR NAME:'
***** *****
DONE   EQU    *
        END

```

Figure 10. Sample Program to Read to a File

Note: When running this program, a status code (generated by the Error subroutine) may appear on your screen. Press any key to continue program execution.

```

        ORG    $1200
**Equates for DOS and ROM routines **
OPEN   EQU    $600
CLOSE  EQU    $602
READ   EQU    $604
POLCAT EQU    $A000
CHROUT EQU    $A002
***** Equates for DCB offsets *****
DEVNUM EQU    $6F
SCREEN EQU    0
DCBTRM EQU    19
DCBDRV EQU    33
DCBBUF EQU    36
DCBLRB EQU    39
*****DOS Programming Convention *****
BEGIN  JMP    MAIN
        FDB    DONE-BEGIN
*****Main Program *****
MAIN   JSR    CLEAR
        JSR    INTDCB
        JSR    SOPEN
        JSR    SREAD
        JSR    SCLOSE
        JSR    SPRINT
        CLR    $71
        JMP    [$FEEE]

*****Routine to Clear the DCB *****
and the Physical and Logical Buffers
CLEAR  LDX    *PBUF
CLEAR1 CLR    ,X+
        CMPX   *PBUF+255
        BNE    CLEAR1
        LDX    *LBUF
CLEAR2 CLR    ,X+
        CMPX   *LBUF+24
        BNE    CLEAR2
        LDX    *DCB+11
CLEAR3 CLR    ,X+
        CMPX   *DCB+48
        BNE    CLEAR3
        RTS

```

```
***** Routine to Insert *****
***** Values in the DCB *****
INTDCB LDU #DCB
LDA DRVNUM
STA DCBDRV ,U
LDA #$0D
STA DCBTRM ,U
LDX #PBUF
STX DCBBUF ,U
LDX #LBUF
STX DCBLRB ,U
RTS

***** Routine to Open a File *****
SOPEN PSHS U,DP
LDU #DCB
LDA #$2F
JSR [OPEN]
PULS U,DP
JSR ERROR
RTS

***** Routine to Read a File *****
SREAD PSHS U,DP
LDU #DCB
LDA #3
JSR [READ]
PULS U,DP
JSR ERROR
RTS

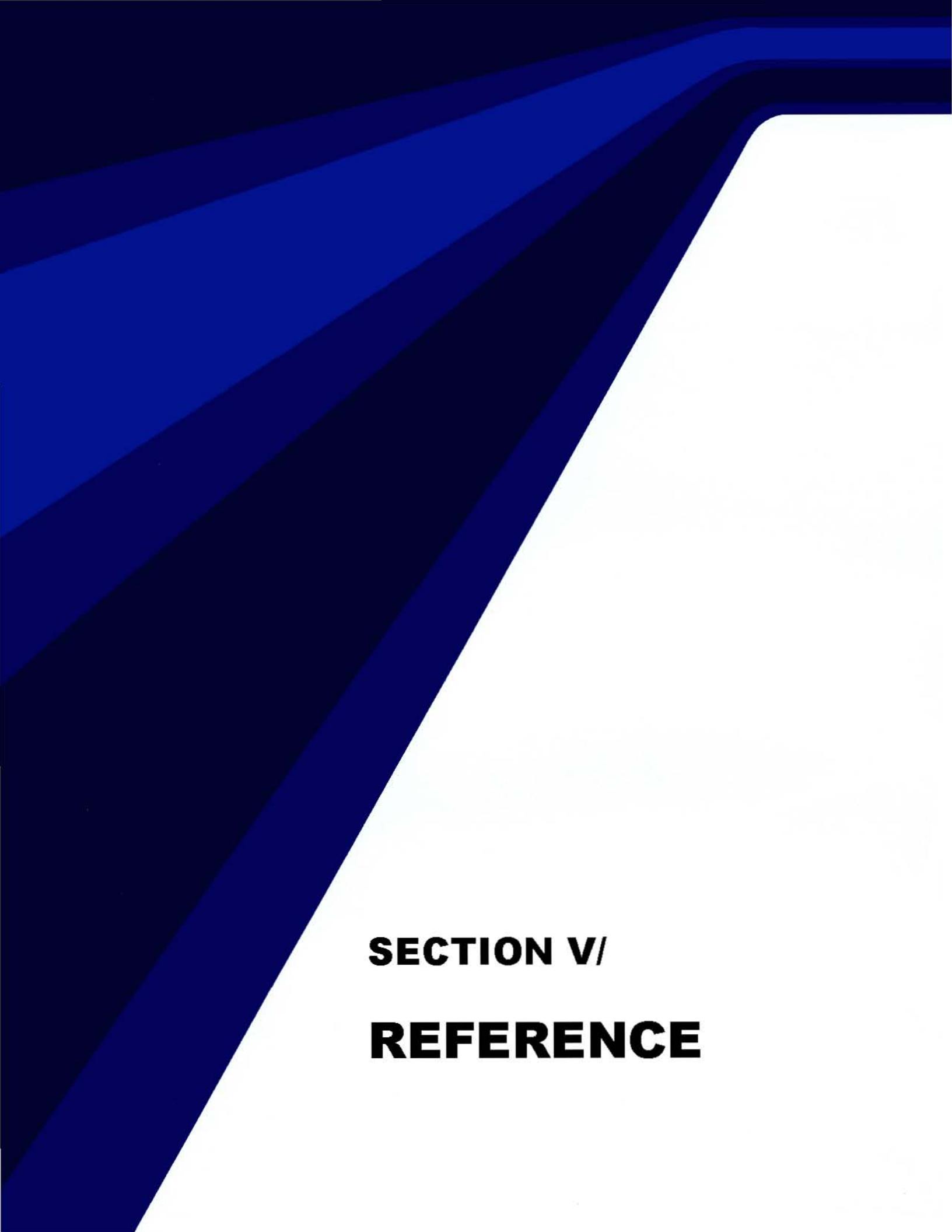
***** Routine to Print Data *****
SPRINT LDB #SCREEN
STB DEVNUM
LDX #LBUF
PRINT LDA ,X+
```

```
JSR [CHROUT]
CMPX #LBUF+24
BNE PRINT
WAIT1 JSR [POLCAT]
BEQ WAIT1
RTS

***** Routine to Close File *****
SCLOSE PSHS U,DP
LDU #DCB
JSR [CLOSE]
PULS U,DP
JSR ERROR
RTS

***** Error Handling Routine *****
ERROR TSTA
BEQ RETURN
STA $450
WAIT2 JSR [POLCAT]
BEQ WAIT2
RETURN RTS

*** Memory for Buffers and Stacks ***
PBUF RMB 256
LBUF RMB 25
***** Memory for Variables *****
DRVNUM FCB 00
***** Memory for DCB *****
DCB EQU *
FCC 'WORKFILE'
FCC 'TXT'
RMB 38
*****
DONE EQU *
END
```



**SECTION V/
REFERENCE**

SECTION V/**REFERENCE**

This section summarizes all the features of the Disk EDTASM.

Reference A/ Editor Commands

Definition of Terms

line

A *line* number in the program. Any *lines* between 0-63999 may be used. These symbols may be used:

- # First line in the program
- * Last line in the program
- . Current line in the program

current line

The last line inserted, edited, or printed.

startline

The line where an operation will begin. In most commands *startline* is optional. If *startline* is omitted, the current line is used.

An asterisk (*) denotes a comment line when used as the first character in the line.

range

The line or lines to use in an operation. If the *range* includes more than one line, they must be specified with one of these symbols:

- : to separate the *startline* from the ending line
- , to separate the *startline* from the number of lines

increment

The *increment* to use between lines. In most commands, *increment* is optional. If the *increment* is omitted, the last specified *increment* is used. On startup, *increment* is set to 10.

filespec

A DOS disk file specification in the format:

filename/ext:drive

COMMANDS	PAGES DISCUSSED

C_{startline}, range, increment

Copies range to a new location beginning with *startline* using the specified *increments*. *startline*, *range*, and *increment* must be included.

C500,100:150,10

D_{range}

Deletes *range*. If *range* is omitted, *current line* is deleted.

D100 D100:150 D

E_{line}

Enters a *line* for editing. If *line* is omitted, *current line* is used.

E100 E

These are the editing subcommands:

A

Cancels all changes and restarts the edit.

nC_{string}

Changes *n* characters to *string*. If *n* is omitted, changes the character at the current cursor position.

nD

Deletes *n* characters. If *n* is omitted, deletes character at current cursor position.

E

Ends line editing and enters all changes without displaying the rest of the line.

H

Deletes rest of line and allows insert.

I *string*

Inserts *string* starting at the current cursor position. While in the mode, deletes a character, and ends the mode.

K

Deletes all characters from the current cursor position to the end of the line.

L

Lists current line and continues edit.

nS_{character}

Searches for *n*th occurrence of *character*. If *n* is omitted, searches for the first occurrence.

X

Extends line.

Ends line editing, enters all changes and displays the rest of the line.

Escapes from subcommand.

n

Moves cursor *n* positions to the right. If *n* is omitted, moves one position.

n

Moves cursor *n* positions to the left. If *n* is omitted, moves the cursor one position.

F_{string}

Finds the string of characters. Search begins with the *current line* and ends each time *string* is found. If *string* is omitted, the last string defined is used.

FABC F

H_{range}

Prints *range* on the printer. If *range* is omitted, the *current line* is printed.

H100 H100:200 H

I_{startline,increment}

Inserts lines up to 127 characters long beginning at *startline*, using the specified *increment*. *startline* and *increment* are optional.

I150,5 1200 I,10

K

Returns to DOS.

LCA *filename*

Loads *filename* from tape into the edit buffer. A is optional. If included, *filename* is appended to the edit buffer. If *filename* is omitted, the next tape file is loaded.

LC SAMPLE/EXT

LCA SAMPLE/EXT

LDA *filespec*

Loads the specified file from disk into the edit buffer. A is optional. If included, *filespec* is appended to the current contents of the edit buffer. If extension is omitted, /ASM is used.

LD SAMPLE/EXT

LDA SAMPLE/EXT

M*startline, range, increment*

Move command, works like copy except the original lines are deleted.

N*startline, increment*

Renumbers beginning at *startline*, using the specified *increment*. *startline* and *increment* are optional.

N100,50 N100 N

O

Shows the hexadecimal values of (1) the first available memory address, (2) the last available address, and (3) USRORG, the address where the assembler originates an /IM assembly with the /MO switch. Then, prompts you to change USRORG.

O

P*range*

Displays *range* on the screen.

```
P100:200    P100!5    P*    P+
P (Prints 15 lines to the screen)
```

Q

Returns to BASIC.

R *startline, increment*

Allows you to replace *startline* and then insert lines using *increment*. *startline* and *increment* are optional.

R100,10 R100 R

S

Shows the current printer parameters and lets you change them.

T*range*

Prints *range* to the printer, without line numbers.

T100 T100:500

V*filename*

Verifies *filename* (a tape file) to ensure that it is free of checksum errors. Works like BASIC's SKIPF command. If *filename* is omitted, this command verifies the next file found.

WC *filename*

Writes *filename* to tape. If *filename* is omitted, NONAME is used.

WD *filespec*

Writes *filespec* to disk. If the extension is omitted, ASM is used.

WD SAMPLE/EXT

Z

Jumps to ZBUG (EDTASM system only).



Scrolls up in memory.



Scrolls down in memory.

(SHIFT) (CLEAR)

Is used to create a backslash (\).

Reference B/ Assembler Commands and Switches

COMMANDS	PAGES DISCUSSED
----------	--------------------

AC *filename* *switch* . . .

Assembles the source program into machine code. If you specify the /IM switch, the assembly is in memory. If you specify *filename*, the assembly is saved on tape as *filename*. If you omit both *filename* and *switch*, the assembly is saved on tape as NONAME.

AD *filespec* *switch* . . .

Assembles the source program into machine code. Either the /IM switch or *filespec* is required: With /IM, the assembly is in memory; with *filespec*, the assembly is on disk. The D is optional.

There must be a space between *filespec* and *switch*.

The switches are:

/AO	Absolute origin.(Applies only If /IM is set.)
/IM	In-memory assembly.
/LP	Assembly listing on the printer.
/MO	Manual origin. (Applies only if /IM is set.)
/NL	No listing printed.
/NO	No object code generated.
/NS	No symbol table generated.
/SR	Single record.
/SS	Short screen.
/WE	Wait on assembly errors.
/WS	With symbols.

Examples:

```
AD SAMPLE
AD/IM/AO
AD SAMPLE /WE/SR
A SAMPLE/TST /WE
AC SAMPLE
AC
```


Reference C/ ZBUG Commands

Definition of Terms

expression

One or more numbers, symbols, or ASCII characters. If more than one is used, you may separate them with these operators:

Multiplication	*	Addition	+
Division	.DIV	Subtraction	-
Modulus	.MOD	Equals	.EQU
Shift	<	Not Equal	.NEG
Local And	.AND	Positive	+
Exclusive Or	.XOR	Negative	-
Logical Or	.OR	Complement	.NOT

address

A location in memory. This may be specified as an expression using numbers or symbols.

filename

A BASIC cassette file specification.

filespec

A DOS file specification. (The same as a BASIC specification.)

COMMANDS	PAGES DISCUSSED

C

Continues execution of the program after interruption at a breakpoint.

D

Displays all breakpoints that have been set.

E

Exits ZBUG and enters the editor. (This applies to the EDTASM ZBUG only, not to Stand-Alone ZBUG.)

Gaddress

Executes the program beginning at **address**.

K

Returns to DOS. (Applies to Stand-Alone ZBUG only.)

LC filename address

Loads *filename* from tape. The optional *address* offsets the file's loading address. If *filename* is omitted, the next file is loaded.

LD filespec address

Loads *filespec* from disk. The optional *address* offsets the file's loading address.

LDS filespec address1 address2

Loads *filespec* from disk with its appended symbol table. The optional *address1* offsets the file's loading address. The optional *address2* offsets the symbol table's loading address. Note that *address2* does not offset the values of the symbols. The D is optional.

PC filename start address end address execution address

Saves memory from *start address* to *end address* to tape. You must also specify an *execution address*, the first address to be executed when the file is loaded. *Filename* is optional; if omitted, NONAME is used.

PD filespec start address end address execution address

Saves memory to disk from *start address* to *end address*. You must also specify an *execution address*, the first address to be executed when the file is loaded. (The D is optional.)

PDS filespec start address end address execution address

Saves memory to disk from *start address* to *end address*, with the current appended symbol table. You must also specify an *execution address*, the first address to be executed when the file is loaded. (The D is optional.)

Q

Returns to BASIC. (Applies to Stand-Alone ZBUG only.)

R

Displays the contents of all the registers.

Taddress1 address2

Displays the memory locations from *address1* to *address2*, inclusive.

THaddress1 address2

Prints the memory locations from *address1* to *address2*, inclusive.

Usource address destination address count

Transfers the contents of memory beginning at *source address* and continuing for *count* bytes to another location in memory beginning with *destination address*.

Vfilename

Verifies date on the specified file or, if no *filename* is specified, the next file on tape.

Xaddress

Sets a breakpoint at *address*. If *address* is omitted, the current location is used. Each breakpoint is assigned a number from 0 to 7. The first breakpoint set is assigned as Breakpoint 0. A maximum of eight breakpoints may be set at one time.

Yn

Deletes the breakpoint referenced by the *n* number. If *n* is omitted, all breakpoints are deleted.

Examination Mode Commands

A	ASCII Mode
B	Byte Mode
M	Mnemonic Mode
W	Word Mode
(The default is M)	

Display Mode Commands

H	Half Symbolic
N	Numeric
S	Symbolic
(The default is S)	

Numbering System Mode Commands

Obase	Output
Ibase	Input
(Base can be 8, 10, or 16. The default is 16)	

Special Symbols

address/
register/

Opens *address* or *register* and displays its contents.

If *address* or *register* is omitted, the last address opened will be reopened. After the contents have been displayed, you may type:

new value	To change the contents.
ENTER	To close and enter any change.
BREAK	To close and delete any change.
	To open next <i>address</i> and enter any change.
	To open preceding <i>address</i> .
address ↵	To branch to the <i>address</i> pointed to by the instruction beginning at <i>address</i> . If <i>address</i> is omitted, the current <i>address</i> is used.
;	To force numeric display mode.
=	To force numeric and byte modes.
:	To force flags.*
"	To force ASCII mode.

***address*,**

Executes *address*, if *address* is omitted, the next instruction is executed.

***expression* =**

Calculates *expression* and displays the results.

* The colon does not actually have anything to do with the CC (status flag) register. It simply interprets the contents of the given address AS IF it contained flag bits.

Reference D/ EDTASM Error Messages

These are error messages you can get while in EDTASM or EDTASMOV:

BAD BREAKPOINT (ZBUG)

You are attempting to set a breakpoint (1) greater than 7, (2) in ROM, (3) at a SWI command, (4) at an address where one is already set.

BAD COMMAND (Editor)

An illegal command letter was used on the command line.

BAD COMMAND (ZBUG)

You are not using a ZBUG command.

BAD FILE DESCRIPTOR (Disk,ZBug)

The filespec is not in the proper DOS format. See "About This Manual" at the beginning of this manual for the proper file specification format.

BAD LABEL (Assembler)

The symbol you are using is (1) not a legal symbol, (2) not terminated with either a space, a tab, or a carriage return, (3) has been used with ORG or END, which do not allow labels, or (4) longer than six characters.

BAD MEMORY (Assembler)

You are attempting to do an in-memory assembly that would (1) overwrite system memory (an address lower than \$1200) (2) overwrite the edit buffer of the symbol table, (3) go into the protected area set by USROG, or (4) go over the top of RAM.

If using the /AO switch, check to see that you've included an ORG instruction. When using /MO, check the addresses you set for BEGTEMP and USRORG. This could also be caused by the data not being stored correctly because of some code generated by an in-memory assembly. See Chapter 7 for more information.

BAD MEMORY (ZBUG)

The data did not store correctly on a memory modification. This error will occur if you try to modify ROM addresses or try to store anything beyond MAXMEM.

BAD OPCODE (Assembler)

The op code is either not valid or is not terminated with a space, tab, or carriage return.

BAD OPERAND (Assembler)

There is some syntax error in the operand field. See Section III for the syntax of assembly language instructions.

BAD PARAMETERS (Editor,ZBug)

Usually this means your command line has a syntax error.

BAD PARAMETERS (ZBUG)

You have specified a filename that has more than eight characters.

BAD RADIX (ZBUG)

You have specified a numbering system other than 10, 8 or 16.

BUFFER EMPTY (Editor)

The specified command requires that there be some text in the Edit Buffer, and there isn't any.

BUFFER FULL (Editor)

There is not enough room in the edit buffer for another line of text.

BYTE OVERFLOW (Assembler)

There is a field overflow in an 8-bit data quantity in an immediate operand, an offset, a short branch, or an FCB pseudo op.

DIRECTORY FULL (Disk)

The directory does not have enough room for another entry. Use another diskette or delete a file (using the BASIC KILL command).

DISK FULL (Disk)

The diskette does not have enough room for another file. Use another diskette or delete a file (using the BASIC KILL command).

DISK WRITE PROTECTED (*Disk*)

You are attempting to write to a diskette that has the write-protect notch covered. Remove the write-protect label or use another diskette.

DOS ERROR (*Disk*)

This indicates an internal DOS error. It usually means either the DOS or the Editor/Assembler has been modified by the user program with harmful results.

DP ERROR (*Assembler*)

Direct Page error. The high order byte of an operand where direct addressing has been forced (.) does not match the value set by the most recent SETDP pseudo op.

DRIVE NOT READY (*Disk*)

The drive is not connected, powered up, working properly, or loaded properly.

END OF FILE (*Disk*)

Your program is attempting to access a record past the end of the file.

ENDC WITHOUT COND (*Assembler*)

The pseudo op ENDC was found without a matching COND having previously been encountered.

ENDM WITHOUT MACRO (*Assembler*)

The pseudo op ENDM was found without a matching MACRO having previously been encountered.

EXPRESSION ERROR (*Assembler and ZBUG*)

Either the syntax for the expression is incorrect (check Chapter 9) or the expression is dividing by zero.

FILE NOT FOUND (*Disk*)

The file is not on the disk's directory.

FM ERROR (*Editor, ZBUG and Disk*)

File Mode Error. The file you are attempting to load is not a TEXT file (if in the Editor) or a CODE file (if in ZBUG).

ILLEGAL NESTING (*Assembler*)

Illegal nesting conditions include the following:

1. Nested macro definitions.
2. Nested macro expansions.
3. Nested INCLUDE pseudo ops.
4. INCLUDE nested within a macro definition.

I/O ERROR (*Editor, ZBUG and Disk*)

Input/Output error. A checksum error was encountered

while loading a file from a cassette tape. The tape may be bad, or the volume setting may be wrong. Try a higher volume.

MACRO FORWARD REFERENCE

(*Assembler*)

A reference to the macro, which is defined on the current line, occurs previous to the macro definition.

MACRO TABLE FULL (*Assembler*)

The macro table is full, any additional entries will overwrite the symbol table. This happens when all memory allocated for the edit buffer, macro table, and symbol table has been used. Adjust USRORG using the Origin (O) command. (See the Chapter 7.)

MISSING END (*Assembler*)

Every assembly language program must have END as its last command.

MISSING INFORMATION (*Assembler*)

(1) There is a missing delimiter in an FCC pseudo op or
(2) there is no label on a SET or EQU pseudo op.

MISSING OPERAND (*Assembler,ZBug*)

The command requires one or more operands.

MULTIPLY DEFINED SYMBOL (*Assembler*)

Your program has defined the same symbol with different values. If the error occurs in a macro expansion, use the ./ notation to name the symbols. See Chapter 12.

NO ROOM BETWEEN LINES (*Editor*)

There is not enough room between lines to use the increment specified. Specify a smaller increment or renumber (N) the text using a larger increment. Remember that the last increment you used is kept until you specify a new one.

NO SUCH LINES (*Editor*)

The specified line or lines do not exist.

REGISTER ERROR (*Assembler*)

(1) No registers have been specified with a PSH/PUL instruction, (2) a register has been specified more than once in a PSH/PUL instruction, or (3) there is a register mismatch with an EXG/TFR instruction.

SEARCH FAILS (*Editor*)

The string specified in the Find (F) command could not be found in the edit buffer beginning with the line specified. If no line is specified the current line is used.

SYMBOL TABLE OVERFLOW (*Assembler*)

The symbol table is extending past USRORG into the protected area of user memory. Adjust USRORG using the O command. See *Chapter 7*.

SYNTAX ERROR (*Assembler*)

There is a syntax error in a macro dummy argument.

UNDEFINED SYMBOL (*Assembler,ZBug*)

Your program has not defined the symbol being used.

Reference E/ Assembler Pseudo Ops

Definition of Terms

symbol

Any *string* from one to six characters long, typed in the symbol field.

expression

Any *expression* typed in the operand field. See Reference C, "ZBUG commands," for a definition of valid expressions.

COMMANDS	PAGES DISCUSSED
----------	--------------------

COND expression

Assembles the instructions between COND and ENDC only if *expression* is true (a non-zero value).

SYMBOL	COND	SYMBOL
	FCB	10
VALUE	FCB	5
	COND	SYMBOL - VALUE

Valid operators for a conditional expression are +, -, /, *. If the expression equals zero, it is false; if non-zero, it is true.

END expression

Ends the assembly. The optional *expression* specifies the start address of the program.

ENDC

Ends a conditional assembly.

ENDM

Ends a macro definition.

symbol EQU expression

Equates *symbol* to an *expression*.

SYMBOL	EQU	\$5000
--------	-----	--------

symbol FCB expression, . . .

Stores a 1-byte expression beginning at the current address.

```
DATA2      FCB      $33+COUNT
```

symbol FCC delimiter string delimiter

Stores *string* in memory beginning with the current address. The *delimiter* can be any character.

```
TABLE      FCC      /THIS IS A STRING/
```

symbol FDB expression

Stores a 2-byte expression in memory begining at the current address.

```
DATA      FDB      $3322
```

INCLUDE source filespec

Includes source *filespec* in the current position of the source program.

```
INCLUDE      SAMPLE/ASM
```

symbol MACRO

Defines the instructions between MACRO and ENDM as a macro named *symbol*.

```
DIVIDE      MACRO
```

OPT switch, . . .

Uses *switch* to control the listing of macros when assembling the program. The switches are:

MC	List macro calls (default)
NOMC	Do not list macro calls
MD	List macro definitions (default)
NOMD	Do not list macro definitions
MEX	List macro expansionns
NOMELEX	Do not list macro expansions (default)
L	Turn on the listing (default)
NOL	Turn off the listing

ORG expression

Originates the program at *expression* address.

```
ORG      $3F00
```

PAGE

Ejects the assembly listing to the next page.

RMB expression

Reserves *expression* bytes of memory for data.

```
DATA      RMB      $06
```

symbol SET expression

Sets or resets *symbol* to *expression*.

```
SYMBOL      SET      $3500
```

SETDP expression

Sets the direct page to *expression*.

SETDP \$20

TITLE string

Prints *string* as the title of each page of the assembly listing. *String* can be up to 32 characters.

TITLE Program 1

Reference F/ Rom Routines

This reference lists the indirect addresses where the Color Computer's ROM routines are stored. It also shows the entry and exit conditions for each routine.

The name of the routine is for documentation only. To jump to the routine, you must use its indirect address (the address contained in the brackets).

COMMANDS	PAGES DISCUSSED
----------	--------------------

BLKIN =[\$A006]

Reads a block from a cassette.

Entry Conditions:

Cassette must be on and in bit sync (see CSRDON).
CBUFAD contains the buffer address.

Exit Conditions:

BLKTYP, located at \$7C, contains the block type:

0 = file header
1 = data
FF = end of file

BLKLEN, located at \$7D, contains the number of data bytes in the block (0-255):

Bit Z in the Register CC, Register A, and CSRERR, located at Address \$81, contains the error:

Z=1, A=CSRERR=0 (if no errors)
Z=0, A=CSRERR=1 (if a checksum error occurs)
Z=0, A=CSRERR=2 (if a memory error occurs)

BLKOUT =[\$A008]

Writes a block to cassette.

Entry Conditions:

If this is the first block write after turning the motor on, the tape should be up to speed and a \$55s should be written first.

CBUFAD, located at \$7E, contains the buffer address.

BLKTYP, located at \$7C, contains the block type.

BLKLEN, located at \$7D, contains the number of bytes.

Exit Conditions:

Interrupts are masked.

X = CBUFAD + BLKLEN.

All registers are modified.

CHROUT = [A002]

Outputs a character to a device.

Entry Conditions:

Register A = character to be output

Address 6F (DEVMNUM) = the device (-2 = printer; 0 = screen)

Exit Conditions:

Register CC is changed; all others are preserved.

CSRDON = [\$A004]

Starts the cassette and gets into bit sync for reading.

Entry Conditions:

None

Exit Conditions:

FIRQ and IRO are masked.

Registers U and Y are preserved. All others are modified.

JOYIN = [\$A00A]

Samples the four joystick pots and stores their values in POTVAL through POTVAL+3.

Left Joystick:

Up/Down 15A

Right/Left 15B

Right Joystick:

Up/Down 15C

Right/Left 15D

For Up/Down, the minimum value equals Up.

For Right/Left, the minimum value equals Left.

POLCAT = [A000]

Polls the keyboard for a character.

Entry Conditions:

None

Exit Conditions:

If no key is seen — Flag Z = 1, Register A = 0

If a key is seen — Flag Z = 0, Register A = key code

Registers B and X are preserved.

All other registers are modified.

Reference G/ DOS Disk Data Control Block (DCB)

DOS uses a 49-byte DCB to access a disk file. This reference shows the contents of each of the bytes (Bytes 0-48) in the DCB.

Bytes 0-31

The first 32 bytes of the DCB correspond to the disk file's 32-byte directory entry. When creating a file, DOS writes the DCB's first 32 bytes to the directory.

When opening an existing file, DOS searches each directory entry for the filename and extension you have set in the DCB. If it finds a match, it overwrites the first 32 bytes of the DCB with the 32-byte directory entry.

When you close the file, DOS overwrites the directory entry with the first 32 bytes of the DCB.

Filename (DCBFNM) **Bytes 0-7**
Contains the name of the file you want to access. You must set this value.

Extension (DCBFNM) **Bytes 8-10**
Contains the extension of the file you want to access. You must set this value.

File Type (DCBFYT) **Byte 11**
Contains the type of file you want to access. DOS ignores this, but BASIC uses it. You need to set this value when creating the file if you want the file compatible with BASIC.

ASCII Flag (DCBASC) **Byte 12**
Contains a flag if the file is in ASCII format. DOS ignores this, but BASIC uses it. You need to set this value when creating the file if you want the file compatible with BASIC.

First Cluster (DCBFCL) **Byte 13**
Contains the number of the first cluster in the file. (When you first create a file, this contains \$FF.) DOS sets this value..Do not change it.

First Sector Bytes (DCBNLS) **Bytes 14-15**
Contains the number of bytes used in the first sector of the file. DOS ignores this. However, to be compatible with BASIC files, you should set this value before closing an output file.

File Mode (DCBCFS) **Byte 16**
Contains the mode you specified with Register A in the OPEN, WRITE, or READ routine. DOS sets this value.

Record Size (DCBRSZ)**Bytes 17-18**

Contains the size of each record. Use this with fixed-length records only. You set this value before reading from or writing to a direct access file.

Record Terminator (DCBTRM)**Byte 19**

Contains the character that DOS uses to terminate each record. You supply this value when reading from or writing to a sequential access file.

Undefined (DCBUSR)**Bytes 20-31**

Contains nothing at present. In future releases, DOS may use part of this.

Bytes 32 – 48

Bytes 32-48 are primarily set by DOS. However, you may use the contents of these bytes as data in your program.

The exceptions to this are the bytes for the drive number, physical buffer address, and logical buffer address. You must set the contents of these bytes before opening a file.

Operation Code (DCBOPC)**Byte 32**

Contains the last physical I/O operation performed on the file. See your Disk System Manual for details. DOS sets this value.

Drive Number (DCBDRV)**Byte 33**

Contains the drive number (0-3 or \$FF). \$FF tells DOS to use the first available drive and then insert the drive number in this segment. You must set this value before opening a file.

Track Number (DCBTRK)**Byte 34**

Contains the number of the last track DOS accessed while doing I/O for this file. DOS sets this value.

Sector Number (DCBSEC)**Byte 35**

Contains the number of the last sector DOS accessed while doing I/O for this file. DOS sets this value.

Physical Buffer Address (DCBBUF)**Bytes 36-37**

Contains the start address of a 256-byte physical buffer. The physical buffer is for storing data before or after disk I/O. You must set this value before opening a file.

Error Code (DCBOK)**Byte 38**

Contains the same value that the DOS routine returns in Register A: a zero if the last DOS routine was successful; the error number if there was an error. DOS sets this value.

Logical Buffer Address (DCBLRN)**Bytes 39-40**

Contains the start address of a logical buffer. The logical buffer is for storing a logical record before or after it goes through the physical buffer. You must set this value before opening a file, unless you have specified the "share" file mode. (See OPEN.)

Physical Record Number (DCBPRN).**Bytes 41-42**

Contains the number of the physical record currently in the physical buffer. DOS uses this to determine whether another physical read or write is required. This contains \$FFFF when the file is opened. It also contains \$FFFF after every read or write when the buffer is "shared." DOS sets this value.

Relative Byte Address (DCBRBA)**Bytes 43-45**

Contains an address which points to the record you want to read or write (zero when the file is first opened). With sequential access, this address always points to the next record. With direct access, this address is the product of DCBRSZ times DCBPRN. DOS sets and updates this value.

Logical Record Number (DCBLRN).**Bytes 46-47**

Contains the number of the next record to be accessed (zero when the file is first opened). Unless you set this value, DOS increments it after accessing each record.

Modified Data Tag (DCBMDT)**Byte 48**

Contains a tag ("1") if the contents of the physical buffer need to be written to disk. DOS sets this tag each time it writes to the logical buffer. The contents of the physical buffer are written to disk only when DOS must access a different sector (because the 256-byte buffer is full) or close the file. If the physical buffer is "shared," the physical buffer is written to disk after each logical write. DOS sets and updates this value.

Reference H/ DOS Routines

This reference lists all the DOS routines that Radio Shack will continue to provide in future releases. Please note that Radio Shack will support only the OPEN, CLOSE, READ, and WRITE routines. The other routines listed in this reference will be provided, but not necessarily supported.

Definition of Terms

root program

The portion of the program that is not an overlay. If you are not using overlays, this is the entire program.

overlay

A portion of the program that DOS loads into memory only when called. This can be your own overlay (called with DOUSR, GOUSR, or LOUSR) or a DOS overlay (called with DO, GO, or LOAD).

DOS programming convention

A convention, which any program using DOS routines must follow:

- The execution address must be the first instruction in the program.
- The first three bytes of the program must contain a JMP or LBR to any part of the root program. (JMP and LBR are both 3-byte instructions.) Example:

START JMP BEGIN

- The next two bytes must contain the length of the root program. If you are not using overlays, this is the entire program. Example:

FDB DONE-START

- If you are using overlays, this is the root program. Example:

FDB DONE-OVY1

DOS overlay conventions

A convention, which any of your own overlays must follow:

- The first two bytes must contain the size of the overlay. Example:

OVY1 FDB Ovy2-OVY1

- The next three bytes must contain a JMP or LBRA to any part of the overlay. Example:

JSR PROV1

- The last instruction should be an RTS, GO, or GOUSR.

- You must assign the overlay a number that is sequential. For example, assign your first overlay the overlay number of 1:

OVY EQU 1

- The overlay must be written with relocatable (rather than absolute) addresses. When DOS loads the overlay, it sets Register X equal to the overlay's base address. Therefore, you can refer to all the local variables as an offset to Register X.

COMMANDS	PAGES DISCUSSED
----------	--------------------

CLOSE = [\$602]

Closes access to a disk file.

Entry Conditions:

Register U = the address of the DCB that was previously opened.
Program must follow DOS programming convention.

Exit Conditions

Register A = status code

Technical Function of CLOSE:

- Checks the drive specified by DCBDRV for a directory entry matching DCBFNM and DCBFEX. When the entry is found, checks to see if the file was previously open by seeing if DCBCFS contains a non-zero value.
- Checks DCBMDT for a modification tag. If found, writes the contents of the physical buffer to the disk.
- Sets DCBCFS to zero.
- Rewrites the directory entry with the first 32 bytes of the DCB. Any changes in the first 32 bytes of the DCB after OPEN and before CLOSE are recorded in the directory.
- Rewrites the diskette's FAT.

DO = [\$60A]

Calls a DOS overlay.

Entry Conditions:

Register A = DOS overlay number

Exit Conditions:

Register A = status code

DOUSR = [\$0610]

Calls one of your own overlays.

Entry Conditions:

Register A = overlay number (the number you have assigned to the overlay)

Exit Conditions:

Register A = status code

GO = [\$60C]

Calls one DOS overlay from another DOS overlay.

Entry Conditions:

Register A = DOS overlay number

Exit Conditions:

Register A = status code

GOUSR = [\$612]

Calls one overlay from another overlay. For example, OVY1 calls OVY2.

Entry Conditions:

Register A = overlay number (the number you have assigned to the overlay)

Exit Conditions:

Register A = "0" if no error; error code if error

LOAD = [\$60E]

Loads a DOS overlay but does not execute it.

Entry Conditions:

Register A = DOS overlay number

Exit Conditions:

Register A = "0" if no error; error code if error

LODUSR = [\$614]

Loads one of your overlays but does not execute it.

Entry Conditions:

Register A = overlay number (the number you have assigned to the overlay)

Exit Conditions:

Register A = "0" if no error; error code if error

OPEN = [\$600]

Opens access to a disk file using the specified file mode.

Entry Conditions:

Register A = file mode

The file modes are:

Bit 0 set — allows reads

Bit 1 set — allows writes

Bit 2 set — allows file creation

Bit 3 set — allows extension past end of file

Bit 4 set — deletes the file when closed (work file)

Bit 5 set — rewrites the directory's file allocation table (FAT) only when the file is closed. (Otherwise, rewrites FAT after each READ; see the *Disk System Manual* for information on the FAT.)

Bit 6 set — shares the physical and logical buffer

Bit 7 set — undefined

Register U = the address where the DCB is stored.

The DCB must contain values for DCBFNM, DCBFEX, DCBDRV, and DCBBUF

Program must follow DOS programming conventions.

Exit Conditions:

Register A = 0 if no error; error code if error

Technical Function of OPEN:

- Checks the drive specified by DCBDRV for a directory entry matching DCBFNM and DCBFEX.
- If a match is found:
 - Uses the directory entry to overwrite the first 32 bytes of the DCB
 - Checks DCBCFS. It indicates a write, create, or extend, the file is opened and Status Code L is returned.
 - Inserts the file mode (contained in Register A) in DCBCFS.
 - Overwrites the directory entry with the first 32 bytes of the DCB.
- If a match is not found and the file mode is "create," creates a directory entry using the first 32 bytes of the DCB

- Sets DCBPRN to \$FFFF
- Clears DCBLRN, DCBMDT, and DCBRBA.

READ = [\$604]

Reads a record from a disk file.

Entry Conditions:

Register A = read option

The read options are:

Bit 0 clear — direct access (read by record number; fixed length records)

Bit 0 set — sequential access (read by terminator character; variable length records)

Bit 1 clear — exit READ pointing to next record

Bit 1 set — exit READ leaving DCBLRN and DCBRBA the same (not pointing to next record)

The other bits can contain any value.

Register U = address pointing to the DCB

Program must follow DOS programming convention

Exit Conditions:

Register A = 0 if no error; error number if error logical buffer (pointed to by DCBLRB) contains the record

Technical Function of READ:

- Checks DCBCFS to see if the file was opened for "read."
- Checks DCBRBA for the record you want to access. (If Bit 0 in Register A is clear, READ calculates DCBRBA as the product of DCBLRN times DCBRSZ).
- Checks to see if the record is in the physical buffer (by comparing the high two bytes of DCBRBA with the contents of DCBPRN).
If the record is not in the physical buffer, READ reads the record into the physical buffer then transfers it to the logical buffer.
- Checks to see if Register A's Bit 1 is set. If so, restore DCBLRN and DCBRBA to their original values.

RELEASE = [\$608]

Frees a physical buffer so that you can use it with another file.

Entry Conditions:

Register U = address where the DCB is stored of the file currently using the physical buffer.

Register A = 0 if no error; error code if error.

Technical Function of RELEASE:

- Check DCBMDT. If the tag is set, the contents of the physical buffer are written to disk and DCBMDT is cleared.
- Sets DCBPRN to \$FFFF.

WRITE = [\$606]

Writes a logical record to disk.

Entry Conditions:

Register A = read/write option

The read/write options are:

Bit 0 clear — direct access (write by record number; fixed length records)

Bit 0 set — sequential access (write by terminator character; variable length records)

Bit 1 clear — exit READ pointing to next record

Bit 1 set — exit READ leaving DCBLRN and DCBRBA the same (not pointing to next record)

The other bits can contain any value.

Register U = address pointing to the DCB logical buffer (pointed to by DCBLRB) contains the record you want to write

Program must follow DOS programming conventions.

Exit Conditions:

Register A = 0 if no error; status code if error

Technical Function of WRITE:

- Checks DCBCFS to see if the file was opened for "write."
- Checks DCBRBA for the record you want to access. (If Bit 0 in Register A is off, WRITE calculates DCBRBA as the product of DCBLRN times DCBRSZ).
- Transfers the contents of the logical buffer to the physical buffer. If all 256 bytes of the physical buffer are full, writes the contents of the physical buffer to disk. If there is still more contents in the logical buffer, WRITE transfer these contents to the physical buffer and sets DCBMDT to 1.
- If the file mode is "share," writes the complete contents of the physical buffer to disk regardless of whether it completely fills the sector. Then, sets DCBPRN to \$FFFF.

Reference I/ DOS Error Codes

Error Code	Hex Code	Character Displayed	Error
00	40	@	No errors
01	41	A	I/O error (drive not ready)
02	42	B	I/O error (write-protected diskette)
03	43	C	I/O error (write fault)
04	44	D	I/O error (seek error or record not found)
05	45	E	I/O error (CER error)
06	46	F	I/O error (lost data)
07	47	G	I/O error (undefined Bit 1)
08	48	H	I/O error (undefined Bit 0)
09	49	I	Register argument is invalid
0A	4A	J	File directory entry not found
0B	4B	K	Full directory
0C	4C	L	File was created by the OPEN function
0D	4D	M	File not closed after changes
0E	4E	N	Attempt to access an opened file
0F	4F	O	Attempt to read a read-protected file
10	50	P	RBA overflow (exceeds 3 bytes -16,777,216)
11	51	Q	Access beyond EOF or extension not allowed
12	52	R	FAT rewrite error
13	53	S	Attempt to close an unopened file
14	54	T	Can't access directly (record size is 0)
15	55	U	Attempt to write on write-protected diskette
16	56	V	Can't extend file (disk capacity exceeded)
17	57	W	Error while loading overlay
18	58	X	Insufficient print space allocated
19	59	Y	I/O error during BASIC line read
1A	5A	Z	Program's load address is too low
1B	5B	[First byte of program file is not equal to zero
1C	5C	\	Not enough space for buffered keyboard
1D	5D]	Not enough memory
1E	5E	^	Output file already exists
1F	5F	-	Wrong diskette

Reference J/ Memory Map

\$0 - \$69	Direct page RAM
\$70-\$FF	System direct page RAM
\$100-\$111	Interrupt vectors
\$112-\$119	System RAM
\$11A	Keyboard alpha lock flag
\$11B-\$159	System RAM
\$15A-\$15D	Joystick pot values
\$15E-\$3FF	System RAM
\$400-\$5FF	Video memory
\$600-\$11FF	DOS
\$1200-\$3FFF	16K user memory
\$1200-\$7FFF	32K user memory
\$8000-\$9FFF	Extended BASIC
\$A000-\$BFFF	BASIC
\$C000-\$DFFF	Disk BASIC
\$E000-\$FEFF	ROM expansion
\$FF00-\$FFEE	Hardware address
\$FFF0-\$FFFF	Interrupt vectors

Reference K/ ASCII Codes

Video Control Codes

Dec	Hex	PRINT CHR\$(code)
8	08	Backspaces and erases current character.
13	0D	Line feed with carriage return.
32	20	Space

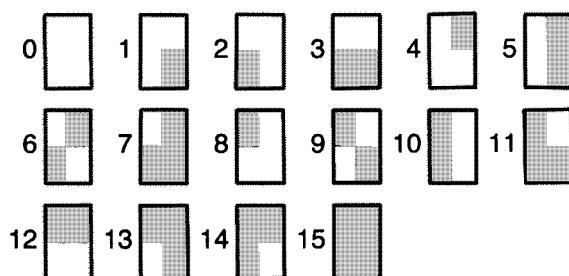
Color Codes

CODE	COLOR
0	Black
1	Green
2	Yellow
3	Blue
4	Red
5	Buff
6	Cyan
7	Magenta
8	Orange

Graphic Character Codes

Given the *color* (1-8) and the *pattern* (0-15), this formula will generate the correct code:

$$\text{code} = 128 + 16 * (\text{color} - 1) + \text{pattern}$$



For example, to print *pattern* 9 in blue (*code* 3), type:

