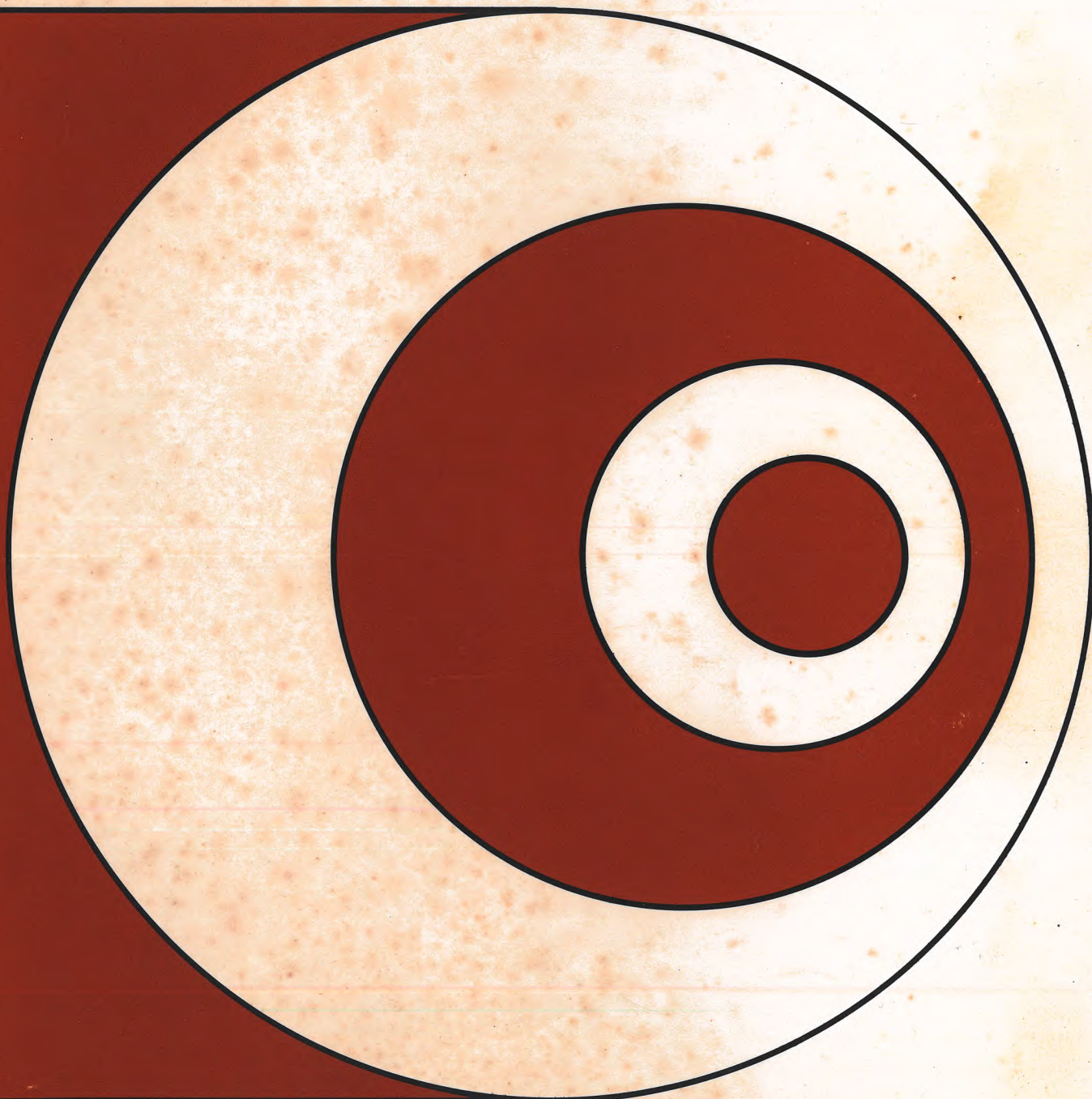


**SIGNETICS TWIN  
OPERATOR'S  
GUIDE**



# **SIGNETICS TWIN OPERATOR'S GUIDE**

**signetics**

a subsidiary of U.S. Philips Corporation

Signetics Corporation  
P.O. Box 9052  
811 East Arques Avenue  
Sunnyvale, California 94086  
Telephone 408/739-7700

# REVISION RECORD

REVISION	DESCRIPTION
A	Total Revision to SDOS 3.0 Level 3-1-79
B	Corrections 6-1-80
TW09003000	

Signetics TWIN Operator's Guide

Address comments concerning this manual to:

Applications Department  
MOS Microprocessors  
Signetics Corporation  
811 E. Arques Avenue  
Sunnyvale, California 94086

©1979  
Signetics Corporation  
Printed in the United States of America



# CONTENTS

<u>Chapter</u>	<u>Title</u>	<u>Page</u>
1	THE TWIN SYSTEM . . . . .	1-1
1.0	Introduction . . . . .	1-1
1.1	TWIN Overview . . . . .	1-1
1.2	About This Book . . . . .	1-5
1.3	Manual Content . . . . .	1-5
2	SYSTEM DESCRIPTION . . . . .	2-1
2.0	Introduction . . . . .	2-1
2.1	Hardware . . . . .	2-1
2.1.1	TWIN Development Computer . . . . .	2-1
2.1.2	Dual Floppy Disk Subsystem . . . . .	2-3
2.2	Peripherals . . . . .	2-4
2.2.1	CRT Terminal . . . . .	2-4
2.2.2	ASR-33 Teletypewriter . . . . .	2-4
2.2.3	Line Printer . . . . .	2-4
2.2.4	User-Supplied Peripherals . . . . .	2-4
2.3	Software . . . . .	2-6
2.3.1	SDOS . . . . .	2-6
2.3.2	The Debug Monitor . . . . .	2-6
2.3.3	PROM Programming . . . . .	2-6
2.3.4	The Editor . . . . .	2-7
2.3.5	The Assemblers . . . . .	2-7
2.3.6	Systems Readiness Test . . . . .	2-7
3	SYSTEM OPERATION . . . . .	3-1
3.0	Introduction . . . . .	3-1
3.1	Unpacking . . . . .	3-1
3.1.1	Unpacking the TWIN Development Computer . . . . .	3-1
3.1.2	Unpacking the CRT Terminal . . . . .	3-1
3.1.3	Unpacking the Floppy Disk Unit . . . . .	3-4
3.1.4	Unpacking the Line Printer . . . . .	3-4
3.1.5	Installing the TWICE Debug Cable . . . . .	3-4
3.2	Interconnection and Physical Installation . . . . .	3-4
3.2.1	Power Requirements . . . . .	3-4
3.2.2	Interconnection . . . . .	3-5
3.3	System Controls and Indicators . . . . .	3-5
3.3.1	Development Computer . . . . .	3-5
3.3.2	Dual Floppy Disk Unit . . . . .	3-9
3.3.3	CRT Terminal . . . . .	3-9
3.3.4	Printer . . . . .	3-9
3.4	Operation . . . . .	3-9
3.4.1	Manual Reset . . . . .	3-11
4	SIGNETICS DISK OPERATING SYSTEM . . . . .	4-1
4.0	Introduction . . . . .	4-1
4.1	System Description . . . . .	4-1
4.1.1	Resident SDOS . . . . .	4-1
4.1.2	SDOS Overlays . . . . .	4-2



<u>Chapter</u>	<u>Title</u>	<u>Page</u>
	4.1.3 System Slave Jobs . . . . .	4-3
	4.1.4 Debug Utility Programs . . . . .	4-3
4.2	File Management and Diskettes . . . . .	4-3
	4.2.1 Files, Devices, and Channels . . . . .	4-4
	4.2.2 File Names . . . . .	4-6
	4.2.3 File Directories . . . . .	4-7
4.3	Entering SDOS Commands . . . . .	4-7
	4.3.1 Command Description . . . . .	4-7
	4.3.2 Command Completion . . . . .	4-9
	4.3.3 Error Reporting . . . . .	4-9
4.4	Special Keys . . . . .	4-12
	4.4.1 Space Bar . . . . .	4-13
	4.4.2 ESC Key . . . . .	4-14
	4.4.3 Control - Z . . . . .	4-16
	4.4.4 Rub Out Key . . . . .	4-17
4.5	System Control Commands . . . . .	4-18
	4.5.1 SUSPEND . . . . .	4-19
	4.5.2 CONT . . . . .	4-20
	4.5.3 ABORT. . . . .	4-21
4.6	System Options . . . . .	4-22
	4.6.1 SYSTEM . . . . .	4-23
	4.6.2 DEVICE . . . . .	4-24
	4.6.3 ICE . . . . .	4-25
	4.6.4 ASSIGN . . . . .	4-26
	4.6.5 CLOSE . . . . .	4-27
4.7	Diskette and File Utilities . . . . .	4-28
	4.7.1 FORMAT . . . . .	4-29
	4.7.2 VERIFY . . . . .	4-31
	4.7.3 DUP . . . . .	4-32
	4.7.4 LDIR . . . . .	4-34
	4.7.5 RENAME . . . . .	4-35
	4.7.6 COPY . . . . .	4-37
	4.7.7 PRINT and PRINTL . . . . .	4-38
	4.7.8 DELETE . . . . .	4-39
	4.7.9 CMPF . . . . .	4-40
	4.7.10 DFIL . . . . .	4-43
4.8	System Utility Commands . . . . .	4-46
	4.8.1 MOVE . . . . .	4-47
	4.8.2 FILL . . . . .	4-48
	4.8.3 READ . . . . .	4-49
	4.8.4 WRITE . . . . .	4-51
	4.8.5 UPR . . . . .	4-54
4.9	Object Program Utilities . . . . .	4-55
	4.9.1 WHEX . . . . .	4-56
	4.9.2 RHEX . . . . .	4-57
	4.9.3 MODULE . . . . .	4-58
	4.9.4 WSMS . . . . .	4-59
	4.9.5 CSMS . . . . .	4-60
4.10	Command Files . . . . .	4-61
	4.10.1 Command Description . . . . .	4-63

<u>Chapter</u>	<u>Title</u>	<u>Page</u>
	4.10.2 * [Comment]. . . . .	4-64
	4.10.3 KILL . . . . .	4-65
	4.10.4 TYPE . . . . .	4-66
4.11	Standard SDOS Command and Utility Files . . . . .	4-67
	4.11.1 COPYSYS . . . . .	4-68
	4.11.2 Configure Optional Drivers . . . . .	4-69
	4.11.3 EQUATES . . . . .	4-71
5	THE TEXT EDITOR . . . . .	5-1
5.0	Introduction . . . . .	5-1
5.1	The EDIT Command . . . . .	5-2
5.2	Sample EDIT Session . . . . .	5-4
5.3	Editor Command Descriptions . . . . .	5-14
	5.3.1 Editor Command Line . . . . .	5-14
	5.3.2 Editor Command Description Conventions . . . . .	5-15
	5.3.3 Insertion . . . . .	5-17
	5.3.4 Deletion . . . . .	5-19
	5.3.5 Alteration . . . . .	5-20
	5.3.6 String Search . . . . .	5-23
	5.3.7 I/O Commands . . . . .	5-24
	5.3.8 Line Pointer Commands . . . . .	5-28
	5.3.9 Utility Commands . . . . .	5-29
	5.3.10 Macros . . . . .	5-35
5.4	Editor Messages . . . . .	5-36
6	THE ABSOLUTE ASSEMBLER . . . . .	6-1
6.0	Introduction . . . . .	6-1
6.1	Pre-Assembly Tasks . . . . .	6-1
6.2	The ASM Command . . . . .	6-2
6.3	Post-Assembly Tasks . . . . .	6-5
6.4	Assembler Errors . . . . .	6-6
6.5	Loading an Assembled Program . . . . .	6-7
6.6	The Assembler Tab Feature . . . . .	6-7
7	THE PROM PROGRAMMER . . . . .	7-1
7.0	Introduction . . . . .	7-1
7.1	On Board TWIN PROM Programming . . . . .	7-1
	7.1.1 R PROM . . . . .	7-3
	7.1.2 W PROM . . . . .	7-4
	7.1.3 C PROM . . . . .	7-6
7.2	Universal PROM Programming Interface . . . . .	7-7
	7.2.1 PROM . . . . .	7-10
8	THE DEBUGGER . . . . .	8-1
8.0	Introduction . . . . .	8-1
8.1	The Debug Package . . . . .	8-2
8.2	The Debug Command . . . . .	8-5
8.3	Sample Debug Session . . . . .	8-6
8.4	Debug SDOS Commands . . . . .	8-13
	8.4.1 GO . . . . .	8-14

<u>Chapter</u>	<u>Title</u>	<u>Page</u>
	8.4.2 LOAD . . . . .	8-15
	8.4.3 XEQ . . . . .	8-16
	8.4.4 DUMP . . . . .	8-17
	8.4.5 EXAM . . . . .	8-19
	8.4.6 PATCH . . . . .	8-21
	8.4.7 STATUS . . . . .	8-22
8.5	Debug Commands . . . . .	8-23
	8.5.1 BKPT . . . . .	8-24
	8.5.2 CLBP . . . . .	8-25
	8.5.3 RESET . . . . .	8-26
	8.5.4 SET . . . . .	8-27
	8.5.5 DSTAT . . . . .	8-29
	8.5.6 TRACE . . . . .	8-31
8.6	TWICE Debug Cable . . . . .	8-34

### APPENDICES

<u>Appendix</u>	<u>Title</u>	<u>Page</u>
A	SDOS and DEBUG COMMAND SUMMARY . . . . .	A-1
B	TEXT EDITOR COMMAND SUMMARY . . . . .	B-1
C	HEXADECIMAL OBJECT FORMAT . . . . .	C-1
D	SMS TAPE FORMAT . . . . .	D-1
E	SYSTEM READINESS TEST . . . . .	E-1
F	TWIN SUPERVISOR CALLS . . . . .	F-1
G	SRB STATUS CODES . . . . .	G-1
H	ADDING A DEVICE DRIVER TO SDOS . . . . .	H-1
I	RS232 DRIVER DESCRIPTION . . . . .	I-1

### FIGURES

<u>Figure</u>	<u>Title</u>	<u>Page</u>
1-1	Elementary Partitioning of 2650 Microcomputer System Logic . . . . .	1-3
1-2	TWIN Slave CPU Emulates User's System CPU . . . . .	1-4
2-1	Diskette . . . . .	2-5



<u>Figure</u>	<u>Title</u>	<u>Page</u>
3-1	Development Computer PC Board Layout . . . . .	3-2
3-2	Development Computer (Top View) . . . . .	3-3
3-3	Computer Front Panel . . . . .	3-6
3-4	Computer Rear Panel . . . . .	3-8
3-5	Inserting a Diskette . . . . .	3-10
4-1	Equates File . . . . .	4-71
5-1	EDIT Sample-Double Precision Add . . . . .	5-4
5-2	Entering Text and Displaying the Buffer . . . . .	5-5
5-3	FIND, SUBSTITUTE and REPLACE Commands . . . . .	5-7
5-4	Displaying the Buffer and Filing . . . . .	5-8
5-5	EDIT Sample Double Precision Add and Subtract . . . . .	5-9
5-6	Adding data to an Existing File . . . . .	5-11
5-7	Inserting Lines into the Buffer . . . . .	5-13
6-1	Sample Program . . . . .	6-3
6-2	Absolute Assembler Output . . . . .	6-4
7-1	TWIN Data I/O Interconnection Cable . . . . .	7-9
7-2	Sample Session - Universal PROM Programmer . . . . .	7-13
8-1	Sample Program To Debug . . . . .	8-7
8-2	Sample Debug Session . . . . .	8-10
E-1	System Readiness Test . . . . .	E-2
H-1	Adding a Driver to SDOS Sample . . . . .	H-3
I-1	RS232 Control Port 1 (Output) . . . . .	I-2
I-2	RS232 Control Port 1 (Input) . . . . .	I-3
I-3	RS232 Control Port 2 (Input Only) . . . . .	I-3
I-4	SRB Usage for Read Status SVC (21H) . . . . .	I-6

**TABLES**

<u>Table</u>	<u>Title</u>	<u>Page</u>
4-1A	Reserved Device Names . . . . .	4-5
4-1B	Optional Device Names . . . . .	4-5
4-2	SDOS System Program Identifiers . . . . .	4-10
4-3	SDOS Error Messages . . . . .	4-11
4-4	ESC Key Usage With SDOS Commands . . . . .	4-15
7-1	PROM Socket Usage . . . . .	7-1

<u>Table</u>	<u>Title</u>	<u>Page</u>
7-2	PROM Programming Retries . . . . .	7-5
8-1	Commands Available in Debug . . . . .	8-3
8-2	TRACE Display Format . . . . .	8-33
F-1	SVC References . . . . .	F-1
F-2	Contents of the Service Request Block . . . . .	F-2
F-3	SVC Function Codes (Hexadecimal) . . . . .	F-4
F-4	Device Identification and Type . . . . .	F-12
F-5	Device Type Code . . . . .	F-12

# CHAPTER 1

## THE TWIN SYSTEM

### 1.0 INTRODUCTION

When designing any product that includes a microprocessor, there are aspects of the development cycle which have no parallel either in combinatorial logic design or in computer program development - the two predecessors of microprocessor product development.

There is no clear-cut demarcation between logic which should be implemented using digital logic packages or logic which should be implemented using programmed instructions; that is what makes microprocessor product development unique. A successful microprocessor development system, such as TWIN, must therefore support digital logic development and object program creation with equal ease. Therein lies the strength of the TWIN system.

### 1.1 TWIN OVERVIEW

TWIN may at first look like any other general purpose minicomputer system; there is a CRT and keyboard which communicates with a box that resembles a minicomputer. Results may be created on a line printer and intermediate data or programs may be stored on diskettes.

Indeed, TWIN offers many of the program creation and execution facilities that any general purpose minicomputer system will offer. Source programs, written in assembly language, may be entered via the CRT terminal and stored on diskette. Subsequently, source programs may be retrieved from diskette, edited and stored back. An Assembler converts source programs into executable object code and a Debugger allows the object code to be conditionally executed as a means of detecting conceptual errors -- that is, instruction sequences which, though they are syntactically correct, do not accurately represent the intended logic or data flows.

