DISK EDTASM™

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.

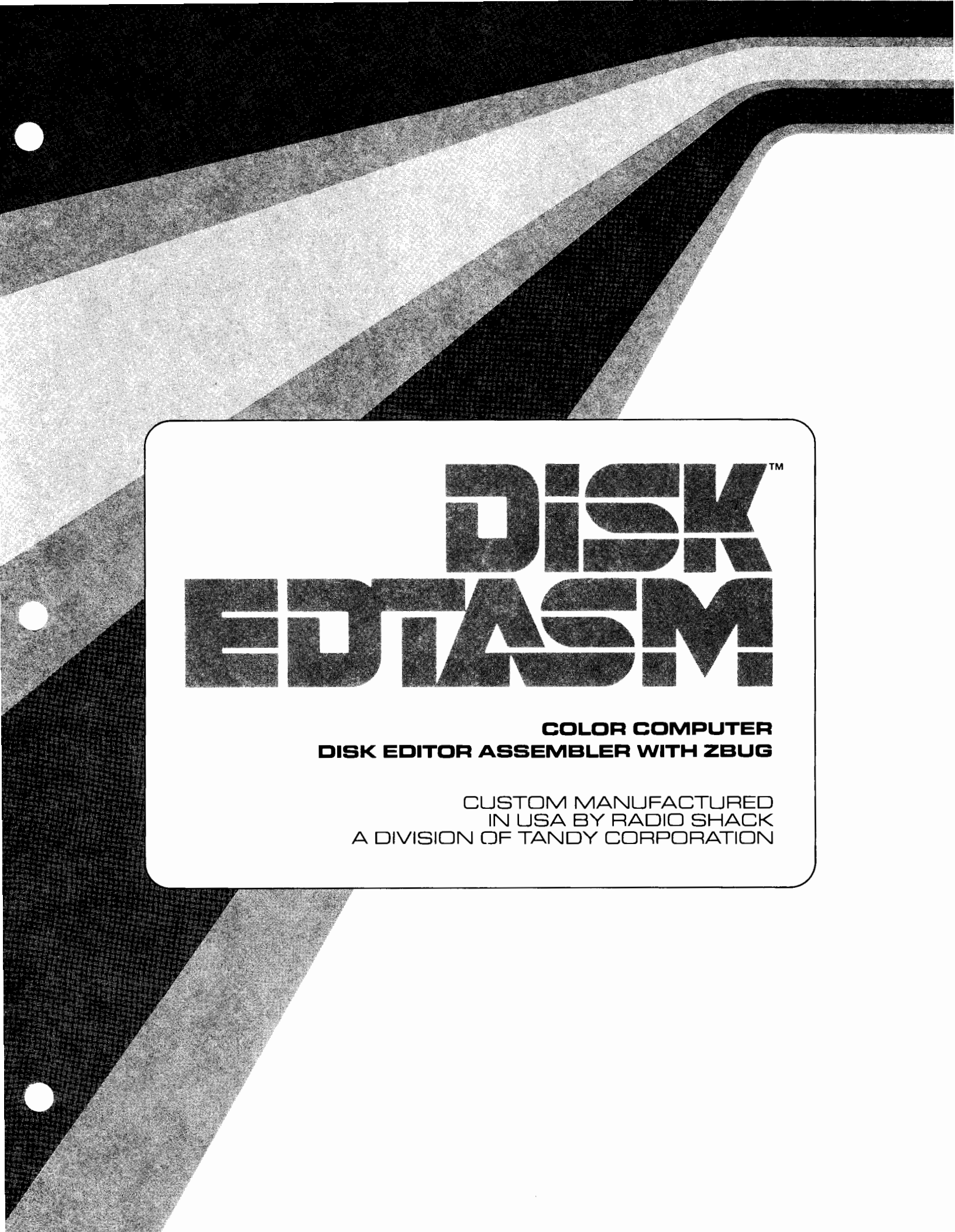
V. APPLICABILITY DF WARRANTY

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 EDTASM™

COLOR COMPUTER
DISK EDITOR ASSEMBLER WITH ZBUG

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

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.
Italics	To denote a value you must supply.
filespec	To denote a DOS file specification. A DOS filespec is in one of these formats: *filename*/ext:drive *filename*.ext:drive *filename* has one to eight characters. *extension* has one to three characters. *drive* 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

Section II/ Commands

Section III/ Assembly Language

Section IV/ ROM and DOS Routines

Section V/ Reference

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:

   `DSKINI0` (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 0 TO 0` (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 0 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/IM/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    80          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

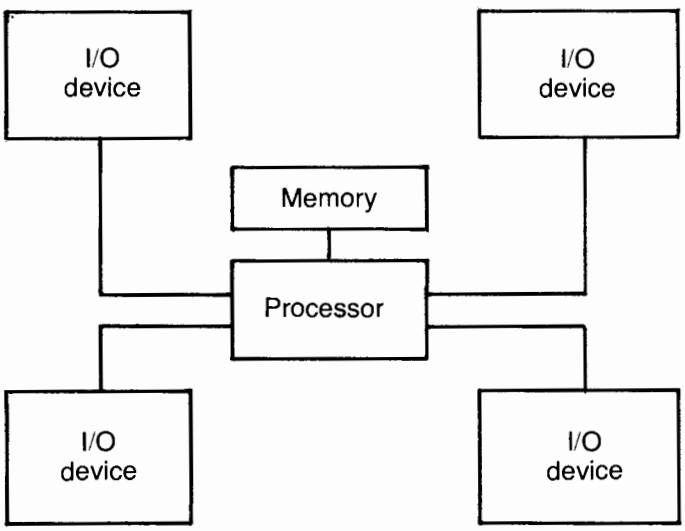
This chapter is for beginning assembly language programmers. It explains some concepts you need. If you're not a beginner, use this chapter as a refresher or skip it.

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 contained 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.