```
C = 128 + 16 * (3-1) + 9
? CHR$ (C)
```

Alphanumeric Character Codes

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
SPACEBAR	32	20
!	33	21
"	34	22
#	35	23
\$	36	24
%	37	25
&	38	26
,	39	27
(40	28
)	41	29
*	42	2A
+	43	2B
,	44	2C
-	45	2D
.	46	2E
/	47	2F
0	48	30
1	49	31
2	50	32
3	51	33
4	52	34
5	53	35
6	54	36
7	55	37
8	56	38
9	57	39
.	58	3A
.	59	3B
.	60	3C
.	61	3D
<	62	3E
>	63	3F
=	64	40
?	65	41
@	66	42
A	67	43
B	68	44
C	69	45
D	70	46
E	71	47
F	72	48
G	73	49
H	74	4A
I	75	4B
J	76	4C
K	77	4D
L	78	4E
M	79	4F
N	80	50
O	81	51
P	82	52
Q	83	53

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
T	84	54
U	85	55
V	86	56
W	87	57
X	88	58
Y	89	59
Z	90	5A
↑ *	94	5E
↓ *	10	0A
← *	8	08
→ *	9	09
BREAK	03	03
CLEAR	12	0C
ENTER	13	0D

*If shifted, the code for these characters are as follows:
(CLEAR) is 92 (hex 5C); **↑** is 95 (hex 5F); **↓** is 91 (hex 5B); **←** is 21 (hex 15); and **→** is 93 (hex 5D).

These are the ASCII codes for lowercase letters. You can produce these characters by pressing **SHIFT** **0** simultaneously to get into an upper-lowercase mode. The lowercase letters will appear on your screen in reversed colors (green with a black background).

CHARACTER	DECIMAL CODE	HEXADECIMAL CODE
a	97	61
b	98	62
c	99	63
d	100	64
e	101	65
f	102	66
g	103	67
h	104	68
i	105	69
j	106	6A
k	107	6B
l	108	6C
m	109	6D
n	110	6E
o	111	6F
p	112	70
q	113	71
r	114	72
s	115	73
t	116	74
u	117	75
v	118	76
w	119	77
x	120	78
y	121	79
z	122	7A

Reference L/ 6809 Mnemonics

Definition of Terms

Source Forms:

This shows all the possible variations you can use with the instruction. *Table 4* gives the meaning of all the notations we use. The notations in italics represent values you can supply.

For example, the BEQ instruction has two source forms. BEQ *dd* allows you to use these instructions:

BEQ \$08 BEQ \$FF BEQ \$A0

Whereas LBEQ DDDD allows you these:

LBEQ \$C000 LBEQ \$FFFF

Operation:

This uses shorthand notation to show exactly what the instruction does, step by step. The meaning of all the codes are also in *Table 4*.

For example, the BEQ operation does this:

"If, (but only if), the zero flag is set, branch to the location indicated by the program counter plus the value of the 8-bit offset."

Condition Codes:

This shows which of the flags in the CC register are affected by the instruction, if any. As you'll note, BEQ does not set or clear any of the flags.

Description:

This is an overall description, in English, of what the instruction does.

Addressing Mode:

This tells you which addressing modes you may use with the instruction. BEQ allows only the Relative addressing mode.

ABBREVIATION	MEANING	ABBREVIATION	MEANING
ACCA or A	Accumulator A.	Us or U	User stack pointer.
ACCB or B	Accumulator B.	P	A memory location with immediate, direct, extended, and indexed addressing modes.
ACCA:ACCB or D	Accumulator D.	Q	A read-write-modify argument with direct, extended and indexed addressing modes.
ACCX	Either accumulator A or accumulator B.	()	The data pointed to by the enclosed (16 bit address).
CCR or CC	Condition code register.	dd	8-bit branch offset.
DPR or DP	Direct page register.	DDDD	16-bit offset.
EA	Effective address.	#	Immediate value follows.
IFF	If and only if.	\$	Hexadecimal value follows.
IX or X	Index register X.	[]	Indirection.
IY or Y	Index register Y.	.	Indicates indexed addressing.
LSN	Least significant nibble.	←	Is transferred to.
M	Memory location.	/	Boolean AND.
MI	Memory immediate.	V	Boolean OR.
MSN	Most significant nibble.	O	Boolean Exclusive OR (XOR).
PC	Program counter.	—	Boolean NOT.
R	A register before the operation.	:	Concatination.
R'	A register after the operation.	+	Arithmetic plus.
TEMP	A temporary storage location.	-	Arithmetic minus.
xxH	Most significant byte of any location.	×	Arithmetic multiply.
xxL	Least significant byte of any location.		
Sp or S	Hardware stack pointer.		

Table 4. Notations and Codes

Add Accumulator B into Index Register X

Source Form: ABX
Operation: $IX' \leftarrow IX + ACCB$

Condition Codes: Not affected.
Description: Add the 8-bit unsigned value in accumulator B into index register X.
Addressing Mode: Inherent.

ABX

Add with Carry into Register

Source Forms: ADCA P; ADCB P
Operation: $R' \leftarrow R + M + C$

Condition Codes:
H — Set if a half-carry is generated; cleared otherwise.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the contents of the C (carry) bit and the memory byte into an 8-bit accumulator.
Addressing Modes: Immediate; Extended; Direct; Indexed.

ADC

Add Memory into Register

Source Forms: ADDA P; ADDB P
Operation: $R' \leftarrow R + M$

Condition Codes:
H — Set if a half-carry is generated; cleared otherwise.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the memory byte into an 8-bit accumulator.
Addressing Modes: Immediate; Extended; Direct; Indexed.

ADD
(8-Bit)

Add Memory into Register

Source Form: ADDD P
Operation: $R' \leftarrow R + M : M + 1$

Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.
C — Set if a carry is generated; cleared otherwise.
Description: Adds the 16-bit memory value into the 16-bit accumulator.
Addressing Modes: Immediate; Extended; Direct; Indexed.

ADD
(16-Bit)

Logical AND Memory into Register

Source Forms: ANDA P; ANDB P
Operation: $R' \leftarrow R \wedge M$

Condition Codes:
H — Not affected.
N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.
Description: Performs the logical AND operation between the contents of an accumulator and the contents of memory location M and the result is stored in the accumulator.
Addressing Modes: Immediate; Extended; Direct; Indexed.

AND

Logical AND Immediate Memory into Condition Code Register

Source Form: ANDCC #xx
Operation: $R' \leftarrow R \wedge MI$

Condition Codes: Affected according to the operation.

Description: Performs a logical AND between the condition code register and the immediate byte specified in the instruction and places the result in the condition code register.
Addressing Mode: Immediate.

AND

Arithmetic Shift Left

Source Forms: ASL Q; ASLA; ASLB

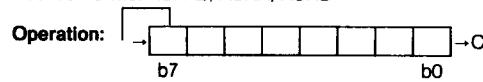
Operation: $C \leftarrow \boxed{} \quad \leftarrow 0$
b7 ← b0

Condition Codes:
H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.

V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.
C — Loaded with bit seven of the original operand.
Description: Shifts all bits of the operand one place to the left. Bit zero is loaded with a zero. Bit seven is shifted into the C (carry) bit.
Addressing Modes: Inherent; Extended; Direct; Indexed.

ASL

ASR

Arithmetic Shift Right**Source Forms:** ASR Q; ASRA; ASRB**Condition Codes:**

H — Undefined.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Not affected.

C — Loaded with bit zero of the original operand.

Description: Shifts all bits of the operand one place to the right. Bit seven is held constant. Bit zero is shifted into the C (carry) bit.**Addressing Modes:** Inherent; Extended; Direct; Indexed.

BCC

Branch on Carry Clear**Source Forms:** BCC dd; LBCC DDDD**Operation:**

TEMP~MI

IFF C = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Tests the state of the C (carry) bit and causes a branch if it is clear.**Addressing Mode:** Relative.**Comments:** Equivalent to BHS dd; LBHS DDDD.

BCS

Branch on Carry Set**Source Forms:** BCS dd; LBCS DDDD**Operation:**

TEMP~MI

IFF C = 1 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Tests the state of the C (carry) bit and causes a branch if it is set.**Addressing Mode:** Relative.**Comments:** Equivalent to BLO dd; LBLO DDDD.

BEQ

Branch on Equal**Source Forms:** BEQ dd; LBEQ DDDD**Operation:**

TEMP~MI

IFF Z = 1 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Tests the state of the Z (zero) bit and causes a branch if it is set. When used after a subtract or compare operation, this instruction will branch if the compared values, signed or unsigned, were exactly the same.**Addressing Mode:** Relative.

BGE

Branch on Greater than or Equal to Zero**Source Forms:** BGE dd; LBGE DDDD**Operation:**

TEMP~MI

IFF (N ⊕ V) = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Causes a branch if the N (negative) bit and the V (overflow) bit are either both set or both clear. That is, branch if the sign of a valid two's complement result is, or would be, positive. When used after a subtract or compare operation on two's complement values, this instruction will branch if the register was greater than or equal to the memory operand.**Addressing Mode:** Relative.

BGT

Branch on Greater**Source Forms:** BGT dd; LBGT DDDD**Operation:**

TEMP~MI

IFF Z ∧ (N ⊕ V) = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Causes a branch if the N (negative) bit and V (overflow) bit are either both set or both clear and the

Z (zero) bit is clear. In other words, branch if the sign of a valid two's complement result is, or would be, positive and not zero. When used after a subtract or compare operation on two's complement values, this instruction will branch if the register was greater than the memory operand.

Addressing Mode: Relative.

BHI

Branch if Higher**Source Forms:** BHI dd; LBHI DDDD**Operation:**

TEMP~MI

IFF (C v Z) = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Causes a branch if the previous operation caused neither a carry nor a zero result. When used after a

subtract or compare operation on unsigned binary values, this instruction will branch if the register was higher than the memory operand.

Addressing Mode: Relative.**Comments:** Generally not useful after INC/DEC, LD/TST, and TST/CLR/COM instructions.

BHS

Branch if Higher or Same**Source Forms:** BHS dd; LBHS DDDD**Operation:**

TEMP \leftarrow MI
IFF C=0 then PC' \leftarrow PC + MI

Condition Codes: Not affected.**Description:** Tests the state of the C (carry) bit and causes a branch if it is clear. When used after a subtract or compare

on unsigned binary values, this instruction will branch if the register was higher than or the same as the memory operand.

Addressing Mode: Relative.

Comments: This is a duplicate assembly-language mnemonic for the single machine instruction BCC. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.

BIT

Bit Test**Source Form:** BIT P**Operation:** TEMP \leftarrow R \wedge M**Condition Codes:**

H — Not affected.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.

V — Always cleared.
C — Not affected.

Description: Performs the logical AND of the contents of accumulator A or B and the contents of memory location M and modifies the condition codes accordingly. The contents of accumulator A or B and memory location M are not affected.

Addressing Modes: Immediate; Extended; Direct; Indexed.

BLE

Branch on Less than or Equal to Zero**Source Forms:** BLE dd; LBLE DDDD**Operation:**

TEMP \leftarrow MI
IFF Z \vee (N \oplus V) = 1 then PC' \leftarrow PC + TEMP

Condition Codes: Not affected.

Description: Causes a branch if the exclusive OR of the N (negative) and V (overflow) bits is 1 or if the Z (zero) bit is set. That is, branch if the sign of a valid two's complement result is, or would be, negative. When used after a subtract or compare operation on two's complement values, this instruction will branch if the register was less than or equal to the memory operand.

Addressing Mode: Relative.**Branch on Lower****Source Forms:** BLO dd; LBLO DDDD**Operation:**

TEMP \leftarrow MI
IFF C = 1 then PC' \leftarrow PC + TEMP

Condition Codes: Not affected.**Description:** Tests the state of the C (carry) bit and causes a

branch if it is set. When used after a subtract or compare on unsigned binary values, this instruction will branch if the register was lower than the memory operand.

Addressing Mode: Relative.

Comments: This is a duplicate assembly-language mnemonic for the single machine instruction BCS. Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.

BLO

BLS

Branch on Lower or Same**Source Forms:** BLS dd; LBLS DDDD**Operation:**

TEMP \leftarrow MI
IFF (C \vee Z) = 1 then PC' \leftarrow PC + TEMP

Condition Codes: Not affected.**Description:** Causes a branch if the previous operation

caused either a carry or a zero result. When used after a subtract or compare operation on unsigned binary values, this instruction will branch if the register was lower than or the same as the memory operand.

Addressing Mode: Relative.

Comments: Generally not useful after INC/DEC, LD/ST, and TST/CLR/COM instructions.

BLT

Branch on Less than Zero**Source Forms:** BLT dd; LBLT DDDD**Operation:**

TEMP \leftarrow MI
IFF (N \oplus V) = 1 then PC' \leftarrow PC + TEMP

Condition Codes: Not affected.**Description:** Causes a branch if either, but not both, of the

N (negative) or V (overflow) bits is set. That is, branch if the sign of a valid two's complement result is, or would be, negative. When used after a subtract or compare operation on two's complement binary values, this instruction will branch if the register was less than the memory operand.

Addressing Mode: Relative.**Branch on Minus****Source Forms:** BMI dd; LBMI DDDD**Operation:**

TEMP \leftarrow MI
IFF N = 1 then PC' \leftarrow PC + TEMP

Condition Codes: Not affected.**Description:** Tests the state of the N (negative) bit and

causes a branch if set. That is, branch if the sign of the two's complement result is negative.

Addressing Mode: Relative.

Comments: When used after an operation on signed binary values, this instruction will branch if the result is minus. It is generally preferred to use the LBLS instruction after signed operations.

BMI

BNE

Branch Not Equal**Source Forms:** BNE dd; LBNE DDDD**Operation:**

TEMP ← MI

IFF Z = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Tests the state of the Z (zero) bit and causes a branch if it is clear. When used after a subtract or compare operation on any binary values, this instruction will branch if the register is, or would be, not equal to the memory operand.**Addressing Mode:** Relative.

BPL

Branch on Plus**Source Forms:** BPL dd; LBPL DDDD**Operation:**

TEMP ← MI

IFF N = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Tests the state of the N (negative) bit and

causes a branch if it is clear. That is, branch if the sign of the two's complement result is positive.

Addressing Mode: Relative.**Comments:** When used after an operation on signed binary values, this instruction will branch if the result (possibly invalid) is positive. It is generally preferred to use the BGE instruction after signed operations.

BRA

Branch Always**Source Forms:** BRA dd; LBRA DDDD**Operation:**

TEMP ← MI

PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Causes an unconditional branch.**Addressing Mode:** Relative.

BRN

Branch Never**Source Forms:** BRN dd; LBRN DDDD**Operation:** TEMP ← MI**Condition Codes:** Not affected.**Description:** Does not cause a branch. This instruction is essentially a no operation, but has a bit pattern logically related to branch always.**Addressing Mode:** Relative.

BSR

Branch to Subroutine**Source Forms:** BSR dd; LBSR DDDD**Operation:**

TEMP ← MI

SP' ← SP - 1, (SP) ← PCL

SP' ← SP - 1, (SP) ← PCH

PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** The program counter is pushed onto the stack. The program counter is then loaded with the sum of the program counter and the offset.**Addressing Mode:** Relative.**Comments:** A return from subroutine (RTS) instruction is used to reverse this process and must be the last instruction executed in a subroutine.

BVC

Branch on Overflow Clear**Source Forms:** BVC dd; LBVC DDDD**Operation:**

TEMP ← MI

IFF V = 0 then PC' ← PC + TEMP

Condition Codes: Not affected.**Description:** Tests the state of the V (overflow) bit and causes a branch if it is clear. That is, branch if the two's complement result was valid. When used after an operation on two's complement binary values, this instruction will branch if there was no overflow.**Addressing Mode:** Relative.

BVS

BVS Branch on Overflow set**Source Forms:** BVS dd; LBVS DDDD**Operation:** Temp ← MI IFF V = 1 then PC' ← PC + TEMP**Condition Codes:** Not affected.**Description:** Tests the state of V (overflow) bit and causes a branch if it is set. That is, branch if two's complement result was invalid. When used after an operation on two's complement binary values, this instruction will branch if there was an overflow.**Addressing Mode:** Relative.

CLR

CLR Clear**Source Forms:** CLR Q**Operation:** TEMP ← M M ← 00 (base 16)**Condition codes:**

H — Not affected.

N — Always cleared.

Z — Always set.

V — Always cleared.

C — Always cleared.

Description: Accumulator A or B or memory location M is loaded with 00000000. Note that the EA is read during this operation.**Addressing Modes:** Inherent, Extended, Direct, Indexed.

Compare Memory from Register

Source Forms: CMPA P; CMPB P

Operation: TEMP \leftarrow R-M

Condition Codes:

- H — Undefined.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description: Compares the contents of memory location to the contents of the specified register and sets the appropriate condition codes. Neither memory location M nor the specified register is modified. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes: Immediate; Extended; Direct; Indexed.

CMP
(8-Bit)

Compare Memory from Register

Source Forms: CMPD P; CMPX P; CMPY P; CMPU P;
CMPS P

Operation: TEMP \leftarrow R-M:M+1

Condition Codes:

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if an overflow is generated; cleared otherwise.

C — Set if a borrow is generated; cleared otherwise.

Description: Compares the 16-bit contents of the concatenated memory locations M:M+1 to the contents of the specified register and sets the appropriate condition codes. Neither the memory locations nor the specified register is modified unless autoincrement or autodecrement are used. The carry flag represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes: Immediate; Extended; Direct; Indexed.

CMP
(16-Bit)

Complement

Source Forms: COM Q; COMA; COMB

Operation: M' \leftarrow O + \bar{M}

Condition Codes:

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Always cleared.
- C — Always set.

Description: Replaces the contents of memory location M or accumulator A or B with its logical complement. When operating on unsigned values, only BEQ and BNE branches can be expected to behave properly following a COM instruction. When operating on two's complement values, all signed branches are available.

Addressing Modes: Inherent; Extended; Direct; Indexed.

COM

Clear CC bits and Wait for Interrupt

Source Form: CWAI #\$XX

E	F	H	I	N	Z	V	C
---	---	---	---	---	---	---	---

Operation:

CCR \leftarrow CCR \wedge MI (Possibly clear masks)

Set E (entire state saved)

SP' \leftarrow SP - 1, (SP) \leftarrow PCL

SP' \leftarrow SP - 1, (SP) \leftarrow PCH

SP' \leftarrow SP - 1, (SP) \leftarrow USL

SP' \leftarrow SP - 1, (SP) \leftarrow USH

SP' \leftarrow SP - 1, (SP) \leftarrow IYL

SP' \leftarrow SP - 1, (SP) \leftarrow IYH

SP' \leftarrow SP - 1, (SP) \leftarrow IXL

SP' \leftarrow SP - 1, (SP) \leftarrow IXH

SP' \leftarrow SP - 1, (SP) \leftarrow DPR

SP' \leftarrow SP - 1, (SP) \leftarrow ACCB

SP' \leftarrow SP - 1, (SP) \leftarrow ACCA

SP' \leftarrow SP - 1, (SP) \leftarrow CCR

Condition Codes: Affected according to the operation.

Description: This instruction ANDs an immediate byte with the condition code register which may clear the interrupt mask bits I and F, stacks the entire machine state on the hardware stack and then looks for an interrupt. When a non-masked interrupt occurs, no further machine state information need be saved before vectoring to the interrupt handling routine. This instruction replaced the MC6800 CLI WAI sequence, but does not place the buses in a high-impedance state. A FIRQ (fast interrupt request) may enter its interrupt handler with its entire machine state saved. The RTI (return from interrupt) instruction will automatically return the entire machine state after testing the E (entire) bit of the recovered condition code register.

Addressing Mode: Immediate.

Comments: The following immediate values will have the following results:

FF = enable neither

EF = enable \bar{IRQ}

BF = enable FIRQ

AF = enable both

CWAI

Decimal Addition Adjust

Source Form: DAA

Operation: ACCA' \leftarrow ACCA + CF (MSN):CF(LSN)

where CF is a Correction Factor, as follows: the CF for each nibble (BCD) digit is determined separately, and is either 6 or 0.

Least Significant Nibble

CF(LSN)=6 IFF 1) C=1
or 2) LSN>9

Most Significant Nibble

CF(MSN)=6 IFF 1) C=1
or 2) MSN>9
or 3) MSN>8 and LSN>9

Condition Codes:

- H — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Undefined.

C — Set if a carry is generated or if the carry bit was set before the operation; cleared otherwise.

Description: The sequence of a single-byte add instruction on accumulator A (either ADDA or ADCA) and a following decimal addition adjust instruction results in a BCD addition with an appropriate carry bit. Both values to be added must be in proper BCD form (each nibble such that: 0 \leq nibble \leq 9). Multiple-precision addition must add the carry generated by this decimal addition adjust into the next higher digit during the add operation (ADCA) immediately prior to the next decimal addition adjust.

Addressing Mode: Inherent.

DAA

DEC

Decrement

Source Forms: DEC Q; DECA; DECB

Operation: $M' \leftarrow M - 1$

Condition Codes:

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if the original operand was 10000000; cleared otherwise.

C — Not affected.

Description: Subtract one from the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only BEQ and BNE branches can be expected to behave consistently. When operating on two's complement values, all signed branches are available.

Addressing Modes: Inherent; Extended; Direct; Indexed.

EOR

Exclusive OR

Source Forms: EORA P; EORB P

Operation: $R' \leftarrow R \oplus M$

Condition Codes:

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Always cleared.

C — Not affected.

Description: The contents of memory location M is exclusive ORed into an 8-bit register.

Addressing Modes: Immediate; Extended; Direct; Indexed.

EXG

Exchange Registers

Source Form: EXG R1,R2

Operation: $R1 \leftarrow R2$

Condition Codes: Not affected (unless one of the registers is the condition code register).

Description: Exchanges data between two designated registers. Bits 3-0 of the postbyte define one register, while bits 7-4 define the other, as follows:

0000 = A:B

0001 = X

1000 = A

1001 = B

0010 = Y

0011 = US

0100 = SP

0101 = PC

0110 = Undefined

0111 = Undefined

1010 = CCR

1011 = DPR

1100 = Undefined

1101 = Undefined

1110 = Undefined

1111 = Undefined

Only like size registers may be exchanged. (8-bit with 8-bit or 16-bit with 16-bit.)

Addressing Mode: Immediate.

INC

Increment

Source Forms: INC Q; INCA; INCB

Operation: $M' \leftarrow M + 1$

Condition Codes:

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if the original operand was 01111111; cleared otherwise.

C — Not affected.

Description: Adds to the operand. The carry bit is not affected, thus allowing this instruction to be used as a loop counter in multiple-precision computations. When operating on unsigned values, only the BEQ and BNE branches can be expected to behave consistently. When operating on two's complement values, all signed branches are correctly available.

Addressing Modes: Inherent; Extended; Direct; Indexed.

JMP

Jump

Source Form: JMP EA

Operation: $PC' \leftarrow EA$

Condition Codes: Not affected.

Description: Program control is transferred to the effective address.

Addressing Modes: Extended; Direct; Indexed.

JSR

Jump to Subroutine

Source Form: JSR EA

Operation:

$SP' \leftarrow SP - 1, (SP) \leftarrow PCL$
 $SP' \leftarrow SP - 1, (SP) \leftarrow PCH$
 $PC' \leftarrow EA$

Condition Codes: Not affected.

Description: Program control is transferred to the effective address after storing the return address on the hardware stack. A RTS instruction should be the last executed instruction of the subroutine.

Addressing Modes: Extended; Direct; Indexed.

LD
(8-Bit)

Load Register from Memory

Source Forms: LDA P; LDB P

Operation: $R' \leftarrow M$

Condition Codes:

- H — Not affected.
- N — Set if the loaded data is negative; cleared otherwise.

Z — Set if the loaded data is zero; cleared otherwise.

V — Always cleared.

C — Not affected.

Description: Loads the contents of memory location M into the designated register.

Addressing Modes: Immediate; Extended; Direct; Indexed.

LD
(16-Bit)

LEA

LSL

LSR

MUL

Load Register from Memory

Source Forms: LDD P; LDX P; LDY P; LDS P; LDU P
Operation: R' ← M: M + 1

Condition Codes:

- H — Not affected.
- N — Set if the loaded data is negative; cleared otherwise.

Z — Set if the loaded data is zero; cleared otherwise.
 V — Always cleared.
 C — Not affected.

Description: Load the contents of the memory location

M: M + 1 into the designated 16-bit register.

Addressing Modes: Immediate; Extended; Direct; Indexed.

Load Effective Address

Source Forms: LEAX, LEAY, LEAS, LEAU
Operation: R' ← EA

Condition Codes:

- H — Not affected.
- N — Not affected.
- Z — LEAX, LEAY: Set if the result is zero; cleared otherwise. LEAS, LEAU: Not affected.
- V — Not affected.
- C — Not affected.

Description: Calculates the effective address from the index addressing mode and places the address in an indexable register.

LEAX and LEAY affect the Z (zero) bit to allow use of these registers as counters and for MC6800 INX/DEX compatibility.

LEAU and LEAS do not affect the Z bit to allow cleaning up the stack while returning the Z bit as a parameter to a calling

routine, and also for MC6800 INS/DES compatibility.

Addressing Mode: Indexed.

Comments: Due to the order in which effective addresses are calculated internally, the LEAX, X + and LEAX, X + do not add 2 and 1 (respectively) to the X register; but instead leave the X register unchanged. This also applies to the Y, U, and S registers. For the expected results, use the faster instruction LEAX 2, X and LEAX 1, X.

Some examples of LEA instruction uses are given in the following table.

Instruction	Operation	Comment
LEAX 10, X	X + 10 → X	Adds 5-bit constant 10 to X.
LEAX 500, X	X + 500 → X	Adds 16-bit constant 500 to X.
LEAY A, Y	Y + A → Y	Adds 8-bit accumulator to Y.
LEAY D, Y	Y + D → Y	Adds 16-bit D accumulator to Y.
LEAU -10, U	U - 10 → U	Subtracts 10 from U.
LEAS -10, S	S - 10 → S	Used to reserve area on stack.
LEAS 10, S	S + 10 → S	Used to clean up' stack.
LEAX 5, S	S + 5 → X	Transfers as well as adds.

Logical Shift Left

Source Forms: LSL Q; LSLA; LSIB

Operation: C ← [] b7 [] b0 ← 0

Condition Codes:

- H — Undefined.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.

V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.

C — Loaded with bit seven of the original operand.

Description: Shifts all bits of accumulator A or B or memory location M one place to the left. Bit zero is loaded with a zero. Bit seven of accumulator A or B or memory location M is shifted into the C (carry) bit.

Addressing Modes: Inherent; Extended; Direct; Indexed.

Comments: This is a duplicate assembly-language mnemonic for the single machine instruction ASL.

Logical Shift Right

Source Forms: LSR Q; LSRA; LSRB

Operation: 0 → [] b7 [] b0 → C

Condition Codes:

- H — Not affected.

N — Always cleared.

Z — Set if the result is zero; cleared otherwise.

V — Not affected.

C — Loaded with bit zero of the original operand.

Description: Performs a logical shift right on the operand.

Shifts a zero into bit seven and bit zero into the C (carry) bit.

Addressing Modes: Inherent; Extended; Direct; Indexed.

Multiply

Source Form: MUL

Operation: ACCA' · ACCB' ← ACCA × ACCB

Condition Codes:

- H — Not affected.
- N — Not affected.
- Z — Set if the result is zero; cleared otherwise.
- V — Not affected.

C — Set if ACCB bit 7 of result is set; cleared otherwise.

Description: Multiply the unsigned binary numbers in the accumulators and place the result in both accumulators (ACCA contains the most-significant byte of the result). Unsigned multiply allows multiple-precision operations.

Addressing Mode: Inherent.

Comments: The C (carry) bit allows rounding the most-significant byte through the sequence: MUL, ADCA #0.

NEG

Negate**Source Forms:** NEG Q; NEGA; NEGB**Operation:** M' ← 0 – M**Condition Codes:**

- H — Undefined.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Set if the original operand was 10000000.

C — Set if a borrow is generated; cleared otherwise.**Description:** Replaces the operand with its two's complement. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry. Note that 80₁₆ is replaced by itself and only in this case is the V (overflow) bit set. The value 00₁₆ is also replaced by itself, and only in this case is the C (carry) bit cleared.**Addressing Modes:** Inherent; Extended; Direct.

NOP

No Operation**Source Form:** NOP**Operation:** Not affected.**Condition Codes:** This instruction causes only the program counter to be incremented. No other registers or memory locations are affected.**Addressing Mode:** Inherent.

OR

Inclusive OR Memory into Register**Source Forms:** ORA P; ORB P**Operation:** R' ← R v M**Condition Codes:**

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.**V** — Always cleared.**C** — Not affected.**Description:** Performs an inclusive OR operation between the contents of accumulator A or B and the contents of memory location M and the result is stored in accumulator A or B.**Addressing Modes:** Immediate; Extended; Direct; Indexed.

OR

Inclusive OR Memory Immediate into Condition Code Register**Source Form:** ORCC #XX**Operation:** R' ← R v MI**Condition Codes:** Affected according to the operation.**Description:** Performs an inclusive OR operation between the contents of the condition code registers and the immediate value, and the result is placed in the condition code register. This instruction may be used to set interrupt masks (disable interrupts) or any other bit(s).**Addressing Mode:** Immediate.

PSHS

Push Registers on the Hardware Stack**Source Form:**

PSHS register list

PSHS #LABEL

Postbyte:

b7	b6	b5	b4	b3	b2	b1	b0
PC	U	Y	X	DP	B	A	CC

push order →

Operation:

IFF b7 of postbyte set, then: SP' ← SP – 1, (SP) ← PCL
 SP' ← SP – 1, (SP) ← PCH

IFF b6 of postbyte set, then: SP' ← SP – 1, (SP) ← USL
 SP' ← SP – 1, (SP) ← USH

IFF b5 of postbyte set, then: SP' ← SP – 1, (SP) ← IYL
 SP' ← SP – 1, (SP) ← IYH

IFF b4 of postbyte set, then: SP' ← SP – 1, (SP) ← IXL
 SP' ← SP – 1, (SP) ← IXH

IFF b3 of postbyte set, then: SP' ← SP – 1, (SP) ← DPR

IFF b2 of postbyte set, then: SP' ← SP – 1, (SP) ← ACCB

IFF b1 of postbyte set, then: SP' ← SP – 1, (SP) ← ACCA

IFF b0 of postbyte set, then: SP' ← SP – 1, (SP) ← CCR

Condition Codes: Not affected.**Description:** All, some, or none of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself).**Addressing Mode:** Immediate.**Comments:** A single register may be placed on the stack with the condition codes set by doing an autodecrement store onto the stack (example: STX, – S).

PSHU

Push Registers on the User Stack**Source Form:**

PSHU register list

PSHU #LABEL

Postbyte:

b7	b6	b5	b4	b3	b2	b1	b0
PC	U	Y	X	DP	B	A	CC

push order →

Operation:

IFF b7 of postbyte set, then: US' ← US – 1, (US) ← PCL

US' ← US – 1, (US) ← PCH

IFF b6 of postbyte set, then: US' ← US – 1, (US) ← SPL

US' ← US – 1, (US) ← SPH

IFF b5 of postbyte set, then: US' ← US – 1, (US) ← IYL
 US' ← US – 1, (US) ← IYH

IFF b4 of postbyte set, then: US' ← US – 1, (US) ← IXL
 US' ← US – 1, (US) ← IXH

IFF b3 of postbyte set, then: US' ← US – 1, (US) ← DPR

IFF b2 of postbyte set, then: US' ← US – 1, (US) ← ACCB

IFF b1 of postbyte set, then: US' ← US – 1, (US) ← ACCA

IFF b0 of postbyte set, then: US' ← US – 1, (US) ← CCR

Condition Codes: Not affected.**Description:** All, some, or none of the processor registers are pushed onto the user stack (with the exception of the user stack pointer itself).**Addressing Mode:** Immediate.**Comments:** A single register may be placed on the stack with the condition codes set by doing an autodecrement store onto the stack (example: STX, – U).

Pull Registers from the Hardware Stack

Source Form:

PULS register list

PULS #LABEL

Postbyte:

b7	b6	b5	b4	b3	b2	b1	b0
PC	U	Y	X	DP	B	A	CC

← pull order

Operation:

IFF b0 of postbyte set, then: CCR' ←(SP), SP'←SP+1
 IFF b1 of postbyte set, then: ACCA' ←(SP), SP'←SP+1
 IFF b2 of postbyte set, then: ACCB' ←(SP), SP'←SP+1
 IFF b3 of postbyte set, then: DPR' ←(SP), SP'←SP+1
 IFF b4 of postbyte set, then: IXH' ←(SP), SP'←SP+1
 IXL' ←(SP), SP'←SP+1

IFF b5 of postbyte set, then: IYH' ←(SP), SP'←SP+1

IYL' ←(SP), SP'←SP+1

IFF b6 of postbyte set, then: USH' ←(SP), SP'←SP+1

USL' ←(SP), SP'←SP+1

IFF b7 of postbyte set, then: PCH' ←(SP), SP'←SP+1

PCL' ←(SP), SP'←SP+1

Condition Codes: May be pulled from stack; not affected otherwise.**Description:** All, some, or none of the processor registers are pulled from the hardware stack (with the exception of the hardware stack pointer itself).**Addressing Mode:** Immediate.**Comments:** A single register may be pulled from the stack with condition codes set by doing an autoincrement load from the stack (example: LDX,S++).

PULS

Pull Registers from the User Stack

Source Form:

PULU register list

PULU #LABEL

Postbyte:

b7	b6	b5	b4	b3	b2	b1	b0
PC	U	Y	X	DP	B	A	CC

← pull order

Operation:

IFF b0 of postbyte set, then: CCR' ←(US), US'←US+1
 IFF b1 of postbyte set, then: ACCA' ←(US), US'←US+1
 IFF b2 of postbyte set, then: ACCB' ←(US), US'←US+1
 IFF b3 of postbyte set, then: DPR' ←(US), US'←US+1
 IFF b4 of postbyte set, then: IXH' ←(US), US'←US+1
 IXL' ←(US), US'←US+1

IFF b5 of postbyte set, then: IYH' ←(US), US'←US+1

IYL' ←(US), US'←US+1

IFF b6 of postbyte set, then: SPH' ←(US), US'←US+1

SPL' ←(US), US'←US+1

IFF b7 of postbyte set, then: PCH' ←(US), US'←US+1

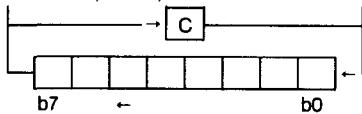
PCL' ←(US), US'←US+1

Condition Codes: May be pulled from stack; not affected otherwise.**Description:** All, some, or none of the processor registers are pulled from the user stack (with the exception of the user stack pointer itself).**Addressing Mode:** Immediate.**Comments:** A single register may be pulled from the stack with condition codes set by doing an autoincrement load from the stack (example: LDX,U++).

PULU

Rotate Left

Source Forms: ROL Q; ROLA; ROLB

Operation:

Condition Codes:

H — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Loaded with the result of the exclusive OR of bits six and seven of the original operand.

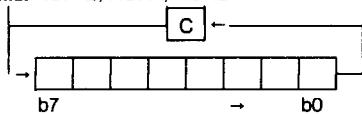
C — Loaded with bit seven of the original operand.

Description: Rotates all bits of the operand one place left through the C (carry) bit. This is a 9-bit rotation.**Addressing Mode:** Inherent; Extended; Direct; Indexed.

ROL

Rotate Right

Source Forms: ROR Q; RORA; RORB

Operation:

Condition Codes:

H — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Not affected.

C — Loaded with bit zero of the previous operand.

Description: Rotates all bits of the operand one place right through the C (carry) bit. This is a 9-bit rotation.**Addressing Modes:** Inherent; Extended; Direct; Indexed.

ROR

RTI

Return from Interrupt**Source Form:** RTI**Operation:** CCR'-(SP), SP'←SP+1, then
IFF CCR bit E is set, then:

ACCA'-(SP), SP'←SP+1
ACCB'-(SP), SP'←SP+1
DPR'-(SP), SP'←SP+1
IXH'-(SP), SP'←SP+1
IXL'-(SP), SP'←SP+1
IYH'-(SP), SP'←SP+1
IYL'-(SP), SP'←SP+1
USH'-(SP), SP'←SP+1
USL'-(SP), SP'←SP+1

PCH'-(SP), SP'←SP+1
PCL'-(SP), SP'←SP+1
IFF CCR bit E is clear, then: PCH'-(SP), SP'←SP+1
PCL'-(SP), SP'←SP+1

Condition Codes: Recovered from the stack.**Description:** The saved machine state is recovered from the hardware stack and control is returned to the interrupted program. If the recovered E (entire) bit is clear, it indicates that only a subset of the machine state was saved (return address and condition codes) and only that subset is recovered.**Addressing Mode:** Inherent.

RTS

Return from Subroutine**Source Form:** RTS**Operation:**

PCH'-(SP), SP'←SP+1
PCL'-(SP), SP'←SP+1

Condition Codes: Not affected.**Description:** Program control is returned from the subroutine to the calling program. The return address is pulled from the stack.**Addressing Mode:** Inherent.

SBC

Subtract with Borrow**Source Forms:** SBCA P; SBCB P**Operation:** R'←R-M-C**Condition Codes:**

H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.

V — Set if an overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description: Subtracts the contents of memory location M and the borrow (in the C (carry) bit) from the contents of the designated 8-bit register, and places the result in that register. The C bit represents a borrow and is set to the inverse of the resulting binary carry.**Addressing Modes:** Immediate; Extended; Direct; Indexed.

SEX

Sign Extended**Source Form:** SEX**Operation:**

If bit seven of ACCB is set then ACCA'←FF₁₆
else ACCA'←00₁₆

Condition Codes:

H — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero, cleared otherwise.

V — Not affected.

C — Not affected.

Description: This instruction transforms a twos complement 8-bit value in accumulator B into a twos complement 16-bit value in the D accumulator.**Addressing Mode:** Inherent.ST
(8-Bit)**Store Register into Memory****Source Forms:** STA P; STB P**Operation:** M'←R**Condition Codes:**

H — Not affected.
N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Writes the contents of an 8-bit register into a memory location.**Addressing Modes:** Extended; Direct; Indexed.ST
(16-Bit)**Store Register into Memory****Source Forms:** STD P; STX P; STY P; STS P; STU P**Operation:** M':M+1'←R**Condition Codes:**

H — Not affected.
N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.
V — Always cleared.
C — Not affected.

Description: Writes the contents of a 16-bit register into two consecutive memory locations.**Addressing Modes:** Extended; Direct; Indexed.SUB
(8-Bit)**Subtract Memory from Register****Source Forms:** SUBA P; SUBB P**Operation:** R'←R-M**Condition Codes:**

H — Undefined.
N — Set if the result is negative; cleared otherwise.
Z — Set if the result is zero; cleared otherwise.

V — Set if the overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description: Subtracts the value in memory location M from the contents of a designated 8-bit register. The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.**Addressing Modes:** Immediate; Extended; Direct; Indexed.

SUB
(16-Bit)

Subtract Memory from Register

Source Forms: SUBD P

Operation: R' ← R - M: M + 1

Condition Codes:

H — Not affected.

N — Set if the result is negative; cleared otherwise.

Z — Set if the result is zero; cleared otherwise.

V — Set if the overflow is generated; cleared otherwise.
C — Set if a borrow is generated; cleared otherwise.

Description: Subtracts the value in memory location

M:M + 1 from the contents of a designated 16-bit register.

The C (carry) bit represents a borrow and is set to the inverse of the resulting binary carry.

Addressing Modes: Immediate; Extended; Direct; Indexed.

SWI

Software Interrupt

Source Form: SWI

Operation:

Set E (entire state will be saved)

SP' ← SP - 1, (SP) ← PCL

SP' ← SP - 1, (SP) ← PCH

SP' ← SP - 1, (SP) ← USL

SP' ← SP - 1, (SP) ← USH

SP' ← SP - 1, (SP) ← IYL

SP' ← SP - 1, (SP) ← IYH

SP' ← SP - 1, (SP) ← IXL

SP' ← SP - 1, (SP) ← IXH

SP' ← SP - 1, (SP) ← DPR
SP' ← SP - 1, (SP) ← ACCB
SP' ← SP - 1, (SP) ← ACCA
SP' ← SP - 1, (SP) ← CCR
Set I, F (mask interrupts)
PC' ← (FFFFA):(FFFFB)

Condition Codes: Not affected.

Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt vector. Both the normal and fast interrupts are masked (disabled).

Addressing Mode: Inherent.

SWI2

Software Interrupt 2

Source Form: SWI2

Operation:

Set E (entire state saved)

SP' ← SP - 1, (SP) ← PCL

SP' ← SP - 1, (SP) ← PCH

SP' ← SP - 1, (SP) ← USL

SP' ← SP - 1, (SP) ← USH

SP' ← SP - 1, (SP) ← IYL

SP' ← SP - 1, (SP) ← IYH

SP' ← SP - 1, (SP) ← IXL

SP' ← SP - 1, (SP) ← IXH

SP' ← SP - 1, (SP) ← DPR
SP' ← SP - 1, (SP) ← ACCB
SP' ← SP - 1, (SP) ← ACCA
SP' ← SP - 1, (SP) ← CCR
PC' ← (FFF4):(FFF5)

Condition Codes: Not affected.

Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt 2 vector. This interrupt is available to the end user and must not be used in packaged software. This interrupt does not mask (disable) the normal and fast interrupts.

Addressing Mode: Inherent.

SWI3

Software Interrupt 3

Source Form: SWI3

Operation:

Set E (entire state will be saved)

SP' ← SP - 1, (SP) ← PCL

SP' ← SP - 1, (SP) ← PCH

SP' ← SP - 1, (SP) ← USL

SP' ← SP - 1, (SP) ← USH

SP' ← SP - 1, (SP) ← IYL

SP' ← SP - 1, (SP) ← IYH

SP' ← SP - 1, (SP) ← IXL

SP' ← SP - 1, (SP) ← IXH

SP' ← SP - 1, (SP) ← DPR
SP' ← SP - 1, (SP) ← ACCB
SP' ← SP - 1, (SP) ← ACCA
SP' ← SP - 1, (SP) ← CCR
PC' ← (FFF2):(FFF3)

Condition Codes: Not affected.

Description: All of the processor registers are pushed onto the hardware stack (with the exception of the hardware stack pointer itself), and control is transferred through the software interrupt 3 vector. This interrupt does not mask (disable) the normal and fast interrupts.

Addressing Mode: Inherent.

SYNC

Synchronize to External Event**Source Form:** SYNC**Operation:** Stop processing instructions.**Condition Codes:** Not affected.

Description: When a SYNC instruction is executed, the processor enters a synchronizing state, stops processing instructions, and waits for an interrupt. When an interrupt occurs, the synchronizing state is cleared and processing continues. If the interrupt is enabled, and it last three cycles or more, the processor will perform the interrupt routine. If the interrupt is masked or is shorter than three cycles, the processor simply continues to the next instruction. While in the synchronizing state, the address and data buses are in the high-impedance state.

This instruction provides software synchronization with a hardware process. Consider the following example for high-speed acquisition of data:

FAST	SYNC Interrupt!	DISC	WAIT FOR DATA
LDA		DATA FROM DISC AND CLEAR INTERRUPT	
STA	.X+	PUT IN BUFFER	
DEC B		COUNT IT, DONE?	
BNE	FAST	GO AGAIN IF NOT.	

The synchronizing state is cleared by any interrupt. Of course, enabled interrupts at this point may destroy the data transfer and, as such, should represent only emergency conditions.

The same connection used for interrupt-driven I/O service may also be used for high-speed data transfers by setting the interrupt mask and using the SYNC instruction as the above example demonstrates.

Addressing Mode: Inherent.

TFR

Transfer Register to Register**Source Form:** TFR R1, R2**Operation:** R1 → R2**Condition Code:** Not affected unless R2 is the condition code register.

Description: Transfers data between two designated registers. Bits 7-4 of the postbyte define the source register, while bits 3-0 define the destination register, as follows:

0000 = A:B
0001 = X1000 = A
1001 = B

0010 = Y	1010 = CCR
0011 = US	1011 = DPR
0100 = SP	1100 = Undefined
0101 = PC	1101 = Undefined
0110 = Undefined	1110 = Undefined
0111 = Undefined	1111 = Undefined

Only like size registers may be transferred. (8-bit to 8-bit, or 16-bit to 16-bit.)

Addressing Mode: Immediate.

TST

Test**Source Forms:** TST Q; TSTA; TSTB**Operation:** TEMP ← M – 0**Condition Codes:**

- H — Not affected.
- N — Set if the result is negative; cleared otherwise.
- Z — Set if the result is zero; cleared otherwise.
- V — Always cleared.
- C — Not affected.

Description: Set the N (negative) and Z (zero) bits according to the contents of memory location M, and clear the V (overflow) bit. The TST instruction provides only minimum information when testing unsigned values; since no unsigned value is less than zero, BLO and BLS have no utility. While BHI could be used after TST, it provides exactly the same control as BNE, which is preferred. The signed branches are available.

Addressing Modes: Inherent; Extended; Direct; Indexed.**Comments:** The MC6800 processor clears the C (carry) bit.

FIRQ

Fast Interrupt Request (Hardware Interrupt)**Operation:**

IFF F bit clear, then: SP' ← SP – 1, (SP) ← PCL
 SP' ← SP – 1, (SP) ← PCH
 Clear E (subset state is saved)
 SP' ← SP – 1, (SP) ← CCR
 Set F, I (mask further interrupts)
 PC' ← (FFF6):(FFFF7)

Condition Codes: Not affected.

Description: A FIRQ (fast interrupt request) with the F (fast interrupt request mask) bit clear causes this interrupt sequence to occur at the end of the current instruction. The program counter and condition code register are pushed

onto the hardware stack. Program control is transferred through the fast interrupt request vector. An RTI (return from interrupt) instruction returns the processor to the original task. It is possible to enter the fast interrupt request routine with the entire machine state saved if the fast interrupt request occurs after a clear and wait for interrupt instruction. A normal interrupt request has lower priority than the fast interrupt request and is prevented from interrupting the fast interrupt request routine by automatic setting of the I (interrupt request mask) bit. This mask bit could then be reset during the interrupt routine if priority was not desired. The fast interrupt request allows operations on memory, TST, INC, DEC, etc. instructions without the overhead of saving the entire machine state on the stack.

Addressing Mode: Inherent.

Interrupt Request (Hardware Interrupt)

Operation:

IFF I bit clear, then: SP'←SP-1, (SP)←PCL
 SP'←SP-1, (SP)←PCH
 SP'←SP-1, (SP)←USL
 SP'←SP-1, (SP)←USH
 SP'←SP-1, (SP)←IYL
 SP'←SP-1, (SP)←IYH
 SP'←SP-1, (SP)←IXL
 SP'←SP-1, (SP)←IXH
 SP'←SP-1, (SP)←DPR
 SP'←SP-1, (SP)←ACCB
 SP'←SP-1, (SP)←ACCA

Set E (entire state saved)
 SP'←SP-1, (SP)←CCR
 Set I (mask further IRQ interrupts)
 PC'←(FFF8):(FFF9)

Condition Codes: Not affected.

Description: If the I (interrupt request mask) bit is clear, a low level on the IRQ input causes this interrupt sequence to occur at the end of the current instruction. Control is returned to the interrupted program using a RTI (return from interrupt) instruction. A FIRQ (fast interrupt request) may interrupt a normal IRQ (interrupt request) routine and be recognized anytime after the interrupt vector is taken.

Addressing Mode: Inherent.

IRQ

Non-Maskable Interrupt (Hardware Interrupt)

Operation:

SP'←SP-1, (SP)←PCL
 SP'←SP-1, (SP)←PCH
 SP'←SP-1, (SP)←USL
 SP'←SP-1, (SP)←USH
 SP'←SP-1, (SP)←IYL
 SP'←SP-1, (SP)←IYH
 SP'←SP-1, (SP)←IXL
 SP'←SP-1, (SP)←IXH
 SP'←SP-1, (SP)←DPR
 SP'←SP-1, (SP)←ACCB
 SP'←SP-1, (SP)←ACCA
 Set E (entire state save)
 SP'←SP-1, (SP)←CCR

Set I, F (mask interrupts)
 PC'←(FFFC):(FFFD)

Condition Codes: Not affected.

Description: A negative edge on the NMI (non-maskable interrupt) input causes all of the processor's registers (except the hardware stack pointer) to be pushed onto the hardware stack, starting at the end of the current instruction. Program control is transferred through the NMI vector. Successive negative edges on the NMI input will cause successive NMI operations. Non-maskable interrupt operation can be internally blocked by a RESET operation and any non-maskable interrupt that occurs will be latched. If this happens, the non-maskable interrupt operation will occur after the first load into the stack pointer (LDS; TFR r,s; EXG r,s; etc.) after RESET.

Addressing Mode: Inherent.

NMI

Restart (Hardware Interrupt)

Operation:

CCR'←X1X1XXXX
 DPR'←00₁₆
 PC'←(FFFE):(FFFF)

Condition Codes: Not affected.

Description: The processor is initialized (required after power-on) to start program execution. The starting address is fetched from the restart vector.

Addressing Mode: Extended; Indirect.

RESTART

Reference M/ Sample Programs

Example 1