The entire process of program creation and correction makes heavy use of the bulk data storage capability of diskettes. Therefore, a disk operating system is provided to automate the process of accessing diskette files by identifying file labels rather than diskette track and sector addresses.



All of the TWIN program creation and execution features are comparable to any general purpose minicomputer system. So complete is this parallel, that there would be nothing preventing TWIN from being used like any other mini-computer system -- as a text editor or even a business machine. User-written programs may access diskettes via the disk operating system; indeed the disk operating system could be included as a utility within a large user-written program.

But TWIN is much more than a general purpose minicomputer system. The typical 2650 user program created on TWIN is subsequently going to become an object program, implemented in PROM or ROM. A microprocessor object program is therefore ultimately to become a package, driving 2650-based logic, in a configuration that may not even remotely resemble a computer. The only constant that may be ascribed to 2650 based products is that they will contain a Signetics 2650 microprocessor, driven by one or more object program packages; additional logic must be present to handle the flow of data or signals to or from the microprocessor. Figure 1-1 therefore generally identifies the ultimate configuration which any microprocessor-based product will have.

Every part of the end product illustrated in Figure 1-1 may be developed using TWIN.

The process of creating an executable object program was described first, since this is the most obvious capability of a configuration that looks like a general purpose minicomputer system. But the similarities between TWIN and a general purpose minicomputer system end at this superficial level.

Consider some of the additional features which TWIN provides to serve as a total microprocessor based product development aid.

To begin with, object programs are likely to be stored in PROM or ROM devices. TWIN allows you to create the PROM, or to define the ROM mask.

The TWIN provides two CPUs. A master CPU performs monitoring and disk operating system functions; functions required by TWIN, but absent in the product being developed. A slave microprocessor takes the place of the 2650 device which must be present in the end product.

Memory is also provided in duplicate. The master CPU has its own memory, out of which it can execute monitoring and disk operating system programs. The slave CPU has separate memory which remains available for user application programs. This is illustrated in Figure 1-2. When appropriate, TWIN allows the master CPU to access slave processor memory. The separation of programs between master and slave memories is not exactly a "system" versus "user" division, but that is of little concern to you as the TWIN user.

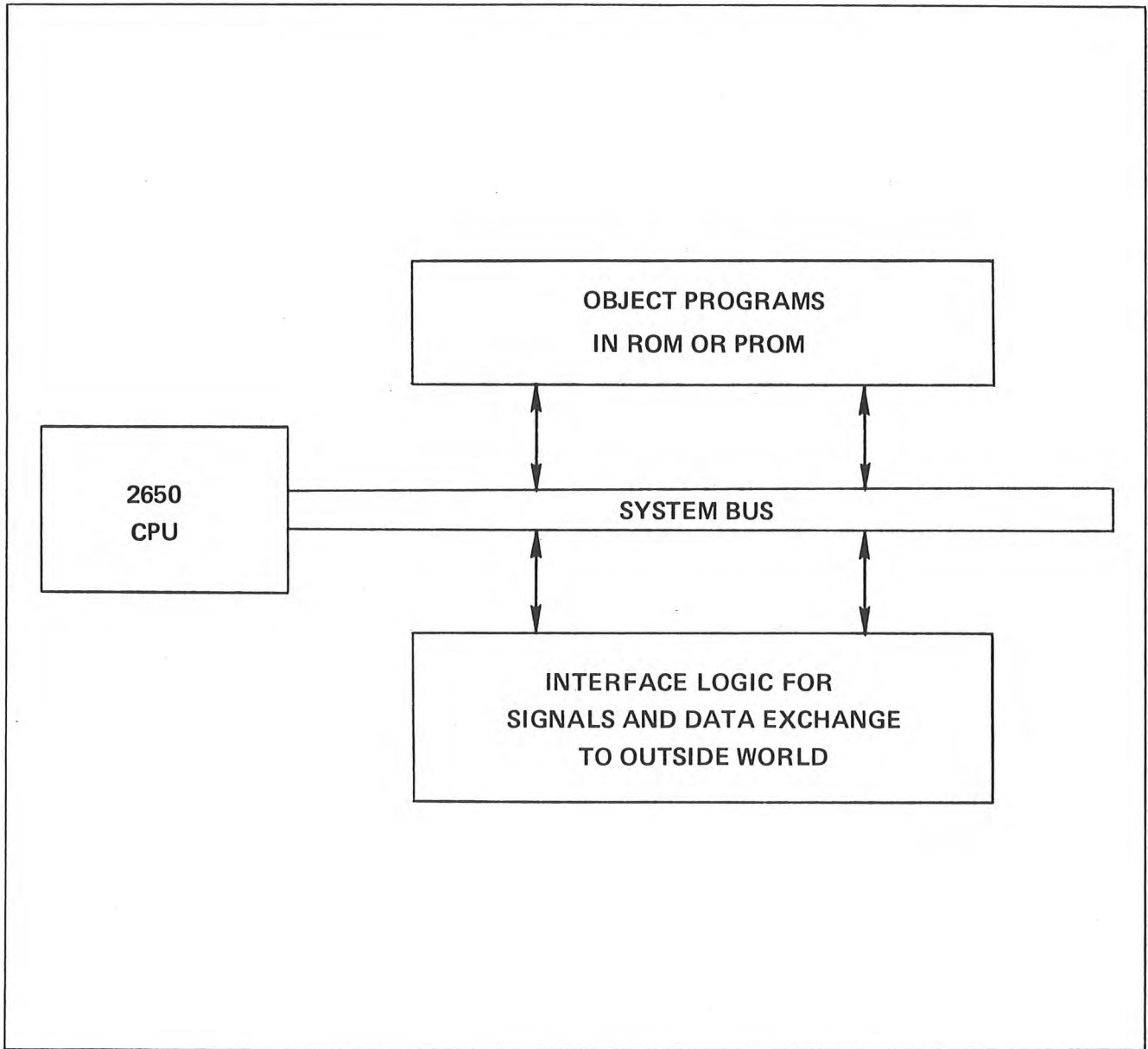


Figure 1-1. Elementary Partitioning of 2650 Microcomputer System Logic

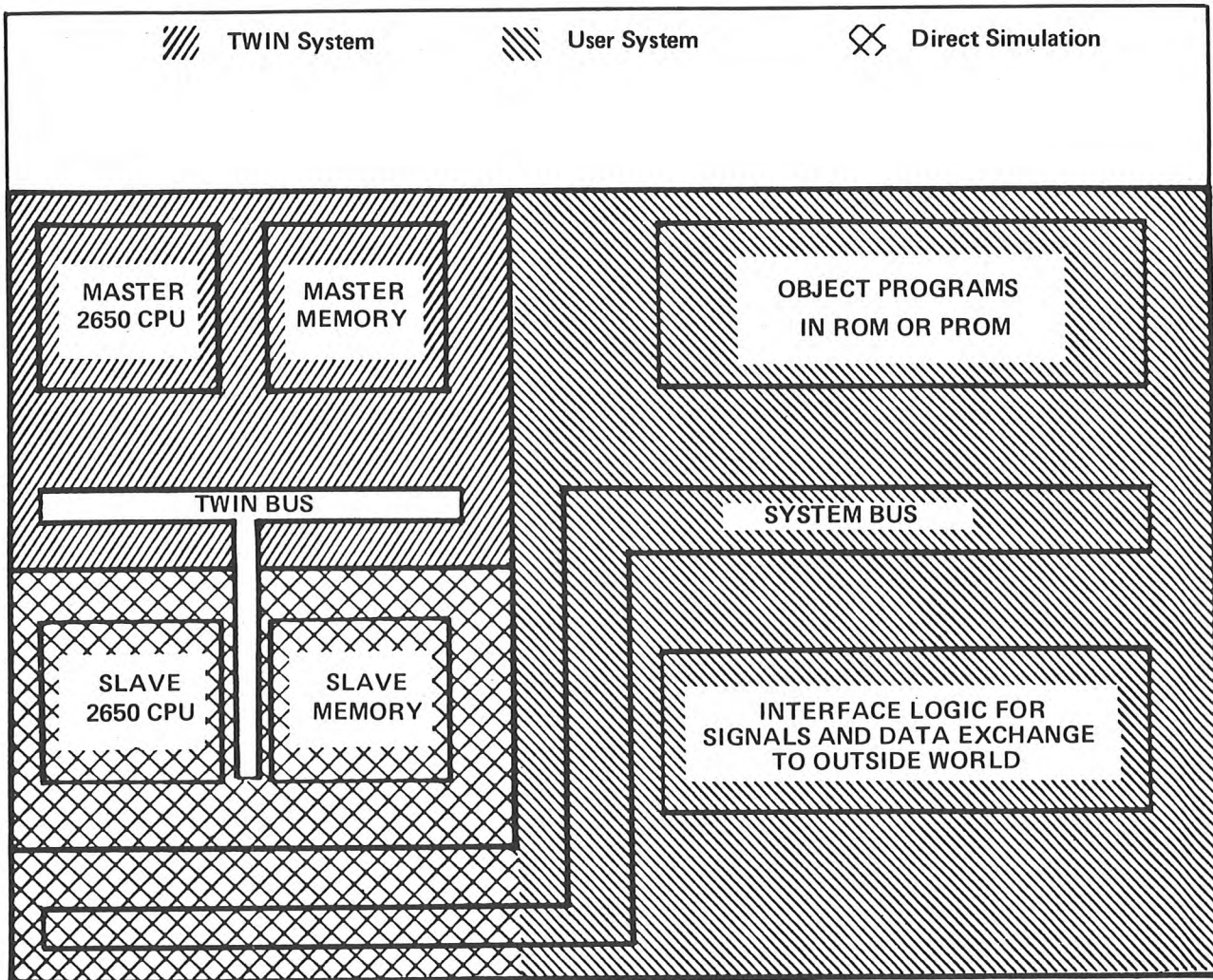


Figure 1-2. TWIN Slave CPU Emulates User's System CPU



TWIN's simulation of I/O logic remains to be described. The problem with this additional logic is that it is completely undefinable. Not only is it impossible to say how far such logic migrates into an end product, it is equally hard to determine, in advance, those functions which will end up as program steps in PROM or ROM as opposed to digital logic packages. TWIN resolves the open-endedness of this additional logic by providing the TWICE cable; any external logic may communicate with the slave microprocessor and its slave memory via the TWICE cable. Moreover, external logic beyond the TWICE cable may, itself, contain program memory. Referring to Figure 1-2, the logic shaded "user system" communicates with the TWIN system via the TWICE cable.

Thus, TWIN becomes a total microprocessor-based product development system. Every aspect of a 2650-based product may be simulated and designed using TWIN. By the time product development is complete, the TWIN user may be certain that no surprises remain. PROMs or ROMs contain object programs which, while being created, were executed by a microprocessor which is identical to the end product microprocessor. While object programs were executed by TWIN, during their creation, they interacted via the TWICE cable with additional logic which, package-for-package, will be identical to the eventual end product. Therefore, when going from TWIN emulation to end product, the only changes will be in physical fabrication.

## **1.2 ABOUT THIS BOOK**

This book is a TWIN Operator's Guide. As such, it describes all aspects of TWIN system operation, from unpacking, through switches and indicators, to the use of the various system development programs.

Additionally, there is a TWIN System Reference Manual, document number TW09004000, which provides a detailed hardware description of the TWIN system and its various components.

A Maintenance Manual, document number TW09006000, helps the user locate and fix malfunctions in the TWIN system, and provides detailed logic diagrams.

The 2650 TWIN Assembly Language Manual, document number TW09005000, describes the 2650 assembly language and the way it should be used to create assembly language source programs.

The Signetics 2650 Microprocessor Manual, document number 2650BM1000, describes the hardware and interfacing aspects of the 2650 and provides detailed explanations of its instruction set.

## **1.3 MANUAL CONTENT**

Chapter 2 of this manual describes system hardware in general terms, and gives an overview of system software. Chapter 3 describes unpacking, installation and initial operation. Chapter 4 gives details of the Signetics Disk Operating System and describes procedures for using it. Chapter 5 describes the Text Editor and gives procedures for using the Editor to create and modify files. Chapter 6 describes the Absolute Assembler and how it is used to

create object programs from assembly language programs. Chapter 7 gives procedures for programming PROMs from assembled user programs. Chapter 8 describes the capabilities of the TWIN debug system.

## CHAPTER 2

### SYSTEM DESCRIPTION

#### 2.0 INTRODUCTION

This chapter outlines system configuration, peripherals, and software provided with the system.

#### 2.1 HARDWARE

The TWIN is a complete microprocessor development system based on the Signetics 2650 microprocessor. This system is used to create and edit assembly language source programs. User's object programs may be executed out of TWIN memory, or object programs may be executed out of external memory that is part of an end product by using the TWICE interconnecting cable assembly. Thus TWIN can simulate an end product, or interface directly to it; therefore, TWIN has the ability to support every phase of product development. A TWIN system consists of a development computer with 16K bytes of master memory, from 16K to 32K bytes of slave memory with a 2650 slave CPU, and a dual drive floppy disk unit. Peripherals include a CRT terminal and a line printer. Options available include an additional floppy disk unit, PROM programmers, and general purpose I/O cards. The computer, disk unit, terminal, and line printer are all desk-top units and are self-contained.

##### 2.1.1 TWIN Development Computer

The development computer consists of a mainframe enclosure and printed circuit board subsystems to implement development functions. The following describes major functions of the development computer hardware.

##### MASTER AND SLAVE CPU

The TWIN operating system runs in a master CPU which is the Signetics 2650. The Editor, Assemblers and user programs run in the slave CPU.

At any point in time, only one CPU within the TWIN system can be active and executing instructions. The master CPU is responsible for determining which CPU is active. The master CPU determines the slave CPU state via a series of control lines which are master CPU interrupts.



## PARTITIONED I/O

The master CPU handles all I/O communication with system peripherals. Programs executed by the slave CPU communicate with system peripherals via the master CPU by issuing requests to the master CPU for their system I/O. This is done through supervisor calls, SVCs, from the slave to the master. SVCs are discussed in Appendix F.

There is separate interface logic available only to the slave CPU. Using this logic, the user can add interface boards for development-oriented peripherals, allowing the slave CPU to communicate with its own peripheral units directly. Thus, programs under development can be executed in a hardware environment nearly identical to that of the user's final product.

## DUAL MEMORIES

The system includes two separate memories: one is the slave memory of up to 64K bytes.\* This memory is accessible by both master and slave CPUs. Three system programs, the Editor and the Assemblers, plus a small debug utility program package, are executed out of the slave memory by the slave CPU. User development programs are also run under the slave CPU in this memory.

The other memory is the master memory in which the operating system and the debug monitor run under the master CPU. This memory is protected completely from the slave CPU and its application programs. The protected portion has an address range from 0000 through 16383. The master CPU also has the ability to map any 16K section of the slave memory into an additional address space available only to the master. This allows the master CPU access to user buffers and pointers and is needed by the Debug Trace program.

Having separate master and slave memories insures that the operating system need not interfere with user programs. This also protects the integrity of the operating system; the operating system in the master memory cannot be inadvertently effected by development programs.

## PROM PROGRAMMING

The development computer contains two optional PROM programming boards and three front-panel PROM sockets. The two programming boards are used for the 82S115 bipolar PROM and the 1702A MOS PROM. Programming of the PROMs is accomplished under program control, with a completely assembled and debugged program. A front-panel switch turns off PROM programmer power so that devices cannot be damaged during insertion and removal.

A universal PROM programming software interface is provided for the DATA I/O PROM Programmers Models 7 and 9, for either Basic I/O (055-0000) or Remote Control (055-0092).

---

\* Although the 2650 slave can address only 32K of memory, a 64K memory capability is provided to allow future use of the TWIN with other slave CPUs.

## DEBUG HARDWARE

The Debug circuitry is an interrupt-driven interface between the master CPU and the active slave CPU. The master CPU can force an interrupt, a reset, or a branch. The slave can also be run in single-step mode. There are two hardware comparator registers available for address breakpoints. The debug interrupt logic is used to handle all I/O service requests from the slave CPU.

## TWICE HARDWARE

The TWICE hardware consists of a cable and driver/receiver circuits that allow in-circuit emulation of user programs in user developed hardware. The user's 2650 microprocessor is removed and replaced by a cable plugged directly into the 2650 socket. The other end of the cable is attached to the TWIN slave CPU circuit board, which contains the multiplexing and other logic to support the TWICE modes. The slave CPU thus becomes the CPU for the user system.

There are three modes of operation:

- 1) The slave CPU runs the program residing in slave memory using the I/O circuits contained in the TWIN system. This is the normal non-TWICE mode.
- 2) The slave CPU runs the program resident in slave memory, but all I/O signals and data are derived from external user developed hardware.
- 3) The slave CPU runs user programs resident in external user development memory. All I/O signals and data are derived from the user developed hardware.

### 2.1.2 Dual Floppy Disk Subsystem

The floppy disk subsystem is the mass-storage medium for the system. The subsystem consists of two disk drives, a microprogrammed controller, power supplies, and cabinet. The disk subsystem communicates directly with the development computer through an interconnecting cable.

## CONTROLLER

The floppy disk controller utilizes a 128-byte sector buffer to allow asynchronous data transfer. Other important features include sector interleaving, automatic data blocking, automatic system boot on power-up, automatic retry on read or write failures, and the ability to expand to a four drive system.

## DISKETTE

The organization of data on a diskette is pictured in Figures 2-1a and 2-1b. On each diskette, there are 77 concentric circles (Figure 2-1a), which can contain data. Each circle is referred to as a track. In Figure 2-1b, a track is divided into its component parts. Each quarter track is referred to as a block. Each block is split into eight sectors. A sector is the basic unit of disk data. Each sector can contain 128 eight-bit bytes. Due to directory limitations, a maximum of 78 files can be contained on one diskette. The disk operating system reserves track 0 for the disk directory, and tracks one through four are normally automatically reserved for the resident portion of SDOS.