```
10 ' This is an example of a BASIC program that calls
20 ' an assembly language program to paint the screen
30 ' yellow.
40 '
50 ' After entering the BASIC program save it on disk.
60 '
70 '
80 ' Run DOS and enter the assembly language program.
90 ' Use the WD and AD assembler commands to write the
100 ' source program to disk and to assemble it.
110 '
120 ' After returning to BASIC, load the assembled
130 ' program into memory with the LOADM command. You
140 ' must load the assembled program before the BASIC
150 ' program.
160 '
170 ' This program demonstrates how much faster
180 ' an assembly program can perform a function than a
190 ' BASIC statement. After you run the program once,
200 ' delete lines 1030, 1040, 1050, and 1120. Insert
210 ' this statement
220 '     1120 PAINT (1,1),2
230 ' and see how much longer it takes BASIC to paint
240 ' the entire screen yellow.
250 '
260 'Specify the highest address BASIC can use. This
270 ' Prevents BASIC from using the memory that contains
280 ' your assembly language subroutine.
290 CLEAR 200,16127
300 PCLEAR 6      'reserve 6 pages of graphics memory
310 DEF USR0=16128 'define the subroutine starting address
320 ' The disk drive uses pages 0 and 1 of video memory.
330 ' You must start at page 2, hex 1200.
340 PMODE 3,2      'select mode 3, starting at page 2
350 PCLS           'clear the screen
360 SCREEN 1,0      'select graphics screen, color set 0
370 COLOR 3,1       'set foreground color to blue
380 A=USR(0)        'call the assembly language subroutine
390 'draw a frame
```

```

1140 LINE (0,0)-(255,191),PSET,B
1150 LINE (12,12)-(242,178),PSET,B
1160 PAINT (2,2),4,3           'fill in the frame with red
1170 FOR X=50 TO 90 STEP 20    'draw top circles
1180 Y=30:ST=.5:EN=0          '      of big cloud
1190 GOSUB 5000
1200 Y=50:ST=0:EN=.5          'draw bottom circles
1210 GOSUB 5000:NEXT X        '      of big cloud
1220 FOR X=160 TO 180 STEP 20
1230 Y=30:ST=.5:EN=0          'draw top circles
1240 GOSUB 5000               '      of little cloud
1250 Y=50:ST=0:EN=.5          'draw bottom circles
1260 GOSUB 5000:NEXT X        '      of little cloud
1270 Y=40:ST=.25:EN=.75       'draw left sides of clouds
1280 GOSUB 5020
1290 X=150:GOSUB 5020
1300 X=100:ST=.75:EN=.25     'draw right sides of clouds
1310 GOSUB 5020
1320 X=190:GOSUB 5020
1330 PAINT (52,30),3,3        'fill the clouds in with blue
1340 PAINT (162,30),3,3
1350 R=60:H=1:GOSUB 5040      'draw the umbrella
1360 R=37:H=1.7:GOSUB 5040    'draw the spokes of the
1370 R=15:H=4.7:GOSUB 5040    '      umbrella
1380 ST=.5:EN=0              'draw the scalloped edges
1390 FOR X=78 TO 184 STEP 23
1400 Y=124:GOSUB 5000
1410 NEXT X
1420 'draw umbrella handle
1430 DRAW "BM121,120;D40;R2;D2;R2;D2;R8;U2;R2;U2;R2;U3;
L2;D2;L2;D2;L2;D2;L3;U2;L2;U2;L2;U40"
1440 PAINT (122,122),3,3      'Paint umbrella handle
1450 PAINT (124,161),3
1460 PAINT (126,163),3
1470 C=8                      'set highest color number
1480 FOR X=68 TO 180 STEP 24  'Paint umbrella Panels
1490 PAINT (X,120),C,3
1500 C=C-1:NEXT X
1510 'Play the song "Raindrops Keep Falling On My Head"
1520 GOSUB 6000:PLAY L$
1530 GOSUB 9000:PLAY L$
1540 PLAY M$:PLAY E$:PLAY N$
1550 PLAY G$:PLAY E$:PLAY O$
1560 PLAY P$:PLAY Q$:PLAY E$
1570 PLAY R$:PLAY S$:PLAY R$
1580 PLAY T$:PLAY P$:PLAY E$
1590 PLAY U$:GOSUB 9000
1600 PLAY V$:PLAY E$:PLAY E$
1610 PLAY W$:PLAY X$
1620 'Keep the image on the screen until a key is pressed.
1630 Z$=INKEY$
1640 IF Z$="" THEN 1630
1650 END

```

```

5000 CIRCLE (X,Y),13,3,,45,ST,EN
5010 RETURN
5020 CIRCLE (X,Y),16,3,,75,ST,EN
5030 RETURN
5040 CIRCLE (124,124),R,3,H,,5,0
5050 RETURN
5060 '
5070 'These lines define the notes of the song.
6000 A$="03;L4A;L8.;A;L16A;L8.;B-;L16A;L8.G;L16F;L4.;A"
6010 B$="P8;P4;P8;P16"
6020 C$="03;L16;C;04;L4C;L8.;C;L16C;L8.;D;L16C;L8.;C"
6030 D$="03;L16A;L4A;B-;G;F;04;E;P4"
6040 E$="P4"
6050 F$="04;L8.;D;L16C;03;L8.;A;L16E;04;L4.E"
6060 G$="P8"
6070 H$="04;L4.;D"
6080 I$="04;L4C;L8.;C;03;L16A;L8.;B-"
6090 J$="04;L16C;03;L8.;B-;L16A"
6100 K$="04;L4.;C;P4"
6110 L$="03;L4F;F;G"
6120 M$="03;L2;A"
6130 N$="04;L8.;C;03;L2G"
6140 O$="03;L8.;A;L4B-;L4A;L4G"
6150 P$="03;L8.;F;L4A;L4.;G"
6160 Q$="03;L4A;L8.;B-;04;L4D;L4C"
6170 R$="P8;P16"
6180 S$="03;L16A;04;L8D;L4C;L2C"
6190 T$="03;L16A;04;L8E;L4D;L2C"
6200 U$="P2;P1"
6210 V$="03;L4F;F;G;L2.;A"
6220 W$="03;L8.;F;L16F;04;L8.;D;L16C;03;L4F"
6230 X$="03;L8A;G;L4F;L2.;F"
9000 PLAY A$;PLAY B$;PLAY C$
9010 PLAY D$;PLAY E$;PLAY F$
9020 PLAY G$;PLAY H$;PLAY G$
9030 PLAY I$;PLAY J$
9040 PLAY I$;PLAY K$
9050 RETURN

00100 * Use EDTASM or EDTASMOV to enter this program. Save
00110 *      the program on disk with WD command and
00120 *      assemble the program with AD command. Do not
00130 *      use the SR switch because this program is
00140 *      called from BASIC, not executed from DOS.
00150 *
00160 * Use the LOADM command to load the assembled code
00170 *      into memory before you load the BASIC program.
00180 *      The ORG statement tells BASIC where in memory
00190 *      to load the program.
00200 *
00210      ORG    $3F00
00220 *

```

```

00230 * Put the hex code for a yellow point (55H) in
00240 *      register A and the address of the first byte
00250 *      of video memory (1200) in register X,
00260 *      The first byte of video memory is 1200 hex
00270 *      because the disk drive uses memory up to that
00280 *      address.
00290 *
00300 START    LDA     #$55
00310         LDX     #$1200
00320 *
00330 * Store the yellow dot at the current video memory
00340 *      address and increment X to the next video
00350 *      memory address.
00360 *
00370 SCREEN   STA     ,X+
00380         CMPX    #$2FFF   Is it the end of video memory?
00390         BNE     SCREEN   If no, continue to store dots
00400         RTS     If yes, exit subroutine and
00410 *                  and return to BASIC
00420 DONE     EQU     *
00430         END     START

```

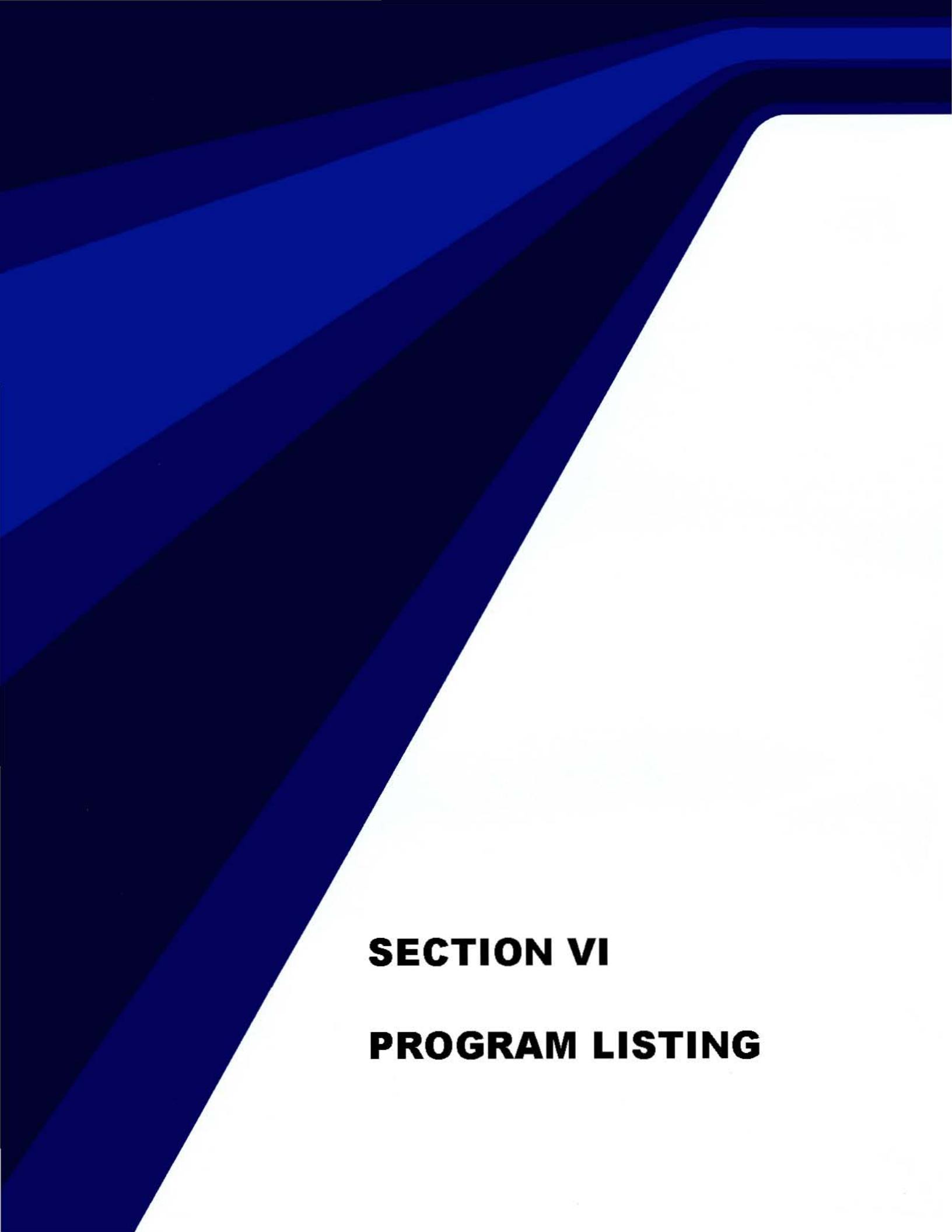
Example 2

```

20 ' After entering the BASIC Program save it on disk.
30 '
40 ' Run DOS and enter the assembly language Program. Use
50 '      the WD and AD assembler commands to write the
60 '      source program to disk and to assemble it.
70 '
80 ' After returning to BASIC, load the assembled
90 '      program into memory with the LOADM command. You
100 '     must load the assembled program before the BASIC
110 '     program.
120 '
130 ' Specify the highest address BASIC can use. This
140 '     prevents BASIC from using the memory that contains
150 '     your assembly language subroutine.
160 CLEAR 200, 16127
170 DEF USR0=16128           'define address of subroutine
180 CLS                     'clear the screen
190 ' Print a prompting message and wait for a response.
200 INPUT "Press [ENTER] when ready"; A$
210 A=USR(0)                 'call subroutine
220 ' Print another prompting message and wait for a response
230 INPUT "Want to do it again"; A$
240 ' If operator types yes, start over. Otherwise end.
250 IF A$="YES" THEN 20 ELSE END

```

```
00100 * Use EDTASM or EDTASMOV to enter this Program. Save
00110 *      the Program on disk with WD command and
00120 *      assemble the Program with AD command. Do not
00130 *      use the SR switch because this Program is
00140 *      called from BASIC, not executed from DOS.
00150 *
00160 * Use the LOADM command to load the assembled code
00170 *      into memory before you load the BASIC Program.
00180 *      The ORG statement tells BASIC where in memory
00190 *      to load the program.
00200 *
00210      ORG    $3F00
00220 *
00230 * Put the hex code for a red checkerboard in
00240 *      register A and the address of the first byte
00250 *      of video memory (400) in register X.
00260 *
00270 START   LDA    #$0F9
00310      LDX    #$400
00320 *
00330 * Store the red checkerboard at the current video
00340 *      memory address and increment X to the next
00350 *      video memory address.
00360 *
00370 SCREEN  STA    ,X+
00380      CMPX   #$600      Is it the end of video memory?
00390      BNE    SCREEN     If no, continue to store red
00400 *          checkerboards
00410      RTS     If yes, exit subprogram and
00420 *          and return to BASIC
00430 DONE    EQU    *
00440      END    START
```

SECTION VI

PROGRAM LISTING

SECTION VI**PROGRAM LISTING**

This section provides a complete source listing of the DOS program.

PAGE 003 DOC .SA:0

```

00630 00112
00640 00113
00650 00114
00660 00115
00670 00116
00680 00117
00690 00118
00700 00119
00710 00120
00720 00121
00730 00122
00740 00123
00750 00124
00760 00125
00770 00126
00780 00127
00790 00128
00800 00129
00810 00130
00820 00131
00830 00132
00840 00133
00850 00134
00860 00135
00870 00136
00880 00137
00890 00138
00900 00139
00910 00140
00920 00141
00930 00142
00940 00143
00950 00144
00960 00145
00970 00146
00980 00147
00990 00148
01000 00149
01010 00150
01020 00151
01030 00152
01040 00153
01050 00154
01060 00155
01070 00156
01080 00157
01090 00158
01100 00159
01110 00160
01120 00161
01130 00162
01140 00163
01150 00164
01160 00165
01170 00166
01180 00167
01190 00168
01200 00169

*****  

* INSTRUCTIONS FOR USE  

*****  

*  

*  

*****  

* ERROR NUMBERS AND THEIR MEANING  

* (THE EQUATES ARE USED SO THAT ERRORS CAN BE RESEARCHED USING XREF LIST)  

* DEFINITIONS START WITH BASIC LINE NUMBER 256 IN DOS  

*****  

*****  

0000 A ERR0 EQU 0 256 NO ERRORS
0001 A ERR1 EQU 1 257 I/O ERROR - DRIVE NOT READY
0002 A ERR2 EQU 2 258 I/O ERROR - WRITE PROTECTED
0003 A ERR3 EQU 3 259 I/O ERROR - WRITE FAULT
0004 A ERR4 EQU 4 260 I/O ERROR - SEEK ERROR OR RECORD NOT FOUND
0005 A ERR5 EQU 5 261 I/O ERROR - CRC ERROR
0006 A ERR6 EQU 6 262 I/O ERROR - LOST DATA
0007 A ERR7 EQU 7 263 I/O ERROR - UNDEFINED BIT 1
0008 A ERR8 EQU 8 264 I/O ERROR - UNDEFINED BIT 0
0009 A ERR9 EQU 9 265 REGISTER ARGUMENT INVALID
000A A ERR10 EQU 10 266 FILE'S DIRECTORY ENTRY NOT FOUND
000B A ERR11 EQU 11 267 DIRECTORY IS FULL
000C A ERR12 EQU 12 268 FILE WAS CREATED BY "OPEN" FUNCTION
000D A ERR13 EQU 13 269 FILE NOT CLOSED AFTER CHANGES
000E A ERR14 EQU 14 270 ATTEMPTING TO ACCESS AN UNOPENED FILE
000F A ERR15 EQU 15 271 ATTEMPT TO READ - READ PROTECTED
0010 A ERR16 EQU 16 272 RBA OVERFLOW (EXCEEDS 3 BYTES - 16,777,216)
0011 A ERR17 EQU 17 273 ACCESS BEYOND EOF - EXTENSION NOT ALLOWED
0012 A ERR18 EQU 18 274 FAT REWRITE ERROR
0013 A ERR19 EQU 19 275 ATTEMPT TO CLOSE UNOPENED FILE
0014 A ERR20 EQU 20 276 CAN'T ACCESS RANDOMLY - REC SIZE IS ZERO!
0015 A ERR21 EQU 21 277 ATTEMPT TO WRITE - WRITE PROTECTED
0016 A ERR22 EQU 22 278 CAN'T EXTEND FILE - DISK CAPACITY EXCEEDED
0017 A ERR23 EQU 23 279 ERROR WHILE LOADING OVERLAY - FUNCTION NOT PERFORMED
0018 A ERR24 EQU 24 280 INSUFFICIENT PRINT SPACE ALLOCATED
0019 A ERR25 EQU 25 281 I/O ERROR DURING BASIC LINE READ
001A A ERR26 EQU 26 282 PROGRAM'S LOAD ADDRESS IS TOO LOW
001B A ERR27 EQU 27 283 FIRST BYTE OF PROGRAM FILE NOT EQUAL TO ZERO
001C A ERR28 EQU 28 284 SPACE FOR BUFFERED KBD NOT BIG ENOUGH
001D A ERR29 EQU 29 285 NOT ENOUGH MEMORY
001E A ERR30 EQU 30 286 OUTPUT FILE ALREADY EXISTS
001F A ERR31 EQU 31 287 WRONG DISKETTE
*  

*****  

* DISK DATA CONTROL BLOCK (DCB) FORMAT
*****
```

PAGE 004 DOC .SA:0

DOS - INSTRUCTIONS

```

01210 00170
01220 00171
01230 00172
01240 00173
01250 00174
01260 00175
01270 00176
01280 00177
01290 00178
01300 00179
01310 00180
01320 00181
01330 00182
01340 00183
01350 00184
01360 00185
01370 00186
01380 00187
01390 00188
01400 00189
01410 00190
01420 00191
01430 00192
01440 00193
01450 00194
01460 00195
01470 00196
01480 00197
01490 00198
01500 00199
01510 00200
01520 00201
01530 00202
01540 00203
01550 00204
01560 00205
01570 00206
01580 00207
01590 00208
01600 00209
01610 00210
01620 00211
01630 00212
01640 00213
01650 00214
01660 00215
01670 00216
01680 00217
01690 00218
01700 00219
01710 00220
01720 00221
01730 00222
01740 00223
01750 00224
01760 00225
01770 00226
01780 00227

*****  

* BYTES CONTENTS  

* -----  

* THESE ITEMS ARE A COPY OF DISK DIRECTORY ENTRY  

* 0-7 FILENAME  

* 8-10 FILE EXTENSION  

* 11 FILE TYPE  

* (0=BASIC PGM,1=BASIC DATA,2=MACHINE LANG. PGM,3=TEXT ED. SOURCE)  

* 12 ASCII FLAG (0=BINARY, FF = ASCII FILE)  

* 13 NUMBER OF FIRST CLUSTER IN FILE  

* 14-15 NUMBER OF BYTES IN USE IN LAST SECTOR OF FILE  

* THESE ITEMS WERE ADDED, USING LAST 16 BYTES OF DIRECTORY ENTRY  

* 16 CURRENT FILE STATUS  

* BIT 0 ON ALLOWS READS  

* BIT 1 ON ALLOWS WRITES  

* BIT 2 ON ALLOWS FILE CREATE IF NON-EXISTANT  

* BIT 3 ON ALLOWS FILE EXTENSION BEYOND EOF ON ACCESS ATTEMPTS  

* BIT 4 ON MEANS WORK FILE - DELETE FILE WHEN CLOSED  

* BIT 5 ON PREVENTS REWRITE OF FAT EVERY TIME A SECTOR IS ADDED TO  

* THE FILE. (MINOR POWER FAILURE INCONSISTANCY COULD RESULT)  

* BIT 6 ON MEANS I/O BUFFER IS SHARED. EACH LOGICAL I/O REQUIRES  

* A PHYSICAL I/O  

* BIT 7 RESERVED FOR FUTURE OPTION(LIKE RELEASE SPACE WHEN FILE SHORTENED)  

* (ALL BITS OFF = FILE CLOSED)  

* 17-18 LOGICAL RECORD SIZE (AS OF LAST TIME FILE WAS CLOSED)  

* ZERO MEANS VARIABLE LENGTH WITH RECORDS TERMINATED BY THE  

* DELIMITER STORED BELOW.  

* $FFF MEANS VARIABLE LENGTH WITH FIRST TWO BYTES OF RECORD  

* CONTAINING SIZE OF THE REST OF THE RECORD.  

* ALL OTHER VALUES MEAN FIXED LENGTH OF SPECIFIED SIZE.  

* 19 VARIABLE LENGTH RECORD TERMINATOR  

* 20-31 AT PRESENT, UNUSED PART OF DIRECTORY ENTRY - USE WITH CAUTION.  

*  

* THESE ITEMS ARE USED FOR PHYSICAL I/O PARAMETERS  

* 32 LAST I/O OPCD  

* 33 LAST I/O DRIVE  

* 34 LAST I/O TRACK  

* 35 LAST I/O SECTOR  

* 36-37 LAST I/O BUFFER POINTER  

* 38 LAST I/O RESULT CODE  

*  

* THESE ITEMS ARE FOR LOGICAL USE  

* 39-40 LOGICAL RECORD BUFFER (CAN BE SAME AS DCBBUF IF DCBRSZ=256)  

* 41-42 LAST I/O PHYSICAL RECORD NUMBER (BEFORE XLATE INTO SECTOR WITHIN  

* GRANULE). THIS IS THE RECORD CURRENTLY IN THE BUFFER.  

* 43-45 CURRENT RELATIVE BYTE ADDRESS (RBA) OF FILE DATA POINTER  

* 46-47 CURRENT LOGICAL RECORD NUMBER  

* 48 MODIFIED DATA TAG - SET NON-ZERO WHEN BUFFER CONTENTS CHANGED  

*  

* EQUATES FOLLOW FOR MEANINGFUL SOURCE CODE WHEN ACCESSING DCB  

* IE: STD DCBLRN,U SAVE NEW LOGICAL RECORD NUMBER  

* (BETTER THAN STD 46,U )  

0000 A DCBFNM EQU 0 FILE NAME
0008 A DCBFEX EQU 8 FILE NAME EXTENSION
000B A DCBFYT EQU 11 FILE TYPE
000C A DCBASC EQU 12 ASCII CODE
000D A DCBFCL EQU 13 FIRST CLUSTER NUMBER

```

PAGE 005 DOC .SA:0 DOS - INSTRUCTIONS

01790 00228	000E	A DCBNLS EQU	14	NUMBER OF BYTES USED IN LAST SECTOR
01800 00229	0010	A DCBCFS EQU	16	CURRENT FILE STATUS
01810 00230	0011	A DCBRSZ EQU	17	RECORD SIZE
01820 00231	0013	A DCBTRM EQU	19	VAR LEN RECORD TERMINATOR
01830 00232	0014	A DCBMRB EQU	20	MAX RBA
01840 00233	0017	A DCBUSR EQU	23	USER AREA
01850 00234	0020	A DCBOPC EQU	32	OPERATION CODE
01860 00235	0021	A DCBDRV EQU	33	DRIVE
01870 00236	0022	A DCBTRK EQU	34	TRACK
01880 00237	0023	A DCBSEC EQU	35	SECTOR
01890 00238	0024	A DCBBUF EQU	36	I/O BUFFER ADDRESS
01900 00239	0026	A DCBOK EQU	38	I/O RESULT CODE
01910 00240	0027	A DCBLRB EQU	39	LOGICAL RECORD BUFFER ADDRESS
01920 00241	0029	A DCBPRN EQU	41	PHYSICAL RECORD NUMBER IN BUFFER
01930 00242	0028	A DCBRAA EQU	43	CURRENT RELATIVE BYTE ADDRESS
01940 00243	002E	A DCBLRN EQU	46	CURRENT LOGICAL RECORD NUMBER
01950 00244	0030	A DCBMDT EQU	48	MODIFIED DATA TAG
01960 00245	0031	A DCBSZ EQU	DCBMDT+1	SIZE OF DCB (CURRENTLY 50 BYTES)
01970 00246	*			
01980 00247				*****
01990 00248				* EQUATES TO SUPPORT ROUTINES IN ROM OPERATING SYSTEM
02000 00249				*****
02010 00250	A000	A POLCAT EQU	\$A000	
02020 00251	0152	A ROLTAB EQU	\$152	KBD ROLLOVER TABLE
02030 00252	A00A	A JOYIN EQU	\$A00A	
02040 00253	A006	A BLKIN EQU	\$A006	
02050 00254	A004	A CSRDON EQU	\$A004	
02060 00255	A00C	A WRTLDR EQU	\$A00C	
02070 00256	A008	A BLKOUT EQU	\$A008	
02080 00257	007C	A BLKTYP EQU	\$7C	
02090 00258	007D	A BLKLEN EQU	\$7D	
02100 00259	007E	A CBUFAD EQU	\$7E	
02110 00260	010C	A IRQ EQU	\$10C	
02120 00261	015A	A POTS EQU	\$15A	JOYSTICK POT VALUES
02130 00262	011A	A ALPHLK EQU	\$11A	KBD RTN'S ALPHA LOCK SWITCH
02140 00263	*			
02150 00264				*****
02160 00265				* EQUATES TO XREF USE OF PIA'S
02170 00266				*****
02180 00267	FF21	A U4ACR EQU	\$FF21	CONTROL REG
02190 00268	FF20	A U4ADR EQU	\$FF20	DATA REG
02200 00269	FF20	A U4ADD EQU	\$FF20	DATA DIRECTION REG
02210 00270	FF23	A U4BCR EQU	\$FF23	
02220 00271	FF22	A U4BDR EQU	\$FF22	
02230 00272	FF22	A U4BDD EQU	\$FF22	
02240 00273	FF01	A UBACR EQU	\$FF01	
02250 00274	FF00	A UBADR EQU	\$FF00	
02260 00275	FF00	A UBADD EQU	\$FF00	
02270 00276	FF03	A UBBCR EQU	\$FF03	
02280 00277	FF02	A UBBDR EQU	\$FF02	
02290 00278	FF02	A UBBDD EQU	\$FF02	
02300 00279	*			
02310 00280				* MISC ADDITIONAL EQUATES
02320 00281	0035	A ENABLE EQU	%00110101	
02330 00282	0034	A DSABLE EQU	%00110100	
02340 00283				* COLOR VALUES
02350 00284	0000	A BUFF EQU	%00000000	
02360 00285	0055	A CYAN EQU	%01010101	

PAGE 006 DOC .SA:0 DOS - INSTRUCTIONS

```

02370 00286      00AA   A MGNTA EQU    %10101010
02380 00287      00FF   A ORANGE EQU   %11111111
02390 00288      0000   A GREEN  EQU   %00000000
02400 00289      0055   A YELLOW EQU   %01010101
02410 00290      00AA   A BLUE   EQU   %10101010
02420 00291      00FF   A RED    EQU   %11111111
02430 00292          * CODES RETURNED BY POLCAT FOR FUNCTION KEYS
02440 00293      005E   A UP     EQU   $5E    UP ARROW
02450 00294      000A   A DOWN   EQU   $0A    DOWN ARROW
02460 00295      0009   A RIGHT   EQU  $09    RIGHT ARROW
02470 00296      0008   A LEFT    EQU  $08    LEFT ARROW
02480 00297      005F   A SUP    EQU   $5F    SHIFT UP ARROW
02490 00298      005B   A SDOWN  EQU   $5B    SHIFT DOWN ARROW
02500 00299      005D   A SRIGHT EQU  $5D    SHIFT RIGHT ARROW
02510 00300      0015   A SLEFT   EQU  $15    SHIFT LEFT ARROW
02520 00301      0003   A BREAK   EQU  $03    BREAK KEY
02530 00302      000C   A CLEAR   EQU  $0C    CLEAR KEY
02540 00303      005C   A SCLEAR  EQU  $5C    SHIFTED CLEAR
02550 00304      000D   A ENTER   EQU  $0D    ENTER KEY
02560 00305      0040   A AT     EQU   $40    "A" KEY
02570 00306      0013   A SAT    EQU   $13    SHIFTED "A" KEY
*
***** DOS MACRO AND LOGICAL EQUATES *****
* D O S   M A C R O   A N D   L O G I C A L   E Q U A T E S
***** DOS MACR          CALL A DOS FUNCTION
2630 LDA #\1 OPTION
2640 JSR [\0] INDIRECT FUNCTION ADDR
2650 ENDM
*
* EQUATES USED WITH DOS MACRO
*
* THE FOLLOWING USED WITH "OPEN"
0600 A OPEN   EQU   $600   OPEN FUNCTION
02710 00320      0004   A CREATE  EQU   4      ALLOWS FILE CREATION ON OPEN IF NOT FOUND
02720 00321      0008   A EXTEND  EQU   8      ALLOWS EXTENSION OF FILE TO POINT OF ACCESS
02730 00322      0001   A INPUT   EQU   1      USED TO SIGNIFY THAT READS ARE ALLOWED
02740 00323      0001   A IN     EQU   1      SHORTER FORM OF ABOVE
02750 00324      0002   A OUT    EQU   2      ALLOWS WRITES
02760 00325      000E   A OUTPUT  EQU   CREATE+EXTEND+OUT USUAL COMBINATION FOR OUTPUT FILES
02770 00326      0010   A WORK   EQU   16     CAUSES FILE TO BE KILLED WHEN CLOSED (WORK FILE)
02780 00327      0020   A FAST    EQU  32      MINIMIZES FAT REWRITES
02790 00328      0040   A SHARE   EQU  64      USED WHEN 2 OR MORE FILES SHARE THE SAME I/O BUFFER
02800 00329          * EXAMPLES:
* DOS OPEN INPUT    TO READ AN EXISTING FILE
* DOS OPEN OUTPUT   TO CREATE & EXTEND AN OUTPUT FILE
* DOS OPEN IN+OUT   TO UPDATE AN EXISTING FILE (NO EXTENSIONS)
* DOS OPEN INPUT+OUTPUT+WORK TO CREATE, EXTEND, READ & WRITE AND KILL
* WHEN CLOSED (A WORK FILE)
* "SHARE" CAN BE ADDED TO ANY OF THE ABOVE EXAMPLES IF 2 OR MORE FILES
* WILL BE USING THE SAME I/O BUFFER AT THE SAME TIME. THIS OPTION CAUSES
* A PHYSICAL I/O TO REFRESH THE BUFFER WITH EVERY LOGICAL I/O OPERATION.
* WITHOUT THIS OPTION, SEVERAL LOGICAL READS OR WRITES TO OR FROM THE
* SAME PHYSICAL SECTOR CAN BE DONE WITH A SINGLE PHYSICAL I/O. "SHARE"
* INCREASES THE AMOUNT OF ACTUAL I/O ACTIVITY, BUT ALLOWS USE OF MANY
* FILES AT THE SAME TIME WITH MUCH LESS MEMORY REQUIREMENTS FOR BUFFERS.
*
* USED WITH "CLOSE" FUNCTION

```

PAGE 007 DOC .SA:0 DOS - INSTRUCTIONS

```

02950 00344      0602 A CLOSE EQU    $602    CLOSE A FILE OPTIONS NOT USED
02960 00345      0000 A IT     EQU    0
02970 00346          * EXAMPLE:
02980 00347          * DOS CLOSE,IT    TO CLOSE A FILE
02990 00348          *
03000 00349          * USED WITH "READ" AND "WRITE" FUNCTIONS
03010 00350      0604 A READ    EQU    $604    READ A RECORD
03020 00351      0606 A WRITE   EQU    $606    WRITE A RECORD
03030 00352      0001 A RBA     EQU    1    TO READ USING REL BYTE ADDR
03040 00353      0000 A RECORD  EQU    0
03050 00354      0000 A REC     EQU    0
03060 00355      0002 A UPDATE   EQU    2    TO PREVENT ADVANCING REC NBR OR RBA AFTER A READ
03070 00356      0008 A NOW     EQU    8    1 = ENSURE I/O BUFFER IS WRITTEN TO DISK AFTER LOGICAL WRITE
03080 00357          * EXAMPLES:
03090 00358          * DOS READ,RECORD           TO RANDOMLY READ BY RECORD NUMBER
03100 00359          *                                     (FIXED LENGTH RECS ONLY)
03110 00360          *                                     (USE THIS FOR NORMAL SEQUENTIAL READ OF FIXED LENGTH)
03120 00361          * DOS READ,RBA             TO READ THE RECORD POINTED AT BY RBA
03130 00362          *                                     (REQUIRED IF USING VARIABLE LENGTH RECORDS)
03140 00363          * DOS READ,UPDATE           TO READ BY REC NBR WITHOUT ADVANCING REC NBR
03150 00364          * DOS READ,RBA+UPDATE        TO READ THE RECORD POINTED AT BY RBA & NOT CHANGE RBA
03160 00365          * DOS WRITE,REC            WRITE VIA RECORD NUMBER (FIXED LENGTH ONLY)
03170 00366          * DOS WRITE,RBA            WRITE FIXED OR VARIABLE RECORD
03180 00367          * DOS WRITE,UPDATE           UNLIKELY OPTION - WRITES RECORD BUT DOES NOT CHANGE
03190 00368          *                                     RBA OR REC NUMBER. COULD BE REWRITTEN AGAIN.
03200 00369          * DOS WRITE,RBA+NOW          SAME AS: DOS WRITE,RBA FOLLOWED BY DOS RELSE,IT
03210 00370          *
03220 00371      0608 A RELSE    EQU    $608    USE TO RELEASE I/O BUFFER WITHOUT CLOSING FILE
03230 00372          * IF CONTENTS OF BUFFER HAVE BEEN CHANGED, IT IS REWRITTEN. THEN DCBPRN
03240 00373          * IS SET TO $FFFF TO ENSURE A PHYSICAL I/O BEFORE THE NEXT LOGICAL I/O.
03250 00374          * USE THIS FUNCTION WHEN USER IS CONTROLLING A SHARED BUFFER.
03260 00375          * EXAMPLE:
03270 00376          * DOS RELSE,IT
03280 00377          *
03290 00378          * USED WITH OVERLAYABLE FUNCTIONS
03300 00379      060A A DO      EQU    $60A    USE TO LOAD IF NECESSARY, THEN EXECUTE AN OVERLAY
03310 00380      060C A GO      EQU    $60C    USE TO XFER CONTROL FROM ONE OVERLAY TO ANOTHER IN SAME AREA
03320 00381      060E A LOAD    EQU    $60E    USE TO LOAD A SYSTEM OVERLAY - IT IS LOADED AT THE
03330 00382          * EXAMPLE:
03340 00383          * DOS DO,MAP
03350 00384          *
03360 00385          * THE FOLLOWING USED WITH "LOAD" AND "DO" FUNCTIONS
03370 00386      0001 A INIT    EQU    1    INITIALIZATION OF DOS
03380 00387          * EXAMPLE:
03390 00388          * DOS DO,INIT EXIT PROGRAM & RE-INITIALIZE DOS
03400 00389          * NOTE: STACK AND OLYLOC SHOULD BE RESET BEFORE USING THIS OVERLAY
03410 00390          *
03420 00391      000E A MENU    EQU    14    DISPLAY DOS MAIN MENU
03430 00392          * EXAMPLE:
03440 00393          * LDS #STACK
03450 00394          * LDD #OVRLAY WHERE OVERLAY AREA SHOULD START
03460 00395          * STD >OLYLOC
03470 00396          * DOS DO,MENU
03480 00397          *
03490 00398      000A A MAP     EQU    10    DISPLAY BASIC LINES
03500 00399          * EXAMPLE:
03510 00400          * LDD #280 FIRST LINE NUMBER TO BE DISPLAYED
03520 00401          * LDY #283 LAST LINE TO BE DISPLAYED

```

PAGE 008 DOC

.SA:0

DOS - INSTRUCTIONS

```

03530 00402          * LDU <CURSOR STARTING DISPLAY ADDRESS
03540 00403          * (IF STARTING ADDR IS ZERO, SCREEN WILL BE CLEARED FIRST AND ROUTINE
03550 00404          * WILL EXIT WITH U->FIRST CHAR AFTER FIRST LEFT BRACKET ON SCREEN)
03560 00405          * PSHS D,Y,U (PARAMETERS ARE PASSED IN THE STACK)
03570 00406          * DOS DO,BASMSG
03580 00407          * PULS D,Y,U NORMALIZE STACK
03590 00408          * BNE ERROR BRANCH ON ANY FAILURE IF DESIRED
03600 00409          *
03610 00410          0002   A RUNIP EQU    2      KEYIN A NAME AND RUN PGM
03620 00411          * EXAMPLE:
03630 00412          * DOS DO,RUNIP
03640 00413          *
03650 00414          0005   A CPYFLE EQU    5      GET INFO FROM USER & COPY A FILE
03660 00415          * EXAMPLE:
03670 00416          * DOS DO,CPYFLE (IF "GO" USED, DOS MENU FOLLOWS COPY FUNCTION)
03680 00417          *
03690 00418          000B   A FIELDI EQU   11      INPUT A MAPPED FIELD
03700 00419          * EXAMPLE:
03710 00420          * LDX DEST WHERE THE DATA GOES IN MEMORY
03720 00421          * LDU FLDADR POINT TO FIELD ON SCREEN
03730 00422          * DOS DO,FIELDI INPUTS THE FIELD
03740 00423          * B IS RETURNED CONTAINING LAST KEystroke ENTERED
03750 00424          *
03760 00425          000C   A EXEC   EQU   12      GIVEN USRDCB CONTENTS, LOAD ROOT & EXECUTE PROGRAM
03770 00426          * EXAMPLE:
03780 00427          * (WHATEVER LOGIC TO PUT NAME IN DCB AT "USRDCB")
03790 00428          * DOS GO,EXEC JUMP TO LOAD & EXECUTE OVERLAY
03800 00429          *
03810 00430          000D   A REALTM EQU   13      CLOCK DISPLAY OVERLAY (SEE SKEL FOR EXAMPLE OF USE)
03820 00431          *
03830 00432          000F   A BUFPRT EQU   15      BUFFERED PRINT OVERLAY
03840 00433          * EXAMPLE:
03850 00434          * LDU #SIZE (TOTAL MEMORY TO BE USED (ROUTINE + BUFFER)
03860 00435          * (ROUTINE IS ABOUT 220 BYTES)
03870 00436          * DOS DO,BUFPRT (SETS IT UP - OVERLAY & BUFFER PROTECTED FROM
03880 00437          * BEING OVERLAYERED).
03890 00438          * FROM THIS POINT ON, CHARACTERS PRINTED BY CALLING "PRNT" WILL GO
03900 00439          * THROUGH BUFFERED I/O. TO WRAP UP AT EOJ, DO THIS:
03910 00440          * CLR A
03920 00441          * JSR [PRNT] REQUEST TO END BUFFERING.
03930 00442          * THIS WILL CAUSE "PRNT" TO WAIT UNTIL THE BUFFER IS EMPTIED (PRINTER
03940 00443          * HAS CAUGHT UP), AND THEN OVERLAY AND BUFFER AREA ARE RELEASED.
03950 00444          *
03960 00445          0011   A COPY   EQU   17      COPY A FILE
03970 00446          * GIVEN:
03980 00447          * U->SOURCE FILE DCB (NOT OPENED)
03990 00448          * Y->DEST FILE DCB (NOT OPENED)
04000 00449          * B (BIT 0) - OFF IF NO DISKETTE SWAPPING, ON FOR DISKETTE SWAPPING
04010 00450          * RETURNED A=ERROR NUMBER
04020 00451          *
04030 00452          * SIMILAR FUNCTIONS FOR USING USER OVERLAYS
04040 00453          0610   A DOUSR EQU   $610    LOAD IF NECESSARY & EXECUTE USER OVERLAY
04050 00454          0612   A GOUSR EQU   $612    JUMP TO A DIFFERENT OVERLAY
04060 00455          0614   A LODUSR EQU   $614    LOAD USER OVERLAY
04070 00456          * USER SHOULD PROVIDE EQUATES FOR HIS OVERLAYS HERE
04080 00457          *
04090 00458          0616   A ERROR  EQU   $616    JSR HERE FOR DISPLAY OF ERR MSG
04100 00459          *

```

PAGE 009 DOC .SA:0 DOS - INSTRUCTIONS

```

04110 00460      0618 A TIME EQU $61B      TURN ON/OFF TIME ROUTINE
04120 00461      0001 A ON EQU 1
04130 00462      0000 A OFF EQU 0
04140 00463      * EXAMPLE:
04150 00464      * LDU #TMERTN LOAD ADDR OF ROUTINE
04160 00465      * DOS TIME,ON GO ACTIVATE THIS ROUTINE
04170 00466      *
04180 00467      061A A PRNT EQU $61A      PRINT A CHARACTER ON PRINTER
04190 00468      * THIS IS CHANGED BY CALLING BUFFERED PRINTER OVERLAY TO POINT
04200 00469      * AT BUFFERED IO ROUTINE
04210 00470      *
04220 00471      061C A KEYIN EQU $61C      POLL KEYBOARD FOR INPUT CHARACTER
04230 00472      * THIS IS CHANGED BY CALLING BUFFERED KEYBOARD OVERLAY TO POINT
04240 00473      * AT BUFFERED IO ROUTINE
04250 00474      *
04260 00475      061E A BASIC EQU $61E      JMP HERE TO RETURN TO BASIC
04270 00476      *
***** OTHER USEFUL MACROS FOLLOW *****
04280 00477      * OTHER USEFUL MACROS FOLLOW
04290 00478      * OTHER USEFUL MACROS FOLLOW
04300 00479      * OTHER USEFUL MACROS FOLLOW
04310 00480      ENABLI MACR      ENABLE INTERRUPTS
00481      4320 ANDCC #%11101111
00482      4330 ENDM
04340 00483      *
04350 00484      DSABLI MACR      DISABLE INTERRUPTS
00485      4360 ORCC #%01010000
00486      4370 ENDM
04380 00487      *
04390 00488      NEGD MACR      NEGATE D
00489      4400 COMA
00490      4410 COMB
00491      4420 ADDD #1
00492      4430 ENDM
04440 00493      *
04450 00494      LSRD MACR      LOGICAL SHIFT RIGHT D
00495      4460 LSRA
00496      4470 RORB
00497      4480 ENDM
04490 00498      *
04500 00499      LSLD MACR      LOGICAL SHIFT LEFT D
00500      4510 LSLB
00501      4520 ROLA
00502      4530 ENDM
04540 00503      *
04550 00504      CLRD MACR      CLEAR D
00505      4560 CLRA
00506      4570 CLR8
00507      4580 ENDM
04590 00508      *
04600 00509      INC8 MACR      ADD 1 TO D
00510      4610 ADDD #1
00511      4620 ENDM
04630 00512      *
***** SYSTEM RAM - DOS *****
04640 00513      * SYSTEM RAM - DOS
04650 00514      *
04660 00515      *
04670 00516      * ADDITIONAL WS USING EXTENDED ADDRESSING
04680 00517A 015E      ORG $15E

```

PAGE 010 DOC .SA:0 DOS - INSTRUCTIONS

```

04690 00518A 0600      ORG    $600
04700 00519               * AREA WHERE USER ACCESSABLE VECTORS & VARIABLES STORED
04710 00520A 0600      0020   A VECTOR RMB 2*16   2 BYTES PER VECTOR
04720 00521               * OPEN  OPEN A DISK FILE
04730 00522               * CLOSE CLOSE A DISK FILE
04740 00523               * READ  READ FROM A DISK FILE
04750 00524               * WRITE WRITE TO A DISK FILE
04760 00525               * RELEASE I/O BUFFER (ALLOW USE FOR ANOTHER FILE)
04770 00526               * DO    LOAD & EXECUTE A SYSTEM OVERLAY
04780 00527               * GO    LOAD ON TOP OF CURRENT OVERLAY & JUMP TO SYSTEM OVERLAY
04790 00528               * LOAD  LOAD SYSTEM OVERLAY
04800 00529               * DOUSR LOAD & EXECUTE USER OVERLAY
04810 00530               * GOUSR LOAD ON TOP OF CURRENT OVERLAY & JUMP TO USER OVERLAY
04820 00531               * LODUSR LOAD USER OVERLAY
04830 00532               * ERROR DISPLAY ERROR NUMBER IN "A"
04840 00533               * TIME  TURN ON/OFF TIME INTERVAL ROUTINE
04850 00534               * PRNT  PRINT A CHARACTER ON PRINTER
04860 00535               * KEYIN INPUT NEXT KEYSTROKE FROM KEYBOARD
04870 00536               * BASIC RETURN TO BASIC CONTROL

04880 00537A 0620      0002   A CLOCK RMB 2    COUNT OF 60THS OF A SECOND
04890 00538A 0622      0001   A RETRY5 RMB 1    NUMBER OF I/O RETRYS INITIALLY SET TO 5
04900 00539A 0623      0002   A RATE RMB 2    TIME CONSTANT THAT CONTROLS PRINTER TRANSMISSION SPEED
04910 00540A 0625      0002   A OLYLOC RMB 2    ADDRESS WHERE CURRENT OVERLAY WAS LOADED
04920 00541A 0627      0002   A USRBSE RMB 2    BASE OF USER'S ROOT + 1. POINTS TO ENTRY ZERO OF OVERLAY'S RBA'
04930 00542A 0629      0002   A HOOK1 RMB 2    JUST BEFORE CHECKING FOR AUTO EXECUTE
04940 00543A 0628      0002   A HOOK2 RMB 2    JUST BEFORE BRANCHING TO USER PROGRAM
04950 00544A 062D      0002   A HOOK3 RMB 2
04960 00545A 062F      0002   A HOOK4 RMB 2
04970 00546A 0631      0002   A HOOK5 RMB 2
04980 00547A 0633      0002   A RETURN RMB 2    CONTAINS TWO RTS CODES - ALL HOOKS RETURN THRU HERE
04990 00548A 0635      0031   A DOSDCB RMB DCBSZ  DCB USED TO READ SYSTEM OVERLAYS
05000 00549A 0666      0031   A MSGDCB RMB DCBSZ  DCB USED TO READ "MAPS" AND MESSAGES
05010 00550A 0697      0031   A USRDCB RMB DCBSZ  DCB USED TO READ USER'S PROGRAM & OVERLAYS
05020 00551A 06C8      0100   A SYSBUF RMB 256   BUFFER FOR SYSTEM USE(DIRECTORY + FAT READS & WRITES)
05030 00552               0045   A FATSZ EQU 69   FILE ALLOCATION TABLE (FAT) SIZE
05040 00553A 07C8      0045   A FAT0 RMB FATSZ  SAVE AREA FOR DRIVE 0 FAT TABLE
05050 00554A 080D      0045   A FAT1 RMB FATSZ
05060 00555A 0852      0045   A FAT2 RMB FATSZ
05070 00556A 0897      0045   A FAT3 RMB FATSZ
05080 00557               07C8   A FATS EQU FAT0
05090 00558A 08DC      0002   A MAXMEM RMB 2    ADDR OF HIGHEST USEABLE MEMORY
05100 00559A 08DE      0001   A DRIVES RMB 1    MAX NBR OF DRIVES TO SEARCH ON GLOBAL OPEN
05110 00560               OPT   L
05120 00561A 08DF      0001   A ENDWSR RMB 1    END OF EXTENDED WS
05130 00562               OPT   NOL
***** D O S S T A R T S H E R E *****
* D O S S T A R T S H E R E
***** ORG ORGIN SEE 1ST MODULE FOR VALUE ASSIGNED
00010 00567               OPT   L
00020 00568               TTL   DOS - I/O ROUTINES
00030 00569               OPT   NOL
***** O P E N D I S K F I L E *****
* GIVEN:
* A=DESIRED FILE STATUS
* U->DCB

```

PAGE 011 IO .SA:0 DOS - I/O ROUTINES

```

00100 00576          * DCBDRV,U = DRIVE TO BE CHECKED ($FF=CHECK ALL DRIVES)
00110 00577          *
00120 00578          * BEFORE CALLING "OPEN", DCB SHOULD CONTAIN: FILENAME, EXTENSION,
00130 00579          * I/O BUFFER ADDRESS. NAME AND EXTENSION ONLY ARE COMPARED
00140 00580          * TO DIRECTORY ENTRIES TO FIND MATCH. TYPE AND ASCII FLAG ARE USED ONLY
00150 00581          * WHEN CREATING FILE (OTHERWISE THEY ARE OVERLAYERD BY EXISTING VALUES).
00160 00582          * ALL I/O NEEDED TO OPEN FILE USES THE 256 BYTE AREA POINTED TO BY
00170 00583          * LAST I/O ADDRESS AS A BUFFER.
00180 00584          *
00190 00585          * OPEN WORKS EXACTLY THE SAME FOR INPUT OR OUTPUT! ACTION IS CONTROLLED
00200 00586          * BY FILE STATUS SUPPLIED IN "A" (SEE DCBCFS IN DCB DESCRIPTION).
00210 00587          *
00220 00588          * OPENING A NON-EXISTANT FILE - IF CREATION IS ALLOWED, FIRST 32 BYTES OF
00230 00589          * DCB ARE PLACED IN DIRECTORY EXCEPT THAT DCBFCL IS SET TO $FF, DCBNLS
00240 00590          * IS SET TO ZERO AND DCBCFS IS SET TO PROVIDED STATUS.
00250 00591          *
00260 00592          * OPENING AN EXISTING FILE - THE 32 BYTE DIRECTORY ENTRY OVERLAYS THE
00270 00593          * FIRST 32 BYTES OF THE DCB EXCEPT FOR DCBCFS WHICH IS SET TO THE PROVIDED
00280 00594          * VALUE.
00290 00595          *
00300 00596          * WHEN FILE IS OPENED, DCBPRN IS SET TO $FFFF (AN INVALID VALUE), DCBRBA
00310 00597          * IS SET TO ZERO, AND DCBLRN IS SET TO ZERO. AT ANYTIME BEFORE OR AFTER
00320 00598          * CALLING OPEN, DCBLRB CAN BE SET OR CHANGED.
00330 00599          *
00340 00600          * FILE TYPE AND ASCII FLAG CAN BE CHANGED AFTER OPEN TO CAUSE THEM TO BE
00350 00601          * CHANGED WHEN FILE IS CLOSED.
00360 00602          *
00370 00603          ****
00380 00604A 0989 E6  C8 21  A DOPEN LDB DCBDRV,U
00390 00605A 098C 34  16   A PSHS D,X
00400 00606A 098E C1  FF   A CMPB #$FF REQUEST FOR SCAN OF ALL DRIVES
00410 00607A 0990 27  0A   099C BEQ D00 IF YES
00420 00608A 0992 C1  04   A CMPB #4 VALID DRIVE REQUESTED?
00430 00609A 0994 25  07   099D BCS D01 IF YES
00440 00610A 0996 86  09   A LDA #ERR9 PARAMETER ERROR
00450 00611A 0998 A7  E4   A DOERR STA ,S
00460 00612A 099A 35  '96  A PULS D,X,PC RETURN WITH ERROR CONDITION
00470 00613A 099C 5F      D00 CLR8 START WITH DRIVE ZERO
00480 00614A 099D E7  C8 21  A D01 STB DCBDRV,U
00490 00615A 09A0 4F      CLRA SAY LOOK FOR MATCH
00500 00616A 09A1 17  02D2 0C76 LBSR CHKDIR CHECK DIRECTORY ON THIS DRIVE FOR MATCH
00510 00617A 09A4 27  50   09F6 BEQ D05 IF MATCH FOUND
00520 00618A 09A6 2B  08   0980 BMI D03 IF NO I/O ERRORS - JUST DIDNT FIND IT
00530 00619          * IT WAS SOME KIND OF I/O ERROR
00540 00620A 09AB 81  01   A CMPA #1 DRIVE NOT READY?
00550 00621A 09AA 26  EC   0998 BNE DOERR IF NO
00560 00622A 09AC 6D  61   A TST 1,S REQUEST FOR SPECIFIC DRIVE?
00570 00623A 09AE 2A  E8   0998 BPL DOERR IF YES, THEN THIS IS AN ERROR
00580 00624A 09B0 6D  61   A D03 TST 1,S REQUEST FOR SPECIFIC DRIVE?
00590 00625A 09B2 2A  09   09BD BPL D04 IF YES, I DIDNT FIND HIS FILE
00600 00626A 09B4 E6  C8 21  A LDB DCBDRV,U LAST DRIVE CHECKED
00610 00627A 09B7 5C      INCB
00620 00628A 09B8 F1  0BDE  A CMPB DRIVES ANOTHER VALID DRIVE TO CHECK?
00630 00629A 09B8 25  E0   099D BCS D01 IF YES
00640 00630          * MATCH NOT FOUND - IS IT OK TO CREATE?
00650 00631A 09BD A6  E4   A D04 LDA ,S (DESIRED STATUS)
00660 00632A 09BF B5  04   A BITA #CREATE CREATE BIT ON?
00670 00633A 09C1 26  04   09C7 BNE D04A IF YES

```

PAGE 012 IO .SA:0 DOS - I/O ROUTINES

```

00680 00634A 09C3 B6 0A A LDA #ERR10 FILE DIRECTORY ENTRY NOT FOUND
00690 00635A 09C5 20 D1 0998 BRA DOERR
00700 00636A 09C7 6D 61 A D04A TST 1,S ANY DRIVE SPECIFIED?
00710 00637A 09C9 2A 03 09CE BPL D04B IF SPECIFIC
00720 00638A 09CB 6F C8 21 A CLR DCBDRV,U CREATE ON DRIVE ZERO
00730 00639A 09CE B6 FF A D04B LDA #$FF SAY LOOK FOR OPEN SLOT
00740 00640A 09D0 17 02A3 0C76 LBSR CHKDIR SCAN THE DIRECTORY
00750 00641A 09D3 27 06 09DB BEQ D04C IF SLOT FOUND
00760 00642A 09D5 2A C1 0998 BPL DOERR IF SOME KIND OF I/O ERROR
00770 00643A 09D7 B6 0B A LDA #ERR11 DIRECTORY IS FULL
00780 00644A 09D9 20 BD 0998 DOERRL BRA DOERR
00790 00645A 09DB A6 E4 A D04C LDA ,S DESIRED STATUS
00800 00646A 09DD A7 61 A STA 1,S SAVE IT
00810 00647A 09DF B6 0C A LDA #ERR12 SAY DIRECTORY WAS CREATED
00820 00648A 09E1 A7 E4 A STA ,S
00830 00649A 09E3 B6 FF A LDA #$FF
00840 00650A 09E5 A7 4D A STA DCBFCL,U SET NUMBER OF 1ST CLUSTER
00850 00651A 09E7 CLRD
00860 00652A 09E9 ED 4E A STD DCBNLS,U CLEAR BYTES IN LAST SECTOR
00870 00653A 09EB ED C8 14 A STD DCBMRB,U CLEAR MAX RBA
00880 00654A 09EE A7 C8 16 A STA DCBMRB+2,U
00890 00655A 09F1 17 0263 0C57 LBSR DCBDIR XFER DATA TO DIRECTORY
00900 00656A 09F4 20 18 0A0E BRA D06 GO CONTINUE PROCESSING
00910 00657 * DIRECTORY ENTRY FOUND
00920 00658A 09F6 A6 E4 A D05 LDA ,S DESIRED STATUS
00930 00659A 09F8 A7 61 A STA 1,S SAVE IT
00940 00660A 09FA 6F E4 A CLR ,S
00950 00661A 09FC A6 88 10 A LDA DCBCFS,X CHK PREVIOUS FILE STATUS
00960 00662A 09FF 27 0D 0A0E BEQ D06 IF IT WAS CLOSED
00970 00663A 0A01 84 0E A ANDA #CREATE+EXTEND+OUT IF LAST OPENED TO MODIFICATION?
00980 00664A 0A03 27 09 0A0E BEQ D06 IF NO
00990 00665A 0A05 6D 88 10 A TST DCBCFS,X CHK PREVIOUS FILE STATUS
01000 00666A 0A08 27 04 0A0E BEQ D06 IF IT WAS CLOSED
01010 00667A 0A0A B6 0D A LDA #ERR13 SAY IT WASN'T PREVIOUSLY CLOSED
01020 00668A 0A0C A7 E4 A STA ,S
01030 00669 * XFER DIRECTORY ENTRY TO DCB
01040 00670A 0A0E A6 61 A D06 LDA 1,S DESIRED STATUS
01050 00671A 0A10 A7 88 10 A STA DCBCFS,X PUT IN DIRECTORY ENTRY
01060 00672A 0A13 17 0249 0C5F LBSR DIRDCB XFER DIRECTORY ENTRY TO DCB
01070 00673A 0A16 A6 C8 10 A LDA DCBCFS,U
01080 00674A 0A19 B4 0E A ANDA #CREATE+EXTEND+OUT WRITES ALLOWED?
01090 00675A 0A1B 27 05 0A22 BEQ D06A IF NO
01100 00676A 0A1D 17 031F 0D3F LBSR SYSWRT REWRITE DIRECTORY RECORD
01110 00677A 0A20 26 B7 09D9 BNE DOERRL IF I/O ERROR
01120 00678A 0A22 B6 02 A D06A LDA #2
01130 00679A 0A24 A7 C8 23 A STA DCBSEC,U
01140 00680A 0A27 17 02FD 0D27 LBSR SYSRED READ FAT RECORD
01150 00681A 0A2A 26 AD 09D9 BNE DOERRL
01160 00682A 0A2C 17 021C 0C4B LBSR ADRFAT POINT "X" AT FAT TABLE IN MEMORY
01170 00683A 0A2F 34 40 A PSHS U
01180 00684A 0A31 CE 06C8 A LDU #SYSBUF POINT TO BUFFER
01190 00685A 0A34 C6 45 A LDB #69 BYTES TO MOVE
01200 00686A 0A36 17 022E 0C67 LBSR XFRUX MOVE THEM
01210 00687A 0A39 35 40 A PULS U
01220 00688 * DO OPEN RESETTING
01230 00689A 0A3B CC FFFF A LDD #$FFFF
01240 00690A 0A3E ED C8 29 A STD DCBPRN,U
01250 00691A 0A41 CLRD

```

PAGE 013 IO .SA:0 DOS - I/O ROUTINES

```

01260 00692A 0A43 ED C8 2B A STD DCBRBA,U
01270 00693A 0A46 A7 C8 2D A STA DCBRBA+2,U
01280 00694A 0A49 ED C8 2E A STD DCBLRN,U
01290 00695A 0A4C 6F C8 30 A CLR DCBMDT,U
01300 00696A 0A4F 16 008C 0ADE LBRA DC5
01310 00697 *
01320 00698 ****
01330 00699 * CLOSE DISK FILE
01340 00700 *
01350 00701 * GIVEN: U -> DCB (CONTAINING FILE STATUS)
01360 00702 *
01370 00703 * FUNCTION:
01380 00704 * FIND DIRECTORY ENTRY AND VERIFY THAT FILE IS OPEN. THEN, IF FILE IS
01390 00705 * TO BE KEPT, UPDATE AND RE-WRITE DIRECTORY ENTRY AND REWRITE FAT TABLE.
01400 00706 * IF FILE IS TO BE PURGED, MARK DIRECTORY ENTRY AS RE-USEABLE AND RE-WRITE
01410 00707 * THEN MARK CLUSTERS AVAILABLE IN FAT TABLE AND REWRITE.
01420 00708 ****
01430 00709A 0A52 4F DCLOSE CLRA (RESULT CODE)
01440 00710A 0A53 34 16 A PSHS D,X
01450 00711A 0A55 4F CLRA SAY LOOK FOR A MATCH
01460 00712A 0A56 17 021D 0C76 LBSR CHkdir CHECK DIRECTORY FOR A MATCH
01470 00713A 0A59 27 07 0A62 BEQ DC1 IF MATCH FOUND
01480 00714A 0A5B 2A 02 0A5F BPL DCERR IF I/O ERR
01490 00715A 0A5D 86 0A A LDA #ERR10 DIRECTORY ENTRY NOT FOUND
01500 00716A 0A5F 16 FF36 0998 DCERR LBRa DOERR
01510 00717A 0A62 A6 C8 10 A DC1 LDA DCBCFS,U IS FILE OPEN?
01520 00718A 0A65 26 04 0A6B BNE DC2
01530 00719A 0A67 B6 13 A LDA #ERR19 CLOSING UNOPENED FILE
01540 00720A 0A69 20 F4 0A5F BRA DCERR
01550 00721A 0A6B EC C8 22 A DC2 LDD DCBTRK,U
01560 00722A 0A6E 34 06 A PSHS D SAVE LOC OF DIR ENT
01570 00723A 0A70 17 02D0 0D43 LBSR REWRTE REWRITE BUFFER IF IT HAD BEEN MODIFIED
01580 00724A 0A73 35 06 A PULS D
01590 00725A 0A75 26 E8 0A5F BNE DCERR IF I/O ERROR OCCURRED IN THE PROCESS
01600 00726A 0A77 ED C8 22 A STD DCBTRK,U RESTORE LOC OF DIR ENT
01610 00727A 0A74 A6 C8 10 A LDA DCBCFS,U
01620 00728A 0A7D 34 02 A PSHS A SAVE FOR DIRECTORY RE-WRITE DECISION
01630 00729A 0A7F 6F C8 10 A CLR DCBCFS,U CLEAR CUR FILE STATUS IN DCB
01640 00730A 0A82 B4 10 A ANDA #WORK WORK FILE TO BE DELETED?
01650 00731A 0A84 27 18 0A9E BEQ DC4 IF NO GO REWRITE DIRECTORY & FAT TABLE
01660 00732A 0A86 6F C4 A CLR ,U MARK DIRECTORY ENTRY AS RE-USEABLE
01670 00733A 0A88 34 10 A PSHS X SAVE ADDR OF DIRECTORY ENTRY
01680 00734 * MARK FAT TABLE ENTRIES AS AVAILABLE
01690 00735A 0A8A 17 01BE 0C4B LBSR ADRFAT POINT "X" AT FAT TABLE IN MEM
01700 00736A 0ABD A6 4D A LDA DCBFCL,U GET FIRST CLUSTER NUMBER
01710 00737A 0ABF 2B 0B 0A9C BMI DC3A IF NO CLUSTERS IN USE
01720 00738A 0A91 E6 86 A DC3 LDB A,X GET NUMBER OF NEXT CLUSTER
01730 00739A 0A93 6F 86 A CLR A,X CLEAR CLUSTER ENTRY
01740 00740A 0A95 6A 86 A DEC A,X SET TO $FF
01750 00741A 0A97 1F 98 A TFR B,A
01760 00742A 0A99 4D TSTA
01770 00743A 0A9A 2A F5 0A91 BPL DC3 IF MORE TO GO
01780 00744A 0A9C 35 10 A DC3A PULS X ADDR OF DIR ENTRY
01790 00745A 0A9E 17 01B6 0C57 DC4 LBSR DCBDIR XFER TO DIRECTORY
01800 00746A 0AA1 35 02 A PULS A PRE-CLOSE CFS
01810 00747A 0AA3 84 0E A ANDA #CREATE+EXTEND+OUT WRITES ALLOWED?
01820 00748A 0AA5 27 15 0ABC BEQ DC4B
01830 00749 * SET DCBNLS TO REFLECT DCBMRB (MAX RBA)

```

PAGE 014 IO .SA:0 DOS - I/O ROUTINES

```

01840 00750A 0AA7 4F           CLRA
01850 00751A 0AA8 E6           LDB   DCBMRB+2,X
01860 00752A 0AA9 26           BNE   DC4A
01870 00753A 0AAD EC           LDD   DCBMRB,X IS IT A NULL FILE
01880 00754A 0AB0 27           BEQ   DC4A IF YES
01890 00755A 0AB2 CC           0100  A
01900 00756A 0AB5 ED           STD   DCBNLS,X
01910 00757A 0AB7 17           0285 0D3F LBSR  SYSWRT RE-WRITE DIRECTORY RECORD
01920 00758A 0ABA 26           BNE   DCERR IF I/O ERROR
01930 00759A 0ABC 17           018C 0C4B DC4B LBSR  ADRFAT
01940 00760A 0ABF 34           40    A PSHS  U SAVE DCB ADDR
01950 00761A 0AC1 CE           06C8  A LDU   #SYSBUF POINT TO SYSTEM'S BUFFER
01960 00762A 0AC4 C6           45    A LDB   #69
01970 00763A 0AC6 17           01A6 0C6F LBSR  XFRXU XFER INTO BUFFER
01980 00764A 0AC9 30           41    A LEAX  1:U
01990 00765A 0ACB C6           BA    A LDB   #256-69-1
02000 00766A 0ACD 17           0197 0C67 LBSR  XFRUX CLEAR REST OF BUFFER TO $FF
02010 00767A 0AD0 35           40    A PULS  U RESTORE DCB ADDR
02020 00768A 0AD2 86           02    A LDA   #2
02030 00769A 0AD4 A7           CB    23   A STA   DCBSEC,U
02040 00770A 0AD7 17           0265 0D3F LBSR  SYSWRT WRITE IT
02050 00771A 0ADA 27           02    0ADE BEQ   DC5
02060 00772A 0ADC A7           E4    A STA   ,S IF I/O ERROR
02070 00773A 0ADE 6D           E4    A DC5   TST   ,S SET COND CODES
02080 00774A 0AE0 35           96    A PULS  D:X,PC
02090 00775
*
***** READ A LOGICAL DISK RECORD *****
02110 00777
02120 00778
02130 00779
* GIVEN: U -> DCB (THAT HAS ALREADY BEEN OPENED!)
* A = FUNCTION DESIRED CODED AS FOLLOWS:
02140 00780
02150 00781
02160 00782
02170 00783
02180 00784
02190 00785
02200 00786
02210 00787
02220 00788
02230 00789
02240 00790
02250 00791
02260 00792
02270 00793
02280 00794
02290 00795
02300 00796
02310 00797
02320 00798
***** RECORD STORAGE ADDRESS AND I/O BUFFER *****
02330 00799A 0AE2 34           32    A DREAD PSHS  A:X,Y
02340 00800A 0AE4 CC           010F  A LDD   #$0100+ERR15
02350 00801A 0AE7 17           0091 0B7B LBSR  RDWR  DO SETUP COMMON TO READ AND WRITE
02360 00802
*
* LOOP TO XFER BYTES TO RECORD AREA
* (X->BUFFER, Y->RECORD AREA)
02380 00803
02390 00805A 0AEA E6           CB    2D   A DR5   LDB   DCBRBA+2,U DISPLACEMENT IN CURRENT SECTOR
02400 00806A 0AED 4F           CLRA
02410 00807A 0AEE A6           BB    A LDA   D,X     GET A BYTE

```

PAGE 015 IO .SA:0 DOS - I/O ROUTINES

```

02420 00808A 0AF0 A7 A0 A STA ,Y+ STORE IN RECORD AREA
02430 00809A 0AF2 6C CB 2D A INC DCBRBA+2,U ADVANCE POINTER IN BUFFER
02440 00810A 0AF5 26 1E 0B15 BNE DR5B IF IN SAME SECTOR
02450 00811A 0AF7 17 0249 0D43 LBSR REWRTE ENSURE PREVIOUSLY MODIFIED DATA GETS WRITTEN
02460 00812A 0AFA 26 0E 0B0A BNE DR5AA IF WRITE ERR
02470 00813A 0AFC EC CB 2B A LDD DCBRBA,U
02480 00814A 0AFF C3 0001 A ADDD #1 POINT TO NEXT SECTOR
02490 00815A 0B02 ED CB 2B A STD DCBRBA,U
02500 00816A 0B05 17 0291 0D99 LBSR CALSEC RECALCULATE TRACK & SECTOR
02510 00817A 0B08 27 06 0B10 BEQ DR5A IF OK
02520 00818A 0B0A 32 67 A DR5AA LEAS 7,S SCRAP STUFF IN STACK
02530 00819A 0B0C A7 E4 A STA ,S
02540 00820A 0B0E 35 B2 A PULS A,X,Y,PC
02550 00821A 0B10 17 01D4 0CE7 DR5A LBSR DSKRED
02560 00822A 0B13 26 F5 0B0A BNE DR5AA IF I/O ERROR
02570 00823A 0B15 EC E4 A DR5B LDD ,S GET COUNT DOWN VALUE
02580 00824A 0B17 27 09 0B22 BEQ DR5C IF VARIABLE LENGTH STRING
02590 00825A 0B19 B3 0001 A SUBD #1
02600 00826A 0B1C ED E4 A STD ,S
02610 00827A 0B1E 26 CA 0AEA BNE DR5 GO GET ANOTHER CHR
02620 00828A 0B20 20 07 0B29 BRA RDWRX GO DO CLEAN-UP COMMON TO READ AND WRITE
02630 00829A 0B22 A6 CB 13 A DR5C LDA DCBTRM,U STRING DELIMITER
02640 00830A 0B25 A1 3F A CMPA -1,Y WAS LAST CHR STORED A DELIMITER?
02650 00831A 0B27 26 C1 0AEA BNE DR5 IF NO, KEEP GOING
02660 00832 * ****
02670 00833 * ****
02680 00834 * CLEAN UP COMMON TO READ AND WRITE
02690 00835 * ****
02700 00836 * RECORD HAS BEEN READ - CLEAN UP
02710 00837A 0B29 35 06 A RDWRX PULS D
02720 00838A 0B2B A6 CB 10 A LDA DCBCFS,U FILE STATUS
02730 00839A 0B2E 85 40 A BITA #SHARE OPTION SET?
02740 00840A 0B30 27 08 0B3A BEQ DR6A IF NO
02750 00841A 0B32 17 020E 0D43 DR6AA LBSR REWRTE FREE UP BUFFER
02760 00842A 0B35 CC FFFF A LDD #$FFFF MARK INVALID SECTOR IN BUFFER
02770 00843A 0B38 20 09 0B43 BRA DR6B
02780 00844A 0B3A A6 65 A DR6A LDA 5,S R/W OPTION
02790 00845A 0B3C 85 08 A BITA #NOW REWRITE NOW?
02800 00846A 0B3E 26 F2 0B32 BNE DR6AA IF YES
02810 00847A 0B40 EC CB 2B A LDD DCBRBA,U LAST SECTOR ACCESSED
02820 00848A 0B43 ED CB 29 A DR6B STD DCBPRN,U MARK WHICH SECTOR IS NOW IN BUFFER
02830 00849 * CHECK FOR NEW DCBMRB
02840 00850A 0B46 EC CB 2B A LDD DCBRBA,U
02850 00851A 0B49 10A3 CB 14 A CMPD DCBMRB,U
02860 00852A 0B4D 25 16 0B65 BCS DR6D IF IN A LOWER SECTOR
02870 00853A 0B4F 26 08 0B59 BNE DR6C IF A HIGHER SECTOR
02880 00854A 0B51 A6 CB 2D A LDA DCBRBA+2,U
02890 00855A 0B54 A1 CB 16 A CMPA DCBMRB+2,U
02900 00856A 0B57 25 0C 0B65 BCS DR6D IF A LOWER BYTE
02910 00857A 0B59 EC CB 2B A DR6C LDD DCBRBA,U
02920 00858A 0B5C ED CB 14 A STD DCBMRB,U
02930 00859A 0B5F A6 CB 2D A LDA DCBRBA+2,U
02940 00860A 0B62 A7 CB 16 A STA DCBMRB+2,U
02950 00861A 0B65 A6 65 A DR6D LDA 5,S READ/WRITE OPTION
02960 00862A 0B67 B4 02 A ANDA #UPDATE SHOULD RBA & LRN BE RESET TO STARTING VALUE?
02970 00863A 0B69 35 32 A PULS A,X,Y
02980 00864A 0B6B 27 0A 0B77 BEQ DR6E IF NO
02990 00865 * RESTORE ORIGINAL POINTERS

```

PAGE 016 IO .SA:0 DOS - I/O ROUTINES

```

03000 00866A 0B6D A7 C8 2D A STA DCBRBA+2,U
03010 00867A 0B70 AF C8 2B A STX DCBRBA,U
03020 00868A 0B73 10AF C8 2E A STY DCBLRN,U
03030 00869A 0B77 6F E4 A DR6E CLR ,S
03040 00870A 0B79 35 B2 A PULS A,X,Y,PC
03050 00871 *
*****SETUP FOR READ OR WRITE*****
03070 00873 * SETUP FOR READ OR WRITE
03080 00874 * GIVEN: A=1 FOR READ, 2 FOR WRITE
03090 00875 * B=ERR NBR FOR POSSIBLE USE
03100 00876 ****
03110 00877A 0B7B 34 06 A RDWR PSHS D SAVE IN CASE NEEDED
03120 00878 * IS FILE OPEN?
03130 00879A 0B7D A6 C8 10 A LDA DCBCFS,U
03140 00880A 0B80 26 08 0B8A BNE RDWR1 IF YES
03150 00881A 0B82 86 0E A LDA #ERR14 IF NOT OPEN
03160 00882A 0B84 32 64 A RDWRER LEAS 4,S (DIDN'T NEED IT AND RET ADDR)
03170 00883A 0B86 A7 E4 A STA ,S
03180 00884A 0B88 35 B2 A PULS A,X,Y,PC
03190 00885 * IS THIS TYPE OF OPERATION ALLOWED (READ OR WRITE)?
03200 00886A 0B8A A5 E4 A RDWR1 BITA ,S (1 FOR READ, 2 FOR WRITE)
03210 00887A 0B8C 26 04 0B92 BNE RDWR2 IF YES
03220 00888A 0B8E A6 61 A LDA 1,S (ERROR NUMBER PROVIDED)
03230 00889A 0B90 20 F2 0B84 BRA RDWRER
03240 00890 *
03250 00891 * CHECK FOR STARTING RBA
03260 00892 * IS I/O BY RBA OR LOGICAL RECORD?
03270 00893A 0B92 A6 64 A RDWR2 LDA 4,S OPTION PROVIDED
03280 00894A 0B94 84 01 A ANDA #RBA
03290 00895A 0B96 26 0E 0BAA BNE RDWR4 IF READ VIA RBA, USE RBA'S CURRENT CONTENTS
03300 00896 * READ BY RECORD NUMBER
03310 00897A 0B98 EC C8 11 A LDD DCBRSZ,U FIXED OR VARIABLE LENGTH RECORDS?
03320 00898A 0B9B 26 04 0BA1 BNE RDWR3 IF FIXED LENGTH
03330 00899A 0B9D 86 14 A LDA #ERR20 CANT CALCULATE - RSZ = ZERO
03340 00900A 0B9F 20 E3 0B84 BRA RDWRER
03350 00901A 0BA1 17 01B3 0D57 RDWR3 LBSR CALRBA CALCULATE RECORD'S STARTING RBA
03360 00902A 0BA4 26 DE 0B84 BNE RDWRER IF OVERFLOW OCCURRED
03370 00903 *
03380 00904 OPT L
03390 00905 * MAKE SURE STARTING RECORD IS IN BUFFER
03400 00906A 0BA6 EC C8 2B A RDWR4 LDD DCBRBA,U (RELATIVE RECORD NEEDED)
03410 00907A 0BA9 10A3 CB 29 A CMPD DCBPRN,U IS NEEDED RECORD IN BUFFER?
03420 00908A 0BAD 27 11 0BC0 BEQ RDWR4A IF YES
03430 00909A 0BAF 17 0191 0D43 LBSR REWRT REWRITE BUFFER IF IT HAS BEEN MODIFIED
03440 00910A 0BB2 26 D0 0B84 BNE RDWRER IF I/O ERROR IN THE PROCESS
03450 00911A 0BB4 17 01E2 0D99 LBSR CALSEC CALCULATE TRACK & SECTOR
03460 00912A 0BB7 26 CB 0B84 BNE RDWRER IF TRYING TO GO BEYOND EOF
03470 00913A 0BB9 17 012B 0CE7 LBSR DSKRED READ THE SECTOR
03480 00914A 0BBC 26 C6 0B84 BNE RDWRER IF I/O ERR
03490 00915A 0BBE 20 05 0BC5 BRA RDWR5
03500 00916A 0BC0 17 01DC 0D9F RDWR4A LBSR CSENT CHECK FOR EOF
03510 00917A 0BC3 26 BF 0B84 BNE RDWRER IF TRYING TO GO PAST EOF
03520 00918 *
03530 00919 * CORRECT STARTING SECTOR IS IN BUFFER
03540 00920 * GET SET TO XFER RECORD
03550 00921A 0BC5 35 26 A RDWR5 PULS D,Y (D=1/0, ERR NBR, Y = RETURN ADDR)
03560 00922 OPT NOL
03570 00923A 0BC7 AE C8 2E A LDX DCBLRN,U

```

PAGE 017 IO .SA:0 DOS - I/O ROUTINES