In order for the disk drive to be able to read or write a diskette, the diskette must have certain information on it. The process of placing this information on the diskette is called formatting. If diskettes are purchased from Signetics, they are pre-formatted. If diskettes are not purchased from Signetics, they MUST be formatted before use. (Section 4.7.1).

## 2.2 PERIPHERALS

Peripherals compatible with the system include a CRT terminal with a full ASCII keyboard, a line printer, and ASR-33 Teletypewriter, and a paper tape reader. In addition, the GPIO card supports any RS-232-C compatible device and contains four 8-bit parallel I/O ports which allow the user to interface TTL compatible peripherals to the TWIN.

### 2.2.1 CRT Terminal

The CRT terminal is the primary I/O device for the operator. The terminal consists of a CRT display and an operator keyboard. The keyboard is a standard typewriter-style unit with additional mode keys. The CRT and keyboard can be separated for operator convenience.

### 2.2.2 ASR-33 Teletypewriter

A standard ASR-33 with a 20 mA current loop or RS-232-C interface can be used as an alternate console I/O device. In addition, the TTY can be used to provide hard copy and to punch paper tapes for file storage off line.

### 2.2.3 Line Printer

A Centronics 306C line printer is available for hard copy output. The standard line printer is connected through a cable to the floppy disk subsystem, and is capable of printing 100 characters per second with an 80 character column width, or 165 characters per second with a 132 character column width.

### 2.2.4 User-Supplied Peripherals

Any RS-232-C compatible peripheral can be connected to the serial I/O port of the GPIO card, or any 8-bit parallel device to one of the four parallel ports on the GPIO card. If these peripherals are to interface to the operating

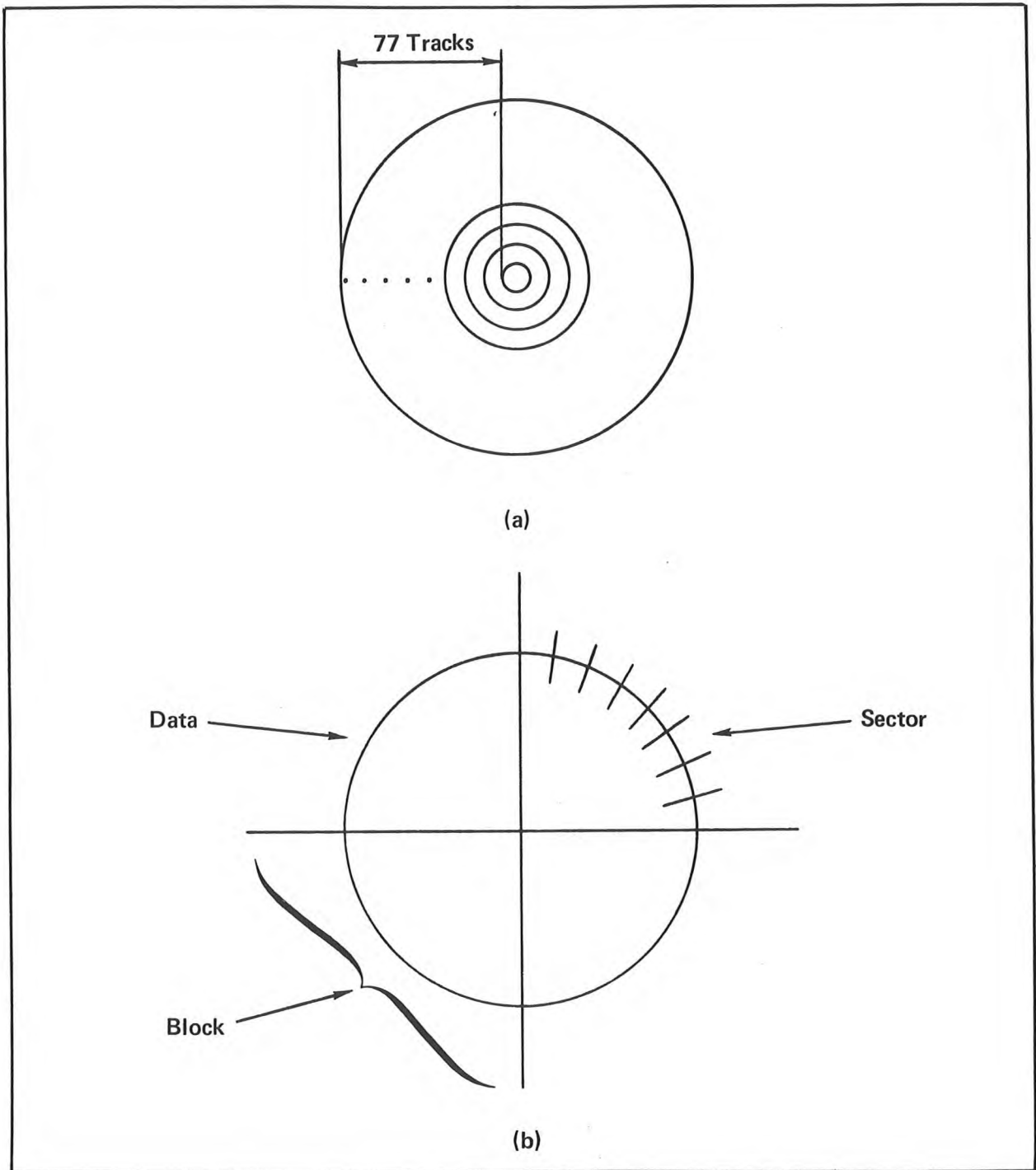


Figure 2-1. Diskette

system, the user may use the general purpose RS-232-C driver supplied with the system, or he may add his own software driver. This driver is added to the TWIN software using the method described in Appendix H.

## **2.3 SOFTWARE**

The TWIN development system software consists of the Signetics Disk Operating System, SDOS, and its associated commands. SDOS commands also invoke the Editor, the Assembler, and the System Readiness Test.

### **2.3.1 SDOS**

SDOS provides the user with a variety of commands that will allow the user to exercise the flexibility of the TWIN system. SDOS provides commands that:

- . Perform disk and file maintenance
- . Set the mode for I/O channels
- . Perform system utility functions
- . Allow the user to control execution of programs
- . Display important system status
- . Manipulate and modify object code

These commands, as well as SDOS, are described in Chapter 4.

### **2.3.2 The Debug Monitor**

The Assembler can only detect syntax errors in a source program. There usually remain a number of logic errors in an object program which cannot be detected by the Assembler. An object program is therefore executed in conjunction with the Debugger in order to detect logic errors. The Debugger is able to control the execution of object programs while examining, changing or tracing the contents of memory, registers or system status.

The Debug monitor, as part of SDOS, executes in master memory. All Debug I/O functions are performed by SDOS. Due to the fact that the master CPU may not access the slave CPU registers directly, a small debug utility program package resides in slave memory to make slave CPU registers available to SDOS for examination and modification.

### **2.3.3 PROM Programming**

SDOS provides a series of commands that allow PROMs to be read, written and compared with slave memory. Most of these commands apply to the PROM sockets located in the front panel. A command is provided for operating the universal PROM programmer interface software.



### **2.3.4 The Editor**

After a source program is conceived and designed, it can be input to the TWIN system with a program called the Editor, which will store a key-entered source program on the floppy disk. The Editor is also used to modify source programs that already exist on mass storage.

The Editor runs in slave memory using the slave CPU. All remaining available slave memory is used for the Editor's text buffer, which is the location of the data operated on by the Editor. SDOS performs all the Editor's I/O requests.

### **2.3.5 The Assemblers**

After a source program has been entered and stored on disk, it must be translated into a machine-executable object program. This function is performed by an assembler, which stores the object code it has assembled from the source program on mass storage.

The Assemblers run in slave memory using the slave CPU. The Assemblers use the available part of slave memory for I/O buffers and to create its symbol tables. SDOS handles all the Assemblers' I/O requests. Both an absolute and relocatable assembler are provided. The relocatable assembler is described in a separate manual titled TWIN 2650 Relocatable Assembler Manual, publication number TW09007000.

### **2.3.6 Systems Readiness Test**

The Systems Readiness Test allows the user to insure that the TWIN system is operational. This test is described in Appendix E.



## CHAPTER 3

### SYSTEM OPERATION

#### 3.0 INTRODUCTION

This chapter describes unpacking, installation, interconnection, and initial operation of the system. Refer to the individual peripheral manuals provided for specific installation procedures for these units.

#### 3.1 UNPACKING

The system is shipped with each major unit in a separate carton. Before unpacking the units, inspect each carton for signs of external damage. If any damage is detected, make a note on the shipper's receipt.

##### 3.1.1 Unpacking the TWIN Development Computer

To unpack the TWIN development computer, open the carton and remove the unit from its packing supports. Place the computer on a bench top and remove the top cover. Remove the packing material from the printed circuit boards and install them in the proper card slots. The correct position for each board is shown in Figure 3-1. The boards are keyed to prevent them from being installed backwards. Push each board firmly into its mother board socket. Untape and remove the power-on switch keys from the chassis and place in the key switch.

Connect the ribbon cable from the front panel to P3 on the Debug card, the ribbon cable from J108 on the rear panel to P2 on the Master CPU card, the ribbon cable from the left-most PROM socket on the front panel to P2 on the 1702A Programmer card (if included in the system), and the ribbon cable from the center socket on the front panel to P2 on the 82S115 Programmer card (if included in the system). Note that the red wire on each cable indicates the end of the cable to be connected to pin 1 of its mating connector. A top view of the computer unit with cards and cables properly installed is shown in Figure 3-2. Do not replace the top cover at this time.

##### 3.1.2 Unpacking the CRT Terminal

Open the carton and remove the packing material from the top of the unit. Lift the terminal and the keyboard out of the carton and set it on a bench top.

No further installation is required until the system is ready for interconnection and operation.

J1	1702A PROM PROGRAMMER *
J2	82S115 PROM PROGRAMMER *
J3	GENERAL PURPOSE I/O *
J4	4K RAM/2K PROM - MASTER
J5	4K RAM - MASTER
J6	4K RAM - MASTER
J7	4K RAM - MASTER
J8	MASTER CPU
J9	DEBUG AND FRONT PANEL I/O
J10	SPARE
J11	4K RAM - SLAVE
J12	4K RAM - SLAVE
J13	4K RAM - SLAVE
J14	4K RAM - SLAVE
J15	SPARE
J16	SPARE
J17	SPARE
J18	SPARE
J19	SPARE
J20	2650 SLAVE CPU

\* Optional

**Figure 3-1. Development Computer PC Board Layout**

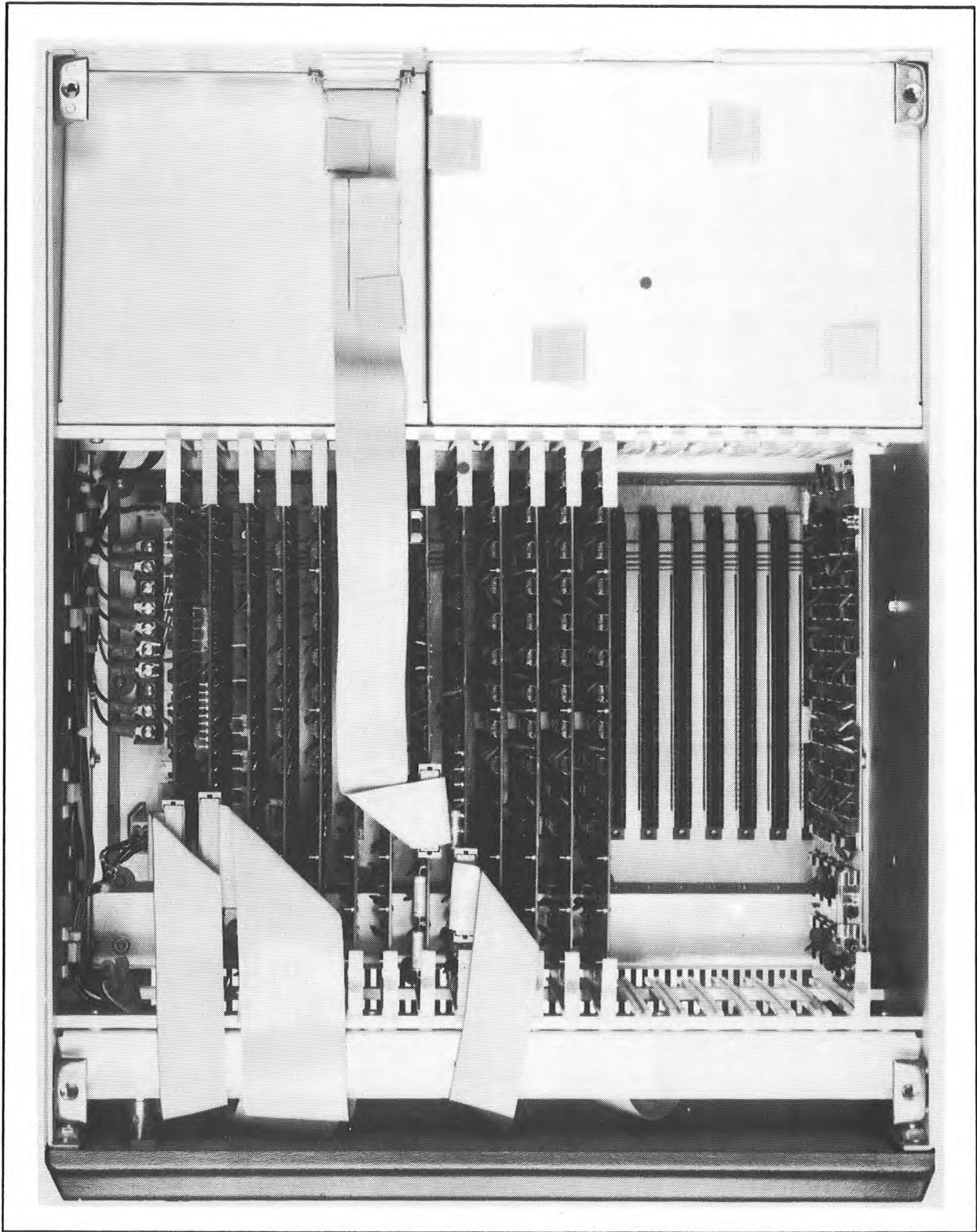


Figure 3-2. Development Computer (Top View)



### 3.1.3 Unpacking the Floppy Disk Unit

To unpack the floppy disk unit, open the carton and remove the packing supports. Lift the unit out of the carton and place on the bench top. Remove the top cover and remove the packing material from around the controller printed circuit board. Make sure the board is secured in its card guides. Unwind the floppy disk and printer interconnect cables and feed them through the channel provided for them in the rear panel. Insure that the ribbon cables are firmly installed in their sockets. Replace the top cover and open the two diskette loading doors.

### 3.1.4 Unpacking the Line Printer

To unpack the line printer you must have the following tools available: 1) 17 mm and 19 mm socket wrenches, or 2) an adjustable wrench. Remove the tape or straps holding the outer cardboard carton to the wooden pallet. Lift the carton off the pallet. Remove the plastic covering the printer. To complete the unpacking, refer to the detailed instructions packed with the printer. These instructions also provide the necessary information on paper installation procedures.

### 3.1.5 Installing the TWICE Debug Cable

It is recommended that the TWICE debug cable be set aside until required for prototype system checkout. At such time, install the cable as follows. Remove the top cover from the computer unit. Unwind the cables from the TWICE interface assembly. Feed the ribbon cables marked P2 and P3 through an access slot in the rear panel and connect them to their corresponding connectors P2 and P3 on the slave CPU card. Replace the top cover. Turn off the power on the user prototype system. Connect the 40 pin TWICE connector to the 2650 socket on the user prototype system, making sure that pin 1 aligns correctly. The TWIN system is now ready for TWICE operation.

## 3.2 INTERCONNECTION AND PHYSICAL INSTALLATION

The units should be placed on a convenient flat surface, close enough to each other for the interconnecting cables to reach. Since the CRT terminal and the TWIN development computer draw cooling air through openings in the bottom of their cabinets, these units should be located where it is unlikely that paper, plastic, carpeting or other materials will be drawn into the air intake and cause overheating. The other units draw cooling air from openings in the rear panel.

### 3.2.1 Power Requirements

Each system unit has a separate power cord and requires a separate outlet for primary power. Current requirements are as follows:

Development Computer:	3.5 amperes at 115 VAC, 60 Hz
	1.8 amperes at 230 VAC, 50 Hz
Dual Floppy Disk Unit:	4.0 amperes at 115 VAC, 60 Hz
	2.0 amperes at 230 VAC, 50 Hz

Line Printer:	3.0 amperes at 115 VAC, 60 Hz
	1.6 amperes at 230 VAC, 50 Hz
CRT Terminal:	2.0 amperes at 115 VAC, 60 Hz
	1.1 amperes at 230 VAC, 50 Hz

### 3.2.2 Interconnection

Before connecting any units to the primary power source, turn all power switches to the OFF position. Rotate the development computer key switch fully counterclockwise. Insure that all units are wired for the primary input voltage used.

Make the system interconnections as follows:

1. Connect the dual floppy disk unit to the development computer by routing the 50 lead ribbon cable (90014021) from the rear of the disk unit through the center cableway on the rear of the computer to P3 of the Master CPU card. Insure that pin 1 of the cable (red stripe) is mated to pin 1 of P3. Replace the top cover on the computer unit.
2. If a line printer is used, connect the ribbon cable (90014172) from the rear of the floppy disk unit to the connector on the rear panel of the printer. Lock the cable in place.
3. Connect the CRT terminal to the development computer by installing the cable (90014191) between J108 on the computer rear panel and the I/O connector on the rear panel of the terminal. The ends of the cable are identical.
4. If multiple disk units are included in the system, refer to the special instructions packed with the system for installation of the additional units.
5. Connect all power cords to the line power source.

## 3.3 SYSTEM CONTROLS AND INDICATORS

The operator controls and indicators for the system units, including peripherals, are described below.