```

03580 00924A 0BCA 34 10 A PSHS X SAVE IN CASE POINTERS DON'T ADVANCE
03590 00925A 0BCC 30 01 A LEAX 1,X POINT TO NEXT RECORD
03600 00926A 0BCE AF CB 2E A STX DCBLRN,U
03610 00927A 0BD1 AE CB 2B A LDX DCBRBA,U
03620 00928A 0BD4 A6 CB 2D A LDA DCBRBA+2,U
03630 00929A 0BD7 34 12 A PSHS A,X SAVE INCASE POINTERS DON'T ADVANCE
03640 00930A 0BD9 EC CB 11 A LDD DCBRSZ,U GET RECORD LENGTH
03650 00931A 0BDC 34 06 A PSHS D SAVE AS COUNT DOWN VALUE FOR LOOP
03660 00932A 0BDE 34 20 A PSHS Y SAVE RET ADDR
03670 00933A 0BE0 AE CB 24 A LDX DCBBUF,U ADDR OF BUFFER
03680 00934A 0BE3 10AE CB 27 A LDY DCBLRB,U ADDR OF LOGICAL RECORD BUFFER
03690 00935A 0BE7 39 RTS RETURN TO READ OR WRITE LOOP
03700 00936 *
***** W R I T E A L O G I C A L D I S K R E C O R D ****
03720 00938 * W R I T E A L O G I C A L D I S K R E C O R D
03730 00939 *
03740 00940 * GIVEN: U -> DCB (THAT HAS ALREADY BEEN OPENED!)
03750 00941 * A = FUNCTION DESIRED CODED AS FOLLOWS:
03760 00942 * BIT 0 ON TO WRITE VIA RBA
03770 00943 * OFF TO WRITE VIA LRN
03780 00944 * BIT 1 ON TO WRITE WITHOUT CHANGING POINTER
03790 00945 * OFF TO EXIT AFTER POINTING AT NEXT (PREVIOUS) RECORD
03800 00946 * BIT 2 ON TO WRITE BACKWARDS
03810 00947 * OFF TO WRITE FORWARD
03820 00948 * BIT 3 ON TO RELEASE BUFFER AFTER WRITE
03830 00949 * OFF TO WAIT UNTIL PHYSICAL I/O IS NECESSARY
03840 00950 * NOTE: FUNCTION IS NEARLY THE SAME AS DREAD - SEE NOTES UNDER DREAD.
03850 00951 ****
03860 00952A 0BEB 34 32 A DWRITE PSHS A,X,Y
03870 00953A 0BEA CC 0215 A LDD #$0200+ERR21
03880 00954A 0BED 8D BC 0B7B BSR RDWR DO SETUP COMMON TO READ AND WRITE
03890 00955 *
03900 00956 * LOOP TO XFER BYTES FROM RECORD AREA
03910 00957 * (X->BUFFER, Y->RECORD AREA)
03920 00958A 0BEF E6 CB 2D A DW5 LDB DCBRBA+2,U DISPLACEMENT IN CURRENT SECTOR
03930 00959A 0BF2 4F CLRA
03940 00960A 0BF3 AE CB 24 A LDX DCBBUF,U ADDR OF BUFFER
03950 00961A 0BF6 30 BB A LEAX D,X DETERMINE ADDR IN BUFFER
03960 00962A 0BF8 A6 A0 A LDA ,Y+ GET BYTE FROM RECORD AREA
03970 00963A 0BFA A7 B4 A STA ,X STORE IN BUFFER
03980 00964A 0BFC 6C CB 2D A INC DCBRBA+2,U ADVANCE POINTER IN BUFFER
03990 00965A 0BFF 26 23 0C24 BNE DW5B IF IN SAME SECTOR
04000 00966A 0C01 17 00E6 0CEA LBSR DSKWRT REWRITE SECTOR
04010 00967A 0C04 26 0E 0C14 BNE DW5AA IF I/O ERROR
04020 00968A 0C06 EC CB 2B A LDD DCBRBA,U
04030 00969A 0C09 C3 0001 A ADDD #1 POINT TO NEXT SECTOR
04040 00970A 0C0C ED CB 2B A STD DCBRBA,U
04050 00971A 0C0F 17 0187 0D99 LBSR CALSEC RECALCULATE TRACK & SECTOR
04060 00972A 0C12 27 06 0C1A BEQ DW5A IF OK
04070 00973A 0C14 32 67 A DW5AA LEAS 7,S SCRAP STUFF IN STACK
04080 00974A 0C16 A7 E4 A STA ,S
04090 00975A 0C18 35 B2 A PULS A,X,Y,PC
04100 00976A 0C1A 17 00CA 0CE7 DW5A LBSR DSKRED
04110 00977A 0C1D 26 F5 0C14 BNE DW5AA IF I/O ERROR
04120 00978A 0C1F B6 01 A LDA #1
04130 00979A 0C21 A7 CB 30 A STA DCBMDT,U MARK NEW REC AS MODIFIED
04140 00980A 0C24 EC E4 A DW5B LDD ,S GET COUNT DOWN VALUE
04150 00981A 0C26 27 09 0C31 BEQ DW5C IF VARIABLE LENGTH STRING

```

PAGE 018 IO .SA:0 DOS - I/O ROUTINES

```

04160 00982A 0C2B 83  0001   A      SUBD #1
04170 00983A 0C2B ED  E4   A      STD ,S
04180 00984A 0C2D 26  C0   0BEF   BNE DW5    GO GET ANOTHER CHR
04190 00985A 0C2F 20  07   0C38   BRA DW6
04200 00986A 0C31 A6  C8 13  A DW5C   LDA DCBTMR,U STRING DELIMITER
04210 00987A 0C34 A1  3F   A      CMPA -1,Y WAS LAST CHR STORED A DELIMITER?
04220 00988A 0C36 26  B7   0BEF   BNE DW5    IF NO, KEEP GOING
04230 00989          *
04240 00990          * RECORD HAS BEEN WRITTEN - CLEAN UP
04250 00991A 0C38 86  01   A DW6   LDA #1
04260 00992A 0C3A A7  C8 30  A      STA DCBMDT,U ENSURE THIS SECTOR GETS REWRITTEN (EVTUALLY)
04270 00993A 0C3D 7E  0B29  A      JMP RDWRX CLEAN UP SAME AS FOR READ
04280 00994          *
04290 00995          ****
04300 00996          * RELEASE THE I/O BUFFER
04310 00997          * (USED WHEN USER WANTS TO CONTROL SHARED BUFFER)
04320 00998          * GIVEN: U->DCB
04330 00999          ****
04340 01000A 0C40 17  0100 0D43 DRELSE LBSR REWRTE REWRITE BUFFER CONTENTS IF NECESSARY
04350 01001A 0C43 CC  FFFF  A      LDD #$FFFF
04360 01002A 0C46 ED  CB 29  A      STD DCBPRN,U FORCE READ NEXT TIME
04370 01003A 0C49 4F  CLRA
04380 01004A 0C4A 39  RTS
00010 01005          OPT L
00020 01006          TTL DOS - SUPPORTING SUBROUTINES
00030 01007          OPT NOL
00040 01008          ****
00050 01009          * POINT "X" AT FAT TABLE IN MEMORY
00060 01010          * GIVEN: U-> DCB CONTAINING DCBDRV
00070 01011          * RETURNED:X
00080 01012          ****
00090 01013A 0C4B 8E  07C8  A ADRFAT LDX #FATS FAT TABLE STORE AREA
00100 01014A 0C4E A6  C8 21  A LDA DCBDRV,U DRIVE CONTAINING FILE
00110 01015A 0C51 C6  45   A LDB #69 NUMBER OF BYTES SAVED
00120 01016A 0C53 3D  MUL
00130 01017A 0C54 30  BB   A LEAX D,X POINT TO CORRECT AREA
00140 01018A 0C56 39  RTS
00150 01019          *
00160 01020          ****
00170 01021          * X F E R B Y T E S R O U T I N E S
00180 01022          ****
00190 01023          * XFER 32 BYTES FROM DCB (AT ,U) TO DIRECTORY (AT ,X)
00200 01024A 0C57 34  56   A DCBDIR PSHS D,X,U
00210 01025A 0C59 C6  20   A LDB #32 BYTES TO XFER
00220 01026A 0C5B 8D  0A   0C67 BSR XFRUX
00230 01027A 0C5D 35  D6   A PULS D,X,U,PC
00240 01028          * XFER 32 BYTES FROM DIRECTORY AT ,X TO DCB AT ,U
00250 01029A 0C5F 34  56   A DIRDCB PSHS D,X,U
00260 01030A 0C61 C6  20   A LDB #32
00270 01031A 0C63 8D  0A   0C6F BSR XFRXU
00280 01032A 0C65 35  D6   A PULS D,X,U,PC
00290 01033          * TRANSFER B BYTES FROM ,U TO ,X
00300 01034A 0C67 A6  C0   A XFRUX LDA ,U+
00310 01035A 0C69 A7  B0   A STA ,X+
00320 01036A 0C6B 5A  DECB
00330 01037A 0C6C 26  F9   0C67 BNE XFRUX
00340 01038A 0C6E 39  RTS
00350 01039          * TRANSFER B BYTES FROM ,X TO ,U

```

PAGE 019 RTN .SA:0 DOS - SUPPORTING SUBROUTINES

```

00360 01040A 0C6F 1E 13 A XFRXU EXG X,U
00370 01041A 0C71 BD F4 0C67 BSR XFRUX
00380 01042A 0C73 1E 13 A EXG X,U
00390 01043A 0C75 39 RTS
00400 01044 *
***** * CHECK DIRECTORY ON THIS DRIVE
00420 01046 * GIVEN: A=ZERO IF LOOKING FOR A MATCH
00430 01047 * A NOT ZERO IF LOOKING FOR AVAILABLE SLOT
00440 01048 * U -> DCB
00450 01049 * RETURNED: A=ZERO IF REQUEST SUCCESSFUL
00460 01050 * A=FF IF NO MATCH FOUND
00470 01051 * A=1-8 IF I/O ERROR
00480 01052 * IF SUCCESSFUL, X-> DIRECTORY ENTRY IN BUFFER
00490 01053
00500 01054
00510 01055A 0C76 34 06 A CHKDIR PSHS D SAVE OPTION
00520 01056A 0C78 CC 1103 A LDD #$1103
00530 01057A 0C7B A7 C8 22 A STA DCBTRK,U SET TO READ DIRECTORY TRACK
00540 01058A 0C7E E7 C8 23 A STB DCBSEC,U SET TO READ FIRST DIRECTORY ENTRIES
00550 01059 * RETRY ONLY IF DRIVE IS READY!
00560 01060A 0CB1 B6 0622 A CD1 LDA >RETRYs
00570 01061A 0CB4 34 02 A PSHS A
00580 01062A 0CB6 86 02 A LDA #2 ** CHANGED IN VER 6 **
00590 01063A 0CB8 B7 0622 A STA >RETRYs
00600 01064A 0CB8 17 0099 0D27 LBSR SYSRED DO PHYSICAL READ
00610 01065A 0CB8 35 04 A PULS B GET ORIG NBR OF RETRYs
00620 01066A 0C90 27 12 0CA4 BEQ CD2 IF I/O OK
00630 01067A 0C92 F7 0622 A STB >RETRYs
00640 01068A 0C95 81 01 A CMPA #ERR1 DRIVE NOT READY?
00650 01069A 0C97 26 04 0C9D BNE CD1A IF I SHOULD TRY SOME MORE
00660 01070A 0C99 A7 E4 A CD1E STA ,S
00670 01071A 0C9B 35 86 A PULS D,PC
00680 01072A 0C9D 17 0087 0D27 CD1A LBSR SYSRED GO TRY SOME MORE
00690 01073A 0CA0 26 F7 0C99 BNE CD1E IF STILL ERROR
00700 01074A 0CA2 20 03 0CA7 BRA CD2A
00710 01075 * CHECK THE DIRECTORY ENTRIES IN THIS RECORD
00720 01076A 0CA4 F7 0622 A CD2 STB >RETRYs
00730 01077A 0CA7 B6 08 A CD2A LDA #8 NUMBER OF DIRECTORY ENTRYS PER REC
00740 01078A 0CA9 A7 61 A STA 1,S
00750 01079A 0CAB BE 06C8 A LDX #SYSBUF POINT AT SYSTEM BUFFER
00760 01080A 0CAE 6D E4 A CD3 TST ,S OPTION?
00770 01081A 0CB0 27 0A 0CBC BEQ CD5 IF LOOKING FOR A MATCH
00780 01082A 0CB2 A6 84 A LDA ,X LOOK AT 1ST BYTE
00790 01083A 0CB4 27 02 0CBB BEQ CD4 IF I FOUND RE-USABLE SPACE
00800 01084A 0CB6 2A 18 0CD0 BPL CD7 IF NOT USEABLE
00810 01085A 0CBB 6F E4 A CD4 CLR ,S
00820 01086A 0CBA 35 86 A PULS D,PC RETURN SUCCESSFULLY
00830 01087 * COMPARE LOOP
00840 01088A 0CBC A6 84 A CD5 LDA ,X LOOK AT 1ST BYTE OF DIRECTORY ENTRY
00850 01089A 0CBE 27 10 0CD0 BEQ CD7 IF DELETED ENTRY
00860 01090A 0CC0 2B 1F 0CE1 BMI CDB IF END OF DIRECTORY ENTRIES
00870 01091A 0CC2 5F CLR B CHARACTER POSITION COUNTER
00880 01092A 0CC3 A6 85 A CD6 LDA B,X CHR IN DIRECTORY FILE NAME
00890 01093A 0CC5 A1 C5 A CMPA B,U CHR IN DCB FILE NAME
00900 01094A 0CC7 26 07 0CD0 BNE CD7 IF NOT A MATCH
00910 01095A 0CC9 5C INCB
00920 01096A 0CCA C1 0B A CMPB #11 MORE CHARACTERS TO COMPARE?
00930 01097A 0CCC 25 F5 0CC3 BCS CD6 IF YES

```

PAGE 020 RTN .SA:0 DOS - SUPPORTING SUBROUTINES

```

00940 01098 * MATCH FOUND
00950 01099A 0CCE 20 E8 0CB8 BRA CD4
00960 01100 *
00970 01101A 0CD0 30 88 20 A CD7 LEAX 32,X POINT TO NEXT DIRECTORY ENTRY
00980 01102A 0CD3 6A 61 A DEC 1,S MORE ENTRIES TO LOOK AT IN THIS REC?
00990 01103A 0CD5 26 D7 0CAE BNE CD3 IF YES
01000 01104A 0CD7 6C C8 23 A INC DCBSEC,U
01010 01105A 0CD8 A6 C8 23 A LDA DCBSEC,U
01020 01106A 0CDD B1 0C A CMPA #12 MORE DIRECTORY RECORDS TO READ?
01030 01107A 0CDF 25 A0 0CB8 BCS CD1 IF YES
01040 01108 * DIRECTORY ENTRY NOT FOUND ON THIS DRIVE
01050 01109A 0CE1 86 FF A CDB LDA #$FF
01060 01110A 0CE3 A7 E4 A STA ,S
01070 01111A 0CE5 35 86 A PULS D,PC
01080 01112 *
01090 01113 ****
01100 01114 *
01110 01115 * PHYSICAL DISK READ
01120 01116 * GIVEN: U->DCB
01130 01117 * FUNCTION: READ INTO DCBBUF
01140 01118 * (NOTE:DSKCON RETRYS ON ERROR 5 TIMES)
01150 01119 * RETURNED:DCBOK = RESULT CODE (ALSO IN A)
01160 01120A 0CE7 86 02 A DSKRED LDA #2 READ SECTOR OP CODE
01170 01121A 0CE9 8C A FCB $BC SKIP OVER NEXT INSTR
01180 01122 *
01190 01123 ****
01200 01124 * PHYSICAL DISK WRITE
01210 01125 * ESSENTIALLY SAME AS ABOVE
01220 01126 ****
01230 01127A 0CEA 86 03 A DSKWRT LDA #3 WRITE OP CODE
01240 01128A 0CEC A7 C8 20 A DSKIO STA DCBOPC,U
01250 01129A 0CEF 6F C8 30 A CLR DCBMDT,U
01260 01130 *
01270 01131 *
01280 01132 ****
01290 01133 * CALL DSKCON
01300 01134 * GIVEN:PARAMETERS IN DCB
01310 01135 * FUNCTION: XFER PARAMS TO [C006]
01320 01136 *
01330 01137 * MOVE RESULT CODE TO DCB
01340 01138 * LEAVE RESULT CODE IN A
01350 01139 ****
01360 01140A 0CF2 34 14 A XFRION PSHS B,X
01370 01141A 0CF4 BE C006 A LDX >$C006
01380 01142A 0CF7 EC C8 20 A LDD DCBOPC,U
01390 01143A 0CFA ED 81 A STD ,X++
01400 01144A 0CFC EC C8 22 A LDD DCBTRK,U
01410 01145A 0CFF ED 81 A STD ,X++
01420 01146A 0D01 EC C8 24 A LDD DCBBUF,U
01430 01147A 0D04 ED 81 A STD ,X++
01440 01148A 0D06 34 08 A XIOENT PSHS DP
01450 01149A 0D08 4F CLRA
01460 01150A 0D09 1F 88 A TFR A,DP
01470 01151A 0D0B 8D 10 0D1D BSR DOIO DO I/O
01480 01152A 0D0D 35 08 A PULS DP
01490 01153A 0D0F 4F CLRA
01500 01154A 0D10 E6 84 A LDB ,X GET RESULT CODE
01510 01155A 0D12 27 07 0D1B BEQ XIOX IF NO ERROR, EXIT

```

PAGE 021 RTN .SA:0 DOS - SUPPORTING SUBROUTINES

```

01520 01156          * GENERATE ERROR NUMBER BASED ON WHICH BIT IS ON
01530 01157A 0D14 58   XIOA  LSLB      IS THIS BIT SET?
01540 01158A 0D15 25   03  0D1A      BCS   XIOB      IF YES
01550 01159A 0D17 4C   INCX
01560 01160A 0D18 20   FA  0D14      BRA   XIOA
01570 01161A 0D1A 4C   XIOB  INCA
01580 01162A 0D18 35   94  A XIOX  PULS  B,X,PC
01590 01163           0024  A ZZ    EQU   ERR1+ERR2+ERR3+ERR4+ERR5+ERR6+ERR7+ERR8
01600 01164          * THE ABOVE LINE SIMPLY PUTS ERR1-8 ON THE XREF MAP
01610 01165A 0D1D 34   76  A DOIO  PSHS  D,X,Y,U
01620 01166A 0D1F B6   0622  A LDA   >RETRY
01630 01167A 0D22 BE   C004  A LDX   >$C004
01640 01168A 0D25 6E   04  A JMP   4,X
01650 01169          *
01660 01170          ****
01670 01171          * PHYSICAL DISK READ - SYSTEM FUNCTIONS
01680 01172          * SAME AS DSKRED EXCEPT SYSTEM'S BUFFER USED
01690 01173          ****
01700 01174A 0D27 B6   02  A SYSRED LDA   #2
01710 01175A 0D29 34   14  A SYSIO  PSHS  B,X
01720 01176A 0D28 E6   C8 21  A LDB   DCBDRV,U
01730 01177A 0D2E BE   C006  A LDX   >$C006
01740 01178A 0D31 ED   B1  A STD   ,X++
01750 01179A 0D33 EC   C8 22  A LDD   DCBTRK,U
01760 01180A 0D36 ED   B1  A STD   ,X++ TRACK & SECTOR
01770 01181A 0D38 CC   06CB  A LDD   #SYSBUF
01780 01182A 0D38 ED   B1  A STD   ,X++
01790 01183A 0D3D 20   C7  0D06  BRA   XIOENT FINISH UP LIKE USER IO
01800 01184          *
01810 01185          ****
01820 01186          * PHYSICAL DISK WRITE - SYSTEM FUNCTIONS
01830 01187          ****
01840 01188A 0D3F 86   03  A SYSWRT LDA   #3
01850 01189A 0D41 20   E6  0D29  BRA   SYSIO
01860 01190          *
01870 01191          ****
01880 01192          * IF DATA IN BUFFER HAS BEEN MODIFIED (DCBMDT NOT = 0) CHECK
01890 01193          * TO SEE IF WRITES ARE ALLOWED. IF NO, DO NOT SET ERROR - JUST EXIT.
01900 01194          * IF YES, REWRITE BLOCK IN BUFFER (EXIT WITH ERROR IN A IF WRITE NO GOOD.)
01910 01195          *
01920 01196          * GIVEN: U->DCB CONTAINING DCBPRN = PHYSICAL REC NUMBER THAT IS IN BUFFER.
01930 01197          ****
01940 01198A 0D43 6D   C8 30  A REWRTE TST   DCBMDT,U DATA IN BUFFER MODIFIED?
01950 01199A 0D46 26   02  0D4A      BNE   RW1   IF YES
01960 01200A 0D48 4F   RWX  CLRA
01970 01201A 0D49 39   RWXX RTS
01980 01202A 0D4A A6   C8 10  A RW1   LDA   DCBCFS,U
01990 01203A 0D4D 84   02  A ANDA #OUT ARE WRITES ALLOWED?
02000 01204A 0D4F 27   F7  0D48  BEQ   RWX   IF NO, EXIT WITH NO ERROR
02010 01205A 0D51 8D   4C  0D9F  BSR   CSENT RE-ESTABLISH TRK & SEC FROM PRN
02020 01206A 0D53 26   F4  0D49  BNE   RWXX IF NG, EXIT WITH ERROR
02030 01207A 0D55 20   93  0CEA  BRA   DSKWRT GO DO REWRITE & RETURN TO CALLER
02040 01208          *
02050 01209          ****
02060 01210          * CALCULATE RELATIVE BYTE ADDRESS FROM LOGICAL RECORD NUMBER
02070 01211          * (DCBRBA = DCBRSZ * DCBLRN)
02080 01212          ****
02090 01213A 0D57 34   70   A CALRBA PSHS  X,Y,U

```

PAGE 022 RTN .SA:0 DOS - SUPPORTING SUBROUTINES

```

02100 01214A 0D59 30 CB 2B A LEAX DCBRBA,U
02110 01215A 0D5C 31 CB 11 A LEAY DCBRSZ,U
02120 01216A 0D5F 33 CB 2E A LEAU DCBLRN,U
02130 01217A 0D62 6F B4 A CLR ,X
02140 01218A 0D64 6F 01 A CLR ,X
02150 01219A 0D66 6F 02 A CLR ,X
02160 01220A 0D68 A6 21 A LDA ,Y
02170 01221A 0D6A E6 41 A LDB ,U
02180 01222A 0D6C 3D MUL
02190 01223A 0D6D ED 01 A STD ,X
02200 01224A 0D6F A6 21 A LDA ,Y
02210 01225A 0D71 E6 C4 A LDB ,U
02220 01226A 0D73 3D MUL
02230 01227A 0D74 E3 B4 A ADDD ,X
02240 01228A 0D76 25 1D 0D95 BCS CRBAER IF CARRY
02250 01229A 0D78 ED B4 A STD ,X
02260 01230A 0D7A A6 A4 A LDA ,Y
02270 01231A 0D7C E6 41 A LDB ,U
02280 01232A 0D7E 3D MUL
02290 01233A 0D7F E3 B4 A ADDD ,X
02300 01234A 0D81 25 12 0D95 BCS CRBAER
02310 01235A 0D83 ED B4 A STD ,X
02320 01236A 0D85 A6 A4 A LDA ,Y
02330 01237A 0D87 E6 C4 A LDB ,U
02340 01238A 0D89 3D MUL
02350 01239A 0D8A EB B4 A ADDB ,X
02360 01240A 0D8C 25 07 0D95 BCS CRBAER
02370 01241A 0D8E E7 B4 A STB ,X
02380 01242A 0D90 4D TSTA
02390 01243A 0D91 26 02 0D95 BNE CRBAER
02400 01244A 0D93 35 F0 A PULS X,Y,U,PC
02410 01245A 0D95 86 10 A CRBAER LDA #ERR16
02420 01246A 0D97 35 F0 A PULS X,Y,U,PC
02430 01247 *
***** * CALCULATE TRACK & SECTOR
02450 01249 *
02460 01250 *
02470 01251 * GIVEN: DCBPRN = RELATIVE RECORD NUMBER
02480 01252 * FUNCTION: FOLLOW CLUSTER CHAIN UNTIL PROPER CLUSTER FOUND
02490 01253 * RESULT: DCBTRK & DCBSEC IF RECORD IN RANGE
02500 01254 * THEY POINT TO LAST SECTOR IF NOT IN RANGE.
02510 01255 * A = ZERO IF SUCCESSFUL
02520 01256 * NON ZERO IF NOT
02530 01257 ****
02540 01258A 0D99 EC CB 2B A CALSEC LDD DCBRBA,U DESIRED REC NUMBER
02550 01259A 0D9C ED CB 29 A STD DCBPRN,U SAVE AS THE REC IN THE BUFFER
02560 01260A 0D9F A6 4D A CSENT LDA DCBFCL,U
02570 01261A 0DA1 34 12 A PSHS A,X
02580 01262A 0DA3 BE 07CB A LDX #FATS
02590 01263A 0DA6 A6 CB 21 A LDA DCBDRV,U
02600 01264A 0DA9 C6 45 A LDB #FATSZ
02610 01265A 0DAB 3D MUL
02620 01266A 0DAC 30 BB A LEAX D,X POINT TO PROPER FAT TABLE
02630 01267A 0DAE EC CB 29 A LDD DCBPRN,U REC NUMBER DESIRED
02640 01268A 0DB1 6D E4 A TST ,S
02650 01269A 0DB3 28 15 0DCA BMI CS3 IF AT END OF CLUSTERS (NULL FILE)
02660 01270A 0DB5 83 0009 A CS1 SUBD #9
02670 01271A 0DB8 25 1B 0DD5 BCS CS4 IF IN THIS CLUSTER

```

PAGE 023 RTN .SA:0 DOS - SUPPORTING SUBROUTINES

```

02680 01272A 0DBA 34 06 A PSHS D
02690 01273A 0DBC A6 62 A LDA ,S
02700 01274A 0DBE A6 86 A LDA A,X GET NEXT CLUSTER POINTER
02710 01275A 0DC0 2B 06 0DCB BMI CS2 IF AT END OF CLUSTERS
02720 01276A 0DC2 A7 62 A STA ,S
02730 01277A 0DC4 35 06 A PULS D
02740 01278A 0DC6 20 ED 0DB5 BRA CS1
02750 01279A 0DC8 35 06 A CS2 PULS D
02760 01280*
02770 01281* REC IS BEYOND END OF CURRENT CLUSTERS
02780 01282* AM I ALLOWED TO ADD ANOTHER CLUSTER?
02790 01283A 0DCA A6 CB 10 A CS3 LDA DCBCFS,U
02800 01284A 0DCF B4 08 A ANDA #EXTEND AM I ALLOWED?
02810 01285A 0DCF 26 66 0E37 BNE CS6 IF YES, GO TRY IT
02820 01286A 0DD1 86 11 A CS3A LDA #ERR17 EXTENSION NOT ALLOWED
02830 01287A 0DD3 20 46 0E1B BRA CSERR
02840 01288*
02850 01289* RECORD IS IN THIS CLUSTER
02860 01290A 0DD5 CB 0A A CS4 ADDB #10 (RESULT IS 1-9)
02870 01291* IS THE SECTOR NUMBER IN B IN USE IN THIS CLUSTER YET?
02880 01292A 0DD7 A6 E4 A LDA ,S (CLUSTER NUMBER)
02890 01293A 0DD9 6D 86 A TST A,X IS THIS CLUSTER THE LAST IN THE FILE?
02900 01294A 0DDB 2A 44 0E21 BPL CS5 IF NO
02910 01295A 0DDD 34 06 A PSHS D CLUSTER NUMBER/SECTOR NUMBER
02920 01296* IS THIS RECORD BEYOND CURRENT LAST SECTOR USED?
02930 01297A 0DDF E6 86 A LDB A,X
02940 01298A 0DE1 C4 3F A ANDB #63 CURRENT LAST SECTOR USED
02950 01299A 0DE3 E1 61 A CMPB ,S THIS ONE
02960 01300A 0DE5 24 0B 0DF2 BCC CS4A IF THIS IS LESS OR EQUAL TO CURRENT END
02970 01301A 0DE7 E6 CB 10 A LDB DCBCFS,U GET FILE STATUS
02980 01302A 0DEA C4 08 A ANDB #EXTEND FILE EXTENSIONS ALLOWED?
02990 01303A 0DEC 26 1B 0E06 BNE CS4B IF YES
03000 01304A 0DEE 35 06 A CS4AE PULS D
03010 01305A 0DF0 20 DF 0DD1 BRA CS3A EXTENSION NOT ALLOWED
03020 01306A 0DF2 26 2B 0E1F CS4A BNE CS4C IF NOT IN LAST SECTOR
03030 01307A 0DF4 A6 CB 10 A LDA DCBCFS,U
03040 01308A 0DF7 84 08 A ANDA #EXTEND ALLOWED?
03050 01309A 0DF9 26 24 0E1F BNE CS4C IF ITS OK
03060 01310* IS REC BEYOND LAST BYTE?
03070 01311A 0DFB E6 CB 2D A LDB DCBRBA+2,U
03080 01312A 0DFE 4F CLRA
03090 01313A 0DFF 10A3 4E A CMPD DCBNLS,U
03100 01314A 0E02 25 1B 0E1F BCS CS4C IF OK
03110 01315A 0E04 20 E8 0DEE BRA CS4AE IF NG
03120 01316* EXTEND LAST SECTOR IN THIS CLUSTER
03130 01317A 0E06 E6 61 A CS4B LDB ,S SECTOR NUMBER
03140 01318A 0E08 CA C0 A ORB #$C0
03150 01319A 0E0A E7 86 A STB A,X PUT IN FAT TABLE
03160 01320* FAT HAS CHANGED - CAN I BYPASS UPDATE THIS TIME?
03170 01321A 0E0C A6 CB 10 A LDA DCBCFS,U
03180 01322A 0E0F 84 20 A ANDA #FAST
03190 01323A 0E11 26 0C 0E1F BNE CS4C IF YES
03200 01324A 0E13 8D 69 0E7E BSR WRTFAT RE-WRITE FAT TABLE TO REFLECT CHANGE
03210 01325A 0E15 27 0B 0E1F BEQ CS4C IF I/O WAS OK
03220 01326A 0E17 35 06 A PULS D
03230 01327A 0E19 86 12 A LDA #ERR18 FAT RW ERR
03240 01328A 0E1B A7 E4 A CSERR STA ,S
03250 01329A 0E1D 35 92 A PULS A,X,PC

```

PAGE 024 RTN .SA:0

DOS - SUPPORTING SUBROUTINES

```

03260 01330A 0E1F 35 06 A CS4C PULS D CONTINUE - IT IS NOW WITHIN RANGE OF FILE
03270 01331 * RECORD IS IN RANGE OF FILE - XLATE CLUSTER INTO TRACK & SECTOR
03280 01332A 0E21 A6 E4 A CS5 LDA ,S CLUSTER NUMBER
03290 01333A 0E23 44 LSRA IS THIS AN ODD CLUSTER?
03300 01334A 0E24 24 02 0E28 BCC CS5A IF NO
03310 01335A 0E26 CB 09 A ADDB #9 IF YES, USE SECTORS 10-18
03320 01336A 0E28 E7 CB 23 A CS5A STB DCBSEC,U
03330 01337A 0E2B 81 11 A CMPA #17 IS CLUSTER BELOW DIRECTORY?
03340 01338A 0E2D 25 01 0E30 BCS CS5B IF YES
03350 01339A 0E2F 4C INCA IF NOT GO ONE TRACK FARTHER
03360 01340A 0E30 A7 CB 22 A CS5B STA DCBTRK,U
03370 01341A 0E33 6F E4 A CLR ,S
03380 01342A 0E35 35 92 A PULS A,X,PC

03390 01343 *
03400 01344 * TRY TO ADD ANOTHER CLUSTER TO THE FILE
03410 01345 * NEXT CLUSTER USED WILL BE THE CLOSEST ONE TO THE LAST ONE USED BY
03420 01346 * THIS FILE. IF FIRST EVER FOR THIS FILE, IT WILL BE CLOSEST TO MIDDLE.
03430 01347A 0E37 E6 E4 A CS6 LDB ,S LAST CLUSTER NUMBER USED
03440 01348A 0E39 2A 02 0E3D BPL CS6A IF NOT VERY FIRST ASSIGNED TO FILE
03450 01349A 0E3B C6 22 A LDB #34 START SEARCH AT CLUSTER 34
03460 01350A 0E3D 4F CS6A CLRA STARTING DISPLACEMENT
03470 01351A 0E3E 34 06 A PSHS D
03480 01352 * LOOP TO LOOK FOR AN AVAILABLE CLUSTER
03490 01353A 0E40 A6 61 A CS7 LDA 1,S LAST CLUSTER OF FILE
03500 01354A 0E42 AB E4 A ADDA ,S ADD DISPLACEMENT
03510 01355A 0E44 B1 44 A CMPA #68 IN RANGE OF TABLE?
03520 01356A 0E46 24 06 0E4E BCC CS7A IF NO
03530 01357A 0E48 E6 86 A LDB A,X GET FAT TABLE BYTE
03540 01358A 0E4A C1 FF A CMPB #$FF IS IT AVAILABLE
03550 01359A 0E4C 27 1B 0E69 BEQ CS8 IF YES
03560 01360A 0E4E A6 61 A CS7A LDA 1,S
03570 01361A 0E50 A0 E4 A SUBA ,S LOOK THE OTHER WAY
03580 01362A 0E52 25 06 0E5A BCS CS7B IF NOT IN RANGE OF THE TABLE
03590 01363A 0E54 E6 86 A LDB A,X GET FAT TABLE BYTE
03600 01364A 0E56 C1 FF A CMPB #$FF AVAILABLE?
03610 01365A 0E58 27 0F 0E69 BEQ CS8 IF YES
03620 01366A 0E5A A6 E4 A CS7B LDA ,S
03630 01367A 0E5C 4C INCA
03640 01368A 0E5D A7 E4 A STA ,S
03650 01369A 0E5F 81 44 A CMPA #68 HAVE I TRIED ALL POSSIBILITIES?
03660 01370A 0E61 25 DD 0E40 BCS CS7 IF NOT YET
03670 01371A 0E63 35 06 A PULS D NORMALIZE STACK
03680 01372A 0E65 86 16 A LDA #ERR22 DISK FULL
03690 01373A 0E67 20 B2 0E1B BRA CSERR
03700 01374A 0E69 E6 62 A CS8 LDB 2,S ORIGINAL ENDING CLUSTER
03710 01375A 0E6B 2A 04 0E71 BPL CSBA
03720 01376A 0E6D A7 4D A STA DCBFCL,U THIS IS FIRST CLUSTER
03730 01377A 0E6F 20 02 0E73 BRA CS8B
03740 01378A 0E71 A7 85 A CS8A STA B,X ADD TO CHAIN
03750 01379A 0E73 C6 C0 A CS8B LDB #$C0 SAY NONE OF THESE SECTORS USED
03760 01380A 0E75 E7 86 A STB A,X
03770 01381A 0E77 35 06 A PULS D
03780 01382A 0E79 35 12 A PULS A,X NORMALIZE STACK
03790 01383A 0E7B 7E 0D9F A JMP CSENT GO TRY AGAIN FROM THE TOP!

03800 01384 *
03810 01385 ****
03820 01386 * REWRITE FAT TABLE ON DIRECTORY TRACK
03830 01387 *

```

PAGE 025 RTN .SA:0 DOS - SUPPORTING SUBROUTINES

```

03840 01388          * GIVEN: X-> CORRECT FAT TABLE IN MEMORY
03850 01389          * U-> DCB CONTAINING CORRECT DRIVE NUMBER
03860 01390          ****
03870 01391A 0E7E 34 10 A WRTFAT PSHS  X
03880 01392A 0EB0 BE C006 A LDX >$C006
03890 01393A 0EB3 86 03 A LDA #3      WRITE
03900 01394A 0EB5 A7 80 A STA ,X+
03910 01395A 0EB7 A6 C8 21 A LDA DCBDRV,U
03920 01396A 0EBA A7 80 A STA ,X+
03930 01397A 0EBC CC 1102 A LDD ##1102  TRACK 17, SECTOR 2
03940 01398A 0EBF ED 81 A STD ,X++
03950 01399A 0E91 EC E4 A LDD ,S      ADDR OF FAT TABLE
03960 01400A 0E93 ED 81 A STD ,X++
03970 01401A 0E95 34 08 A PSHS DP
03980 01402A 0E97 4F          CLRA
03990 01403A 0E98 1F 88 A TFR A,DP
04000 01404A 0E9A AD 9F C004 A JSR [$C004] DO IO
04010 01405A 0E9E 35 08 A PULS DP
04020 01406A 0EA0 A6 84 A LDA ,X      RESULT
04030 01407A 0EA2 35 90 A PULS X,PC
00010 01408          TTL DOS - PAGING & OVERLAYS
00020 01409          *
00030 01410          ****
00040 01411          * ON DISK, THIS PROGRAM BEGINS HERE! EVERY THING THAT PRECEEDS THIS POINT
00050 01412          * IS RECORDED ON DISK AFTER THE END OF THE OVERLAYS. WHEN DOS IS FIRST
00060 01413          * LOADED INTO MEMORY, THE ROUTINE CALLED "OVERLAY" SHIFTS THOSE ROUTINES
00070 01414          * DOWN TO THEIR PROPER PLACE.
00080 01415          ****
00090 01416A 0EA4 7E 0ECF A DOS JMP DOS1 JUMP OVER DISPLACEMENTS TO OVERLAYS
00100 01417A 0EA7 0329 A FDB B1-DOS
00110 01418A 0EA9 035C A FDB B2-DOS
00120 01419A 0EAB 037E A FDB B3-DOS
00130 01420A 0EAD 03AE A FDB B4-DOS
00140 01421A 0EAF 0440 A FDB B5-DOS
00150 01422A 0EB1 0529 A FDB B6-DOS
00160 01423A 0EB3 0607 A FDB B7-DOS
00170 01424A 0EB5 0610 A FDB B8-DOS
00180 01425A 0EB7 0619 A FDB B9-DOS
00190 01426A 0EB9 0622 A FDB B10-DOS
00200 01427A 0EBB 06EA A FDB B11-DOS
00210 01428A 0EBD 0762 A FDB B12-DOS
00220 01429A 0EBF 07D9 A FDB B13-DOS
00230 01430A 0EC1 088A A FDB B14-DOS
00240 01431A 0EC3 0881 A FDB B15-DOS
00250 01432A 0EC5 0984 A FDB B16-DOS
00260 01433A 0EC7 0AA9 A FDB B17-DOS
00270 01434A 0EC9 0C08 A FDB B18-DOS
00280 01435A 0ECB 0CA1 A FDB B19-DOS
00290 01436A 0ECD 0D2A A FDB B20-DOS
00300 01437          * MINIMUM INITIALIZATION FOLLOWS
00310 01438A 0ECF DOS1 DOS DO,INIT GO INITIALIZE (MENU ETC)
00320 01439A 0ED5 4D          TSTA
00330 01440A 0ED6 27 04 0EDC BEQ DOS2
00340 01441A 0EDB AD 9F 0616 A JSR [ERROR]
00350 01442A 0EDC 7E 0FF6 A DOS2 JMP OBASIC
00360 01443A 0EDF CC 10A2 A DOS3 LDD #OVRLAY
00370 01444A 0EE2 FD 0625 A STD >OLYLOC
00380 01445A 0EE5 39          RTS

```

PAGE 026 ML

.SA:0

DOS - PAGING & OVERLAYS

```

00390 01446          *
00400 01447          ****
00410 01448          * 8 BIT PRINTER DRIVER
00420 01449          * GIVEN: A=CHR TO BE SENT TO PRINTER
00430 01450          * RETURNED:A IN TACT
00440 01451          *           CC = Z CONDITION IF SENT OK
00450 01452          *           CC = NON-Z IF PRINTER NOT READY - TRY AGAIN
00460 01453          ****
00470 01454          * IS PRINTER READY?

00480 01455A 0EE6 34  16   A DPRNT  PSHS  D,X
00490 01456A 0EE8 B6  FF22   A   LDA    >U4BDR
00500 01457A 0EEB 44          LSRA
00510 01458A 0EEE 24  04   0EF2    BCC    DP1    IF READY
00520 01459A 0EEE 86  01   A     LDA    #1     SET NON-Z CONDITION
00530 01460A 0EF0 35  96   A     PULS   D,X,PC
00540 01461A 0EF2 34  01   A DP1    PSHS   CC    SAVE INTERRUPT STATUS
00550 01462A 0EF4          DSABLI NO   INTERRUPTS DURING HARD LOOP TIMING
00560 01463A 0EF6 A6  61   A     LDA    1,S    CHR TO SEND
00570 01464A 0EF8 5F          CLRB
00580 01465A 0EF9 8D  1A   0F15   BSR    LPSND  SEND START BIT
00590 01466A 0EFB C6  08   A     LDB    #8    BITS TO SEND
00600 01467A 0EFD 34  04   A     PSHS   B     LOOP COUNTER
00610 01468A 0EFF 5F          DP2    CLRB
00620 01469A 0F00 44          LSRA
00630 01470A 0F01 59          ROLB
00640 01471A 0F02 59          ROLB
00650 01472A 0F03 8D  10   0F15   BSR    LPSND  SEND THE BIT
00660 01473A 0F05 6A  E4   A     DEC    ,S
00670 01474A 0F07 26  F6   0EFF   BNE    DP2    GO BACK FOR NEXT BIT
00680 01475A 0F09 35  04   A     PULS   B
00690 01476          * INITIATE STOP BIT (IT CONTINUES UNTIL PRINTER SAYS "READY")
00700 01477A 0F0B C6  02   A     LDB    #2
00710 01478A 0F0D F7  FF20   A     STB    >U4ADR
00720 01479A 0F10 35  01   A     PULS   CC    RESTORE INTERRUPT STATUS
00730 01480A 0F12 4F          CLRA   SET ZERO CONDITION CODES
00740 01481A 0F13 35  96   A     PULS   D,X,PC
00750 01482A 0F15 F7  FF20   A LPSND  STB    >U4ADR  LATCH BIT TO OUTPUT
00760 01483A 0F18 BE  0623   A     LDX    >RATE TIME CONSTANT FOR TRANSMISSION
00770 01484A 0F1B 30  1F   A LPDLP   LEAX   -1,X
00780 01485A 0F1D 26  FC   0F1B   BNE    LPDLP
00790 01486A 0F1F 39          RTS

00800 01487          *
00810 01488          ****
00820 01489          * TURN ON OR OFF A TIME DRIVEN ROUTINE
00830 01490          * GIVEN: U-> START OF ROUTINE THAT FOLLOWS SPECS
00840 01491          ****
00850 01492A 0F20 34  46   A DTMEON PSHS  D,U
00860 01493A 0F22 4D          TSTA   REQ FOR ON OR OFF?
00870 01494A 0F23 27  0C   0F31   BEQ    DTMEOF  IF OFF
00880 01495A 0F25 FC  010D   A     LDD    >IRQ+1
00890 01496A 0F28 ED  41   A     STD    1,U
00900 01497A 0F2A 33  43   A     LEAU   3,U
00910 01498A 0F2C FF  010D   A     STU    >IRQ+1
00920 01499A 0F2F 35  C6   A     PULS   D,U,PC
00930 01500          *
00940 01501          ****
00950 01502          * TURN OFF A TIME DRIVEN ROUTINE
00960 01503          * GIVEN: U -> START OF ROUTINE

```

PAGE 027 ML .SA:0 DOS - PAGING & OVERLAYS

```

00970 01504 ****
00980 01505A 0F31 33 43 A DTMEOF LEAU 3,U ADDR STORED IN CHAIN
00990 01506A 0F33 34 50 A PSHS X,U
01000 01507A 0F35 CE 010D A LDU #IRQ+1
01010 01508A 0F38 AE C4 A DTO LDX ,U LOOK AT ADDR OF NEXT ROUTINE
01020 01509A 0F3A 8C 0F58 A CMPX #STDTME IS IT END OF CHAIN?
01030 01510A 0F3D 27 0A 0F49 BEQ DTO2 IF YES, GET OUT
01040 01511A 0F3F AC 62 A CMPX 2,S IS IT THE ONE SOUGHT?
01050 01512A 0F41 27 0A 0F4D BEQ DTO3 IF YES
01060 01513A 0F43 1F 03 A TFR D,U
01070 01514A 0F45 33 5E A LEAU -2,U
01080 01515A 0F47 20 EF 0F38 BRA DTO
01090 01516A 0F49 35 50 A DTO2 PULS X,U
01100 01517A 0F4B 35 C6 A PULS D,U,PC
01110 01518 * X POINTING AT DESIRED ROUTINE
01120 01519 * U POINTING AT WHERE THAT POINTER CAME FROM
01130 01520A 0F4D AE 1E A DTO3 LDX -2,X GET ADDR THAT DESIRED ROUTINE POINTS TO
01140 01521A 0F4F AF C4 A STX ,U UNLINK HIS ROUTINE
01150 01522A 0F51 35 50 A PULS X,U
01160 01523A 0F53 35 C6 A PULS D,U,PC
01170 01524 *
01180 01525 ****
01190 01526 * STANDARD TIME ROUTINE - LINKED IN BY INITIAL START UP ROUTINE
01200 01527 ****
01210 01528A 0F55 7E 0000 A STMX JMP >0
01220 01529A 0F58 FC 0620 A STDTME LDD >CLOCK
01230 01530A 0F5B INCX
01240 01531A 0F5E FD 0620 A STD >CLOCK
01250 01532A 0F61 4F CLRA
01260 01533A 0F62 1F 88 A TFR A,DP ENSURE ROM ROUTINE USES PAGE ZERO
01270 01534A 0F64 20 EF 0F55 BRA STMX
01280 01535 *
01290 01536 ****
01300 01537 * CALL A SYSTEM OVERLAY (OR USER OVERLAY)
01310 01538 * GIVEN: OVERLAY NUMBER IN "A"
01320 01539 * OVERLAY IS LOADED IF NOT PRESENT IN MEMORY
01330 01540 * NOTE: X IS NOT PRESERVED - USED FOR OVERLAY BASE ADDRESS
01340 01541 ****
01350 01542A 0F66 8A 80 A USROLY ORA #$80
01360 01543A 0F68 BE 0625 A SYSOLY LDX >OLYLOC POINT AT CURRENT OVERLAY LOAD AREA
01370 01544A 0F6B A1 1F A CMPA -1,X IS THE DESIRED OVERLAY ALREADY THERE?
01380 01545A 0F6D 27 11 0F80 BEQ SYS03 IF YES
01390 01546A 0F6F 34 04 A PSHS B
01400 01547A 0F71 4D TSTA SYSTEM OR USER?
01410 01548A 0F72 2B 04 0F78 BMI SYS01 IF USER
01420 01549A 0F74 8D 5C 0FD2 BSR SYSLOD LOAD THE OVERLAY
01430 01550A 0F76 20 06 0F7E BRA SYS02
01440 01551A 0F78 BD 45 0FBF SYS01 BSR USRLOD LOAD THE OVERLAY
01450 01552A 0F7A 27 02 0F7E BEQ SYS02 IF OK
01460 01553A 0F7C 35 84 A PULS B,PC IF LOAD ERROR
01470 01554A 0F7E 35 04 A SYS02 PULS B
01480 01555A 0F80 BE 0F9B A SYS03 LDX #SYS04 WHERE TO GO ON THE WAY BACK FROM THE OVERLAY
01490 01556A 0F83 34 10 A PSHS X
01500 01557A 0F85 BE 0625 A LDX >OLYLOC OVERLAY LOAD AREA
01510 01558A 0F88 30 02 A LEAX 2,X ENTRY POINT WITHIN OVERLAY
01520 01559A 0F8A 34 10 A PSHS X
01530 01560A 0F8C 30 1E A LEAX -2,X PROVIDE USER WITH HIS BASE ADDRESS
01540 01561A 0F8E 34 14 A PSHS B,X

```

PAGE 028 ML .SA:0 DOS - PAGING & OVERLAYS

```

01550 01562A 0F90 EC 84 A LDD ,X GET SIZE OF OVERLAY
01560 01563A 0F92 30 8B A LEAX D,X POINT TO END OF OVERLAY
01570 01564A 0F94 30 03 A LEAX 3,X POINT TO BASE OF NEXT OVERLAY AREA
01580 01565A 0F96 BF 0625 A STX >OLYLOC
01590 01566A 0F99 35 94 A PULS B,X,PC BASE ADDR OF OVERLAY
01600 01567 *
01610 01568 * ON THE WAY BACK, ADJUST OLYLOC
01620 01569A 0F9B 34 17 A SYS04 PSHS CC,D,X
01630 01570A 0F9D BE 0625 A LDX >OLYLOC
01640 01571A 0FA0 30 1D A LEAX -3,X
01650 01572A 0FA2 EC 84 A LDD ,X GET SIZE OF THIS OVERLAY
01660 01573A 0FA4 NEGD
01670 01574A 0FA9 30 8B A LEAX D,X POINT AT BEGINNING OF OVERLAY I AM EXITING
01680 01575A 0FAB BF 0625 A STX >OLYLOC SAVE IT
01690 01576A 0FAE 35 97 A PULS CC,D,X,PC
01700 01577 *
***** FROM ONE OVERLAY & XFER CONTROL TO ANOTHER
01720 01579 * RETURN FROM ONE OVERLAY & XFER CONTROL TO ANOTHER
01730 01580 * GIVEN: STACK NORMALIZED AS IF READY TO RTS FROM AN OVERLAY
01740 01581 * A = DESIRED OVERLAY NUMBER
01750 01582 ****
01760 01583A 0FB0 8A 80 A DUSRGO ORA #$80
01770 01584A 0FB2 34 06 A DGO PSHS D SAVE D
01780 01585A 0FB4 EC 64 A LDD 4,S (RET ADDR TO SYS04)
01790 01586A 0FB6 ED 62 A STD 2,S
01800 01587A 0FB8 CC 0F6B A LDD #SYSOLY CAUSE "RETURN" TO SYSOLY AFTER "UNDOING"
01810 01588A 0FB8 ED 64 A STD 4,S
01820 01589A 0FBD 35 86 A PULS D,PC RETURNS TO SYS04
01830 01590 *
***** LOAD A SYSTEM OVERLAY (OR USER OVERLAY)
01840 01591 * GIVEN: A = OVERLAY NUMBER
01850 01592 *
01860 01593 *
01870 01594 ****
01880 01595A 0FBF 8A 80 A USRLOD ORA #$80
01890 01596A 0FC1 34 60 A PSHS Y,U
01900 01597A 0FC3 10BE 0627 A LDY >USRSE
01910 01598A 0FC7 CE 0697 A LDU #USRDCB
01920 01599A 0FC8 BD 7A 1046 BSR PAGEIN LOAD THE OVERLAY
01930 01600A 0FCC 27 02 0FD0 BEQ SLDX IF LOADED OK
01940 01601A 0FCE 86 17 A LDA #ERR23
01950 01602A 0FD0 35 E0 A SLDX PULS Y,U,PC
01960 01603A 0FD2 34 60 A SYSLOD PSHS Y,U
01970 01604A 0FD4 10BE 0EA5 A LDY #DOS+1 LOC OF OVERLAY'S RBA TABLE IN MEMORY
01980 01605A 0FD8 CE 0635 A LDU #DOSDCB POINT AT SYSTEM'S DCB
01990 01606A 0FDB BD 69 1046 BSR PAGEIN LOAD THE OVERLAY
02000 01607A 0FDD 26 02 0FE1 BNE ABORT IF SYSTEM FAILURE
02010 01608A 0FDF 35 E0 A PULS Y,U,PC
02020 01609 *
02030 01610 ****
02040 01611 * FATAL ERROR OCCURRED IN DOS - CAN'T PROCEED
02050 01612 ****
02060 01613A 0FE1 8E 0400 A ABORT LDX #$400 VID
02070 01614A 0FE4 CE 100D A LDU #ABTMSG
02080 01615A 0FE7 C6 10 A LDB #16
02090 01616A 0FE9 17 FC7B 0C67 LBSR XFRUX
02100 01617A 0FEC CE 0400 A LDU #$400
02110 01618A 0FEF C6 F0 A LDB #256-16
02120 01619A 0FF1 17 FC73 0C67 LBSR XFRUX

```

PAGE 029 ML .SA:0 DOS - PAGING & OVERLAYS

```

02130 01620A 0FF4 BD 3F 1035 BSR DERR WAIT FOR A KEYSTROKE
02140 01621A 0FF6 4D OBASIC TSTA
02150 01622A 0FF7 27 0D 1006 BEQ OBAS1
02160 01623A 0FF9 10CE 0400 A LDS #STACK
02170 01624A 0FFD BD 0EDF A JSR DOS3 RESET STACK & OLYLOC
02180 01625A 1000 DOS DO,MENU
02190 01626A 1006 7F 0071 A OBAS1 CLR >$71
02200 01627A 1009 6E 9F FFFE A JMP [$FFFE]
02210 01628A 100D 53 A ABTMSG FCC /SYSTEM/
02220 01629A 1013 60 A FCB $60
02230 01630A 1014 46 A FCC /FAILURE/
02240 01631A 101B 6060 A FDB $6060
02250 01632 * ****
02260 01633 * ****
02270 01634 * USER ABORT ROUTINE
02280 01635 * GIVEN: ERROR NUMBER IN A
02290 01636 * ****
02300 01637A 101D 1F 89 A DERROR TFR A,B
02310 01638A 101F 86 01 A LDA #1 (ADD 256 TO IT)
02320 01639A 1021 34 06 A PSHS D SAVE FOR LATER
02330 01640A 1023 CC 0032 A LDD #50 START OF INSTRUCTIONS
02340 01641A 1026 108E 0045 A LDY #69 END OF INSTRUCTIONS
02350 01642A 102A CE 0000 A LDU #0 CLEAR SCREEN FIRST
02360 01643A 102D BD 0D 103C BSR DOMAP GIVE INSTRUCTIONS
02370 01644A 102F 35 06 A PULS D
02380 01645A 1031 1F 02 A TFR D,Y
02390 01646A 1033 BD 07 103C BSR DOMAP DISPLAY ERROR
02400 01647A 1035 DERR SYSTEM POLCAT WAIT FOR ANY KEYSTROKE
02410 01648A 1039 27 FA 1035 BEQ DERR
02420 01649A 103B 39 RTS
02430 01650 *
02440 01651 * ****
02450 01652 * DO MAP DISPLAY FUNCTION
02460 01653 * ****
02470 01654A 103C 34 66 A DOMAP PSHS D,Y,U
02480 01655A 103E DOS DO,MAP
02490 01656A 1044 35 E6 A PULS D,Y,U,PC
02500 01657 *
02510 01658 * ****
02520 01659 * LOAD OVERLAY ROUTINE
02530 01660 *
02540 01661 * GIVEN: A=OVERLAY NUMBER
02550 01662 * U-> PROGRAM DCB
02560 01663 * Y-> TABLE CONTAINING RBA'S OF OVERLAYS
02570 01664 * THE FILE MUST HAVE PREVIOUSLY BEEN OPENED!
02580 01665 * ****
02590 01666A 1046 BE 0625 A PAGEIN LDX >OLYLOC
02600 01667A 1049 A7 1F A STA -1,X
02610 01668A 104B 1F 89 A TFR A,B
02620 01669A 104D C4 7F A ANDB #$7F
02630 01670A 104F 58 LSLB 2 BYTES PER VECTOR
02640 01671A 1050 4F CLRA
02650 01672A 1051 EC AB A LDD D,Y GET RBA OF START OF OVERLAY
02660 01673A 1053 C3 0005 A ADDD #5 ADJUST TO RBA WITHIN DISK FILE
02670 01674A 1056 6F CB 2B A CLR DCBRBA,U
02680 01675A 1059 ED CB 2C A STD DCBRBA+1,U
02690 01676A 105C CC 0002 A LDD #2 LENGTH OF A SIZE FIELD
02700 01677A 105F ED CB 11 A STD DCBRSZ,U SET TO READ 2 BYTES

```

PAGE 030 ML .SA:0 DOS - PAGING & OVERLAYS

```

02710 01678A 1062 AF C8 27 A STX DCBLRB,U
02720 01679A 1065 CC FFFF A LDD #$FFFF
02730 01680A 1068 ED C8 29 A STD DCBPRN,U FORCE INITIAL PHYSICAL READ
02740 01681A 106B BD 18 1085 BSR PIRD
02750 01682A 106D 30 02 A LEAX 2,X
02760 01683A 106F EC D8 27 A LDD [DCBLRB,U] LENGTH OF ROUTINE (INCLUDING SIZE WORD)
02770 01684A 1072 AF C8 27 A STX DCBLRB,U WHERE REST OF OVERLAY GOES
02780 01685A 1075 30 BB A LEAX D,X POINT TO END OF OVERLAY + 2
02790 01686A 1077 ED 1E A STD -2,X SAVE HIS SIZE AT END
02800 01687A 1079 6F 84 A CLR ,X SAY NO VALID OVERLAYS FOLLOW
02810 01688A 107B 83 0002 A SUBD #2 SIZE OF THE REST
02820 01689A 107E ED C8 11 A STD DCBRSZ,U SAVE AS RECORD SIZE
02830 01690A 1081 BD 02 1085 BSR PIRD
02840 01691A 1083 4F CLRA
02850 01692A 1084 39 RTS
02860 01693A 1085 PIRD DOS READ,RBA
02870 01694A 1088 27 08 1095 BEQ PIERX
02880 01695A 108D 32 62 A LEAS 2,S BYPASS RET ADDR
02890 01696A 108F BE 0625 A PIERR LDX >OLYLOC
02900 01697A 1092 6F 84 A CLR ,X SAY THIS OVERLAY DOESN'T EXIST IN MEMORY
02910 01698A 1094 4D TSTA SET COND CODES
02920 01699A 1095 39 PIERX RTS
02930 01700 *
***** * MINIMUM LOGIC TO LOAD & PASS CONTROL TO USER PROGRAM
02940 01701 *
02950 01702 *
02960 01703 *
02970 01704 *
02980 01705A 1096 B12A DOS READ,RBA READ IN THE ROOT SEGMENT
02990 01706A 109C AE C8 27 A LDX DCBLRB,U BASE OF PROGRAM
03000 01707A 109F 1F 15 A TFR X,PC JUMP TO ROOT
03010 01708 *
03020 01709A 10A1 00 A FCB 0 PLACE WHERE NUMBER OF 1ST OVERLAY LOADED GOES
03030 01710 *
***** * OVERLAY SECTION FOLLOWS
03040 01711 *
03050 01712 *
03060 01713 *
03070 01714 *
03080 01715 *
03090 01716 *
03100 01717 *
03110 01718 *
03120 01719 *
03130 01720 *
03140 01721A 10A2 BE 1BD0 A OVRLAY LDX #LASTPG
03150 01722A 10A5 CE 0989 A LDU #$989
03160 01723A 10A8 10BE 051B A LDY #DOS-ORGIN AMOUNT OF PGM TO XFER
03170 01724A 10AC A6 80 A OVLP LDA ,X+
03180 01725A 10AE A7 C0 A STA ,U+
03190 01726A 10B0 31 3F A LEAY -1,Y
03200 01727A 10B2 26 FB 10AC BNE OVLP
03210 01728 * INITIALIZE VECTORS AT $600
03220 01729A 10B4 CE 0600 A LDU #$600
03230 01730A 10B7 BE 1104 A LDX #VECINI
03240 01731A 10B8 C6 C8 A LDB #ENDVEC-VECINI
03250 01732A 10BC BD 0C6F A JSR XFRXU MOVE IT TO $600
03260 01733 * FROM THIS POINT ON, VECTORS AT $600 MAY BE USED
03270 01734A 10BF 10CE 0400 A LDS #STACK
03280 01735A 10C3 BE 010D A LDX >IRQ+1 VECTOR TO DISK ROM TIME ROUTINE

```

PAGE 031 ML .SA:0 DOS - PAGING & OVERLAYS

```

03290 01736A 10C6 30 05 A LEAX 5:X BYPASS CHECK FOR WHICH INTERRUPT IT IS
03300 01737A 10CB BF 010D A STX >IRQ+1 STORE REVISED ENTRY POINT
03310 01738A 10CB CE 0F55 A LDU #STMX
03320 01739A 10CE DOS TIME,ON
03330 01740A 10D4 FC A000 A LDD >POLCAT ADDR OF ROM KBD SCAN ROUTINE
03340 01741A 10D7 FD 061C A STD >KEYIN SAVE IN KEYIN VECTOR
03350 01742 * DETERMINE MEMORY SIZE
03360 01743A 10DA BE 7FFF A LDX #$7FFF END OF 32K
03370 01744A 10DD A6 84 A LDA ,X
03380 01745A 10DF 43 COMA
03390 01746A 10E0 A7 84 A STA ,X
03400 01747A 10E2 A1 84 A CMPA ,X
03410 01748A 10E4 27 03 10E9 BEQ OVLPI IF 32K MACHINE
03420 01749A 10E6 BE 3FFF A LDX #$3FFF FOR 16K
03430 01750A 10E9 BF 08DC A OVLPI STX >MAXMEM
03440 01751A 10EC B6 04 A LDA #4 MAX NUMBER OF DRIVES
03450 01752A 10EE B7 08DE A STA DRIVES
03460 01753A 10F1 CE 0635 A LDU #DOSDCB
03470 01754A 10F4 10CE 0400 A LDS #STACK
03480 01755A 10FB BD 0EDF A JSR DOS3
03490 01756A 10FB DOS OPEN,INPUT READ ONLY
03500 01757A 1101 7E 0EA4 A JMP DOS
03510 01758 * 
03520 01759A 1104 0989 A VECINI FDB DOPEN POINTER TO OPEN FUNCTION
03530 01760A 1106 0A52 A FDB DCLOSE
03540 01761A 1108 0AE2 A FDB DREAD
03550 01762A 110A 0BEB A FDB DWRITE
03560 01763A 110C 0C40 A FDB DRELSE RELEASE I/O BUFFER
03570 01764A 110E 0F68 A FDB SYSOLY CALL SYSTEM OVERLAY
03580 01765A 1110 0FB2 A FDB DGO JUMP BETWEEN SYSTEM OVERLAYS
03590 01766A 1112 0FD2 A FDB SYSLOD LOAD A SYSTEM OVERLAY
03600 01767A 1114 0F66 A FDB USROLY CALL USER OVERLAY
03610 01768A 1116 0FB0 A FDB DUSRGO JUMP BETWEEN USER OVERLAYS
03620 01769A 1118 0FBF A FDB USRLOD LOAD A USER OVERLAY
03630 01770A 111A 101D A FDB DERROR USER FATAL ERROR EXIT
03640 01771A 111C 0F20 A FDB DTMEON TIME ROUTINE ON/OFF
03650 01772A 111E 0EE6 A FDB DPRNT 8 BIT PRINTER DRIVER
03660 01773A 1120 0000 A FDB 0 SLOT FOR KEYIN
03670 01774A 1122 0FF6 A FDB OBASIC RETURN TO BASIC
03680 01775A 1124 0000 A FDB 0 INITIAL CLOCK VALUE
03690 01776A 1126 05 A FCB 5 INITIAL RETRY COUNT
03700 01777A 1127 00AE A FDB $AE PRINTER TIME CONSTANT
03710 01778A 1129 10A2 A FDB OVRLAY LOAD ADDRESS FOR NEXT OVERLAY
03720 01779A 112B 0000 A FDB 0 BASE ADDR OF USER PGM + 1
03730 01780A 112D 0633 A FDB RETURN HOOK1
03740 01781A 112F 0633 A FDB RETURN HOOK2
03750 01782A 1131 0633 A FDB RETURN HOOK3
03760 01783A 1133 0633 A FDB RETURN HOOK4
03770 01784A 1135 0633 A FDB RETURN HOOK5
03780 01785A 1137 3939 A FDB $3939 RETURN CODE FOR HOOKS
03790 01786 * INIT COPY OF DOSDCB
03800 01787A 1139 44 A FCC /DOS BIN/
03810 01788A 1144 00 A FCB 0,0,0,0,0,0,0,0,0
03820 01789A 114D 00 A FCB 0,0,0,0,0,0,0,0,0,0,0,0
03830 01790A 1159 00 A FCB 0,$FF,0,0
03840 01791A 115D 06CB A FDB SYSBUF
03850 01792A 115F 00 A FCB 0
03860 01793A 1160 06CB A FDB SYSBUF

```

PAGE 032 ML .SA:0 DOS - PAGING & OVERLAYS

```

03870 01794A 1162    00      A     FCB    0:0,0,0,0,0,0,0,0
03880 01795          00      * INIT COPY OF MSGDCB
03890 01796A 116A    44      A     FCC    /DOS    BAS/
03900 01797A 1175    00      A     FCB    0,0,0,0,0,0,0,0
03910 01798A 117E    00      A     FCB    0,0,0,0,0,0,0,0
03920 01799A 118A    00      A     FCB    0,$FF,0,0
03930 01800A 118E    06C8    A     FDB    SYSBUF
03940 01801A 1190    00      A     FCB    0
03950 01802A 1191    0000    A     FDB    0           (SET WHEN USED)
03960 01803A 1193    00      A     FCB    0,0,0,0,0,0,0,0
03970 01804          00      * INIT COPY OF USER PGM DCB
03980 01805A 119B    20      A     FCC    /BIN/
03990 01806A 11A6    00      A     FCB    0,0,0,0,0,0,0,0
04000 01807A 11AF    00      A     FCB    0,0,0,0,0,0,0,0
04010 01808A 11BB    00      A     FCB    0,$FF,0,0
04020 01809A 11BF    06C8    A     FDB    SYSBUF
04030 01810A 11C1    00      A     FCB    0
04040 01811A 11C2    0000    A     FDB    00
04050 01812A 11C4    00      A     FCB    0,0,0,0,0,0,0,0
04060 01813          *
04070 01814A 11CC    00      A ENDVEC FCB  0           END OF PRESET DATA
04080 01815          *
00010 01816          ****
00020 01817          * INITIAL START UP - CHECK FOR AUTO EXECUTE
00030 01818          *
00040 01819          ****
00050 01820A 11CD    0033    A B1     FDB    B2-B1   SIZE OF OVERLAY
00060 01821          * CHECK FOR AUTO PROGRAM EXECUTION
00070 01822A 11CF CC  0001    A LDD    #$1
00080 01823A 11D2 108E 0001    A LDY    #$1
00090 01824A 11D6 CE  0000    A LDU    #$0
00100 01825A 11D9 BD  103C    A JSR    DOMAP   CLR SCREEN & IF AUTO EXISTS, DISPLAY IT
00110 01826A 11DC BE  0697    A LDX    #USRDCB
00120 01827A 11DF C6  08      A LDB    #8        NAME LENGTH
00130 01828A 11E1 A6  C4      A LDA    ,U        GET 1ST CHR FROM SCREEN
00140 01829A 11E3 81  60      A CMPA   #$60    IS IT A BLANK?
00150 01830A 11E5 27  13      11FA    BEQ    MENU0   IF YES, NO AUTO FUNCTION
00160 01831A 11E7 A6  C0      A STRT1  LDA    ,U+
00170 01832A 11E9 81  60      A CMPA   #$60
00180 01833A 11EB 25  02      11EF    BCS    STRT2
00190 01834A 11ED 88  40      A EORA   #$40
00200 01835A 11EF A7  80      A STRT2  STA    ,X+
00210 01836A 11F1 5A          DECB
00220 01837A 11F2 26  F3      11E7    BNE    STRT1
00230 01838A 11F4          DOS    GO,EXEC  GO LOAD & EXECUTE PROGRAM
00240 01839A 11FA          MENU0  DOS    GO,MENU  GO DISPLAY MENU & RE-INITIALIZE
00250 01840          *
00260 01841          ****
00270 01842          * MAIN MENU SELECTION 2 - EXECUTE A PROGRAM
00280 01843          ****
00290 01844A 1200    0022    A B2     FDB    B3-B2   SIZE OF OVERLAY
00300 01845A 1202 CC  0200    A LDD    #512    STARTING LINE NUMBER
00310 01846A 1205 108E 0225    A LDY    #549    ENDING NUMBER
00320 01847A 1209 CE  0000    A LDU    #0
00330 01848A 120C BD  103C    A JSR    DOMAP   DISPLAY SCREEN FORMAT & GET ADDR OF INPUT FIELD
00340 01849          * NOTE U -> FIRST INPUT FIELD ON SCREEN
00350 01850A 120F BE  0697    A LDX    #USRDCB  POINT AT DCB
00360 01851A 1212 34  50      A PSHS   X,U    ADDR OF VID AREA & DEST AREA

```

PAGE 033 OLY .SA:0

DOS - PAGING & OVERLAYS

```

00370 01852A 1214          DOS DO,FIELDI INPUT A FIELD
00380 01853A 121A 35 50    A PULS X,U
00390 01854A 121C          DOS GO,EXEC GO EXECUTE IT
00400 01855               *
00410 01856               *****
00420 01857               * MAIN MENU SELECTION 3 - TURN ON CLOCK DISPLAY
00430 01858               *****
00440 01859A 1222 0030    A B3 FDB B4-B3 SIZE OF OVERLAY
00450 01860A 1224 FC 0BDC  A LDD MAXMEM
00460 01861A 1227 83 0086  A SUBD #B14-B13+5 ALLOW ROOM FOR CLOCK ROUTINE
00470 01862A 122A FD 0BDC  A STD MAXMEM
00480 01863A 122D 10BE 0625 A LDY OLYLOC
00490 01864A 1231 34 20   A PSHS Y
00500 01865A 1233 FD 0625 A STD OLYLOC
00510 01866A 1236 CE 0418  A LDU #$400+32-8 DISPLAY AT TOP RIGHT CORNER
00520 01867A 1239 5F     CLRBL
00530 01868A 123A 10BE 0007 A LDY #7
00540 01869A 123E          DOS DO,REALTM TURN ON DISPLAY
00550 01870A 1244 32 64   A LEAS 4,S NORMALIZE STACK
00560 01871A 1246 35 20   A PULS Y
00570 01872A 1248 10BF 0625 A STY OLYLOC
00580 01873A 124C          DOS GO,MENU
00590 01874               *
00600 01875               *****
00610 01876               * MAIN MENU SELECTION 4 - DISPLAY FREE SPACE MAP
00620 01877               *****
00630 01878A 1252 0092    A B4 FDB B5-B4 SIZE
00640 01879A 1254 CC 0046  A LDD #70 START OF SCREEN FORMAT
00650 01880A 1257 10BE 0063 A LDY #99 END OF FORMAT
00660 01881A 125B CE 0000  A LDU #
00670 01882A 125E BD 103C  A JSR DOMAP DISPLAY FORMAT
00680 01883A 1261 4F     CLRA
00690 01884A 1262 34 02   A PSHS A (DRIVE COUNTER)
00700 01885A 1264 CE 0400  A LDU #$400 VID BUFFER
00710 01886A 1267 BD 6C   12D5 BSR FRES1 FIND STARTING DISPLAY POSN
00720 01887               * LOOP ONCE PER DRIVE
00730 01888A 1269 1183 05FF A FRE1 CMPU #$5FF MORE DISPLAY ROOM?
00740 01889A 126D 24 3D   12AC BCC FREX IF NO
00750 01890A 126F 34 40   A PSHS U SAVE NEXT DISPLAY ADDRESS
00760 01891A 1271 BE 0006  A LDX >$C006 POINT AT PARAMETERS
00770 01892A 1274 B6 02   A LDA #2 (READ)
00780 01893A 1276 E6 62   A LDB 2,S (DRIVE)
00790 01894A 1278 ED 81   A STD ,X+++
00800 01895A 127A CC 1102  A LDD #$1102 (TRK 17, SEC 2)
00810 01896A 127D ED 81   A STD ,X+++
00820 01897A 127F CC 06CB  A LDD #SYSBUF
00830 01898A 1282 ED 81   A STD ,X+++
00840 01899A 1284 B6 02   A LDA #2 (ONLY 2 RETRYS)
00850 01900A 1286 B7 0622 A STA >RETRY
00860 01901A 1289 BD 0D1D  A JSR DOIO
00870 01902A 12BC B6 05   A LDA #5 (RESTORE TO 5)
00880 01903A 12BE B7 0622 A STA >RETRY
00890 01904A 1291 35 40   A PULS U (DISPLAY LOC)
00900 01905A 1293 10BE 06CB A LDY #SYSBUF
00910 01906A 1297 C6 44   A LDB #68 LOOP COUNT
00920 01907A 1299 A6 84   A LDA ,X RESULT
00930 01908A 129B 27 1B   12B8 BEQ FRE5 IF OK
00940 01909A 129D B6 58   A FRE3 LDA #$58 (X)

```

PAGE 034 OLY .SA:0 DOS - PAGING & OVERLAYS

```

00950 01910A 129F 8D 32 12D3     BSR    FRESET
00960 01911A 12A1 5A               DECB
00970 01912A 12A2 26 F9 129D     BNE    FRE3
00980 01913A 12A4 6C E4          A FRE4   INC    ,S      DRIVE COUNT
00990 01914A 12A6 A6 E4          A       LDA    ,S
01000 01915A 12AB 81 04          A       CMPA   #4      MORE DRIVES TO GO?
01010 01916A 12AA 25 BD 1269     BCS    FRE1   IF YES
01020 01917A 12AC BD 1035     A FREX   JSR    DERR   WAIT FOR A KEYSTROKE
01030 01918A 12AF 35 02          A       PULS   A
01040 01919A 12B1               DOS    GO,MENU
01050 01920A 12B7 39               RTS
01060 01921          * DISPLAY FOR THIS DRIVE
01070 01922A 12B8 A6 A0          A FRE5   LDA    ,Y+
01080 01923A 12B8 28 04          12C0   BMI    FRE6   IF PART OR ALL AVAILABLE
01090 01924A 12B8 86 58          A       LDA    #$58  (X)
01100 01925A 12BE 20 0C          12CC   BRA    FREB
01110 01926A 12C8 81 FF          A FRE6   CMPA   #$FF  ALL AVAILABLE?
01120 01927A 12C2 26 04          12C8   BNE    FRE7   IF PART USED
01130 01928A 12C4 86 6E          A       LDA    #$6E  (PERIOD)
01140 01929A 12C6 20 04          12CC   BRA    FRE8
01150 01930A 12C8 84 0F          A FRE7   ANDA   #$F
01160 01931A 12CA 8A 70          A       ORA    #$70
01170 01932A 12CC 8D 05          12D3   FRES  BSR    FRESET
01180 01933A 12CE 5A               DECB
01190 01934A 12CF 26 E7 12B8     BNE    FRE5
01200 01935A 12D1 20 D1 12A4     BRA    FRE4   GO BACK FOR NEXT DRIVE
01210 01936          *
01220 01937          * STORE CHR ON SCREEN & FIND NEXT DISPLAY POSN
01230 01938A 12D3 A7 5F          A FRESET STA   -1,U
01240 01939A 12D5 A6 C0          A FRES1  LDA    ,U+
01250 01940A 12D7 81 6E          A       CMPA   #$6E  PERIOD?
01260 01941A 12D9 27 08          12E3   BEQ    FRESX
01270 01942A 12DB 1183 0600     A       CMPU   #$600 END OF SCREEN?
01280 01943A 12DF 26 F4 12D5     BNE    FRES1 IF NO
01290 01944A 12E1 33 5F          A       LEAU   -1,U
01300 01945A 12E3 39               FRESX RTS
01310 01946          *
01320 01947          *****
01330 01948          * MAIN MENU SELECTION 5 - COPY FILES
01340 01949          *****
01350 01950A 12E4 00E9 A B5     FDB    B6-B5  SIZE OF OVERLAY
01360 01951A 12E6 34 10          A       PSHS   X
01370 01952A 12E8 20 62 134C     BRA    B5A
01380 01953A 12EA 0031 A B5DCB1 RMB   DCBSZ
01390 01954A 131B 0031 A B5DCB2 RMB   DCBSZ
01400 01955A 134C CC 0226 A B5A     LDD    #550  START OF FORMAT
01410 01956A 134F 10BE 0257 A     LDY    #599  END OF FORMAT
01420 01957A 1353 CE 0000 A     LDU    #0    CLEAR SCREEN FIRST
01430 01958A 1356 BD 103C A     JSR    DOMAP DISPLAY SCREEN
01440 01959A 1359 C6 07          A     LDB    #7
01450 01960A 1358               DOS    DO,INPTS
01460 01961A 1361 C1 03          A     CMPB   #BREAK
01470 01962A 1363 27 5F 13C4     BEQ    B5X
01480 01963          *
01490 01964          * ENTER PUSHED SET UP DCBS
01500 01965A 1365 8E 0400 A B5J     LDX    #$400
01510 01966A 1368 EE E4          A     LDU    ,S      BASE ADDR
01520 01967A 136A 33 46          A     LEAU   B5DCB1-B5,U POINT AT SOURCE DCB

```

PAGE 035 OLY .SA:0 DOS - PAGING & OVERLAYS

```

01530 01968A 136C 8D 07 1375 BSR B5K SET UP SOURCE DCB
01540 01969A 136E 33 C8 31 A LEAU DCBSZ,U POINT AT DEST DCB
01550 01970A 1371 8D 02 1375 BSR B5K
01560 01971A 1373 20 2F 13A4 BRA B5L
01570 01972 * SETUP A DCB
01580 01973A 1375 8D 18 138F B5K BSR B5TAB
01590 01974A 1377 C6 08 A LDB #8
01600 01975A 1379 31 C4 A LEAY ,U
01610 01976A 1378 8D 19 1396 BSR B5MOV
01620 01977A 137D 8D 10 138F BSR B5TAB
01630 01978A 137F 31 48 A LEAY DCBFEX,U
01640 01979A 1381 C6 03 A LDB #3
01650 01980A 1383 8D 11 1396 BSR B5MOV MOVE EXTENTION
01660 01981A 1385 8D 08 138F BSR B5TAB
01670 01982A 1387 A6 84 A LDA ,X
01680 01983A 1389 80 70 A SUBA #$70 (ZERO)
01690 01984A 138B A7 C8 21 A STA DCBDRV,U
01700 01985A 138E 39 RTS
01710 01986A 138F A6 80 A B5TAB LDA ,X+
01720 01987A 1391 81 5B A CMPA #$5B
01730 01988A 1393 26 FA 138F BNE B5TAB
01740 01989A 1395 39 RTS
01750 01990A 1396 A6 80 A B5MOV LDA ,X+
01760 01991A 1398 81 60 A CMPA #$60
01770 01992A 139A 25 02 139E BCS B5MOV1
01780 01993A 139C 80 40 A SUBA #$40
01790 01994A 139E A7 A0 A B5MOV1 STA ,Y+
01800 01995A 13A0 5A DECB
01810 01996A 13A1 26 F3 1396 BNE B5MOV
01820 01997A 13A3 39 RTS
01830 01998 *
01840 01999A 13A4 8D E9 138F B5L BSR B5TAB TO Y/N
01850 02000A 13A6 E6 84 A LDB ,X
01860 02001A 13AB C1 59 A CMPB #$59 Y
01870 02002A 13AA 27 04 13B0 BEQ B5M
01880 02003A 13AC C1 4E A CMPB #$4E N
01890 02004A 13AE 26 14 13C4 BNE B5X
01900 02005A 13B0 AE E4 A B5M LDX ,S BASE
01910 02006A 13B2 33 06 A LEAU B5DCB1-B5,X
01920 02007A 13B4 31 88 37 A LEAY B5DCB2-B5,X
01930 02008A 13B7 DOS DO,COPY
01940 02009A 13BD 4D TSTA
01950 02010A 13BE 27 04 13C4 BEQ B5X
01960 02011A 13C0 AD 9F 0616 A JSR [ERROR]
01970 02012 *
01980 02013A 13C4 35 10 A B5X PULS X
01990 02014A 13C6 DOS GO,MENU
02000 02015A 13CC 39 RTS
02010 02016 *
02020 02017 ****
02030 02018 * DISPLAY SELECTED DIRECTORY LIST
02040 02019 ****
02050 02020A 13CD 00DE A B6 FDB B7-B6
02060 02021A 13CF 30 89 000A A LEAX B6ARG-B6,X
02070 02022A 13D3 34 10 A PSHS X
02080 02023A 13D5 20 2D 1404 BRA B6A
02090 02024A 13D7 00 A B6ARG FCB 0,0
02100 02025A 13D9 20 A FCC /

```

PAGE 036 OLY .SA:0 DOS - PAGING & OVERLAYS