### 3.3.1 Development Computer

Referring to Figure 3-3, the following controls are located on the computer front panel:

1. The key-operated switch controls primary power to the unit. When the key is rotated fully clockwise, power is applied; when the key is rotated fully counterclockwise, power is off and the key may be removed.

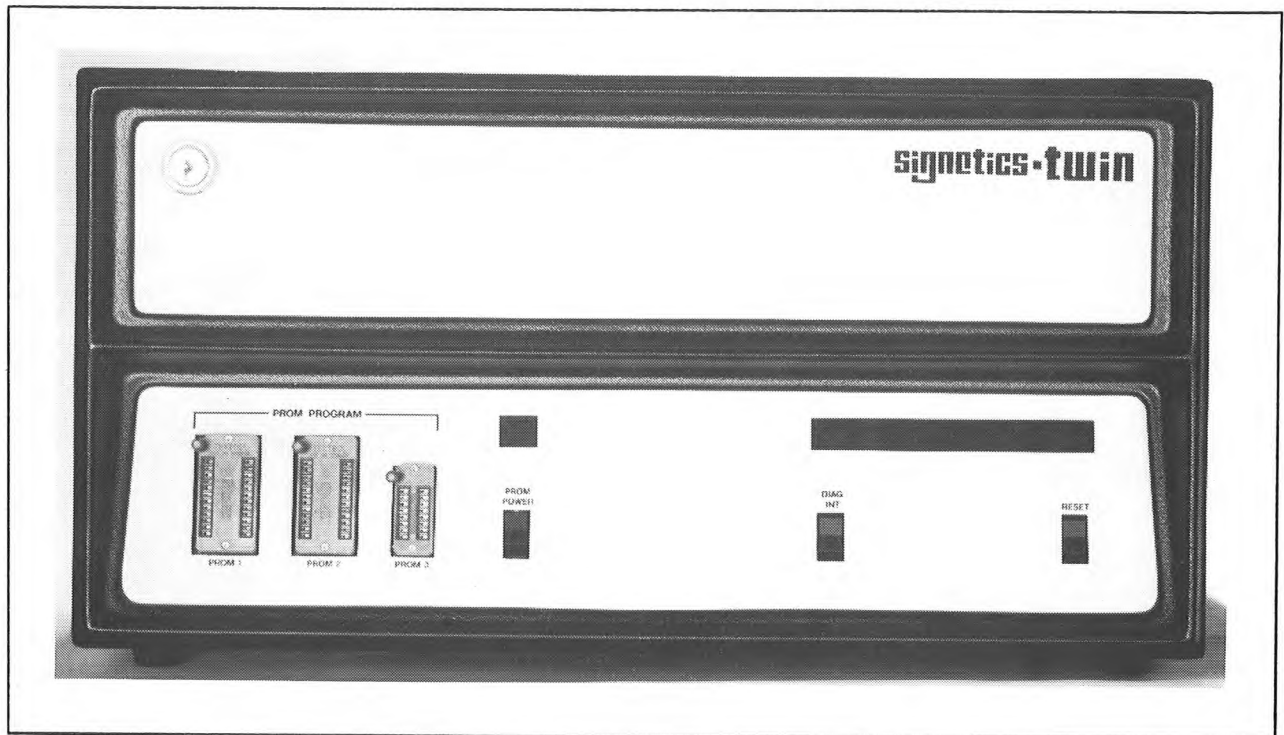


Figure 3-3. Computer Front Panel

2. The back-lighted display has the following legends:
  - PWR lights when primary power is applied.
  - MSTR lights when the master CPU has control.
  - SLV lights when the slave CPU has control.
  - RUN lights when the system is running.
3. The DIAG INT switch initiates a reload of SDOS when the system is in the RUN state. Control is returned to the master CPU. This switch is used with the maintenance diagnostic software.
4. The RESET switch terminates any program in progress. The hardware is initialized, and the operating system is reloaded from the system diskette.
5. The PROM PWR switch enables or disables PROM programming power at the front-panel PROM sockets. When enabled, the PPWR indicator above the switch is lighted. PROM PWR should be off whenever devices are inserted or removed from the PROM sockets.
6. PROM programming sockets:

The leftmost socket (PROM1) is used for programming type 1702A MOS PROMs.

The center (PROM2) is used for programming type 82S115 bipolar PROMs.

The rightmost socket (PROM3) is reserved for future use.

All three sockets are zero insertion force sockets.

Referring to Figure 3-4, the following items are located on the rear panel.

1. AC IN is the connector for primary power, using the power cable supplied with the unit.
2. The 115/230 slide switch selects the internal voltage taps for 115V or 230V operation. Insure that F3 and F4 contain the proper fuses for the selected voltage.
3. The barrier terminal strip allows connection of an external supply to a separate motherboard bus line and allows the user the choice of chassis grounded or floating logic. To connect signal ground to chassis ground, connect the terminals so marked together.
4. Fuses protect the internal power supplies. F4 is the fuse for primary power input. F3 independently fuses the +12V power supply. F1 and F2 fuse the PROM programmer AC secondary voltage.



Figure 3-4. Computer Rear Panel



5. J108 is a female connector used to connect the CRT terminal or the teletype to the computer.

### 3.3.2 Dual Floppy Disk Unit

The floppy disk unit has a single front-panel power on/off push-button switch that is lighted when primary power is on.

The disk unit rear panel contains a connector for disk expansion, a fuse for primary power, and a connector for the power cable.

### 3.3.3 CRT Terminal

The terminal consists of a CRT unit and keyboard. The keyboard layout closely approximates an ASR-33 Teletype. Refer to the CRT terminal operator's manual for details of terminal operation.

### 3.3.4 Printer

The optional printer is a Centronics 306C. Refer to the Centronics 306C operator's manual for details of printer operation.

## 3.4 OPERATION

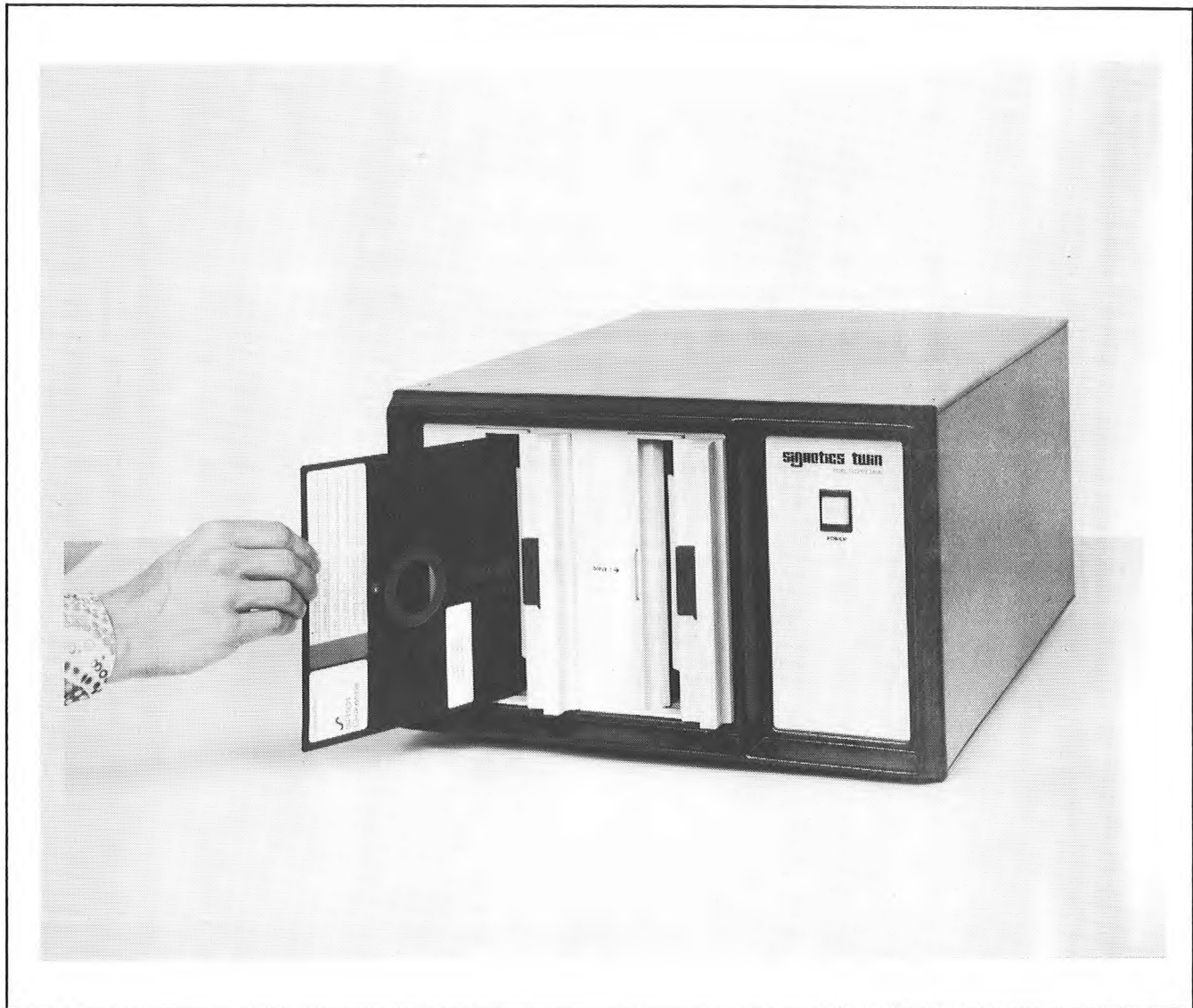
To power up the TWIN system and load the operating system, SDOS, into memory, perform the following steps.

- 1) Power up the CRT terminal. After a brief warm-up period, the cursor will appear on the screen. Adjust the intensity to the desired level.
- 2) Power up the floppy disk unit.

### CAUTION

DO NOT TURN POWER ON OR OFF THE DISK UNIT  
WITH DISKETTES INSTALLED AND DOORS CLOSED AS  
MEDIA DATA MIGHT BE DESTROYED.

- 3) Allow a five minute warm-up time to allow the disk drive electronics to reach stable temperature.
- 4) Insert the system diskette into drive 0. The correct method for inserting a diskette is shown in Figure 3-5. Insure that the label area is toward the power switch on the disk unit and is the last part of the diskette inserted into the drive. Close disk drive door only after power has been applied to the TWIN development computer.
- 5) Apply power to the printer.
- 6) Apply power to the TWIN development computer. This will cause an automatic read from drive 0 which will load SDOS into master memory.



**Figure 3-5. Inserting a Diskette**

When SDOS has been loaded, this welcoming message will be displayed on the terminal:

```
>SDOS VER X.Y 2650  
>
```

The > is the SDOS prompt character and it informs you that SDOS is ready to accept commands.

If the welcoming message does not appear with 15 seconds, depress the RESET switch. If the system does not respond correctly again, an improper diskette or a faulty drive may be the problem. Try again with a new system diskette and/or using drive 1. If trouble persists, request service assistance.

#### NOTE

The TWIN computer will automatically switch the initialization process to drive 1 if attempts to load from drive 0 fail repeatedly.

If the welcoming message is incorrect, the baud rate-setting of the CRT may not correspond to the rate selected on the Master CPU card. Select the correct baud rate on the CRT terminal rear panel. Depending on the version level of the master CPU either a slide switch or a thumbwheel switch is present to select the baud rate. For the thumbwheel switch see fig. 3.6 for baud selection. If slide switch is present then there are two baud rates directly available 110 and the rate selected by the jumpers on the master CPU. Refer to the TWIN System Reference Manual for information on changing the baud rate on the Master CPU card. page 3-19.

#### NOTE

The 110 baud indication on the Master CPU (slide switch version) is incorrect. 110 baud is available when switch is not in 110 baud position.

#### 3.4.1. Manual Reset

If a reinitialization of the system is desired during operation, the user may reload SDOS by pressing the RESET switch on the front panel. The welcoming message and the prompt character will be issued after SDOS has been loaded.

Thumbwheel POSITION	Baud RATE	Format
9	110	Binary (2 Stop Bits)
8	110	ASCII + Parity (2 Stop Bits)
7	150	ASCII + Parity (1 Stop Bit)
6	300	ASCII + Parity (1 Stop Bit)
5	600	ASCII + Parity (1 Stop Bit)
4	1200	ASCII + Parity (1 Stop Bit)
3	2400	ASCII + Parity (1 Stop Bit)
2	Illegal	
1	Illegal	
0	Illegal	

It is possible to select Binary for Thumbwheel positions 3 - 8 by removing jumper J1 (situated top left when viewed from component side) on master CPU

Fig. 3-6

BAUD SELECTION THUMBWHEEL SWITCH

## CHAPTER 4

### SIGNETICS DISK OPERATING SYSTEM

#### 4.0 INTRODUCTION

This section describes the Signetics Disk Operating System, SDOS, for the TWIN system. General topics include the use of the keyboard to enter commands or request control of the system, an overview of the SDOS file structure, a catalog of the SDOS commands and their functions, and a study of the command file capability and overlay areas. In addition, summaries of the SDOS commands and SDOS error messages are provided.

#### 4.1 SYSTEM DESCRIPTION

The Signetics Disk Operating System executes in system master memory. However, due to the size of SDOS, only a portion of the system may be memory resident at a time. This resident portion of the operating system is loaded into master memory whenever power is turned on to the system or the RESET switch is toggled. Remaining portions are overlayed into master memory as needed to execute system commands.

##### 4.1.1 Resident SDOS

The portion of SDOS which is resident in master memory at all times includes:

- Command Line Processor
- Job Dispatcher
- Supervisor Call (SVC) Processor
- Device Drivers
- File Manager
- Four Resident System Commands

Each of these modules is describe in detail below.

**Command Line Processor:** The command line processor operates on commands entered from the system console or from a command file stored on diskette. The command line processor interprets the commands and prepares a parameter list. Then the function is performed by transferring control to the appropriate resident procedure or by loading and executing a system command.

**Job Dispatcher:** The job dispatcher controls execution of the active jobs in the system. The job dispatcher transfers control to the highest priority job whose I/O operation has been completed or which is ready to run.



Supervisor Call Processor: The supervisor call processor operates on internal requests for input and output (I/O) or an SDOS function. All of the I/O communication between the slave CPU and system peripherals is performed by the supervisor call processor.

File Manager and Device Drivers: The flexible disk drive file manager and other device drivers control operation of the peripheral devices in the system.

Resident Commands: GO, LOAD, XEQ, and SYSTEM are resident to facilitate system initial program load and debugging.

#### 4.1.2 SDOS Overlays

The SDOS overlays consist of all SDOS commands except the four memory-resident commands and the Editor and Assemblers which run as slave jobs.

Master memory contains two overlay areas into which the SDOS command overlays are loaded and executed. The overlay areas are referred to as Overlay Area 1 and Overlay Area 2. Some SDOS commands are executed in Overlay Area 1, some are executed in Overlay area 2, and some occupy both overlay areas during execution.

The SDOS commands are categorized in the following list by the overlay area in which they are executed:

<u>OVERLAY AREA 1</u>	<u>OVERLAY AREA 2</u>	<u>OVERLAY AREA 1 &amp; 2</u>
COPY	ABORT	CMPF
RHEX	ICE	DFIL
CPROM	ASSIGN	KILL
RPPROM	BKPT	MOVE
DEBUG	CLBP	PATCH
RSMS	CLBP	RENAME
DUP	CLOSE	RESET
VERIFY	CONT	SET
FORMAT	DELETE	STATUS
WHEX	DEVICE	SUSPEND
WSMS	DSTAT	TRACE
WPROM	DUMP	TYPE
	EXAM	UPR
	FILL	*

SDOS commands can be executed concurrently as long as they do not occupy the same overlay area. In addition, the concurrent execution must be consistent with the current state of the peripheral devices and must not cause any system conflicts.

To carry out concurrent command execution do the following steps.

- 1) Start execution of the Overlay Area 1 command
- 2) Depress the ESC key
- 3) Type the Overlay Area 2 command and depress the carriage return key.

As an example of concurrent command execution, suppose a paper tape was being read into the slave memory. This would be accomplished using the RHEX command.

```
> R TTYR
```

```
ESC
```

```
>> DEL FILE1/1 DATA1/1 SOURCE/1 (r)
```

While the tape is being processed, file maintenance could be performed. Pressing the ESCAPE key suspends RHEX execution and displays the SDOS prompt character, >>. The DELETE command is then entered.

When the (r) is entered, the RHEX command continues execution and the DEL command starts. Note that RHEX executes in Overlay Area 1, while DELETE operates in Overlay Area 2, allowing concurrent execution of these programs.

#### 4.1.3 System Slave Jobs

The system commands EDIT, ASM, MAC, RASM, and LINK run as slave jobs under the slave CPU. The size of the programs and data buffers needed for these jobs prohibits them from running in master memory under the command overlay scheme.

#### 4.1.4 Debug Utility Program

To communicate with the slave CPU, SDOS requires that a minimum package of debug utility programs be slave memory resident. These programs perform such functions as save and restore slave CPU registers for TRACE output, and buffering data being passed between the master and slave. The SDOS commands which utilize the debug utility programs are PATCH, EXAM, MOVE, FILL, READ, WRITE, the Debugger, and system slave jobs; in short, things which write to or read from slave memory.

The debug utility programs are loaded into the top-most 256 bytes of slave memory whenever the system is powered on or the RESET switch is toggled. This block of slave memory is write-protected, rendering it inaccessible to the user. However, these programs are not protected when the LOAD or XEQ commands are executed.

#### NOTE

To utilize the top-most memory locations in a user program, relocate the debug utility programs to another area of slave memory with the system command UPR.

## 4.2 FILE MANAGEMENT AND DISKETTES

Before using SDOS, several concepts regarding diskettes and disk drives require explanation.

- 1) Before using any new diskette not purchased from Signetics, THE NEW DISKETTE MUST BE FORMATTED AND VERIFIED. Follow the procedures outlined in the discussion of the SDOS commands FORMAT and VERIFY.

All diskettes purchased from Signetics are pre-formatted and verified and therefore do not require these operations before use.

2. Every diskette has a write protect slot (see Figure 3-5).
  - A) If the slot is covered, the diskette is write enabled. The diskette can be read from or written to by the operating system.
  - B) If the slot is not covered, the diskette is write protected. The diskette can only be read from. Any attempt to write to a write-protected diskette will cause an SDOS error to appear on the system console.
- 3) The typical TWIN system has two disk drives, although there may be up to four.

Drive 0 is normally the system drive. The diskette loaded on the system drive is known as the system diskette and must contain the system programs. The system drive is the drive which SDOS accesses when it must load a system command. The system drive is also the drive used when a drive number is not specified with a file name in an SDOS command. The system diskette can be write protected to ensure that the system programs are not altered.

Drive 1 usually contains a user diskette which contains user files. This diskette is used for modifying user files or as a scratch data area, and may or may not contain the system programs. This diskette is not write protected, since it may be used as a scratch area.

#### **4.2.1 Files, Devices, and Channels**

SDOS is a file-oriented system. The understanding of a file-oriented system is greatly enhanced by understanding the concepts of a file, a device and a channel.

##### Files

A file is a discrete set of data. The set has a logical beginning and a logical end. For example, the government's file on a person's tax return might begin with the first return filed by the person and end with his last return filed. In between the first return and the last return there could be other returns, audits, etc. All the information beginning with the first return and ending with the last return is the file.

In the TWIN system, files are stored on floppy diskettes. A file can be accessed through its logical beginning address (a map that indicates where the data in the file is located on the disk), and a logical ending address.

##### Devices

Devices are physical peripherals that provide input and output services for SDOS. The four standard devices are the console input device, the console output device, the Centronics line printer, and the teletype reader. These devices all have reserved names through which the user must access them. These names appear in Table 4-1A.

TABLE 4-1A. RESERVED DEVICE NAMES

DEVICE NAME	DEVICE
CONI CONO LPT1 TTYR	CONSOLE INPUT CONSOLE OUTPUT LINE PRINTER 1 TELETYPE READER

The system may be optionally configured to include one additional device of the user's choice. A software driver is provided for each of these devices, as well as procedures to configure these drives into the system. These optional devices are the Printronix 300 line printer, the high speed paper tape reader, and the RS232 port on the GPIO card. The names by which these devices are accessed are shown in Table 4-1B.

TABLE 4-1B. OPTIONAL DEVICE NAMES

DEVICE NAME	DEVICE
LPT2 HSPT R232	PRINTRONIX 300 LINE PRINTER HIGH SPEED PAPER TAPE READER RS232 PORT

NOTE

Only one of the optional devices may be configured into SDOS at a time. The device name is then reserved only as long as that device driver is configured into SDOS

SDOS is shipped configured for the optional device R232.

For example, using device names, enter the command:

```
>COPY TTYR LPT1
```

which copies the information from the teletype paper tape reader to the line printer.

NOTE

Although SDOS software supports a high speed paper tape reader, this peripheral is not currently available for the TWIN system.

Files may also be viewed as devices. Files can be specified as either input or output devices. To refer to a file as a device, you must refer to the file name for that file. In addition, if the file is not located on the diskette installed in the system drive it is necessary to specify the drive on which the file is located.

SDOS is only aware of diskettes that are loaded in the available disk drives. For this reason, diskettes are referred to by drive number rather than by diskette name.

As an example, suppose you had diskettes loaded in drives 0 and 1. Drive 0 is the system drive. There is a file named DATA1 on drive 0 and a file named DATA1 on drive 1. If it were necessary to copy DATA1 to the line printer, how would this be accomplished? The action is performed by specifying a drive number to indicate which DATA1 is to be copied. To copy DATA1 on drive 1 to the line printer, the following command would be performed:

```
>COPY DATA1/1 LPT1
```

In this example, the /1 following the file name specifies that the file on drive 1 is to be copied.

### Channels

Channels are used by the program running on the slave CPU. A channel is assigned to a physical device to enable the slave CPU to perform input or output operations to the device through that channel. The device specified in the assignment may be either a physical device or a file. Channels must be assigned to physical devices prior to performing I/O.

#### **4.2.2 File Names**

A file name has the following properties:

- 1) The file name must contain at least one but not more than eight characters.
- 2) The characters in the name must come from the following set:  
The alphabetic characters (A-Z)  
The numeric characters (0-9)  
The special characters ! " # % & ' ( ) \* ; = ?
- 3) The file name may not begin with a numeric character.
- 4) The file name must not be one of the reserved names which identify physical devices: CONO, CONI, LPT1, TTYR or the current optional system device name.
- 5) The file name must be unique to the diskette containing the file.

File names entered as parameters in an SDOS command line may include the file's drive number by appending an /n, where n is the drive number, to the file name.



### 4.2.3 File Directories

Every diskette has a system area, called the directory, where system information is kept concerning all files on that diskette. This information includes the filename, disk sectors used, beginning and ending disk addresses, and load module ID. The directory also includes system information that prevents the allocation of bad disk sectors for file usage.

### 4.3 ENTERING SDOS COMMANDS

The SDOS command line consists of the command name and in most cases, one or more parameters with delimiting characters. Most commands require that parameters be specified. The command is always separated from its parameter by one or more spaces or by a comma. When two or more parameters are present in a command line, the parameters must also be separated by spaces or a comma. The following two command lines are interpreted by SDOS in the same way:

```
>LDIR 0 /  
>LIDR,0, /
```

The command line is entered after the prompt character > is displayed. In the example above, each command line is preceded by the prompt character. LDIR is the command to be executed, the 0 (zero) is the first parameter, and the "/" is the second parameter.

#### 4.3.1 Command Description

##### Syntax

<b>COMMAND</b> { file name } [ <sup>device</sup> file name [/disk drive] ] [ {line number 1} {line number 2} ] ...
--

##### Command Name

A minimum set of characters is required for each command. This minimum set of characters is underlined in the syntactical description.

In addition to the minimum set of characters in the command name, a maximum set (long form) is also given for each command name. Any number of characters in the command name ranging from the short form spelling to the long form spelling may be used as long as the exact spelling is followed.

##### Delimiters

The components in the command line must be separated by delimiters when entered into the computer. A space is used as the main delimiter. The slash "/" is used to delimit a file name and the disk drive number.

The comma may be used as a delimiter in most cases. In the text editor a comma may not be used as a delimiter between a command the parameters. Two

commas are used to specify null or empty fields in a parameter list. Three commas are used to specify two adjacent null fields.

Special delimiters may be required by you in some text editor commands, as for the FIND and SUBSTITUTE commands. The delimiters you specify in these cases must not be any of the characters in the string being sought or replaced. For example, if you are trying to find the string \$15 in the text, you might use the ampersand "&" character as the delimiter in this way:

```
* FIND &$15&
```

### Parameters

The parameters or controlling conditions of each command line are shown in the syntactical description above. These parameters may be names, numbers, characters or symbols. When the parameter is shown capitalized it must be entered exactly as shown. A parameter shown in lower case letters is a descriptive term to signify the type of entry, as shown above.

### Braces and Brackets

When the parameter is enclosed in braces, { }, the parameter must be present in the command line. Parameters enclosed in brackets, [ ], are optional. Brackets and braces may be nested. The following is an example of braces nested in brackets:

```
[{line number 1} {line number 2}]
```

The use of braces and brackets are for syntactical representation and should not be entered as part of the command line.

### Stacked Item

Parameters stacked within either braces or brackets indicate that only one of the enclosed items should be selected. In the example below a peripheral device name may be selected or a file name with a disk drive number, but not both.

```
[device  
file name [/disk drive]]
```

### Trailing Dots

A line of dots following a parameter indicate that the parameter may be repeated a number of times not to exceed the length of the system console display line, normally 80 characters including the carriage return. In the example below the line number parameters can be repeated:

```
[{line number 1} {line number 2}]...
```

## Numeric Values

A parameter calling for a numeric value may be referenced in the explanation by "n". A parameter range may be referenced by "a" and "b". Multiple numeric parameters may be referenced in order by "a", "b", "c", etc.

### 4.3.2 Command Completion

Most SDOS commands indicate that they have completed their function by displaying and End-of-Job message. The form of this message is:

```
*id* E0J
```

where:

id	is the SDOS system program identifier (see Table 4-2)
E0J	is the end of job message

Completion of any user-entered command causes the SDOS prompt character '>' to be displayed.

### 4.3.3 Error Reporting

When an error is encountered during the execution of a command, command processing aborts and the command completion is with an error message rather than with the normal completion message. The form of the error message is:

```
*id* ERR xx
```

where:

id	is the SDOS system program identifier (See Table 4-2)
ERR	is the error condition indicator
xx	is the SDOS error code (see Table 4-3)

Completion of the command results in display of the SDOS prompt character '>'. Error conditions which can occur for each command are listed under the command descriptions on the following pages.

TABLE 4-2. SDOS SYSTEM PROGRAM IDENTIFIERS

*ABT*	ABORT OVERLAY
*ASN*	ASSIGN OVERLAY
*CLS*	CLOSE OVERLAY
*CMP*	CMPF OVERLAY
*CON*	CONT OVERLAY
*COP*	COPY OVERLAY
*DEB*	DEBUG OVERLAY
*DEL*	DELETE OVERLAY
*DEV*	DEVICE OVERLAY
*DFL*	DFIL OVERLAY
*DIR*	LDIR OVERLAY
*DMP*	DUMP OVERLAY
*DOS*	SDOS RESIDENT PROGRAM
*DUP*	DUP OVERLAY
*EXM*	EXAM OVERLAY
*FIL*	FILL OVERLAY
*FMT*	FORMAT OVERLAY
*ICE*	ICE OVERLAY
*KIL*	KILL OVERLAY
*MOD*	MODULE OVERLAY
*MOV*	MOVE OVERLAY
*PAT*	PATCH OVERLAY
*PRM*	PROM PROGRAMMING OVERLAY
*PRN*	PRINT OVERLAY
*RDW*	READ/WRITE OVERLAY
*REN*	RENAME OVERLAY
*RHX*	RHEX OVERLAY
*SLJ*	PROGRAM RUNNING UNDER SLAVE CPU
*SMS*	SMS OVERLAYS
*SUS*	SUSPEND OVERLAY
*TYP*	TYPE OVERLAY
*UPR*	UPR OVERLAY
*VER*	VERIFY OVERLAY
*WHX*	WHEX OVERLAY

TABLE 4-3. SDOS ERROR MESSAGES

ERROR	DESCRIPTION	ERROR	DESCRIPTION
1	DIRECTORY READ ERROR	35	INVALID START ADDRESS
2	DIRECTORY WRITE ERROR	36	INVALID END ADDRESS
3	COMMAND FILE NOT FOUND	37	INVALID GO ADDRESS
4	COMMAND FILE INPUT ERROR	38	UNUSED
5	COMMAND FILE BUSY	39	INVALID HEX CHARACTER
6	DEVICE READ ERROR	40	INVALID RHEX INPUT FORMAT
7	DEVICE WRITE ERROR OR END-OF-DEVICE	41	INVALID BREAKPOINT ACCESS MODE
8	DRIVE NOT SPECIFIED WHEN REQUIRED	42	INVALID REGISTER PARAMETER
9	INVALID DRIVE NUMBER	43	INVALID DATA PARAMETER
10	OVERLAY LOAD FAILURE	44	INVALID TRACE MODE PARAMETER
11	OVERLAY AREA IN USE	45	INVALID SLAVE SRB ADDRESS
12	INVALID FILE NAME	46	SLAVE HALTED
13	INPUT FILE NOT FOUND	47	SYSTEM AREA BAD
14	INVALID INPUT DEVICE	48	LOAD FILE NOT FOUND
15	INVALID OUTPUT DEVICE	49	LOAD FILE ASSIGN FAILURE
16	INPUT DEVICE ASSIGN FAILURE	50	FILE NOT A LOAD MODULE
17	OUTPUT DEVICE ASSIGN FAILURE	51	INVALID LOAD REQUEST
18	DEVICE IN USE	52	INVALID DEVICE
19	INVALID CHANNEL NUMBER	53	UNUSED
20	CHANNEL IN USE	54	INVALID MODE
21	CHANNEL ASSIGN FAILURE	55	INVALID MEMORY
22	COMMAND LINE BUFFER OVERFLOW	56	INVALID DEVICE ADDRESS
23	INVALID COMMAND	57	FILE NAME IN USE
24	JOB NOT ACTIVE	58	DEVICE ASSIGN FAILURE
25	JOB NOT SUSPENDED	59	MEMORY WRITE ERROR
26	JOB ALREADY SUSPENDED	60	END OF MEDIA
27	JOB EXECUTING	61	FILE IN USE
28	JOB UNDER DEBUG CONTROL	62	DEVICE NOT OPERATIONAL
29	PROM POWER FAILURE (front panel)	63	DIRECTORY FULL
30	INVALID PARAMETER	64	INVALID DISKETTE
31	PARAMETER REQUIRED	65	MASTER MEMEORY PARITY ERROR
32	TOO MANY PARAMETERS	66	SLAVE MEMORY PARITY ERROR
33	BIAS PARAMETER ERROR	67	UNUSED
34	INVALID ADDRESS	68	DEBUG UTILITY ROUTINE CON- FLICT OR MEMORY WRAPAROUND
		69	UNUSED



#### 4.4 SPECIAL KEYS

SDOS pays special attention to certain keys in order to facilitate the entry of command lines and operator control of the system or to control system or slave programs. These keys are:

<u>KEY NAME</u>	<u>DESCRIPTION</u>
Space Bar	Suspends console display.
ESC	Suspends or terminates any action.
CTRL-Z	Issues an end-of-file character during an ASCII read operation.
RUB OUT (or DELETE)	Deletes the last character from the line buffer.

These key functions are described on the following pages.

#### 4.4.1 Space Bar



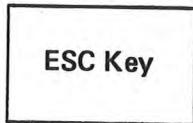
#### PURPOSE

The space bar is used to halt the display output to the console or cause the display to continue.

#### EXPLANATION

Striking the space bar during console display temporarily halts the display. Striking the space bar again causes the display to continue. The display may be halted and continued as many times as needed.

## 4.4.2 ESC Key



### PURPOSE

Striking the ESC key (escape) causes suspension or termination of program execution and returns control to SDOS.

### EXPLANATION

Striking the ESC key twice (ESC ESC) suspends all active programs. A program suspended by this means (ESC ESC) does not resume execution unless a CONT (continue execution) command is entered from the console.

The response to striking the ESC key once varies, depending on whether input is being performed or a command is being executed. Some commands abort when the ESC key is depressed, while some commands are not interruptable at all. See Table 4-4.

#### ESC During Console Input

Striking the ESC key during an input operation produces differing results as follows:

- If an SDOS command line is being entered, the system deletes the command line and responds to the display console with a double prompt character >> .
- If the EXAM command is being executed, the command is terminated and a double prompt character is displayed. Any memory locations that were altered prior to striking the ESC key remain altered.
- If the EDIT command is being executed, the Editor prompt character \* is displayed. When the Editor is in the Input mode, striking the ESC key once deletes the current line being entered and moves the display cursor to the next line. In this case a prompt character is not displayed. Striking the ESC key twice causes the Editor to suspend and return to SDOS. In this case, EDIT can be resumed with the CONT command.
- If a user program is being executed, the response depends on the function that the user has programmed for the ESC key.

#### ESC During Program Execution

Striking the ESC key during the execution of an SDOS command causes the execution of that command to pause. Execution of the command may be continued

either by striking the RETURN key or by entering the CONT command. This allows you to do concurrent command execution as described under SDOS Overlays, Section 4.1.2.

The execution of certain SDOS commands, as shown in Table 4-4, will be terminated by striking the ESC key instead of just being paused.

TABLE 4-4. ESC KEY USAGE WITH SDOS COMMANDS

Commands which abort upon depression of the ESC key:	
CMPF	EXAM
DFIL	LDIR
DUMP	STATUS
	TRACE
Commands which cannot be interrupted with the ESC key:	
All Overlay Area 2 commands	
All other commands can be temporarily interrupted with the ESC key.	

### 4.4.3 Control-Z

**CTRL-Z**

#### PURPOSE

The CTRL-Z (control-Z) provides an end-of-file character during an ASCII read operation.

#### EXPLANATION

A CTRL-Z is sent by holding down the CTRL (control) key while striking the Z key.

The CTRL-Z is treated as an end-of-file character when an ASCII read is being performed from the console or other system input device. It is the logical end-of-file character.

CTRL-Z does not send a visual character to the console.



#### 4.4.4 Rub Out Key

RUB OUT Key

##### PURPOSE

The RUB OUT key is used to delete an incorrectly entered character at the system console.

##### EXPLANATION

Striking the RUB OUT key deletes the last character input from the console keyboard. This function deletes the last character in the line buffer and echoes that character to the display. If more than one character has been entered incorrectly, the RUB OUT key may be used to delete each character in the string. The entry can then be completed as if the incorrect characters were never entered.

##### EXAMPLE

When entering the command line to copy paper tape information from the teletype reader to the console, if you enter the device name by mistake before the file name, the entry may be corrected with the RUB OUT key as follows:

```
> COPY,CONOONOCTTYR,CONO
```

The underlined portion of the command line above shows the incorrect entry and the effect of using the RUB OUT key four times followed by the correct entry.

#### 4.5 SYSTEM CONTROL COMMANDS

You may control the execution of system or slave programs with the following commands:

<u>COMMAND NAME</u>	<u>DESCRIPTION</u>
SUSPEND	Suspends program execution.
CONT	Continue execution of suspended programs.
ABORT	Terminate program execution.

## 4.5.1 SUSPEND

<u>S</u> SUSPEND { program name } * /
---

### PURPOSE

The SUSPEND command suspends the execution of active programs.

### EXPLANATION

The SUSPEND command may be used with any active program except DEBUG.

The primary use for this command is in conjunction with the command file capability. Inserting the SUSPEND command in a command file suspends system operation and allows some required user action, such as inserting a special diskette into one of the drives.

The SUSPEND command must be accompanied by one of the parameters.