```

02110 02026A 13E4    20    A      FCC   /
02120 02027A 1404 CC  0258  A B6A   LDD   #600
02130 02028A 1407 108E 0289  A      LDY   #649
02140 02029A 140B CE  0000  A      LDU   #0
02150 02030A 140E BD  103C  A      JSR   DOMAP  DISPLAY INPUT SCREEN
02160 02031          * GET USER INPUTS
02170 02032A 1411 C6  03    A      LDB   #3   NUMBER OF FIELDS
02180 02033A 1413          DOS   D0,INPTS GET INPUTS
02190 02034          * SETUP ARGUMENTS
02200 02035A 1419 EE  E4    A      LDU   ,S
02210 02036A 141B 33  42    A      LEAU  2,U   POINT TO NAME
02220 02037A 141D BE  0400  A      LDX   ##$400
02230 02038A 1420 BD  74    1496  BSR   B6TAB
02240 02039A 1422 C6  08    A      LDB   #8
02250 02040A 1424 BD  77    149D  BSR   B6MOV
02260 02041A 1426 BD  6E    1496  BSR   B6TAB
02270 02042A 1428 C6  03    A      LDB   #3
02280 02043A 142A BD  71    149D  BSR   B6MOV
02290 02044A 142C BD  68    1496  BSR   B6TAB
02300 02045A 142E A6  80    A      LDA   ,X+
02310 02046A 1430 84  03    A      ANDA  #3
02320 02047A 1432 EE  E4    A      LDU   ,S
02330 02048A 1434 A7  C4    A      STA   ,U
02340 02049A 1436 6F  41    A      CLR   1,U
02350 02050          * PREPARE LISTING
02360 02051A 1438 A6  80    A B6D   LDA   ,X+
02370 02052A 143A B1  6E    A      CMPA  ##$6E
02380 02053A 143C 27  15    1453  BEQ   B6E
02390 02054A 143E 91  6F    A      CMPA  $6F
02400 02055A 1440 27  11    1453  BEQ   B6E
02410 02056A 1442 BC  0600  A      CMPX  ##$600
02420 02057A 1445 25  F1    1438  BCS   B6D
02430 02058A 1447 BD  1035  A B6D1  JSR   DERR  WAIT FOR A KEYSTROKE
02440 02059A 144A 35  40    A      PULS  U
02450 02060A 144C          DOS   GO, MENU
02460 02061A 1452 39          RTS
02470 02062A 1453 EE  E4    A B6E   LDU   ,S
02480 02063A 1455 34  50    A      PSHS  X+U
02490 02064A 1457          DOS   DO, SCNDIR
02500 02065A 145D 35  50    A      PULS  X+U
02510 02066A 145F A6  41    A      LDA   1,U   ENTRY FOUND?
02520 02067A 1461 2B  E4    1447  BMI   B6D1  IF NO
02530 02068A 1463 33  4D    A      LEAU  13,U  POINT AT NAME FOUND
02540 02069A 1465 30  1F    A      LEAX  -1,X
02550 02070A 1467 C6  08    A      LDB   #8   MAX NAME LENGTH
02560 02071          * DISPLAY NAME
02570 02072A 1469 A6  84    A B6F   LDA   ,X
02580 02073A 146B B1  6E    A      CMPA  ##$6E
02590 02074A 146D 26  09    1478  BNE   B6G
02600 02075A 146F A6  C0    A      LDA   ,U+
02610 02076A 1471 BA  40    A      ORA   ##$40
02620 02077A 1473 A7  B0    A      STA   ,X+
02630 02078A 1475 5A          DECB
02640 02079A 1476 26  F1    1469  BNE   B6F
02650 02080          * DISPLAY EXTENT
02660 02081A 1478 EE  E4    A B6G   LDU   ,S
02670 02082A 147A 33  C8  15  A      LEAU  21,U  POINT AT EXT
02680 02083A 147D A6  80    A      LDA   ,X+

```

PAGE 037 OLY .SA:0

DOS - PAGING & OVERLAYS

```

02690 02084A 147F 81 6F      A      CMPA    ##$6F      /
02700 02085A 1481 26 B5      143B    BNE    B6D      GO GET NEXT ONE
02710 02086A 1483 C6 03      A      LDB    #3
02720 02087A 1485 A6 84      A      B6H      LDA    ,X
02730 02088A 1487 81 6E      A      CMPA    ##$6E
02740 02089A 1489 26 AD 143B    BNE    B6D
02750 02090A 148B A6 C0      A      LDA    ,U+
02760 02091A 148D 8A 40      A      ORA    ##$40
02770 02092A 148F A7 B0      A      STA    ,X+
02780 02093A 1491 5A
02790 02094A 1492 26 F1 1485    BNE    B6H
02800 02095A 1494 20 A2 143B    BRA    B6D
02810 02096A 1496 A6 80      A      B6TAB   LDA    ,X+
02820 02097A 1498 81 5B      A      CMPA    ##$5B
02830 02098A 149A 26 FA 1496    BNE    B6TAB
02840 02099A 149C 39
02850 02100A 149D A6 80      A      B6MOV   LDA    ,X+
02860 02101A 149F 81 60      A      CMPA    ##$60
02870 02102A 14A1 25 02 14A5    BCS    B6MOV1
02880 02103A 14A3 80 40      A      SUBA    ##$40
02890 02104A 14A5 A7 C0      A      B6MOV1  STA    ,U+
02900 02105A 14A7 5A
02910 02106A 14AB 26 F3 149D    BNE    B6MOV
02920 02107A 14AA 39
02930 02108*
02940 02109*****
02950 02110* FILL FOR ROUTINES NOT YET WRITTEN
02960 02111*****
02970 02112* (OTHER MAIN MENU FUNCTIONS)
02980 02113A 14AB 0009 A B7  FDB  B8-B7  SIZE OF OVERLAY
02990 02114A 14AD
03000 02115A 14B3 39
03010 02116*
03020 02117A 14B4 0009 A B8  FDB  B9-B8  SIZE OF OVERLAY
03030 02118A 14B6
03040 02119A 14BC 39
03050 02120*
03060 02121A 14BD 0009 A B9  FDB  B10-B9  SIZE OF OVERLAY
03070 02122A 14BF
03080 02123A 14C5 39
03090 02124*
03100 02125*****
03110 02126* GET SCREEN LINES OUT OF BASIC FILE & DISPLAY
03120 02127*
03130 02128* GIVEN IN THE STACK(PUSHED BEFORE CALLING:
03140 02129* (,S = RET ADDR TO UNDO)
03150 02130* (2,S = RET ADDR TO CALLER)
03160 02131* 4,S STARTING LINE NUMBER DESIRED
03170 02132* 6,S ENDING LINE NUMBER DESIRED
03180 02133* B,S INITIAL DISPLAY LOC
03190 02134*****
03200 02135A 14C6 00C8 A B10  FDB  B11-B10 OVERLAY SIZE
03210 02136 14C6 A MAPBSE EQU  B10  (ONLY THIS LINE & ONE ABOVE MUST CHG TO USE DIF OVRLAY NBR)
03220 02137A 14CB 20 03 14CD BRA  MAP1  BYPASS LOCALS
03230 02138A 14CA 00 A MAPOSW FCB  0  FILE OPEN SW - 0 WHEN OVERLAY 1ST LOADED; 1 FROM THEN ON
03240 02139A 14CB 0000 A MAPLN FDB  0  LAST LINE NUMBER READ
03250 02140*
03260 02141A 14CD CE 0666 A MAP1  LDU  #MSGDCB POINT AT DCB

```

PAGE 038 OLY .SA:0 DOS - PAGING & OVERLAYS

```

03270 02142A 14D0 10BE 0625 A LDY >OLYLOC (POINTS BEYOND THIS OVERLAY (WHERE NEXT OVRLAY WOULD GO)
03280 02143A 14D4 10AF C8 27 A STY DCBLRB,U USE AS LOGICAL RECORD BUFFER
03290 02144A 14DB 6D 04 A TST MAPOSW-MAPBSE,X FILE OPENED?
03300 02145A 14DA 26 16 14F2 BNE MAP3 IF YES
03310 02146 * IF FIRST TIME CALLED, OPEN DISK FILE
03320 02147A 14DC DOS OPEN,INPUT OPEN DISK FILE
03330 02148A 14E2 B6 01 A LDA #1
03340 02149A 14E4 A7 04 A STA MAPOSW-MAPBSE,X SAY FILE IS OPEN
03350 02150 * RESET TO BEGINNING OF FILE
03360 02151A 14E6 MAP2 CLR'D
03370 02152A 14E8 ED 05 A STD MAPLN-MAPBSE,X RESET LAST LINE READ
03380 02153A 14EA ED C8 2B A STD DCBRBA,U
03390 02154A 14ED B6 03 A LDA #3 (START READING AT RBA 00 00 03)
03400 02155A 14EF A7 C8 2D A STA DCBRBA+2,U
03410 02156 * CHECK TO SEE IF FILE NEEDS TO BE RESET
03420 02157 * (REQUEST MUST BE > LAST LINE READ)
03430 02158A 14F2 CC FFFF A MAP3 LDD #$FFFF
03440 02159A 14F5 ED C8 29 A STD DCBPRN,U TO FORCE RE-READ INTO BUFFER
03450 02160A 14F8 EC 05 A LDD MAPLN-MAPBSE,X LAST LINE READ
03460 02161A 14FA 10A3 64 A CMPD 4,S 1ST LINE TO BE DISPLAYED
03470 02162A 14FD 24 E7 14E6 BCC MAP2 GO START OVER AT BOF
03480 02163 * CHECK DISPLAY LOC OPTION
03490 02164A 14FF EC 68 A LDD 8,S STARTING DISPLAY LOC
03500 02165A 1501 26 12 1515 BNE MAP5 IF ADDRESS GIVEN
03510 02166 * CLEAR THE SCREEN
03520 02167A 1503 CE 0400 A LDU #$400
03530 02168A 1506 EF 68 A STU 8,S START DISPLAY AT TOP OF SCREEN
03540 02169A 1508 CC 6060 A LDD #$6060 BLANKS
03550 02170A 150B 10BE 0100 A LDY #256
03560 02171A 150F ED C1 A MAP4 STD ,U++
03570 02172A 1511 31 3F A LEAY -1,Y
03580 02173A 1513 26 FA 150F BNE MAP4
03590 02174 *
03600 02175 * READ/DISPLAY LOOP
03610 02176 * READ A LINE
03620 02177A 1515 CE 0666 A MAP5 LDU #MSGDCB POINT AT DCB
03630 02178A 1518 CC 0004 A LDD #4 LENGTH OF LINE NBR & MEM ADDR
03640 02179A 151B ED C8 11 A STD DCBRSZ,U SET TO READ 4 BYTE RECORD
03650 02180A 151E DOS READ,RBA
03660 02181A 1524 26 65 158B BNE MAPERR IF I/O ERROR
03670 02182A 1526 10BE 0625 A LDY >OLYLOC (LOGICAL REC BUFFER)
03680 02183A 152A EC A4 A LDD ,Y GET "MEMORY ADDRESS"
03690 02184A 152C 27 45 1573 BEQ MAP10 IF AT EOF
03700 02185A 152E EC 22 A LDD 2,Y GET LINE NUMBER
03710 02186A 1530 ED 05 A STD MAPLN-MAPBSE,X SAVE FOR FUTURE REFERENCE
03720 02187A 1532 34 06 A PSHS D
03730 02188A 1534 6F C8 12 A CLR DCBRSZ+1,U SET FOR VARIABLE LENGTH RECORDS
03740 02189A 1537 DOS READ,RBA READ A STRING
03750 02190A 153D 35 06 A PULS D
03760 02191A 153F 26 4A 158B BNE MAPERR IF I/O ERROR
03770 02192A 1541 10A3 64 A CMPD 4,S IS AT LEAST AS FAR AS STARTING LINE NUMBER?
03780 02193A 1544 25 CF 1515 BCS MAP5 NOT FAR ENOUGH, GO READ ANOTHER
03790 02194A 1546 10A3 66 A CMPD 6,S IS IT BEYOND LAST ONE?
03800 02195A 1549 27 02 154D BEQ MAP6 IF THIS IS THE LAST ONE
03810 02196A 154B 24 26 1573 BCC MAP10 IF AT END OF RANGE
03820 02197 * LINE FOUND - XFER IT TO SCREEN
03830 02198A 154D 34 30 A MAP6 PSHS X,Y
03840 02199A 154F BE 0625 A LDX >OLYLOC

```

PAGE 039 OLY .SA:0

DOS - PAGING & OVERLAYS

```

03850 02200A 1552 30 01      A     LEAX   1,X    (SKIP THE "REM" CODE)
03860 02201A 1554 10AE 6C      A     LDY    8+4,S  DESTINATION ADDRESS
03870 02202          * MOVE CHARACTER LOOP
03880 02203A 1557 A6 80      A MAP7  LDA    ,X+    GET A CHARACTER
03890 02204A 1559 A1 C8 13  A  CMPA   DCBTRM,U IS IT THE TERMINATOR BYTE?
03900 02205A 155C 27 0A 1568  BEQ    MAP9    IF YES
03910 02206A 155E 81 40      A     CMPA   #$40    IS IT SPL CHR?
03920 02207A 1560 24 02 1564  BCC    MAPB    IF NO
03930 02208A 1562 8A 40      A     ORA    #$40
03940 02209A 1564 A7 A0      A MAP8  STA    ,Y+
03950 02210A 1566 20 EF 1557  BRA    MAP7
03960 02211A 1568 AE 6C      A MAP9  LDX    8+4,S
03970 02212A 156A 30 88 20  A     LEAX   32,X
03980 02213A 156D AF 6C      A     STX    8+4,S
03990 02214A 156F 35 30      A     PULS   X,Y
04000 02215A 1571 20 A2 1515  BRA    MAP5    GO GET NEXT LINE
04010 02216          * FIND START OF INPUT FIELD
04020 02217A 1573 CE 0400  A MAP10 LDU   #$400
04030 02218A 1576 10BE 0200  A     LDY    #512    MAX CHR'S TO TEST
04040 02219A 157A 86 5B      A     LDA    #$5B    (LEFT BRACKET ON SCREEN)
04050 02220A 157C A1 C0      A MAP11 CMPA   ,U+
04060 02221A 157E 27 07 1587  BEQ    MAP12
04070 02222A 1580 31 3F      A     LEAY   -1,Y
04080 02223A 1582 26 F8 157C  BNE    MAP11
04090 02224A 1584 CE 0400  A     LDU   #$400    IF NO FIELD FOUND
04100 02225A 1587 EF 68      A MAP12 STU    8,S
04110 02226A 1589 4F          CLRA
04120 02227A 158A 39          RTS
04130 02228A 158B 86 19      A MAPERR LDA   #ERR25
04140 02229A 158D 39          RTS
04150 02230          *
*****INPUT FIELD FROM THE KEYBOARD (ECHO ON THE SCREEN)*****
04160 02231          *
04170 02232          *
04180 02233          *
04190 02234          * GIVEN: (,S = RET TO UNDO)
04200 02235          * (2,S = RET TO CALLER
04210 02236          * 4,S = ADDR OF INPUT FIELD IN WS
04220 02237          * 6,S = ADDR OF INPUT FIELD ON SCREEN
04230 02238          *****
04240 02239A 158E 0140  A B11  FDB   B12-B10  SIZE OF OVERLAY
04250 02240A 1590 EE 66      A     LDU   6,S
04260 02241A 1592 10AE 64      A     LDY   4,S
04270 02242A 1595 1183 0400  A     CMPU   #$400  NO FIELD DEFINED?
04280 02243A 1599 27 10 15AB  BEQ    FLDI2  IF NO FIELD MARKERS
04290 02244          * MOVE ORIG CONTENTS TO SCREEN
04300 02245A 159B A6 C4      A FLDI1 LDA   ,U    LOOK AT DESTINATION POSITION
04310 02246A 159D 81 5B      A     CMPA   #$5B  LEFT BRACKET?
04320 02247A 159F 27 0A 15AB  BEQ    FLDI2  IF YES
04330 02248A 15A1 81 5D      A     CMPA   #$5D  RIGHT BRACKET?
04340 02249A 15A3 27 06 15AB  BEQ    FLDI2  IF YES
04350 02250A 15A5 A6 A0      A     LDA    ,Y+
04360 02251A 15A7 A7 C0      A     STA    ,U+
04370 02252A 15A9 20 F0 159B  BRA    FLDI1
04380 02253A 15AB BD 1035  A FLDI2 JSR   DERR  WAIT FOR A KEYSTROKE
04390 02254A 15AE 1F 89      A     TFR    A,B
04400 02255A 15B0 EE 66      A     LDU   6,S
04410 02256A 15B2 10AE 64      A     LDY   4,S
04420 02257A 15B5 1183 0400  A     CMPU   #$400  NO FIELD MARKERS?

```

PAGE 040 OLY .SA:0

DOS - PAGING & OVERLAYS

```

04430 02258A 15B9 27 4A 1605 BEQ FLDIXX IF NO FIELD MARKERS, EXIT WITH KEY IN A & B
04440 02259A 15BB 81 20 A CMPA #$20 WAS IT LOW CONTROL KEY?
04450 02260A 15BD 25 2A 15E9 BCS FLDIX IF YES
04460 02261A 15BF 81 5B A CMPA #$5B SPL CHR/NUMBERS/UPPER CASE?
04470 02262A 15C1 25 04 15C7 BCS FLDI4 IF YES
04480 02263A 15C3 81 60 A CMPA #$60 HIGH CONTROL CODES?
04490 02264A 15C5 25 22 15E9 BCS FLDIX IF YES
04500 02265 * FALL THRU WITH LOWER CASE
04510 02266A 15C7 A6 C4 A FLDI4 LDA ,U
04520 02267A 15C9 81 5B A CMPA #$5B IS CURSOR OVER START OF FIELD?
04530 02268A 15CB 27 1C 15E9 BEQ FLDIX IF YES
04540 02269A 15CD 81 5D A CMPA #$5D OVER END OF FIELD?
04550 02270A 15CF 27 18 15E9 BEQ FLDIX IF YES
04560 02271A 15D1 1F 98 A TFR B,A
04570 02272A 15D3 A7 A0 A STA ,Y+ SAVE CHR IN INPUT AREA
04580 02273A 15D5 81 40 A CMPA #$40 SPL CHR?
04590 02274A 15D7 24 02 15DB BCC FLDI5 IF YES
04600 02275A 15D9 8A 40 A ORA #$40
04610 02276A 15DB A7 C0 A FLDI5 STA ,U+
04620 02277A 15DD 10AF 64 A FLDI5A STY 4,S
04630 02278A 15E0 EF 66 A STU 6,S
04640 02279A 15E2 5F CLRB
04650 02280A 15E3 A6 C4 A LDA ,U
04660 02281A 15E5 81 5D A CMPA #$5D FIELD OVERFLOW?
04670 02282A 15E7 26 C2 15AB BNE FLDI2
04680 *
04690 02284 * EXIT WITH LAST KEY PUSHED IN B (ZERO IF FIELD OVERFLOW)
04700 02285A 15E9 81 08 A FLDIX CMPA #LEFT (LEFT ARROW?)
04710 02286A 15EB 26 18 1605 BNE FLDIXX
04720 02287A 15ED 86 20 A LDA #$20
04730 02288A 15EF A7 A4 A STA ,Y
04740 02289A 15F1 A7 C4 A STA ,U
04750 02290A 15F3 A6 5F A LDA -1,U
04760 02291A 15F5 81 5B A CMPA #$5B IN FIRST POSN NOW?
04770 02292A 15F7 27 04 15FD BEQ FLDIX1 IF YES
04780 02293A 15F9 31 3F A LEAY -1,Y
04790 02294A 15FB 33 5F A LEAU -1,U
04800 02295A 15FD 86 20 A FLDIX1 LDA #$20
04810 02296A 15FF A7 A4 A STA ,Y
04820 02297A 1601 A7 C4 A STA ,U
04830 02298A 1603 20 DB 15DD BRA FLDI5A
04840 02299A 1605 39 FLDIXX RTS
04850 02300 *
04860 02301 ****
04870 02302 * ACTUALLY LOAD AND EXECUTE PROGRAM
04880 02303 * GIVEN: DCB FOR THE PROGRAM FILE STORED
04890 02304 * IN USRDCE
04900 02305 ****
04910 02306A 1606 0077 A B12 FDB B13-B12 SIZE OF OVERLAY
04920 02307A 1608 34 10 A PSHS X SAVE MY BASE (LOWEST LOAD ADDRESS ALLOWED)
04930 02308 * STEP 1 OPEN THE PROGRAM FILE - DOES IT EXIST?
04940 02309A 160A CE 0697 A LDU #USRDCB
04950 02310A 160D 86 FF A LDA #$FF
04960 02311A 160F A7 C8 21 A STA DCBDRV,U SEARCH ALL DRIVES
04970 02312A 1612 DOS OPEN, INPUT
04980 02313A 1618 27 10 162A BEQ EX1 IF OK
04990 02314A 161A 81 0D A CMPA #ERR13 NOT PREV CLOSED IS OK
05000 02315A 161C 27 0C 162A BEQ EX1

```

PAGE 041 OLY .SA:0 DOS - PAGING & OVERLAYS

```

05010 02316A 161E AD 9F 0616 A EXERR JSR [ERROR]
05020 02317A 1622 35 10 A PULS X
05030 02318A 1624 DOS GO, MENU
05040 02319 * 
05050 02320 * READ FILE PREFIX DATA (LOAD ADDR, RBA OF 1ST OVERLAY, ETC)
05060 02321A 162A BE 0625 A EX1 LDX >OLYLOC POINT BEYOND ME
05070 02322A 162D AF C8 27 A STX DCBLRB,U USE AS LOGICAL REC BUFFER
05080 02323A 1630 CC 000A A LDD #10 READ 1ST 10 BYTES OF PROGRAM FILE
05090 02324A 1633 ED C8 11 A STD DCBRSZ,U
05100 02325A 1636 DOS READ, RBA
05110 02326A 163C 26 E0 161E BNE EXERR
05120 02327A 163E 6D 84 A TST ,X IS 1ST BYTE ZERO?
05130 02328A 1640 27 04 1646 BEQ EX2 IF YES, OK
05140 02329A 1642 86 1B A LDA #ERR27 WRONG TYPE FILE
05150 02330A 1644 20 D8 161E BRA EXERR
05160 02331A 1646 EC 03 A EX2 LDD 3,X (LOAD ADDRESS)
05170 02332A 1648 27 0C 1656 BEQ EX3A IF BASED AT ZERO, ASSUME RELOCATABLE
05180 02333A 164A 10A3 E4 A CMPD ,S HE MUST LOAD ABOVE THIS POINT
05190 02334A 164D 24 04 1653 BCC EX3 IF HE IS OK
05200 02335A 164F 86 1A A LDA #ERR26 LOAD ADDR IS TOO LOW
05210 02336A 1651 20 CB 161E BRA EXERR
05220 02337 * LOAD ADDRESS IS HIGH ENOUGH
05230 02338A 1653 ED 08 27 A EX3 STD DCBLRB,U SET THIS AS LOGICAL RECORD BUFFER
05240 02339A 1656 EC 08 27 A EX3A LDD DCBLRB,U
05250 02340A 1659 INCD
05260 02341A 165C FD 0627 A STD >USRSE
05270 02342A 165F EC 08 A LDD 8,X (SHOULD BE RBA OF 1ST OVERLAY)
05280 02343A 1661 ED 08 11 A STD DCBRSZ,U THAT IS ALSO HOW BIG ROOT SECTION IS
05290 02344A 1664 E3 C8 27 A ADDD DCBLRB,U RESULT IS WHERE END OF ROOT WILL BE IN MEMORY
05300 02345A 1667 C3 0003 A ADDD #3
05310 02346A 166A FD 0625 A STD >OLYLOC SET THIS AS BASE OF FUTURE OVERLAYS
05320 02347A 166D 1F 02 A TFR D,Y
05330 02348A 166F 86 FF A LDA #$FF INVALIDATE WHICH OVERLAY IS IN OVERLAY AREA
05340 02349A 1671 A7 3F A STA -1,Y
05350 02350A 1673 86 05 A LDA #5
05360 02351A 1675 A7 C8 2D A STA DCBRBA+2,U START READING WITH 6TH BYTE
05370 02352A 1678 35 10 A PULS X
05380 02353A 167A 7E 1096 A JMP B12A GO LOAD ROOT & XFER CONTROL TO IT
05390 02354 *
*****RELOCATABLE REAL-TIME CLOCK ROUTINE*****
00010 02355
00020 02356
00030 02357
00040 02358
* DESIGNED TO BE LOADED BY MAINLINE OF USER'S PROGRAM, SAVING ITS
00050 02359 * LOAD ADDRESS. THEN ACCESSED THRU THE SAVED VECTOR TO PERFORM
00060 02360 * FUNCTIONS.
00070 02361
00080 02362 * GIVEN: B=0 - INITIAL CALL, LINK SELF INTO TIME INTERRUPT AND PROTECT
00090 02363 * MYSELF FROM BEING OVERLAYERED
00100 02364 * B=FF - UNLINK AND RELEASE OVERLAY SPACE
00110 02365 * B=1 - GET TIME
00120 02366 * B=2 - SET TIME
00130 02367 * WITH GET & SET TIME, Y CONTAINS SECONDS AND 60THS OF SECONDS
00140 02368 * U CONTAINS HOURS AND MINUTES
00150 02369 * WITH INITIAL CALL, U -> DISPLAY ADDRESS (0=NO DISPLAY DESIRED)
00160 02370 * Y = 1 FOR HOURS, 2 FOR MINUTES, 4 FOR SECONDS
00170 02371 * OR ANY COMBINATION (ADDED TOGETHER)
00180 02372 ****OVERLAY SIZE****
00190 02373A 167D 00B1 A B13 FDB B14-B13

```

PAGE 042 OLY2 .SA:0 DOS - PAGING & OVERLAYS

```

00200 02374      167D A CLK   EQU   B13   (TO ALLOW CHANGING TO DIFFERENT OVERLAY DURING DEVELOPMENT)
00210 02375A 167F 20 07 168B BRA   CLK1   JUMP OVER LOCALS
00220 02376A 1681 00 A HRS   FCB   0       HOURS (COUNTS TO 255)
00230 02377A 1682 00 A MIN   FCB   0       MINUTES (ALL VALUES SET TO ZERO WHEN LOADED)
00240 02378A 1683 00 A SEC   FCB   0       SECONDS
00250 02379A 1684 00 A CNT   FCB   0
00260 02380A 1685 0000 A TMELOC FDB   0       TIME DISPLAY LOC
00270 02381A 1687 00 A TMEOPT FCB   0       HR,MIN,SEC OPTION
00280 02382A 1688 5D          CLK1   TSTB
00290 02383A 1689 27 16 16A1 BEQ   CLKGO
00300 02384A 168B 30 1E A LEAX  -2,X
00310 02385A 168D 5D          TSTB
00320 02386A 168E 2B 2B 16BB BMI   CLKSTP
00330 02387A 1690 5A          DECB
00340 02388A 1691 27 07 169A BEQ   CLKGET
00350 02389A 1693 EF 04 A CLKSET STU   HRS-CLK,X
00360 02390A 1695 10AF 06 A STY   SEC-CLK,X
00370 02391A 1698 4F          CLRA
00380 02392A 1699 39          RTS
00390 02393A 169A EE 04 A CLKGET LDU   HRS-CLK,X
00400 02394A 169C 10AE 06 A LDY   SEC-CLK,X
00410 02395A 169F 4F          CLRA
00420 02396A 16A0 39          RTS
00430 02397          *
00440 02398A 16A1 EF 08 A CLKGO STU   TMELOC-CLK,X SAVE DISPLAY ADDRESS
00450 02399A 16A3 1F 20 A TFR   Y,D
00460 02400A 16A5 E7 0A A STB   TMEOPT-CLK,X SAVE DISPLAY OPTION
00470 02401A 16A7 33 89 004E A LEAU  CLKTME-CLK,X POINT AT INTERVAL ROUTINE
00480 02402A 16AB AF 44 A STX   4,U   SET LDX COMMAND TO LOAD CURRENT X VALUE
00490 02403A 16AD          DOS   TIME,ON PLUG IN THE CLOCK
00500 02404A 16B3 EC 62 A LDD   2,S   RET ADDR TO CALLER
00510 02405A 16B5 34 06 A PSHS  D     PUT IN TOP OF STACK TO BYPASS NORMAL EXIT OF OVERLAY
00520 02406A 16B7 4F          CLRA
00530 02407A 16B8 30 02 A LEAX  2,X   TELL USER WHERE TO ENTER ME
00540 02408A 16B8 39          RTS   RETURN TO CALLER
00550 02409          *
00560 02410A 16B8 33 89 004E A CLKSTP LEAU  CLKTME-CLK,X POINT AT INTERVAL ROUTINE
00570 02411A 16BF          DOS   TIME,OFF PULL THE PLUG
00580 02412A 16C5 35 06 A PULS  D     RET ADDR TO CALLER
00590 02413A 16C7 ED 62 A STD   2,S   SET TO RET TO HIM AFTER EXITING FROM OVERLAY
00600 02414A 16C9 4F          CLRA
00610 02415A 16CA 39          RTS
00620 02416          *
00630 02417A 16CB 7E 0000 A CLKTME JMP  >0
00640 02418A 16CE 8E 0000 A LDX   #0     THIS INSTR MODIFIED BY ABOVE ROUTINE
00650 02419A 16D1 EC 06 A LDD   SEC-CLK,X LOAD SEC & 60THS
00660 02420A 16D3 5C          INCB
00670 02421A 16D4 ED 06 A STD   SEC-CLK,X
00680 02422A 16D6 C1 38 A CMPB  #56   FULL SECOND?
00690 02423A 16D8 25 F1 16CB BCS   CLKTME IF NO, EXIT
00700 02424A 16DA 5F          CLRBL
00710 02425A 16DB 88 01 A ADDA  #1
00720 02426A 16DD 19          DAA
00730 02427A 16DE ED 06 A STD   SEC-CLK,X
00740 02428A 16E0 81 60 A CMPA  #60   FULL MINUTE?
00750 02429A 16E2 25 1A 16FE BCS   CLKDSP IF NO
00760 02430A 16E4 4F          CLRA
00770 02431A 16E5 A7 06 A STA   SEC-CLK,X

```

PAGE 043 OLY2 .SA:0 DOS - PAGING & OVERLAYS

```

00780 02432A 16E7 EC 04 A LDD HRS-CLK,X
00790 02433A 16E9 CB 01 A ADDB #1
00800 02434A 16EB 1E 89 A EXG A,B
00810 02435A 16ED 19 DAA
00820 02436A 16EE 1E 89 A EXG A,B
00830 02437A 16F0 E7 05 A STB MIN-CLK,X
00840 02438A 16F2 C1 60 A CMPB #$60 FULL HOUR?
00850 02439A 16F4 25 08 16FE BCS CLKDSP IF NO
00860 02440A 16F6 5F CLRBL
00870 02441A 16F7 BB 01 A ADDA #1
00880 02442A 16F9 19 DAA
00890 02443A 16FA ED 04 A STD HRS-CLK,X
00900 02444A 16FC 20 CD 16CB BRA CLKTME
00910 02445 * DISPLAY RESULTS IF NECESSARY
00920 02446A 16FE EE 08 A CLKDSP LDU TMELOC-CLK,X DISPLAY LOC
00930 02447A 1700 27 C9 16CB BEQ CLKTME EXIT
00940 02448A 1702 E6 0A A LDB TMEOPT-CLK,X DISPLAY OPTION
00950 02449A 1704 54 LSRB
00960 02450A 1705 24 04 170B BCC CLK2 IF NO
00970 02451A 1707 A6 04 A LDA HRS-CLK,X
00980 02452A 1709 BD 10 171B BSR CLKEDT
00990 02453A 170B 54 CLK2 LSRB MINUTES DESIRED?
01000 02454A 170C 24 04 1712 BCC CLK3 IF NO
01010 02455A 170E A6 05 A LDA MIN-CLK,X
01020 02456A 1710 BD 09 171B BSR CLKEDT
01030 02457A 1712 54 CLK3 LSRB SECONDS DESIRED?
01040 02458A 1713 24 B6 16CB BCC CLKTME IF NO
01050 02459A 1715 A6 06 A LDA SEC-CLK,X
01060 02460A 1717 BD 02 171B BSR CLKEDT
01070 02461A 1719 20 B0 16CB BRA CLKTME
01080 02462 * EDIT THE BCD NUMBER IN A - DISPLAY AT U
01090 02463A 171B 34 02 A CLKEDT PSHS A
01100 02464A 171D 44 LSRA
01110 02465A 171E 44 LSRA
01120 02466A 171F 44 LSRA
01130 02467A 1720 44 LSRA
01140 02468A 1721 BB 30 A ADDA #$30
01150 02469A 1723 A7 C0 A STA ,U+
01160 02470A 1725 35 02 A PULS A
01170 02471A 1727 84 0F A ANDA #$0F
01180 02472A 1729 BB 30 A ADDA #$30
01190 02473A 172B A7 C1 A STA ,U++
01200 02474A 172D 39 RTS
01210 02475 *
01220 02476 ****
01230 02477 * DOS MAIN MENU DISPLAY
01240 02478 ****
01250 02479A 172E 0027 A B14 FDB B15-B14 SIZE OF OVERLAY
01260 02480 * DISPLAY DOS MENU SCREEN
01270 02481A 1730 CC 0064 A LDD #100 STARTING LINE NUMBER
01280 02482A 1733 108E 00C7 A LDY #199 END OF RANGE
01290 02483A 1737 CE 0000 A LDU #0 SAY CLEAR SCREEN FIRST
01300 02484A 173A BD 103C A JSR DOMAP DISPLAY MENU MAP
01310 02485A 173D MENU1 SYSTEM POLCAT
01320 02486A 1741 27 FA 173D BEQ MENU1
01330 02487A 1743 80 31 A SUBA #31 LESS THAN 1?
01340 02488A 1745 27 0B 1752 BEQ MENU2 IF 1 ENTERED (RET TO BASIC)
01350 02489A 1747 25 F4 173D BCS MENU1 IF YES

```

PAGE 044 OLY2 .SA:0

DOS - PAGING & OVERLAYS

```

01360 02490A 1749 81 06 A      CMPA #6      NUMBER OF MENU SELECTIONS THAT HAVE BEEN WRITTEN
01370 02491A 174B 24 F0 173D    BCC MENU1     IF NOT IN RANGE
01380 02492A 174D 4C           INCA          TO GET OVERLAY NUMBER OF SERVICE ROUTINE
01390 02493A 174E AD 9F 060C A   JSR [GO]    PAGE IT IN & GO TO IT
01400 02494A 1752 7E 0FF6 A    MENU2 JMP OBASIC

01410 02495 *
01420 02496 ****
01430 02497 * BUFFERED PRINT I/O OVERLAY
01440 02498 *
01450 02499 * TO ACTIVATE:
01460 02500 * LDU #SIZE (TOTAL MEMORY TO USE FOR THIS PURPOSE)
01470 02501 * DOS DO,BUFPR
01480 02502 *
01490 02503 * TO USE:
01500 02504 * LDA CHARACTER TO PRINT
01510 02505 * AGAIN CLR B (SAYS "I AM NOT SHUTTING DOWN")
01520 02506 * JSR [PRNT]
01530 02507 * BNE AGAIN IF BUFFER WAS FULL, TRY AGAIN (OR GO DISPLAY MSG)
01540 02508 *
01550 02509 * TO TERMINATE:
01560 02510 * LDB #1 (ANY NON-ZERO SAYS SHUT DOWN)
01570 02511 * JSR [PRNT]
01580 02512 ****

01590 02513A 1755 00D3 A B15 FDB B16-B15 SIZE OF OVERLAY
01600 02514 1755 A BP EQU B15 (FOR USE IN RELATIVE ADDRESSING
01610 02515A 1757 20 0C 1765 BRA BP1 JUMP OVER LOCALS
01620 02516A 1759 0000 A PRTBUF FDB 0 POINTER TO PRINT BUFFER
01630 02517A 175B 0000 A BUFSZ FDB 0 SIZE OF PRINT BUFFER
01640 02518A 175D 0000 A BUFcnt FDB 0 NUMBER OF CHRS IN BUFFER
01650 02519A 175F 0000 A SNDCHR FDB 0 POINTER INTO BUFFER FOR CHR BEING SENT
01660 02520A 1761 0000 A STRCHR FDB 0 POINTER INTO BUFFER FOR CHR BEING STORED
01670 02521A 1763 0000 A PRNTSV FDB 0 SAVE AREA FOR VECTOR TO ORIG PRNT ROUTINE
01680 02522 *
01690 02523 * SEE IF ENOUGH ROOM PROVIDED
01700 02524A 1765 1F 30 A BP1 TFR U,D PUT SPACE ALLOWED IN D
01710 02525A 1767 83 00D8 A SUBD #BPSZ+5 AMOUNT NOT AVAILABLE FOR BUFFER
01720 02526A 176A 24 03 176F BCC BP1A IF ROOM FOR AT LEAST 1 BYTE BUFFER
01730 02527A 176C 86 18 A LDA #ERR24 BUFFER NOT BIG ENOUGH
01740 02528A 176E 39 RTS

01750 02529 * SET UP FOR BUFFERED PRINTING
01760 02530A 176F C3 0001 A BP1A ADDD #1 (ACTUAL SIZE OF BUFFER)
01770 02531A 1772 ED 06 A STD BUFSZ-BP,X SAVE BUFFER SZ
01780 02532A 1774 31 89 00D6 A LEAY BPSZ+3,X POINT AT BASE OF BUFFER
01790 02533A 1778 10AF 04 A STY PRTBUF-BP,X SAVE IT
01800 02534A 177B 31 AB A LEAY D,Y POINT BEYOND END OF BUFFER
01810 02535A 177D 6F A0 A CLR ,Y+ SAY NO VALID OVERLAY FOLLOWS
01820 02536A 177F 10BF 0625 A STY >OLYLOC THIS IS WHERE NEXT OVERLAY GOES
01830 02537A 1783 AF 89 00AB A STX BPTME+4-BP,X MODIFY LDX COMMAND
01840 02538A 1787 AF 89 0055 A STX BPOUT+3-BP,X (SO IT KNOWS WHERE LOCAL WS IS)
01850 02539A 178B 33 89 00A7 A LEAU BPTME-BP,X POINT AT TIME ROUTINE
01860 02540A 178F DOS TIME,ON PLUG IT IN
01870 02541A 1795 FE 061A A LDU >PRNT GET ADDR OF ORIGINAL PRINT ROUTINE
01880 02542A 1798 EF 0E A STU PRNTSV-BP,X SAVE IT
01890 02543A 179A 33 89 0052 A LEAU BPOUT-BP,X POINT AT ENTRY FOR BUFFERED PRINT
01900 02544A 179E FF 061A A STU >PRNT
01910 02545A 17A1 EE 62 A LDU 2,S RET ADR TO USER
01920 02546A 17A3 34 40 A PSHS U (BYPASS NORMAL RETURN THRU UN-DO)
01930 02547A 17A5 4F CLRA SAY DONE OK

```

PAGE 045 OLY2 .SA:0

DOS - PAGING & OVERLAYS

```

01940 02548A 17A6 39          RTS
01950 02549               *
01960 02550               * SEND A CHARACTER TO THE PRINTER VIA BUFFERED I/O
01970 02551A 17A7 34      52   A BPOUT PSHS A,X,U
01980 02552A 17A9 8E      0000  A LDX #0      (THIS INSTR MODIFIED BY SETUP LOGIC)
01990 02553A 17AC 5D      TSTB REQUEST TO SHUT DOWN?
02000 02554A 17AD 26      30   17DF BNE BPO3
02010 02555A 17AF EC      08   A BPO1 LDD BUFcnt-BP,X
02020 02556A 17B1 10A3     06   A CMPD BUFSZ-BP,X ROOM FOR MORE?
02030 02557A 17B4 25      06   17BC BCS BP01A IF ROOM
02040 02558A 17B6 86      01   A LDA #1      SET NON-Z COND
02050 02559A 17B8 35      52   A PULS A,X,U
02060 02560A 17B8 24      F3   17AF BCC BP01 IF NO ROOM
02070 02561A 17BC          BPO1A DSABLI
02080 02562A 17BE EE      04   A LDU PRTBUF-BP,X
02090 02563A 17C0 EC      0C   A LDD STRCHR-BP,X DISPLACEMENT IN BUFFER
02100 02564A 17C2 33      CB   A LEAU D,U POINT AT NEXT STORE POSITION
02110 02565A 17C4 C3      0001  A ADDD #1
02120 02566A 17C7 10A3     06   A CMPD BUFSZ-BP,X WRAP AROUND?
02130 02567A 17CA 25      02   17CE BCS BP02 IF NO
02140 02568A 17CC          CLRD
02150 02569A 17CE ED      0C   A BPO2 STD STRCHR-BP,X
02160 02570A 17D0 A6      E4   A LDA ,S (CHR TO BE PRINTED)
02170 02571A 17D2 A7      C4   A STA ,U
02180 02572A 17D4 EC      08   A LDD BUFcnt-BP,X
02190 02573A 17D6 C3      0001  A ADDD #1
02200 02574A 17D9 ED      08   A STD BUFcnt-BP,X
02210 02575A 17DB          ENABLI
02220 02576A 17DD 35      D2   A PULS A,X,U,PC
02230 02577               * WAIT FOR BUFFER TO EMPTY
02240 02578A 17DF EC      08   A BPO3 LDD BUFcnt-BP,X EMPTY YET?
02250 02579A 17E1 26      FC   17DF BNE BPO3 IF NO WAIT
02260 02580A 17E3 33      89  00A7 A LEAU BPTME-BP,X POINT AT TIME ROUTINE
02270 02581A 17E7          DOS TIME,OFF UN PLUG IT
02280 02582A 17ED EC      0E   A LDD PRNTSV-BP,X GET ADDR OF ORIG DRIVER
02290 02583A 17EF FD      061A A STD >PRNT RESTORE IT
02300 02584A 17F2 EC      04   A LDD PRTBUF-BP,X WHERE NEXT OVERLAY SHOULD HAVE GONE
02310 02585A 17F4 FD      0625  A STD >OLYLOC
02320 02586A 17F7 35      40   A PULS U (RET ADDR)
02330 02587A 17F9 ED      62   A STD 2,S I'M SET TO RETURN VIA UN-DO)
02340 02588A 17FB 39      RTS
02350 02589               *
02360 02590               * TIME INTERVAL DRIVEN PRINT LOGIC
02370 02591A 17FC 7E      0000  A BPTME JMP >0 (TO NEXT TIME ROUTINE)
02380 02592A 17FF 8E      0000  A LDX #0 (INSTRUCTION MODIFIED BY ABOVE LOGIC)
02390 02593               * IS THERE DATA IN THE BUFFER TO BE SENT TO PRINTER?
02400 02594A 1802 EC      08   A LDD BUFcnt-BP,X
02410 02595A 1804 27      F6   17FC BEQ BPTME IF NO, EXIT
02420 02596               * TRY TO SEND IT (PRINTER MIGHT NOT BE READY)
02430 02597A 1806 EE      04   A LDU PRTBUF-BP,X POINT AT BUFFER
02440 02598A 1808 EC      0A   A LDD SNDCHR-BP,X DISPLACEMENT WITHIN BUFFER
02450 02599A 180A A6      CB   A LDA D,U GET CHR OUT OF BUFFER
02460 02600A 180C AD      98  0E   A JSR [PRNTSV-BP,X]
02470 02601A 180F 26      EB   17FC BNE BPTME IF PRINTER WAS NOT READY
02480 02602               * ADVANCE BUFFER POINTER
02490 02603A 1811 EC      0A   A LDD SNDCHR-BP,X
02500 02604A 1813 C3      0001  A ADDD #1
02510 02605A 1816 10A3     06   A CMPD BUFSZ-BP,X IS POINTER WRAPPING AROUND END OF BUFFER?

```

PAGE 046 OLYZ .SA:0 DOS - PAGING & OVERLAYS