- . SUSPEND,\* suspends all active programs.
- . SUSPEND,/ suspends any active slave program.
- . SUSPEND,program name suspends the specified program.

### \*SUS\* Error Responses

24-Job not active  
26-Job already suspended  
31-Parameter required

## 4.5.2 CONT

<u>CONT</u> { * / } (program name)
--

### PURPOSE

The CONT command continues the execution of a suspended program.

### EXPLANATION

A suspended program may be continued by entering the CONT command. Execution of the suspended program resumes from the point where execution was suspended.

When the ESC key has been used to terminate execution of the SDOS commands LDIR, CMPF, DFIL, TRACE, STATUS, EXAM and DUMP, they cannot be continued.

The CONT command must be accompanied by one of the following parameters.

- . CONT,\* causes execution of all active programs to be resumed.
- . CONT,/ causes execution of an active slave program to be resumed.
- . CONT,program name causes execution of the specified program to be continued.

### \*CON\* Error Responses

24-Job not active  
25-Job not suspended  
31-Parameter required

### 4.5.3 ABORT

<u>ABORT</u> { * / program name }
---

#### PURPOSE

The ABORT command terminates execution of an active program.

#### EXPLANATION

The ABORT command causes the execution of an active program to be terminated. The ABORT command must be accompanied by one of the parameters.

- . ABORT,\* causes all active programs to be terminated.
- . ABORT,/ causes an active slave program to be terminated.
- . ABORT,program name causes the named program to be terminated.

#### \*ABT\* Error Responses

24-Job not active  
31-Parameter required



## 4.6 SYSTEM OPTIONS

You may set the value of various system options and control the slave channels with the following commands:

<u>COMMAND NAME</u>	<u>DESCRIPTION</u>
SYSTEM	Designates the system drive.
DEVICE	Specifies device status to SDOS.
ICE	Sets operation mode for the slave CPU.
ASSIGN	Assigns devices to slave I/O channels.
CLOSE	Disconnects slave channels from device assignments.

These commands are discussed on the following pages.

### NOTE

The SEARCH command has been deleted from SDOS in version 3.0.

#### 4.6.1 SYSTEM

SYSTEM {drive number}

##### PURPOSE

The SYSTEM command is used to specify the disk drive to be used as the system drive.

##### EXPLANATION

The SYSTEM command allows you to designate any disk drive as the system drive. The default value for the system drive is 0.

Drive number must be in the range 0-3.

At power-up or on RESET, the system selects the system drive as the first disk drive that contains a diskette. The search starts with drive 0.

##### \*DOS\* Error Responses

9-Invalid drive number

## 4.6.2 DEVICE

<u>DEVICE</u> {device name} {U} {D}
--

### PURPOSE

The DEVICE command informs SDOS of the availability of a peripheral device.

### EXPLANATION

The DEVICE command specifies the availability of the named device. The device named must be one of the system device names (CONI, CONO, LPT1, TTYR; see Tables 4-1A and B).

The second parameter, either U or D, must be specified. If U is specified, the system is informed that the device is up and available for use. If D is specified, the system is informed that the device is down and not available for use.

### \*DEV\* Error Responses

- 30-Invalid parameter
- 31 Parameter required
- 52-Invalid device

### 4.6.3 ICE

**ICE {mode}**

#### PURPOSE

The ICE command is used to specify slave CPU operational mode.

#### EXPLANATION

The ICE command mode sets the slave CPU mode of operation. The possible values for mode are:

- OFF - puts the slave in normal mode, using TWIN slave memory, I/O, and clock.
- 1 - puts the slave into partial TWICE mode, using TWIN slave memory with user prototype I/O and clock.
- 2 - puts the slave into full TWICE mode, using user prototype memory, I/O, and clock.

The default is OFF.

#### \*ICE\* Error Responses

- 32-Too many parameters
- 54-Invalid mode

#### CAUTION

The ICE command replaces the SLAVE command from SDOS version 2.0.

#### 4.6.4 ASSIGN

<code>ASSIGN {channel number} { device name file name [/diskdrive] } [ channel number { device name file name [/disk drive] } ] ...</code>
--

#### PURPOSE

The ASSIGN command causes the connection of the logical slave I/O channel to the specified device.

#### EXPLANATION

The ASSIGN command causes a slave I/O channel to be connected to a device. The channel number must be in the range from 0 to 7. The device named may be a file on a diskette or one of the system device names (CONI, CONO, LPT1 TTYR; see Tables 4-1A and B).

Every disk file is viewed as an independent physical device. When a disk file name is used as DEVICE in the ASSIGN command, the directory of the diskette is searched for the file name. If the file name is not found, the file is created in the directory.

The specified channel is connected to the device which results in all subsequent I/O operations on the channel being performed on that device.

A user application (slave) program may have only 7 of the 8 assignable channels assigned to files. When the command is executed from within a procedure file, only 6 channels may be assigned to disk files.

#### \*ASN\* Error Responses

- 1-Directory read error
- 9-Invalid drive number
- 12-Invalid file name
- 18-Device in use
- 19-Invalid channel number
- 20-Channel in use
- 21-Channel assign failure
- 31-Parameter required
- 61-File in use
- 62-Device not operational

#### 4.6.5 CLOSE

**CLOSE {channel number} [channel number]...**

#### PURPOSE

The CLOSE command disconnects the specified slave I/O channel from its associated device.

#### EXPLANATION

The CLOSE command disconnects the channel from the device that was connected to it by the ASSIGN Command.

If the channel was assigned to a disk file the data remaining in the SDOS deblocking buffer is written to the file before closing.

The channel number must be in the range 0-7.

#### \*CLS\* Error Responses

- 2-Directory write error
- 7-Device write error
- 19-Invalid channel number
- 20-Channel in use
- 31-Parameter required
- 62-Device not operational
- 64-Invalid diskette



## 4.7 DISKETTE AND FILE UTILITIES

You can perform diskette and file utilities with these commands.

<u>COMMAND NAME</u>	<u>DESCRIPTION</u>
FORMAT	Initializes a diskette.
VERIFY	Finds and catalogs defective blocks on a diskette.
DUP	Duplicates all files on a diskette.
LDIR	Lists diskette directory.
RENAME	Changes file name or diskette ID.
COPY	Moves data between system devices.
PRINT and PRINTL	Prints out lines of data to devices.
DELETE	Removes a file from a diskette.
CMPF	Compares two files.
DFIL	Dumps a file in Hex format.

These commands are explained in detail on the following pages.

## 4.7.1 FORMAT

**FORMAT {disk drive} [ident]**

### PURPOSE

A new diskette must be formatted and verified before it can be used by SDOS.

The formatting process prepares a blank diskette for use with SDOS by writing such information as clock bits, sync patterns, track and sector numbers, data patterns and CRC characters on the diskette. The formatting process also pre-sets the diskette directory to reserve tracks 1 through 4 for SDOS.

### EXPLANATION

The FORMAT command causes the diskette on the specified drive to be formatted. The drive specified must not be the designated system drive. For example, if the designated system drive is drive 1, the appropriate command format is FORMAT 0. In this case the diskette on drive 0 is formatted. Since tracks 0 through 4 are reserved for SDOS, any bad sector detected on these tracks causes the process to abort.

The IDENT portion of the command is optional but serves to identify the diskette. This identification is always displayed when the diskette directory is listed. If the IDENT is not specified at the time of formatting, a string of blanks is used to identify the diskette. IDENT is truncated to 48 characters if more than that are entered.

If the diskette is not used for storage of system software, the area reserved for SDOS may be freed for other uses after formatting by entering the DELETE SDOS/n command. This prevents the use of this diskette for system programs unless it is again formatted, in which case any files on it are destroyed.

### CAUTION

The DELETE SDOS command can delete SDOS from your system diskette. If this happens, the system is non-recoverable and you will need to obtain a new system disk.

## PROCEDURE FOR FORMATTING A FLEXIBLE DISK

1. Power up the TWIN as previously described in Section 3.4 of this manual.
2. Insert a system diskette into drive 0 of the Floppy Disk Unit.
3. Insert a blank diskette into drive 1.
4. Enter the following command string after the prompt character.

>FORMAT 1 identification

5. Execution of this command takes approximately three minutes. When the formatting process is completed, SDOS responds with:

\*FMT\* EOJ

### \*FMT\* Error Responses

2-Directory write error  
8-Drive not specified  
9-Invalid drive number or system drive number  
17-Output device assign failure  
18-Device in use  
47-System area bad

### NOTE

Diskettes purchased from Signetics are pre-formatted and verified for you.

## 4.7.2 VERIFY

```
VERIFY {disk drive}
```

### PURPOSE

The verification process determines if any sectors on a diskette are defective, then records the location of the defective track on the bad block bit map.

### EXPLANATION

This command causes the diskette on the specified disk drive to be verified.

This verification process consists of reading every sector on the diskette and noting all errors that occur. When an error on a sector is found, the four blocks on the track in which the defective sector resides are recorded in the bad block bit map. In addition, the track and sector number of the defective sector are printed on the console. When all the sectors have been read, the bad block bit map is written on the diskette. This is a read only process which does not destroy data on the diskette; only the bad block bit map is rewritten to the diskette.

Whenever files are created and disk space allocation for the file is performed, reference is made to the bad block bit map and any defective blocks are not allocated.

If a defective sector is detected on tracks 0 through 4 (the SDOS area) during the verification process, the process is aborted and an appropriate message displayed on the console.

### \*VER\* Error Responses:

- 1-Directory read error
- 2-Directory write error
- 9-Invalid drive number
- 16-Input device assign failure
- 18-Device in use
- 31-Parameter required
- 47-System area bad

### 4.7.3 DUP

**DUP {disk drive 1} {disk drive 2} [diskette identifier]**

#### PURPOSE

The DUP command causes an exact duplication of the contents of the diskette in drive 1 to be created on the diskette in drive 2.

#### EXPLANATION

The DUP command causes the files and directory data stored on the diskette on disk drive 1 to be copied onto the diskette on disk drive 2. The disk drive number entered for 1 may not be the same as the number used for 2. In addition, the number used for 2 may not specify the system drive.

The IDENTIFIER portion of the command is optional but serves to identify the new diskette. This identification is always displayed when the disk directory is listed. If the IDENTIFIER is not specified at the time of formatting, a string of blanks is used to identify the new diskette. The IDENTIFIER is truncated to 48 characters if more than that are entered.

If a disk read or write error occurs during a file copy, the output file is deleted from the flexible disk on drive 2, a warning message is displayed on the system console, and the DUP process continues with the next file.

The diskette on drive 2 should be verified before the DUP command is executed in order to establish the bad block bit map for the diskette.

Uses for the DUP command include making backup diskettes for your system. At times you may want to make a backup diskette for a non-system diskette. If your system has more than two disk drives, the DUP command can be used if you have the system disk in the system drive and the other two flexible disks in other drives. The system disk is needed to provide the DUP command.

If you have the minimum system with only two disk drives, it might seem impossible to DUP a non-system diskette. The following procedure allows you to use your system to duplicate non-system diskettes.

## Procedure for Duplicating Non-system Diskettes with Two Disk Drives

1. Insert the system disk into drive 0 and toggle the RESET switch.
2. After the prompt character is displayed on the console, enter the following command:

```
> DUP 1 0
```

3. The following error will be displayed along with an End-Of-Job message:

```
*DUP* ERROR  
*DUP* EOJ
```

Now the DUP command program is resident in system memory and the system diskette is not required for execution of the DUP command.

4. Insert the non-system diskette to be copied into disk drive 1.
5. Remove the system diskette from drive 0 and insert a blank diskette into drive 0.
6. Enter the following command:

```
> SYS 1
```

7. Next enter the command:

```
> DUP 1 0
```

8. When the files on the diskette in drive 1 have been copied onto the blank diskette drive 0, the following message will be displayed on the console:

```
*DUP* EOJ
```

### \*DUP\* Error Responses:

- 1-Directory read error
- 2-Directory write error
- 6-Read error, DUP continues
- 7-Write error, DUP continues
- 9-Invalid drive number or system drive number
- 16-Input device assign failure
- 17-Output device assign failure
- 21-Channel assign failure



#### 4.7.4 LDIR

<code><u>L</u>DIR [disk drive] [.] [/] [device name file name/disk drive]</code>
--

#### PURPOSE

The LDIR command causes the contents of a floppy disk directory to be sent to the specified device.

#### EXPLANATION

The LDIR command lists the contents of the disk directory that is mounted on the specified disk drive. If the disk drive is not specified, the system disk directory is listed. When a decimal point "." is specified as a parameter, all of the SDOS system files will be included in the directory listing. When a slash "/" is specified as one of the parameters, space allocation information and file identification information are included in the listing.

Any valid output device name (CON0, LPT1, R232) is a valid DEVICE NAME entry (see Table 4-1A and B). In addition, any valid file may be named as an output device; however, you must specify the disk drive where the flexible disk with that file is located. The listing of the directory to a file will overlay any data in that file. A new file will be created if a file with the specified name does not exist.

The diskette identification precedes the directory listing and a summary of diskette sector usage follows it. The SDOS system file sector usage is included in the summary, whether or not the "." option was specified.

#### \*DIR\* Error Responses

- 1-Directory read error
- 7-Device write error
- 8-Drive not specified for FILE NAME
- 10-Overlay load failure
- 15-Invalid output device
- 17-Output device assign failure
- 30-Invalid parameter

## 4.7.5 RENAME

<pre>RENAME {old file name/disk drive} {new file name} or RENAME {disk drive} {diskette identifier}</pre>
---

### PURPOSE

The RENAME command is used to:

- 1) rename a file
- 2) rename a diskette

### EXPLANATION

#### Renaming a File

The first form of the RENAME command causes the name of a file on the specified disk drive to be changed. This form requires that a disk drive number be specified with the OLDFILE name. A disk drive number may be specified with the NEWFILE name, however, it must be the same as that specified for the OLDFILE name. The following is a typical transaction.

```
>REN TEST/0 TEXT  
*REN* EOJ
```

#### Renaming a Flexible Disk

The second form renames the diskette on the specified disk drive with the character string IDENTIFIER. The IDENTIFIER is truncated if longer than 48 characters. The following is a typical transaction:

```
>REN 0 MASTER SYSTEM DISK  
*REN* EOJ
```

The system diskette cannot be renamed from within a command file. To do so results in an \*REN\* ERROR 18.

\*REN\* Error Responses

- 1-Directory read error
- 2-Directory write error
- 8-Drive not specified
- 9-Invalid drive number
- 12-Invalid file name
- 13-Input file not found
- 16-Input device assign failure
- 18-Device in use
- 30-Invalid parameter
- 31-Parameter required
- 32-Too many parameters
- 57-File name in use

#### 4.7.6 COPY

<code>COPY</code> {input device name input file name [/disk drive]} [input device name input file name [/disk drive]] ... {output device name output name [/disk drive]}
---

#### PURPOSE

The COPY command is used to transfer data from one or more devices or files to another device or file.

#### EXPLANATION

The COPY command transfers data from the input device or file to an output device or file. More than one input device or file may be specified; however, the output device or file may not be used as an input device or file.

Data is transferred from the input device or file to the specified output device until an end-of-file condition is encountered on the input. If more than one input device or file is specified, the data are concatenated in the following manner:

1. The data from the first input device or file is transferred to the output device or file until an end-of-file condition is reached.
2. The data from the second input device or file is transferred to the output device or file and concatenated directly to the first set of data.
3. The data from the third input file or device is then transferred to the output device or file, etc.
4. When the end-of-file condition is encountered in the last input device or file the output device or file is closed.

When an ASCII file is being input from one of the system devices (CONI, TTYR, or R232), the CTRL-Z character is interpreted as the end-of-file condition.

#### \*COP\* Error Responses

- 6-Input read error
- 7-Output write error
- 13-Input file not found
- 14-Not an input device
- 15-Not an output device
- 16-Input device assign failure
- 17-Output device assign failure
- 30-Parameter error
- 52-Invalid device name

#### 4.7.7 PRINT and PRINTL

$\left. \begin{array}{l} \text{PRINT} \\ \text{PRINTL} \end{array} \right\} \text{file name [ /disk drive] } \left[ \begin{array}{l} \text{device} \\ \text{file name [ /disk drive]} \end{array} \right] \left[ \begin{array}{l} \text{begin line number} \text{ } \text{end line number} \\ \text{end line number} \end{array} \right]$
---

#### PURPOSE

The PRINT and PRINTL commands transfer lines of data from an input file to an output device or file. The PRINTL command numbers each line upon output.

#### EXPLANATION

The PRINT and PRINTL commands transfer lines of data from the specified input file to the specified output device or file. When the output device or file is not specified, the data lines are printed to the line printer LPT1.

The line numbers must be greater than or equal to one and less than 32,768, and BEGIN must be less than or equal to END. END line number must be greater than or equal to BEGIN line number. When the line range is specified using BEGIN and END, only the lines from BEGIN through END are output. For example, when the first number is 4 and the second number is 7, then lines 4, 5, 6 and 7 are transferred from the specified file. If only one line number is specified, all lines from the first line in the file through the specified line are transferred. When a line range is not specified the entire file is transferred.

When the PRINTL command form is used the lines are numbered as they are output.

#### \*PRN\* Error Reponse

- 6-Input read error
- 7-Output write error
- 13-Input file not found
- 14-Invalid input device
- 15-Invalid output device
- 16-Input device assign failure
- 17-Output device assign failure
- 30-Invalid parameter

#### 4.7.8 DELETE

```
DELETE {file name/disk drive} [file name/disk drive] ...
```

##### PURPOSE

The DELETE command is used to delete specified files from a diskette.

##### EXPLANATION

The DELETE command causes the file named to be deleted from the specified disk drive. Each file specified must have a disk drive number associated with it. [file name/disk drive]... indicates that more than one file can be specified for deletion in a single command line.

Upon execution of the DELETE command, each file specified in the parameter list is deleted from the directory of the flexible disk on which it resides. The sector blocks allocated to the deleted file are released for reallocation.

If an error occurs, command processing aborts; however, all files processed up to the time of the error have already been deleted.

##### \*DEL\* Error Responses

- 2-Directory write error
- 8-Drive not specified
- 9-Invalid drive number
- 12-Invalid file name
- 13-File not found
- 18-Device in use
- 21-Channel assign failure
- 30-Invalid parameter
- 31-Parameter required
- 61-File in use



#### 4.7.9 CMPF

<code>CMPF {file name 1 [/disk drive]} {file name 2 [/disk drive]} [output device name output file name [/disk drive]] [mode]</code>
--

#### PURPOSE

The CMPF command is used to compare two files according to the compare mode specified and list an exception report on the specified device. Mode may be:

- 1) B-Binary
- 2) A-ASCII

#### EXPLANATION

The CMPF command compares the first file specified with the second file. An exception report lists any differences on the OUTPUT DEVICE. If a device is not named, the exception report is sent to the system console, CONO. Any output device name (CONO, LPT1, R232) is a valid entry. In addition, any valid file may be named an OUTPUT DEVICE. The output to a file overlays any data in that file. If a file with the specified name does not exist, a new file is created.

ASCII mode compares the files line by line, while BINARY mode compares them byte by byte. The default is binary mode.

Independent of the MODE specified, the compare continues until an end-of-file is encountered in one of the files. Then, all remaining data in the longer file is output. A double asterisk \*\* is printed in lieu of data for the shorter file.

The disk drive number for either file is optional, but if either file is not stored on the system disk then the drive number must be specified.

The exception report begins with the two file names and ends with a total exception count in Hexadecimal.

#### ASCII Mode Compare

The first file named is compared line by line with the second file named. When an exception is found, the line number of the exception line is output followed by the line data from the first and second files, respectively.

Example:

The following data is contained in two files which reside on a diskette in disk drive 1.

File one:

This text file describes the  
operation of the text editing  
program 'edit'  
(EOF)

(r)  
(r)  
(r)

File two:

This text flie describes the  
operation of the text editing  
program 'edit'  
(EOF)

(r)  
(r)  
(r)

The differences in these files are underlined. To compare the files line by line enter:

```
>CMPF,ONE/1,TWO/1,,A
```

An exception report will be listed to CONO as follows:

F1= ONE  
F2= TWO

Line 1

This text file describes the  
This text flie describes the

(r)  
(r)

TOTAL EXCEPTIONS = 0001  
\*CMP\* EOJ

### BINARY Mode Compare

The first file named is compared byte for byte with the second file named. When exceptions are found, they are listed on the output device, along with the byte number in Hex into the file where the exception was found.

Example:

Using the previous example, suppose you preferred to do a binary compare of the two files. Enter the command:

>CMPF,ONE/1,TWO/1

In the first line of the files, two letters in the word "file" do not compare.

The following exception report is output:

F1= ONE  
F2= TWO

BYTE	F1	F2
000C	49	52
000D	52	49

TOTAL EXCEPTIONS = 0002  
\*CMP\* EOJ

### Comparing Two Load Modules

When a BINARY mode compare is requested, both files are checked to see if they are load modules before the actual compare begins. This is done by looking for a proper load module header at the beginning of each file. If these checks indicate both files are load modules, the module identifications are printed below the file name header on the exception report. In Sector 0 the applicable load module data is compared, then all bytes of successive sectors are compared.

In rare instances, the files may be determined to be load modules, but the entire first sector (Sector 0) is compared without encountering an "End of Load" code. When this happens, the compare is restarted and done as a normal binary mode compare.

### \*CMP\* Error Responses

- 6-Device read error
- 7-Device write error
- 10-Overlay load failure
- 13-Input file not found
- 14-Invalid input device
- 15-Invalid output device
- 16-Input device assign failure
- 17-Output device assign failure
- 30-Invalid mode
- 31-Parameter required

#### 4.7.10 DFIL

<code>DFIL {file name [/disk drive]} [output device name output file name [/disk drive]] [start byte] [end byte]</code>
---

#### PURPOSE

The DFIL command is used to dump a file in Hexadecimal format to the specified device.

#### EXPLANATION

The DFIL command dumps a file to the specified output device or file name. If an output device is not named, the file dump is listed on CONO.

For readability, the output is formatted in blocks of 128 bytes of data, sixteen data bytes per line.

ASCII interpretation of the Hex data is printed to the right of each data line. Bytes which are not ASCII characters are interpreted as a period.

The file is dumped according to the following rules.

- 1) The file is dumped from beginning of file if START is not specified.
- 2) The file is dumped to end-of-file, if END is not specified.
- 3) START byte and END byte must be valid Hexadecimal numbers in the range 0000 through FFFF.
- 4) START must be less than or equal to END.
- 5) If START equals END only one byte is output.
- 6) If END byte is greater than the file size, the dump continues until an end-of-file is reached.

#### Example 1

To dump the entire contents of the file named SAM, of H'91' bytes, in disk drive 1 to the line printer, enter the command:

```
>DFIL,SAM/1,LPT1
```

Output is produced as follows:

```
FILE = SAM

0 1 2 3 4 5 6 7 8 9 A B C D E F

0000
54 48 49 53 20 49 53 20 41 20 46 49 4C 45 20 54      THIS IS A FILE T
4F 20 44 45 4D 4F 4E 53 54 52 41 54 45 20 54 48      O DEMONSTRATE TH
45 20 44 46 49 4C 20 43 4F 4D 4D 41 4E 44 0D 0D      E DFIL COMMAND.
00 20 00 20 00 49 54 20 43 4F 4E 54 41 49 4E 53      . . . IT CONTAINS
20 41 20 54 4F 54 41 4C 20 4F 46 20 39 31 20 42      A TOTAL OF 91 B
59 54 45 53 2C 20 4E 55 4D 42 45 52 45 44 20 46      YTES, NUMBERED F
52 4F 4D 20 30 20 54 4F 20 39 30 2E 0D 4C 41 53      ROM 0 TO 90. LAS
54 20 4C 49 4E 45 20 2D 20 41 42 43 44 45 46 47      T LINE - ABCDEFG

0080
48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57      HIJKLMNOPQRSTUWV
0D
```

Example 2

To dump a portion of this file, from byte H'09' through byte H'83', enter the command:

```
>DFIL,SAM/1,LPT1,9,83
```

Output is produced as follows:

```
FILE = SAM

0 1 2 3 4 5 6 7 8 9 A B C D E F

0009
                20 46 49 4C 45 20 54      . . . FILE T
4F 20 44 45 4D 4F 4E 53 54 52 41 54 45 20 54 48      O DEMONSTRATE TH
45 20 44 46 49 4C 20 43 4F 4D 4D 41 4E 44 0D 0D      E DFIL COMMAND.
00 20 00 20 00 49 54 20 43 4F 4E 54 41 49 4E 53      . . . IT CONTAINS
20 41 20 54 4F 54 41 4C 20 4F 46 20 39 31 20 42      A TOTAL OF 91 B
59 54 45 53 2C 20 4E 55 4D 42 45 52 45 44 20 46      YTES, NUMBERED F
52 4F 4D 20 30 20 54 4F 20 39 30 2E 0D 4C 41 53      ROM 0 TO 90. LAS
54 20 4C 49 4E 45 20 2D 20 41 42 43 44 45 46 47      T LINE - ABCDEFG

0080
48 49 4A 4B      HIJK
```

\*DFL\* Error Responses

- 6-Device read error
- 7-Device write error
- 10-Overlay load failure
- 13-Input file not found
- 14-Input is not a valid file name
- 15-Invalid output device
- 16-Input device assign failure
- 17-Output device assign failure
- 31-File name parameter required
- 32-Too many parameters
- 35-Invalid start address
- 39-Invalid Hex character in START or END.



## 4.8 SYSTEM UTILITY COMMANDS

The following utility commands allow you to access and manipulate slave memory and your prototype memory.

<u>COMMAND NAME</u>	<u>DESCRIPTION</u>
MOVE	Move a block of data in slave or user prototype memory.
FILL	Fill slave or user prototype memory with a Hex string constant.
READ	Read a memory location or data bus.
WRITE	Write to a memory location or data bus.
UPR	Relocates slave-resident debug utility program.

## 4.8.1 MOVE

**MOVE {source-destination} {start source address} {end source address} {destination address}**

### PURPOSE

The MOVE command moves the block of data defined by the START and END source addresses to the DESTINATION address.

### EXPLANATION

The MOVE command moves a block of data from SOURCE memory to DEST memory according to the SOURCE-DESTINATION parameter table below.

<u>SOURCE-DEST</u>	<u>ACTION</u>
CC	moves a block of data within TWIN slave memory
UU	moves a block of data within user memory
CU	moves a block of data from TWIN slave memory to user
UC	moves a block of data from user memory to TWIN slave memory.

The two characters, C and U, of the SOURCE-DEST parameter must be juxtaposed.

The START, END, and DEST addresses must all be hexadecimal numbers in the range 0000 through FFFF. Additionally, START must be less than or equal to END.

All parameters are required.

### \*MOV\* Error Responses

- 30-Invalid SOURCE-DEST parameter
- 31-Parameter required
- 32-Too many parameters
- 34-START > END or Invalid DEST address
- 35-Invalid START address
- 36-Invalid END address
- 59-Memory write error
- 68-Attempt to clobber Debug utility routines.

## 4.8.2 FILL

`FILL {start address} {end address} {hex-string}`

### PURPOSE

The FILL command fills an area of slave or user prototype memory with a Hexadecimal string constant.

### EXPLANATION

The FILL command fills either TWIN slave memory or user prototype with multiples of the specified HEX-STRING constant. The fill function is executed according to the current ICE mode:

<u>ICE MODE</u>	<u>ACTION</u>
OFF	fill TWIN slave memory
1	
2	fill user prototype memory

The START and END addresses must be entered as hexadecimal numbers in the range 0000 through FFFF, and START must be less than or equal to END.

If the specified memory area to fill is not an integral multiple of the HEX STRING length then the HEX STRING is truncated at the END address and a warning error is output to the system console. HEX STRING must be an even number of hexadecimal digits, not less than 2 nor more than 68 characters in length.

All parameters are required.

### \*FIL\* Error Responses

- 30-Invalid parameter
- 31-Parameter required
- 32-Too many parameters
- 34-START address > END address
- 35-Invalid START address
- 36-Invalid END address
- 39-Invalid Hex character
- 59-Memory write error
- 68-Attempt to clobber Debug utility routines

### 4.8.3 READ

<code>READ</code> { <code>memory address</code> <code>2650 read instruction type</code> } [ <code>output option</code> ]
---

#### PURPOSE

The READ command reads from either a memory address or data bus in the user prototype system.

#### EXPLANATION

The READ command is used to generate and execute a small slave program in order to read a user memory location or to execute one of the three 2650 Input Instructions as follows:

```
REDC,RO
REDD,RO
REDE,RO Device-Address
```

The device address must be specified if you select the REDE instruction.

Note that the READ command does not operate with the TWIN common memory.

You may elect to perform the read only once, and display the data bus or memory contents; or you may elect to perform the read continually with no data displayed. In the case of a continuous read, the results may be monitored with an oscilloscope since no data is displayed.

The OPTION parameter selects either a one-time or continuous read as follows:

```
* - continuous read
device name - one-time read with data displayed on the specified device
```

OPTION default is a one-time read with the data output to CONO. Use the ESC key to terminate continual read mode.

#### Read Memory Location

To READ a memory location, enter the memory address as the first parameter and select the continuous read or a one-time read output device.

Example:

The comand

```
> READ,FFF,LPT1
```

causes the contents of address FFF to be read once and its contents written on the line printer.

#### Read from Data Bus

To READ from the data bus, enter the 2650 INSTRUCTION TYPE shown below:

#### 2650 Instruction

REDC,RO  
REDD,RO  
REDE,RO Device Address

#### Enter Instruction Type

RC  
RD  
RE-dd

where:

$0 \leq dd \leq EF_H$  or  
 $F8_H \leq dd \leq FF_H$

Note that device addresses  $F0_H$  through  $F7_H$  are reserved for SDOS supervisor calls (SVCs).

As for read memory, you may select either the continual read or one-time read with output displayed.

Example:

The command

```
> READ,RE-F8,*
```

causes device address F8 to be read continually via the REDE,RO F8 instruction. No data is displayed. The read will repeat until you depress the ESC key.

#### \*RDW\* Error Responses

- 15-Invalid output device
- 17-Output device assign failure
- 27-Slave CPU is in use
- 30-Invalid parameter
- 31-Address parameter required
- 34-Invalid memory address
- 56-Invalid device address or F0-F7 reserved for SVCs

#### 4.8.4 WRITE

<u>WRITE</u> { <sup>memory address</sup> 2650 write instruction type } [option] [hex-string]
---

#### PURPOSE

The WRITE command writes a string of hexadecimal data to either a memory location or data bus in the user prototype system.

#### EXPLANATION

The WRITE command is used to generate and execute a small slave program in order to write a specified HEX-STRING to a user memory location or to the user data bus via one of the three 2650 Output Instructions as follows:

```
WRTC,RO  
WRTD,RO  
WRTE,RO Device-Address
```

The device address must be specified if you select the WRTE instruction.

Note that the WRITE command does not operate with TWIN common memory.

You may elect to perform the output of the HEX-STRING data or single default data byte only once; or you may elect to perform the write continually.

The OPTION parameter selects either a one-time or continuous write as follows:

```
* - continuous write of HEX-STRING  
1 - write HEX-STRING one time
```

OPTION default is a one-time write. Use the ESC key to terminate continual write mode.

HEX-STRING can be any even number of hexadecimal digits up to a maximum of 60 digits. The default value of HEX-STRING is FF<sub>H</sub> (all ones).

Execution of the WRITE instruction must be monitored with an oscilloscope.

#### Write to Memory Location

To WRITE to memory location enter the memory address as the first parameter, and select the option as desired. If you specify a HEX-STRING of length greater than 2 digits, the data will be written beginning at the specified memory address and continue through successive memory addresses until all of HEX STRING has been written.



Continuous write begins each iteration at the specified memory address.

Example:

The command

```
>WRITE,100,*,FF00
```

writes the data FF<sub>H</sub> and 00<sub>H</sub> to memory addresses 0100<sub>H</sub> and 0101<sub>H</sub>, respectively. The sequence is repeated until you depress the ESC key.

Write to the Data Bus

To WRITE to the data bus, enter the 2650 INSTRUCTION TYPE as shown below:

2650 INSTRUCTION

ENTER INSTRUCTION TYPE

WRTC,RO  
WRTD,RO  
WRTE,RO Device Address

WC  
WD  
WE-dd

where

0 ≤ dd ≤ EF<sub>H</sub>

F8<sub>h</sub> ≤ dd ≤ FF<sub>H</sub>

Note that device addresses F0<sub>H</sub> through F7<sub>H</sub> are reserved for SDOS supervisor calls (SVCs).

As for write memory, you may select either the continual write or one time write. If you specify a HEX-STRING of length greater than 2 digits, the string is written, a byte at a time, according to INSTRUCTION TYPE. Continuous write simply loops through HEX-STRING until the ESC Key is depressed.

Example:

The command

```
>WRITE,WE-F8,1,0102040816
```

executes the 2650 instruction WRTE,RO F8 once for each byte in the HEX-STRING.

The command

```
>WRITE,WE-F8,*
```

executes the 2650 instruction WRTE,RO F8 continually until you depress the ESC key. In this example, the default HEX-STRING value FF<sub>H</sub> is output.

**\*RDW\* Error Responses**

- 27-Slave CPU is in use
- 30-Invalid parameter
- 34-Invalid memory address
- 43-Invalid HEX-STRING
- 56-Invalid device address or F0-F7, reserved for SVCs

#### 4.8.5 UPR

<u>UPR</u> [address]
----------------------

#### PURPOSE

The UPR command, Utility Program Relocate, moves the SDOS slave-resident debug utility programs from the top 256 bytes of slave memory to the specified slave memory address.

#### EXPLANATION

So that the user can have access to the highest 256 bytes of slave memory, the UPR Command is provided to move the debug utility programs to another area of slave memory. ADDRESS must be evenly divisible by 256.

When the utility programs are moved out of the initial memory block they are not write-protected at the new location. If no ADDRESS parameter is entered, the utility programs are moved to the initial memory location in the top-most 256-byte slave memory block with write protect. Refer to Section 4.1.4, Debug Utility Programs.

Note that if the default location is actually entered for ADDRESS, the programs are not protected.

#### CAUTION

At no time are the utility programs write-protected against the LOAD or XEQ commands.

#### \*UPR\* Error Responses

- 6-Cannot read utility programs from diskette
- 34-Invalid Hex address or syntax error
- 35-ADDRESS is not 256 aligned or ADDRESS is greater than memory size
- 58-SDOS system failure

## 4.9 OBJECT PROGRAM UTILITIES

The commands in this section are used to move object code between program memory and flexible disk storage or a peripheral device. The object code may be stored on a flexible disk either in binary or hexadecimal format. The object code is loaded into program memory in binary format.

<u>COMMAND NAME</u>	<u>DESCRIPTION</u>
WHEX	Converts binary code to hexadecimal format and writes it on the diskette or to a device.
RHEX	Reads hexadecimal formatted code, converts it to binary code and loads the binary code into slave memory.
MODULE	Creates a binary object module on diskette or a device from binary code in slave memory.
WSMS	Writes a block of slave memory in SMS format to a diskette or a device.
CSMS	Translates an SMS file and compares it with slave memory.

SMS format is used by Signetics for the generation of PROMs and is described in Appendix D.

Hexadecimal format is described in Appendix C.

## 4.9.1 WHEX

```
WHEX {address 1} {address 2},, [ {address 1} {address 2} ] ..., [ {address 3} {device  
file name [/disk drive]} ]
```

### PURPOSE

The WHEX command converts binary code in slave memory to hexadecimal format and writes the hexadecimal code blocks on the specified device. Hexadecimal format is described in Appendix C.