```

02520 02606A 1819 25 02 181D     BCS    BPT1    IF NO
02530 02607A 181B                 CLRD
02540 02608A 181D ED 0A         A BPT1    STD    SNDCHR-BP,X SAVE POINTER TO NEXT CHR
02550 02609                   * ADJUST BUFFER COUNT
02560 02610A 181F EC 08         A LDD    BUFCNT-BP,X
02570 02611A 1821 83 0001      A SUBD   #1
02580 02612A 1824 ED 08         A STD    BUFCNT-BP,X
02590 02613A 1826 20 D4 17FC    BRA    BPTME   EXIT (ONLY SEND ONE CHR PER INTERRUPT!)
02600 02614                   *
02610 02615                   ****
02620 02616                   * BUFFERED KEYBOARD INPUT OVERLAY
02630 02617                   ****
02640 02618A 1828 00C5      A B16    FDB    B17-B16
02650 02619                   00D3      A BPSZ   EQU    B16-B15 (FOR PREVIOUS ROUTINE'S USE)
02660 02620                   1828      A BK     EQU    B16
02670 02621A 182A 20 0C 1838    BRA    BK1    JUMP OVER LOCALS
02680 02622A 182C 0000      A KEYBUF  FDB    0       ADDR OF KEYBOARD BUFFER
02690 02623A 182E 0000      A KEYSZ   FDB    0       SIZE OF KBD BUFFER
02700 02624A 1830 0000      A KEYCNT  FDB    0       NUMBER OF KEYSTROKES IN BUFFER
02710 02625A 1832 0000      A SNDKEY  FDB    0       DISPLACEMENT TO NEXT KEY TO GIVE USER
02720 02626A 1834 0000      A STRKEY  FDB    0       DISPLACEMENT FOR STORING NEXT KEYSTROKE
02730 02627A 1836 0000      A KEYSV   FDB    0       SAVE AREA FOR ADDR OF ORIGINAL KBD ROUTINE
02740 02628                   *
02750 02629                   * SET UP FOR BUFFERED KBD
02760 02630A 1838 1F 30         A BK1    TFR   U,D    PUT SPACE ALLOWED IN D
02770 02631A 183A 83 00CA      A SUBD   #BKSZ+5 AMOUNT NOT AVAILABLE FOR BUFFER
02780 02632A 183D 24 03 1842    BCC    BK1A   IF ROOM FOR AT LEAST 1 BYTE BUFFER
02790 02633A 183F B6 1C         A LDA    #ERR2B BUFFER NOT BIG ENOUGH
02800 02634A 1841 39           RTS
02810 02635A 1842 C3 0001      A BK1A   ADDD   #1       (ACTUAL SIZE OF BUFFER)
02820 02636A 1845 ED 06         A STD    KEYSZ-BK,X SAVE BUF SZ
02830 02637A 1847 31 89 00CB A LEAY   BKSZ+3,X POINT AT BASE OF BUFFER
02840 02638A 1848 10AF 04         A STY    KEYBUF-BK,X
02850 02639A 184E 31 AB         A LEAY   D,Y
02860 02640A 1850 6F A0         A CLR    ,Y+     SAV NO VALID OVERLAY FOLLOWS
02870 02641A 1852 10BF 0625    A STY    >OLYLOC NEXT OVERLAY GOES HERE
02880 02642A 1856 AF 89 0082 A STX    BKTME+4-BK,X MODIFY LDX INSTR
02890 02643A 185A AF 89 0055 A STX    BKGIVE+3-BK,X DITTO
02900 02644A 185E 33 89 007E A LEAU   BKTME-BK,X
02910 02645A 1862           DOS    TIME,ON PLUG IN TIME RTN
02920 02646A 1868 FE 061C      A LDU    >KEYIN
02930 02647A 186B EF 0E         A STU    KEYSV-BK,X
02940 02648A 186D 33 89 0052 A LEAU   BKGIVE-BK,X
02950 02649A 1871 FF 061C      A STU    >KEYIN
02960 02650A 1874 EE 62         A LDU    2,S
02970 02651A 1876 34 40         A PSHS   U
02980 02652A 1878 4F           CLRA
02990 02653A 1879 39           RTS
03000 02654                   *
03010 02655                   * POLL FOR A CHARACTER TO GIVE USER
03020 02656A 187A 34 54         A BKGIVE PSHS   B,X,U
03030 02657A 187C 8E 0000      A LDX    #0     (THIS INSTRUCTION MODIFIED BY SETUP)
03040 02658A 187F EC 08         A LDD    KEYCNT-BK,X COUNT OF BUFFERED CHRS
03050 02659A 1881 26 02 1885    BNE    BK1    IF ONE TO SEND
03060 02660A 1883 35 D4         A PULS   B,X,U,PC IF NONE, EXIT WITH A=ZERO
03070 02661A 1885 EE 04         A BKG1   LDU    KEYBUF-BK,X ADDR OF BUFFER
03080 02662A 1887 EC 0A         A LDD    SNDKEY-BK,X DISPLACEMENT
03090 02663A 1889 33 CB         A LEAU   D,U     POINT AT CHARACTER

```

PAGE 047 OLY2 .SA:0

DOS - PAGING & OVERLAYS

```

03100 02664 * ADJUST POINTER TO NEXT POSITION
03110 02665A 188B C3 0001 A ADDD #1
03120 02666A 188E 10A3 06 A CMPD KEYSZ-BK,X WRAP AROUND?
03130 02667A 1891 25 F2 1885 BCS BKG1 IF NO
03140 02668A 1893 CLRD
03150 02669A 1895 ED 0A A BKG2 STD SNDKEY-BK,X
03160 02670A 1897 A6 C4 A LDA ,U
03170 02671A 1899 34 02 A PSHS A
03180 02672A 1898 EC 08 A LDD KEYCNT-BK,X
03190 02673A 189D B3 0001 A SUBD #1
03200 02674A 18A0 ED 08 A STD KEYCNT-BK,X
03210 02675A 18A2 6D E4 A TST ,S
03220 02676A 18A4 35 D6 A PULS D,X,U,PC
03230 02677 *
03240 02678 * TIME INTERVAL KEYBOARD SCAN ROUTINE
03250 02679A 18A6 7E 0000 A BKTME JMP >0 TO NEXT TIME ROUTINE
03260 02680A 18A9 BE 0000 A LDX #0 (MODIFIED BY SETUP)
03270 02681A 18AC EC 08 A BKTMEA LDD KEYCNT-BK,X
03280 02682A 18AE 10A3 06 A CMPD KEYSZ-BK,X IS BUFFER FULL?
03290 02683A 18B1 25 13 18C6 BCS BKT1 IF NO
03300 02684 * BUFFER IS FULL - GO BEEP
03310 02685A 18B3 BE FF22 A LDX #U4BDR
03320 02686A 18B6 A6 84 A LDA ,X
03330 02687A 18B8 88 02 A EORA #2 COMPLIMENT SOUND BIT
03340 02688A 18BA A7 84 A STA ,X
03350 02689A 18BC 10BE 0020 A LDY #$20 PULSE WIDTH
03360 02690A 18C0 31 3F A BKT0 LEAY -1,Y
03370 02691A 18C2 26 FC 18C0 BNE BKT0
03380 02692A 18C4 20 E0 18A6 BRA BKTME EXIT
03390 02693A 18C6 AD 98 0E A BKT1 JSR [KEYSV-BK,X] GO POLL KEYBOARD
03400 02694A 18C9 4D TSTA
03410 02695A 18CA 27 DA 18A6 BEQ BKTME IF NO NEW KEYSTROKES, EXIT
03420 02696A 18CC 34 02 A PSHS A SAVE KEY
03430 02697A 18CE EE 04 A LDU KEYBUF-BK,X
03440 02698A 18D0 EC 0C A LDD STRKEY-BK,X DISPLACEMENT TO SAVE LOC
03450 02699A 18D2 33 CB A LEAU D,U POINT AT SAVE LOC
03460 02700A 18D4 C3 0001 A ADDD #1 POINT TO NEXT SAVE LOC
03470 02701A 18D7 10A3 06 A CMPD KEYSZ-BK,X WRAP AROUND?
03480 02702A 18DA 25 02 18DE BCS BKT2 IF NO
03490 02703A 18DC CLRD
03500 02704A 18DE ED 0C A BKT2 STD STRKEY-BK,X
03510 02705A 18E0 EC 08 A LDD KEYCNT-BK,X
03520 02706A 18E2 C3 0001 A ADDD #1
03530 02707A 18E5 ED 08 A STD KEYCNT-BK,X
03540 02708A 18E7 35 02 A PULS A
03550 02709A 18E9 A7 C4 A STA ,U
03560 02710A 18EB 20 BF 18AC BRA BKTMEA GO CHECK FOR ANOTHER KEY DOWN
03570 02711 *
*****COPY FILE OVERLAY*****
03580 02712
03590 02713 * COPY FILE OVERLAY
03600 02714 * GIVEN: B (BIT 0) = ZERO IF NO DISK SWAPPING, 1 IF SWAPPING
03610 02715 * U-> SOURCE FILE DCB (UNOPENED)
03620 02716 * Y-> DEST FILE DCB (UNOPENED)
03630 02717 * USES MEMORY FROM "OLYLOC" TO "MAXMEM"
03640 02718 * USES LAST LINE ON SCREEN FOR PROMPTS IF SWAPPING DISKETTES
03650 02719 ****
03660 02720A 18ED 01BF A B17 FDB B18-B17
03670 02721A 18EF 34 76 A PSHS D,X,Y,U

```

PAGE 048 OLY2 .SA:0 DOS - PAGING & OVERLAYS

```

03680 02722A 18F1 32 7A      A       LEAS   -6,S
03690 02723A 1BF3 20 60      1955    BRA    B17A
03700 02724          * ,S COUNT OF SECTORS IN MEMORY
03710 02725          * 1,S EOF SW
03720 02726          * 2,S=NEXT INPUT PRN
03730 02727          * 4,S=NEXT OUTPUT PRN
03740 02728          * 6,S=PGS AVAIL
03750 02729          * 7,S=SWAP SW
03760 02730          * 8,S=BASE
03770 02731          * 10,S=DEST DCB ADDR
03780 02732          * 12,S=SOURCE DCB ADDR
03790 02733          * 14,S=RET ADDR

03800 02734A 18F5      4C      A B17M1 FCC  /LOAD SOURCE DISKETTE /
03810 02735A 1915      4C      A B17M2 FCC  /LOAD DESTINATION DISKETTE /
03820 02736A 1935      4C      A B17M3 FCC  /LOAD S Y S T E M DISKETTE /
03830 02737          *
03840 02738          * SETUP STACK
03850 02739A 1955 C4      01      A B17A ANDB #1      SET TO 1 OR 0
03860 02740A 1957 A6      C8 21  A       LDA    DCBDRV,U
03870 02741A 195A A1      AB 21  A       CMPA   DCBDRV,Y SAME DRIVE?
03880 02742A 195D 27      01      1960    BEQ    B17B   IF YES
03890 02743A 195F 5F      CLRB
03900 02744A 1960 E7      67      A B17B STB    7,S
03910 02745A 1962 4F      CLRA
03920 02746A 1963 5F      CLR B
03930 02747A 1964 ED      E4      A       STD    ,S
03940 02748A 1966 ED      62      A       STD    2,S  STARTING INPUT PRN
03950 02749A 1968 ED      64      A       STD    4,S  STARTING OUTPUT PRN
03960 02750A 196A FC      08DC    A       LDD    >MAXMEM
03970 02751A 196D B3      0625    A       SUBD   >OLYLOC HOW MUCH MEM TO WORK WITH
03980 02752A 1970 25      03      1975    BCS    B17B1 IF NOT ENOUGH
03990 02753A 1972 4D      TSTA
04000 02754A 1973 26      51      19C6    BNE    B17C   IF AT LEAST 1 PAGE
04010 02755A 1975 86      1D      A B17B1 LDA   #ERR29 NOT ENOUGH MEM
04020 02756          *
04030 02757          * COMMON EXIT
04040 02758A 1977 A7      66      A B17X STA   6,S
04050 02759A 1979 6D      67      A       TST    7,S
04060 02760A 197B 27      10      198D    BEQ    B17XIT
04070 02761          * RECOVER SYSTEM DISKETTE
04080 02762A 197D AE      68      A       LDX    8,S
04090 02763A 197F 30      88 48  A       LEAX   B17M3-B17,X
04100 02764A 1982 8D      0D      1991    BSR    B17WTE
04110 02765A 1984 CE      0666    A       LDU    #MSGDCB
04120 02766A 1987          DOS   OPEN,INPUT TO RE-LOAD FAT TABLE
04130 02767A 198D 32      66      A B17XIT LEAS  6,S
04140 02768A 198F 35      F6      A       PULS   D,X,Y,U,PC
04150 02769          *
04160 02770          * DISPLAY FLASHING MSG & WAIT FOR DISKETTE SWAP
04170 02771A 1991 10BE 05E0  A B17WTE LDY  #$400+512-32 (LAST LINE)
04180 02772A 1995 C6      20      A       LDB    #32
04190 02773A 1997 A6      80      A B17WT1 LDA   ,X+
04200 02774A 1999 A7      A0      A       STA    ,Y+
04210 02775A 199B 5A      DECB
04220 02776A 199C 26      F9      1997    BNE    B17WT1
04230 02777A 199E 7F      0621    A       CLR    >CLOCK+1
04240 02778A 19A1          B17WT2 SYSTEM POLCAT WAIT FOR KEYSTROKE
04250 02779A 19A5 B1      0D      A       CMPA   #$0D

```

PAGE 049 OLY2 .SA:0 DOS - PAGING & OVERLAYS

```

04260 02780A 19A7 27 1C 19C5 BEQ B17WTX
04270 02781A 19A9 8E 05E0 A LDX #$400+512-32
04280 02782A 19AC B6 0621 A LDA >CLOCK+1
04290 02783A 19AF 84 20 A ANDA #$20
04300 02784A 19B1 48 LSLA
04310 02785A 19B2 34 02 A PSHS A
04320 02786A 19B4 C6 20 A LDB #32
04330 02787A 19B6 A6 84 A B17WT3 LDA ,X
04340 02788A 19B8 84 BF A ANDA #%10111111
04350 02789A 19BA AA E4 A ORA ,S
04360 02790A 19BC A7 80 A STA ,X+
04370 02791A 19BE 5A DECB
04380 02792A 19BF 26 F5 19B6 BNE B17WT3
04390 02793A 19C1 35 02 A PULS A
04400 02794A 19C3 20 DC 19A1 BRA B17WT2
04410 02795A 19C5 39 B17WTX RTS
04420 02796 *  

04430 02797A 19C6 A7 66 A B17C STA 6,S PAGES AVAILABLE
04440 02798 *  

04450 02799 * LOOP TO COPY FILE
04460 02800A 19CB EE 6C A B17D LDU 12,S SOURCE
04470 02801A 19CA FC 0625 A LDD >OLYLOC
04480 02802A 19CD ED CB 24 A STD DCBBUF,U
04490 02803A 19D0 6D 67 A TST 7,S SWAPPING?
04500 02804A 19D2 27 06 19DA BEQ B17D0 IF NO
04510 02805 * WAIT FOR SOURCE DISKETTE
04520 02806A 19D4 AE 68 A LDX 8,S
04530 02807A 19D6 30 08 A LEAX B17M1-B17,X
04540 02808A 19D8 BD B7 1991 BSR B17WTE
04550 02809A 19DA B17D0 DOS OPEN,INPUT
04560 02810A 19E0 26 95 1977 BNE B17X IF NOT FOUND
04570 02811A 19E2 EC 62 A LDD 2,S
04580 02812A 19E4 ED CB 29 A STD DCBPRN,U SET STARTING SECTOR NUMBER
04590 02813A 19E7 26 12 19FB BNE B17E IF NOT FIRST TIME
04600 02814 * FIRST TIME - SAVE DIRECTORY DATA IN OUTPUT DCB
04610 02815A 19E9 10AE 6A A LDY 10,S
04620 02816A 19EC 33 4B A LEAU 11,U
04630 02817A 19EE 31 2B A LEAY 11,Y EXCEPT FOR NAME
04640 02818A 19F0 C6 15 A LDB #32-11
04650 02819A 19F2 A6 C0 A B17D1 LDA ,U+
04660 02820A 19F4 A7 A0 A STA ,Y+
04670 02821A 19F6 5A DECB
04680 02822A 19F7 26 F9 19F2 BNE B17D1
04690 02823A 19F9 EE 6C A LDU 12,S SOURCE
04700 02824A 19FB 6F E4 A B17E CLR ,S SECTORS IN MEMORY
04710 02825 *  

04720 02826 * LOAD LOOP
04730 02827A 19FD BD 0D9F A B17F JSR CSENT XLATE PRN INTO TRACK & SECTOR
04740 02828A 1A00 26 1E 1A20 BNE B17F1 IF OUT OF RANGE
04750 02829A 1A02 BD 0CE7 A JSR DSKRED DO PHYSICAL I/O
04760 02830A 1A05 26 16 1A1D BNE B17XX IF I/O ERR
04770 02831A 1A07 EC CB 29 A LDD DCBPRN,U
04780 02832A 1A0A C3 0001 A ADDD #1
04790 02833A 1A0D ED CB 29 A STD DCBPRN,U
04800 02834A 1A10 6C CB 24 A INC DCBBUF,U
04810 02835A 1A13 6C E4 A INC ,S COUNT SECTORS READ
04820 02836A 1A15 E6 E4 A LDB ,S
04830 02837A 1A17 E1 66 A CMPB 6,S IS BUFFER FULL

```

PAGE 050 OLY2 .SA:0 DOS - PAGING & OVERLAYS

```

04840 02838A 1A19 26 E2 19FD     BNE   B17F     IF NO
04850 02839A 1A1B 20 05 1A22     BRA   B17G     GO WRITE IT
04860 02840A 1A1D 16 FF57 1977 B17XX LBRA   B17X     THIS STMT USED AS AN UP-LINK
04870 02841          *
04880 02842          * INPUT AT END - SET EOF SW
04890 02843A 1A20 6C 61 A B17F1 INC 1,S
04900 02844          *
04910 02845          * CLOSE INPUT
04920 02846A 1A22 EC CB 29 A B17G LDD DCBPRN,U
04930 02847A 1A25 ED 62 A STD 2,S SAVE FOR NEXT BATCH
04940 02848A 1A27          DOS CLOSE,IT
04950 02849A 1A2D A6 E4 A LDA ,S ANY SECTORS READ?
04960 02850A 1A2F 27 EC 1A1D BEQ B17XX IF NO, I'M DONE
04970 02851          *
04980 02852          * OPEN OUTPUT
04990 02853A 1A31 6D 67 A TST 7,S SWAPPING?
05000 02854A 1A33 27 08 1A3D BEQ B17H IF NO
05010 02855A 1A35 AE 6B A LDX 8,S
05020 02856A 1A37 30 88 28 A LEAX B17M2-B17,X
05030 02857A 1A3A 17 FF54 1991 LBSR B17WTE WAIT FOR DESTINATION DISKETTE
05040 02858A 1A3D EE 6A A B17H LDU 10,S OUTPUT FILE DCB
05050 02859A 1A3F FC 0625 A LDD >OLYLOC START OF BUFFER
05060 02860A 1A42 ED CB 24 A STD DCBBUF,U
05070 02861A 1A45          DOS OPEN,OUTPUT+FAST
05080 02862A 1A4B 27 06 1A53 BEQ B17H1 IF FILE EXISTS
05090 02863A 1A4D B1 0C A CMPA #12
05100 02864A 1A4F 27 10 1A61 BEQ B17H2 IF CREATED
05110 02865A 1A51 20 CA 1A1D BRA B17XX IF OTHER ERROR
05120 02866          *
05130 02867          * FILE EXISTS
05140 02868A 1A53 EC 64 A B17H1 LDD 4,S
05150 02869A 1A55 26 12 1A69 BNE B17I IF NOT FIRST TIME, ITS OK
05160 02870A 1A57          DOS CLOSE,IT
05170 02871A 1A5D 86 1E A LDA #ERR30
05180 02872A 1A5F 20 BC 1A1D BRA B17XX
05190 02873          *
05200 02874          * FILE CREATED
05210 02875A 1A61 EC 64 A B17H2 LDD 4,S
05220 02876A 1A63 27 04 1A69 BEQ B17I IF FIRST TIME, OK
05230 02877A 1A65 86 1F A LDA #ERR31 MISC ERR
05240 02878A 1A67 20 B4 1A1D BRA B17XX
05250 02879          *
05260 02880A 1A69 ED CB 29 A B17I STD DCBPRN,U
05270 02881          *
05280 02882          * WRITE LOOP
05290 02883A 1A6C BD 0D9F A B17J JSR CSENT XLATE PRN INTO TRACK & SECTOR
05300 02884A 1A6F 26 AC 1A1D BNE B17XX
05310 02885A 1A71 BD 0CEA A JSR DSKWRT WRITE SECTOR
05320 02886A 1A74 1026 FEFF 1977 LBNB B17X
05330 02887A 1A78 EC CB 29 A LDD DCBPRN,U
05340 02888A 1A7B C3 0001 A ADDD #1
05350 02889A 1A7E ED CB 29 A STD DCBPRN,U
05360 02890A 1A81 6C CB 24 A INC DCBBUF,U
05370 02891A 1A84 6A E4 A DEC ,S COUNT DOWN SECTORS WRITTEN
05380 02892A 1A86 26 E4 1A6C BNE B17J
05390 02893          *
05400 02894          * CLOSE OUTPUT
05410 02895A 1A88 EC CB 29 A LDD DCBPRN,U

```

PAGE 051 OLY2 .SA:0 DOS - PAGING & OVERLAYS

```

05420 02896A 1A88 ED 64 A STD 4,S SAVE FOR NEXT BATCH
05430 02897A 1ABD B3 0001 A SUBD #1
05440 02898A 1A90 ED C8 14 A STD DCBMRB,U
05450 02899A 1A93 AE 6C A LDX 12,S SOURCE DCB
05460 02900A 1A95 EC 0E A LDD DCBNLS,X
05470 02901A 1A97 ED 4E A STD DCBNLS,U
05480 02902A 1A99 E7 C8 16 A STB DCBMRB+2,U
05490 02903A 1A9C DOS CLOSE,IT
05500 02904A 1AA2 4F CLRA
05510 02905A 1AA3 6D 61 A TST 1,S AT EOF?
05520 02906A 1AA5 1026 FF74 1A1D LBNE B17XX I'M DONE
05530 02907A 1AA9 16 FF1C 19C8 LBRA B17D GO COPY ANOTHER BATCH OF SECTORS
05540 02908*
05550 02909*****
05560 02910* GET MULTIPLE USER INPUTS
05570 02911* GIVEN B=NUMBER OF INPUTS
05580 02912*****
05590 02913A 1AAC 0099 A B1B FDB B19-B18
05600 02914 0012 A INPTS EQU 18
05610 02915A 1AAE 86 01 A LDA #1
05620 02916A 1AB0 34 06 A PSHS D
05630 02917A 1AB2 CE 0400 A B1BB LDU $$400
05640 02918A 1AB5 E6 E4 A LDB ,S
05650 02919A 1AB7 A6 C0 A B1BC LDA ,U+
05660 02920A 1AB9 81 5B A CMPA $$5B E
05670 02921A 1ABB 27 09 1AC6 BEQ B18D
05680 02922A 1ABD 1183 0600 A CMPU $$600
05690 02923A 1AC1 25 F4 1AB7 BCS B1BC
05700 02924A 1AC3 CE 0401 A LDU $$401
05710 02925A 1AC6 5A B1BD DECB
05720 02926A 1AC7 26 EE 1AB7 BNE B1BC
05730 02927* INPUT A FIELD
05740 02928A 1AC9 7F 0621 A CLR CLOCK+1
05750 02929A 1ACC 30 5F A LEAX -1,U
05760 02930A 1ACE F6 0621 A B1BE LDB CLOCK+1
05770 02931A 1AD1 C4 10 A ANDB #16
05780 02932A 1AD3 27 04 1AD9 BEQ B18E1
05790 02933A 1AD5 86 5B A LDA $$5B
05800 02934A 1AD7 20 02 1ADB BRA B18E2
05810 02935A 1AD9 86 18 A B18E1 LDA $$1B
05820 02936A 1ADB A7 84 A B18E2 STA ,X
05830 02937A 1ADD 34 50 A PSHS X,U
05840 02938A 1ADF SYSTEM POLCAT
05850 02939A 1AE3 35 50 A PULS X,U
05860 02940A 1AE5 4D TSTA
05870 02941A 1AE6 27 E6 1ACE BEQ B1BE
05880 02942A 1AE8 81 03 A CMPA #BREAK
05890 02943A 1AEA 27 50 1B3C BEQ B1BX
05900 02944A 1AEC 81 0A A CMPA #DOWN
05910 02945A 1AEE 27 1A 1B0A BEQ B1BF
05920 02946A 1AF0 81 5E A CMPA #UP
05930 02947A 1AF2 27 21 1B15 BEQ B1BG
05940 02948A 1AF4 81 0D A CMPA #ENTER
05950 02949A 1AF6 27 44 1B3C BEQ B1BX
05960 02950A 1AF8 81 08 A CMPA #LEFT
05970 02951A 1AFA 27 34 1B30 BEQ B1BI
05980 02952A 1AFC 81 20 A CMPA $$20
05990 02953A 1AFE 25 CE 1ACE BCS B1BE

```

PAGE 052 OLY2 .SA:0 DOS - PAGING & OVERLAYS

```

06000 02954A 1B00 81 5B     A      CMPA   #$5B
06010 02955A 1B02 25 20    1B24   BCS    B1BH
06020 02956A 1B04 81 60     A      CMPA   #$60
06030 02957A 1B06 25 1C    1B24   BCS    B1BH
06040 02958A 1B08 20 C4   1ACE   BRA    B18E
06050 02959          * DOWN
06060 02960A 1B0A A6 E4   A B18F   LDA    ,S
06070 02961A 1B0C A1 61   A      CMPA   ,S
06080 02962A 1B0E 24 0E   1B1E   BCC    B18G1 IF AT END ALREADY
06090 02963A 1B10 4C
06100 02964A 1B11 A7 E4   A      STA    ,S
06110 02965A 1B13 20 09   1B1E   BRA    B18G1
06120 02966          * UP
06130 02967A 1B15 A6 E4   A B18G   LDA    ,S
06140 02968A 1B17 81 01   A      CMPA   #1
06150 02969A 1B19 27 03   1B1E   BEQ    B18G1
06160 02970A 1B1B 4A
06170 02971A 1B1C A7 E4   A      STA    ,S
06180 02972A 1B1E 86 5B   A B18G1  LDA    #$5B
06190 02973A 1B20 A7 84   A      STA    ,X
06200 02974A 1B22 20 8E   1B22   BRA    B18B
06210 02975          * TEXT CHR
06220 02976A 1B24 8A 40   A B18H   ORA    #$40
06230 02977A 1B26 A7 C0   A      STA    ,U+
06240 02978A 1B28 A6 C4   A      LDA    ,U
06250 02979A 1B2A 81 5D   A      CMPA   #$5D  J
06260 02980A 1B2C 27 DC   1B0A   BEQ    B18F
06270 02981A 1B2E 20 9E   1ACE   BRA    B18E
06280 02982          * BACK ARROW
06290 02983A 1B30 A6 C2   A B18I   LDA    ,-U
06300 02984A 1B32 84 BF   A      ANDA   #$BF
06310 02985A 1B34 81 1B   A      CMPA   #$1B
06320 02986A 1B36 26 96   1ACE   BNE    B18E
06330 02987A 1B38 A6 C0   A      LDA    ,U+
06340 02988A 1B3A 20 F4   1B30   BRA    B18I
06350 02989          * BREAK OR ENTER
06360 02990A 1B3C 1F 89   A B18X   TFR    A,B
06370 02991A 1B3E 32 62   A      LEAS   2,S
06380 02992A 1B40 86 5B   A      LDA    #$5B
06390 02993A 1B42 A7 84   A      STA    ,X
06400 02994A 1B44 39
06410 02995          *
06420 02996          *****
06430 02997          * SCAN FOR SELECTED DIRECTORY ENTRY
06440 02998          *****
06450 02999A 1B45 0089  A B19   FDB    B20-B19
06460 03000 0013  A SCNDIR EQU   19
06470 03001A 1B47 BE 0006  A      LDX    $C006 PARAMETER AREA
06480 03002A 1B4A 86 02   A      LDA    #2 READ
06490 03003A 1B4C A7 80   A      STA    ,X+
06500 03004A 1B4E A6 C4   A      LDA    ,U DRIVE
06510 03005A 1B50 A7 80   A      STA    ,X+
06520 03006A 1B52 CC 1103  A      LDD    #$1103 TRACK & SECTOR
06530 03007A 1B55 ED 80   A      STD    ,X+ LEAVE X -> SECTOR
06540 03008A 1B57 108E 06CB  A      LDY    #SYSBUF
06550 03009A 1B5B 10AF 01   A      STY    1,X
06560 03010A 1B5E A6 41   A      LDA    1,U STARTING OCCURANCE
06570 03011A 1B60 81 48   A B19A  CMPA   #72 ANY MORE ON THIS DRIVE?

```

PAGE 053 OLY2 .SA:0

DOS - PAGING & OVERLAYS

```

06580 03012A 1B62 24 65 1BC9     BCC   B19NO
06590 03013A 1B64 80 08 A B19B    SUBA  #8
06600 03014A 1B66 25 04 1B6C    BCS   B19C   IF IN THIS SECTOR
06610 03015A 1B68 C6 84 A INC   ,X
06620 03016A 1B6A 20 F8 1B64    BRA   B19B
06630 03017A 1B6C 88 08 A B19C    ADDA  #8
06640 03018A 1B6E 27 07 1B77    BEQ   B19D
06650 03019A 1B70 C6 20 A LDB   #32
06660 03020A 1B72 3D           MUL
06670 03021A 1B73 31 AB A LEAY  D,Y   DISPLACEMENT IN THIS SECTOR
06680 03022A 1B75 20 0C 1B83    BRA   B19D1  OFFSET TO 1ST ENT TO SCAN
06690 03023A 1B77 34 70 A B19D    PSHS  X,Y,U
06700 03024A 1B79 AD 9F C004 A   JSR   [$C004]
06710 03025A 1B7D 35 70 A PULS  X,Y,U
06720 03026A 1B7F A6 03 A LDA   3,X   RESULT
06730 03027A 1B81 26 46 1BC9    BNE   B19NO  IF I/O ERR
06740 03028           * COMPARE AGAINST ARGUMENT
06750 03029           * REGISTERS:X->SECTOR NBR
06760 03030           * Y->ENTRY IN BUFFER
06770 03031           * U->SEARCH ARGUMENT
06780 03032A 1B83 34 60 A B19D1 PSHS  Y,U
06790 03033A 1B85 C6 0B A LDB   #11   BYTES TO COMPARE
06800 03034A 1B87 33 42 A LEAU  2,U   TO START OF ARGUMENT
06810 03035A 1B89 A6 A4 A LDA   ,Y
06820 03036A 1B8B 27 0C 1B99    BEQ   B19E1  IF EMPTY ENTRY
06830 03037A 1B8D 2B 0A 1B99    BMI   B19E1  IF END OF DIRECTORY
06840 03038A 1B8F A6 C0 A B19E    LDA   ,U+
06850 03039A 1B91 81 2A A CMPA  #'*   WILDCARD?
06860 03040A 1B93 27 1F 1BB4    BEQ   B19F
06870 03041A 1B95 A1 A0 A CMPA  ,Y+
06880 03042A 1B97 27 1D 1BB6    BEQ   B19G
06890 03043           * NO MATCH
06900 03044A 1B99 35 60 A B19E1 PULS  Y,U
06910 03045A 1B9B 6C 41 A INC   1,U
06920 03046A 1B9D A6 41 A LDA   1,U
06930 03047A 1B9F B1 48 A CMPA  #72   ANY MORE?
06940 03048A 1BA1 24 26 1BC9    BCC   B19NO
06950 03049A 1BA3 31 A8 20 A LEAY  32,Y   POINT AT NEXT ENTRY
06960 03050A 1BA6 108C 07C8 A   CMPY  #$SYSBUF+256
06970 03051A 1BAA 25 D7 1B83    BCS   B19D1
06980 03052           * READ NEXT SECTOR
06990 03053A 1BAC 108E 06C8 A   LDY   #$SYSBUF
07000 03054A 1BB0 6C 84 A INC   ,X
07010 03055A 1BB2 20 C3 1B77    BRA   B19D
07020 03056A 1BB4 A6 A0 A B19F    LDA   ,Y+   BYPASS SOURCE CHR
07030 03057A 1BB6 5A           B19G    DECB
07040 03058A 1BB7 26 D6 1BBF    BNE   B19E
07050 03059           * MATCH FOUND
07060 03060A 1BB9 35 60 A PULS  Y,U
07070 03061A 1BBB 6C 41 A INC   1,U   SEARCH CONTINUES WITH NEXT ENTRY
07080 03062A 1BBD 33 4D A LEAU  2+11,U
07090 03063A 1BBF C6 20 A LDB   #32
07100 03064A 1BC1 A6 A0 A B19H    LDA   ,Y+
07110 03065A 1BC3 A7 C0 A STA   ,U+
07120 03066A 1BC5 5A           DECB
07130 03067A 1BC6 26 F9 1BC1    BNE   B19H
07140 03068A 1BC8 39           RTS
07150 03069A 1BC9 B6 FF A B19NO LDA   #$FF

```

PAGE 054 OLY2 .SA:0 DOS - PAGING & OVERLAYS

07160	03070A	1BCB	A7	41	A	STA	1,U	SAY NO MORE
07170	03071A	1BCD	39			RTS		
07180	03072				*			
07190	03073A	1BCE		0001	A B20	RMB	1	
07200	03074A	1BCF		0001	A B21	RMB	1	
07210	03075				*			
07220	03076				OPT	L		
07230	03077A	1BD0		0001	A LASTPG	RMB	1	END OF OVERLAYS
07240	03078			1BD0	A B22	EQU	LASTPG	
07250	03079			00C5	A BKSZ	EQU	B17-B16	
07260	03080			0718	A PGMSZ	EQU	OVRLAY-ORGIN-1	
07270	03081			1246	A TOTSZ	EQU	LASTPG-ORGIN-1	
07280	03082			0EA4	A START	EQU	DOS - START OF DISK FILE	
07290	03083			20EB	A END	EQU	LASTPG+DOS-ORGIN END OF DISK FILE	
07300	03084			10A2	A ENTRY	EQU	OVRLAY INITIAL ENTRY POINT INTO PROGRAM	
07310	03085			10A5	A LOWUSR	EQU	OVRLAY+3 LOWEST POINT WHERE USER PGM CAN LOAD	
07320	03086			0BB9	A FIXIT	EQU	\$1E00-LASTPG+ORGIN POINT THAT BASIC CLOBBERS	
07330	03087				TTL		DOS - CROSS REFERENCE	
07340	03088				END			
TOTAL ERRORS 00000--00000								
TOTAL WARNINGS 00000--00000								

Index

&	35
\$	35
@	35
[.	43
]	43
/AO (Absolute origin)	25
/IM (Assemble into memory)	25
/IM Switch	27
/LP (Assembler listing)	25
/MO (Manual origin)	25, 29
/NL (No listing)	25
/NO (No object code in memory)	25
/NS (No symbol table)	25
/SR (Single record)	25
/SR "switch"	7
/SS (Short screen listing)	25
/WE (Wait on assembly errors)	25
/WS (With symbols)	25
6809	41
6809 Mnemonics, Reference	109
6809 Registers	41
 — A —	
Absolute Origin Assembly	28
Addressing-Mode Characters	10
Addressing Modes	42
Direct Addressing	45
Extended Addressing	43
Immediate Addressing	43
Indexed Addressing	43
Indirect Addressing	43
Inherent Addressing	43
Relative Addressing	44
Alphanumeric Character Codes	106
Arithmetic Operators	36
Addition (+)	36
Subtraction (-)	36
Multiplication (*)	36
Division (.DIV.)	36
Modulus (.MOD.)	36
Positive (+)	36
Negative (-)	36
ASCII Codes	105
Alphanumeric Character	106
Color	105
Graphic Character	105
Video Control	105
ASCII Mode	18
Assembler Commands	25
Assembler Commands and Switches, Reference	75
 — B —	
Assembler Pseudo Ops, Reference	85
Assembling	25
Assembling for DOS	30
Assembling for Stand-Alone ZBUG	30
Assembly Display Listing	26
Assembly Listing, Changing	49
COND	49
ENDC	49
INCLUDE	50
OPT	49
PAGE	49
TITLE	49
 — C —	
Backups	3
BASIC Command	23
Breakpoints	32
Buffers	61
Byte Mode	17
 — D —	
Data Control Block	61
Data Control Block (DCB), Reference	91
Defining Symbols	47
EQU	48
SET	48
Delete Command	22
Drange	22
Direct Access	65
Direct Addressing	45
Directory	15
Disk Allocation Map	15
Disk Assembly	30
Assembling for DOS	30
Assembling Stand-Alone ZBUG	30
Display Modes	31
Half-Symbolic Mode	32

Numeric Mode.....	32
Symbolic Mode	32
DOS Error Codes, Reference	101
DOS Routines.....	10, 61
DOS Routines, Reference	95
Drange	22
 — E —	
Edit Command	21
Eline	21
Editor Commands, Reference	71
EDTASM	5
EDTASMOV	5
Eline	21
END	47
ENDC	49
EQU	48
Error Codes, DOS Reference	101
Error Messages, EDTASM Reference	81
Examination Modes	17
ASCII Mode	18
Byte Mode	17
Mnemonic Mode	18
Word Mode	18
Examining Memory	17
Examining Registers and Flags	33
Executing a Program from ZBUG	32
Extended Addressing.....	43
Indirect Addressing.....	43
Extended Indirect Addressing	43
 — F —	
FCB.....	48
FCC	48
FDB.....	48
Flags, Examining	33
FLDFLG	27
Formatting	3
 — G —	
Graphic Character Codes.....	105
 — H —	
Half-Symbolic Mode.....	32
Hrange	21
 — I —	
Immediate Addressing.....	43
INCLUDE.....	50
Indexed Addressing.....	43
Indirect Addressing.....	44
Indexed Indirect Addressing.....	44
Indirect Addressing	43
Inherent Addressing.....	43
Input Mode.....	35
Insert Command	22
Istartline, increment	22
Inserting Data	48
FCB.....	48
FCC.....	48
FDB.....	48
RMB	48
Istartline, increment	22
 — L —	
Label.....	42
LD filespec	23
LDA filespec	23
Left bracket (I)	6
LINCNT	27
Load Command	23
LD filespec	23
LDA filespec	23
Logical Operators.....	37
Shift (<)	37
LogicalAND (.AND.)	37
InclusiveOR (.OR.)	37
ExclusiveOR (.XOR.)	37
Complement (.NOT.)	37
 — M —	
Macro Call	53
Macro, Calling	51
Macro, Defining	51
Macro, Dummy Values	53
Macro, Format	52
Macro Definition	52
Macro, Passing Values	52
Macros	51
Manual Origin Assembly	29
Memory Map	103
Mnemonic Mode	18
Mnemonics	10
Mnemonics, 6809 Reference	109
 — N —	
Nstartline, increment	22
Numbering System Modes	35
Input Mode	35
Output Mode	35
Numeric Mode	32
 — O —	
Opcode	9
Opening a Disk File	62
Operands	36
Operations	36

Operands	36	Pseudo Ops.....	10, 47
Operators	36	Pseudo Ops, Reference	85
Arithmetic	36		
Logical	37		
Relational	37		
Complex Operations.....	37		
Operators.....	10, 36	— R —	
Arithmetic	36	Read/Write Option	66
Logical	37	Reading a Disk File	65
Relational	37	Read to a File Sample Program	67
OPT	49	Registers	9
ORG	47	6809	41
Origination Offset Assembly.....	28	Registers, Examining	33
Output Mode	35	Relational Operators	37
		Equal to (.EQU.)	37
		Not Equal to (.NEQ.)	37
		Relative Addressing	44
		Renumber Command	22
		<i>Nstartline, increment</i>	22
		Replace Command	22
		<i>Rstartline, increment</i>	22
		Right bracket (])	6
		RMB	48
		ROM Routines	10, 57
		CHROUT	58
		POLCAT	57
		ROM Routines, Reference	89
		Routines	
		DOS	10
		ROM	10
		<i>Rstartline, increment</i>	22
		— S —	
		Sample Program.....	5, 11
		Sample Programs	125
		Saving Memory from ZBUG	34
		SET	48
		Sequential Access	65
		Single Stepping	33
		Switches	
		/AO	25
		/IM	25
		/LP	25, 27
		/MO	25
		/NL	25, 27
		/NO	25
		/NS	25, 27
		/SR	7, 25
		/SS	25
		/WE	25, 27
		/WS	25
		Symbolic Mode	32
		Symbols	10
		Examine Memory	32

— T —	
TITLE	49
Trange	21
Transferring Memory Blocks.....	33
— V —	
Video Control Codes.....	105
— W —	
WD <i>filespec</i>	23
Word Mode	18
— Z —	
Write Command	23
WD <i>filespec</i>	23
Write to a File Sample Program	67
Writing a Disk File	65
ZBUG Calculator	35
ZBUG Command	22
ZBUG Commands	17, 31
ZBUG Commands Reference.....	77

RADIO SHACK, A DIVISION OF TANDY CORPORATION

**U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5**

TANDY CORPORATION

AUSTRALIA

91 KURRAJONG ROAD
MOUNT DRAUIT, N.S.W. 2770

BELGIUM

PARC INDUSTRIEL DE NANINNE
5140 NANINNE

U. K.

BILSTON ROAD WEDNESBURY
WEST MIDLANDS WS10 7JN