### EXPLANATION

The WHEX command program causes an absolute hexadecimal format file to be written from the binary code in slave memory to the diskette or device. ADDRESS 1 and ADDRESS 2 are the addresses of the lower and upper bounds respectively, of the user program in slave memory and must be entered in pairs separated by a pair of commas. The ADDRESSES 1, 2, and 3 are to be entered in hexadecimal. ADDRESS 3 is an optional starting address.

The DEVICE is an optional output device or file. When the DEVICE is specified, the starting ADDRESS 3 must be specified. When the DEVICE is not specified, the output is to the console output device, CONO.

The WHEX command writes the data beginning with ADDRESS 1 through ADDRESS 2 for each ADDRESS 1,2 pair present in the parameter list.

### NOTE

Two commas are required between address pairs if multiple address pairs are specified.

### \*WHX\* Error Responses

- 7-Device write error
- 15-Invalid output device
- 17-Output device assign failure
- 30-Invalid parameter

## 4.9.2 RHEX

<code>RHEX [/bias amount] [device file name [/disk drive]]</code>
---

### PURPOSE

The RHEX command reads hexadecimal formatted code from the diskette or device, converts it to binary code and loads the binary code into slave memory.

### EXPLANATION

The RHEX command loads the absolute hexadecimal code into slave memory. The absolute hexadecimal code is read into memory from the specified device or file. The DEVICE defaults to the teletype paper tape reader TTYR.

The BIAS amount is used to alter the absolute load address for the file entered as a signed hexadecimal address constant. When the sign is not specified, the bias amount is assumed to be positive. The default value for the BIAS amount is zero. When BIAS is specified, the initial load address is altered by the BIAS amount.

The program start address given at the end of the object file will be ignored by SDOS. It must be entered as part of the GO command when execution of the program is requested.

### \*RHX\* Error Responses

- 6-Device read error
- 14-Invalid input device
- 16-Input device assign failure
- 33-Bias parameter error
- 40-Invalid input format

### 4.9.3 MODULE

<code>MODULE {file name [/disk drive]} {address 1} {address 2} {address 3} [module identifier]</code>
---

#### PURPOSE

The MODULE command writes binary code in binary load module format onto the diskette from slave memory.

#### EXPLANATION

The MODULE command writes binary code into the specified file from slave memory. ADDRESS 1 and ADDRESS 2 are the addresses of the lower and upper bound respectively of the user program in slave memory. ADDRESS 2 must be greater than or equal to ADDRESS 1. ADDRESS 3 is the starting execution address of the program. The ADDRESSES 1, 2, and 3, are to be entered in hexadecimal. The load module is preceded by a one sector header which contains the memory bounds of the ADDRESSES 1 and 2 of the binary object module and its starting address, ADDRESS 3. This starting address is loaded by LOAD, allowing you to execute the program by simply typing GO.

#### \*MOD\* Error Responses

- 7-Device write error
- 10-Overlay load failure
- 12-Invalid file name
- 32-Too many parameters
- 34-Invalid address



#### 4.9.4 WSMS

<u>WSMS</u> [address] [file name [/disk drive]] [device]
---

#### PURPOSE

The WSMS command outputs a 512-byte block of slave memory in SMS format.

#### EXPLANATION

ADDRESS specifies the first location of memory to be written. The default value of ADDRESS is 0. DEVICE specifies the output device or disk file to which the SMS data is to be written. The default value of DEVICE is CONO. SMS format is described in Appendix D.

#### \*SMS\* Error Responses

- 7-Device write error
- 15-Invalid output device
- 21-Channel assign failure
- 30-Invalid parameter
- 35-Invalid address

#### 4.9.5 CSMS

<u>CSMS</u> [address] [file name [/disk drive] device]
---

#### PURPOSE

The CSMS command reads a file that is written in SMS format from DEVICE, translates the data to binary, and compares the data with slave memory.

#### EXPLANATION

ADDRESS refers to the first location in slave memory that will be compared with the SMS file. The default value of ADDRESS is 0. DEVICE is the input device or disk file where the SMS data resides. The default value of DEVICE is TTYR, the teletype paper tape reader. CONI cannot be the input device.

The SMS file is compared with a 512-byte block of memory. If an SMS byte and the contents of a memory location are not equal, the memory location will be displayed on the console. SMS format is described in Appendix D.

#### \*SMS\* Error Responses

- 6-Device read error
- 13-Input file not found
- 14-Invalid input device
- 21-Channel assign failure
- 30-Invalid parameter
- 35-Invalid address

## 4.10 COMMAND FILES

SDOS provides the user with the capability of executing a sequence of SDOS commands by issuing a single command. This capability is implemented through the use of COMMAND FILES. A COMMAND FILE consists of a sequence of SDOS command lines. When the name of the command file is used as an SDOS command, SDOS first determines that the command file name is not one of the basic SDOS commands. It then searches the system directory for the file name. When the file is located, it treats the first line as an SDOS command and executes it. Then the second line is executed, and so forth, until an end-of-file condition is reached on the command file.

Example:

Suppose the Editor was used to create the following file named LISTALL on the diskette in Drive 0. LISTALL contains the following primary SDOS commands.

```
LDIR 0 LPT1
LDIR 1 LPT1
LDIR 2 LPT1
LDIR 3 LPT1
```

If "LISTALL" is entered as an SDOS command, SDOS will locate LISTALL on the diskette in drive 0 and execute the first line as an SDOS command. This will result in the directory of the diskette on drive 0 being printed on the line printer. Execution of the next three lines will result in the directories of the diskettes on drives 1, 2 and 3 being printed on the line printer.

### Parameters

Parameters may be entered in the command line with the command file filename. Command file parameters are positional and are passed to a command file in the same manner in which parameters are passed to a subroutine via a subroutine call. That is, the body of the command file contains primitive SDOS commands with formal parameters rather than the actual parameter values. Then, the actual parameters are entered along with the command file filename on the system console.

Formal parameters are identified by a \$ followed by an integer. The integers correspond to the order in which the parameters are entered in the command file request.

Example:

LISTALL is changed to:

```
LDIR 0 LPT1 $1 $2
LDIR 1 LPT1 $1 $2
LDIR 2 LPT1 $1 $2
LDIR 3 LPT1 $1 $2
```

A command file invocation to execute this command file is

```
>LISTALL . /
```

The '.', the first parameter, replaces all the \$1s and the '/', the second parameter, replaces all the \$2s in the LISTALL file. This results in execution of the following command stream execution:

```
LDIR 0 LPT1 . /  
LDIR 1 LPT1 . /  
LDIR 2 LPT1 . /  
LDIR 3 LPT1 . /
```

If the command:

```
>LISTALL /
```

were entered, the '/' would replace all the \$1s in the LISTALL file, but the \$2s will be replaced with blanks, resulting in execution of the command stream:

```
LDIR 0 LPT1 /  
LDIR 1 LPT1 /  
LDIR 2 LPT1 /  
LDIR 3 LPT1 /
```

Note that if a required parameter represented by a \$n is omitted when invoking the command file, an error may occur.

#### Command File Control

The SDOS commands available for use in conjunction with command files and a general description of the command file invocation are described on the following pages.

<u>COMMAND NAME</u>	<u>DESCRIPTION</u>
*	Prefaces all comments.
KILL	Aborts the command file on a system command error.
TYPE	Prints the SDOS commands in a command file as they are interpreted.

#### 4.10.1 Command Description

`filename [/disk drive] [parameter 1 ... parameter n]`

##### PURPOSE

The command file FILENAME is entered as an SDOS command to invoke the command file.

##### EXPLANATION

The command file is identified by a single name that must conform to the file naming conventions specified in Section 4.2.2, File Names.

When the command file is not resident on the system disk the disk drive must also be specified. This value defaults to the system disk.

Parameters specified in the command line may be used by the system commands that make up the command file body. For example, you may specify a program file name as a parameter in the command line and then have several system commands within the command file use or modify that file. Later you can have that same command file perform the same action on another program file. Parameters omitted from the sequential parameter list must be indicated by two consecutive commas. Parameters omitted from the end of the parameter list may be indicated by a number of commas one greater than the number of omitted parameters. (This is only necessary if the omitted parameter number referenced in an SDOS command inside the procedure has additional data after it which must be included in the command line for proper command execution.)

Command files cannot be nested but they can be chained. That is, if the last system command in a command file is the name of another command file, the command file being executed is terminated and the next command file is started. The parameters are passed from one command file to another in the same way they are passed to the system commands within the command file.

If a device read error is encountered in a command file, the entire file execution will be aborted, except when the value of the KILL switch is OFF. (See Section 4.10.3).

A maximum of six disk files instead of the normal seven can be assigned to a slave program while a command file is in progress.

#### 4.10.2 \* Comment

\* (The Asterisk) [comment]

#### PURPOSE

The asterisk \* is used to insert comments into the job flow of the command file.

#### EXPLANATION

The asterisk is entered into the first character position and must be followed by a space. The comment inserted is printed on the console as its turn comes up on the job flow. The print out of the comment may be inhibited by invoking the TYPE OFF command.

Comments must be entered as separate lines and not entered on the same line as a system command.

#### \* Error Responses

\*DOS\* ERROR 3 - The \* is not followed by a space.

### 4.10.3 KILL

<u>K</u> ILL {ON } {OFF }
------------------------------

#### PURPOSE

The KILL command causes termination of command file execution upon detection of an error in one of the system commands.

#### EXPLANATION

After the KILL ON command has been invoked, either from the keyboard or within the command file, a command file will be terminated if an error is encountered during the execution of any of the system commands within that file.

The KILL OFF command allows execution of a command file to continue after an error occurs, starting with the next system command in the file. Error messages are printed on the system console.

System level commands, those flagged by the \*DOS\* error ID, are not screened by the kill flag, and always cause termination of the command file.

The KILL command defaults to ON at power up and reset.

#### \*KIL\* Error Responses

- 30-Invalid parameter
- 31-Parameter required



#### 4.10.4 TYPE

<u>TYPE</u> {ON } {OFF }
-----------------------------

#### PURPOSE

The TYPE command causes each system command in the command file to be printed on the system console as the command is interpreted.

#### EXPLANATION

After the TYPE ON command has been invoked, either from the keyboard or within the command file, the command line for each system command within the command file is printed on the console at the start of the command execution.

The TYPE OFF command inhibits the printing of the command line and comments and only error messages are output.

The system defaults to TYPE ON at power-up and reset.

#### \*TYP\* Error Responses

- 30-Invalid parameter
- 31-Parameter required

#### 4.11 STANDARD SDOS COMMAND AND UTILITY FILES

Several standard files are provided to ease use of the system. They are:

<u>FILE NAME</u>	<u>TYPE</u>	<u>DESCRIPTION</u>
COPYSYS	Command File	Copies the operating system.
MAK3ORS	Command File	Reconfigures the operating system with the RS232 driver.
MAK3OPT	Command File	Reconfigures the operating system with the high speed paper tape driver.
MAK3OLP	Command File	Reconfigures the operating system with the Printronix 300 line printer driver.
R232;M HSPT;M LPT2;M	Utility Files	Load modules for the optional drivers.
EQUATES	Utility File	Contains standard assembler symbols equates.

#### 4.11.1 COPYSYS

```
COPYSYS [/disk drive] {disk drive 1} {disk drive 2}
```

#### PURPOSE

COPYSYS is a command file that copies the SDOS operating system from one diskette to another.

#### EXPLANATION

The COPYSYS command causes the system files on the diskette mounted on DISK DRIVE 1 to be copied to the diskette on DISK DRIVE 2. The COPYSYS command is entered after the SDOS prompt character > is displayed. The following example illustrates the use of this command to copy the SDOS system from a disk on disk drive 0 to a disk on disk drive 1:

```
> COPYSYS 0 1  
.  
.  
.  
>
```

The underlined portion is entered from the keyboard. When execution of the command has been completed, the following files have been copied over:

- . The resident SDOS binary load module.
- . All SDOS command overlays.
- . All SDOS slave jobs, i.e., the assemblers and the editor.
- . The COPYSYS command file.
- . The standard SDOS command, utility files, and system readiness test (Appendix E).

The operating system should be copied onto a diskette before any other files to achieve the most rapid system response to commands. This allocates the system files to the tracks on the outside of the disk and minimizes read head movement when the commands are brought into memory.

#### NOTE

The file SDOS must reside on tracks 0 through 4 of a diskette to be bootable.

#### 4.11.2 Configure Optional Drivers

<pre>{ MAK30RS } { MAK30PT } [ /disk drive ] { source disk drive } { destination disk drive } { SDOS version } { date } { MAK30LP }</pre>
---

#### PURPOSE

The MAK30xx command files reconfigure the SDOS operating system for one of the optional drives.

#### EXPLANATION

These command files reconfigure the SDOS operating system with one of the optional drivers according to the table below.

<u>Command File</u>	<u>Driver</u>	<u>Device Name</u>	<u>Driver Load Module Name</u>
MAK30RS	RS232	R232	R232;M
MAK30PT	High Speed Paper Tape Reader	HSPT	HSPT;M
MAK30LP	Printronix 300 Line Printer	LPT2	LPT2;M

The command file reads SDOS into slave memory, links the selected driver load module into SDOS, enters the device name into the system Device Definition Table, and creates a new SDOS load module file on the specified DESTINATION disk drive.

SOURCE disk drive specifies the drive on which SDOS and the drive object modules reside. DESTINATION disk drive is the drive number of the diskette to which the reconfigured SDOS is written.

SDOS Version and Date are identification fields which you may use to identify the new SDOS load module.

When SDOS is reconfigured with an optional driver, the associated device name is a reserved system device name in that version of SDOS only. If you reconfigure SDOS again and write it to the same diskette, then the new device name replaces the first one as a reserved name.

The new reconfigured SDOS must be booted in order to use the driver and de name.

The digits '30' in the command file filename correspond to the version of SDOS on which the command files operate. These command files operate on SDOS version 3.0.

### 4.11.3 EQUATES

The EQUATES file (Figure 4-1) is an assembly language source file which defines all of the standard assembler symbols for the 2650 registers, PSW, condition codes. This file may be appended to the beginning of any assembly language source file before assembly.

```
*****
* STANDARD SYMBOL DEFINITION - THIS FILE MAY BE APPENDED TO THE
*                               FRONT OF ANY USER'S SOURCE DECK
* REGISTER EQUATES
R0 EQU 0 REGISTER 0
R1 EQU 1 REGISTER 1
R2 EQU 2 REGISTER 2
R3 EQU 3 REGISTER 3
* CONDITION CODES
P EQU 1 POSITIVE RESULT
Z EQU 0 ZERO RESULT
N EQU 2 NEGATIVE RESULT
LT EQU 2 LESS THAN
EQ EQU 0 EQUAL TO
GT EQU 1 GREATER THAN
UN EQU 3 UNCONDITIONAL
* PSW LOWER EQUATES
CC EQU H'00' CONDITIONAL CODES
IDC EQU H'20' INTERDIGIT CARRY
RS EQU H'10' REGISTER BANK
WC EQU H'08' 1=WITH 0=WITHOUT CARRY
OVF EQU H'04' OVERFLOW
COM EQU H'02' 1=LOGIC 0=ARITHMETIC COMPARE
C EQU H'01' CARRY/BORROW
* PSW UPPER EQUATES
SENS EQU H'80' SENSE BIT
FLAG EQU H'40' FLAG BIT
II EQU H'20' INTERRUPT INHIBIT
SP EQU H'07' STACK POINTER
* END OF EQUATES
EJE
```

Figure 4-1. EQUATES File





## CHAPTER 5

### THE TEXT EDITOR

#### 5.0 INTRODUCTION

The major function of the TWIN Text Editor is to create new source programs or to change existing source programs. The Text Editor is also used for the creation and modification of COMMAND FILES. The Editor performs these functions by processing command lines entered by the user. Each command line specifies one action or a series of actions for the Editor to undertake, such as entering new source lines or searching the file for a specified string.

The Editor will be discussed by examining the SDOS command EDIT, presenting a sample edit, detailing all the Editor commands, and listing all the messages which the Editor may display to the operator.

The Editor resides in slave memory and occupies approximately seven thousand bytes of the memory. The remainder of the slave memory is available for the text that is being worked on. This is approximately 150 60-character lines in a system with 16K bytes of slave memory..

Throughout this discussion, there are two terms and a keyboard input convention which are used. These are:

Buffer

(or Workspace): The buffer is the slave memory area that contains the text that the Editor operates on. Data is written into and read from the buffer by the Editor. The buffer can be seen as having a top (or first) line and a bottom (or last) line. The Editor can operate on any line in the buffer. In this chapter, the terms workspace and buffer are used interchangeably.

Line Pointer: Data in the buffer is edited by examining, changing, inserting and replacing lines. The Editor keeps track of which line the operator is working on by keeping a pointer at the current line.

Ⓡ: This symbol will indicate the RETURN key.

If you are familiar with Editors, the section on the EDIT command, Section 5.1, the detailed description of the commands, Section 5.3, and the Editor messages, Section 5.4, will be most helpful.

If you are not familiar with Editors, Section 5.2, which describes a typical edit session, will be helpful in illustrating the use of the Editor commands.

## 5.1 THE EDIT COMMAND

You start the Editor with the SDOS EDIT command. This command has three forms:

- 1) EDIT INFILENAME OUTFILENAME
- 2) EDIT FILENAME
- 3) EDIT

### Form 1

INFILENAME designates the PRIMARY INPUT file and OUTFILENAME designates the PRIMARY OUTPUT file. The PRIMARY INPUT file will be the default file in any Editor command that asks for data from the disk. The PRIMARY OUTPUT file will be the default file in any Editor command that writes data to the disk. If INFILENAME is the same as OUTFILENAME, the file will be edited to itself. A temporary work file to be used as OUTFILENAME is created by changing the first character of INFILENAME to an \*. When you finish your edit session, INFILENAME is deleted, and then the temporary work file is renamed INFILENAME. For example, if:

```
EDIT DATA1 DATA1 (r)
```

is requested, DATA1 is the input file, and \*ATA1 is the temporary output file. After you complete your edit session, DATA1 is deleted, then \*ATA1 is renamed DATA1. In the event of disk read or write errors during the edit session, both the DATA1 and \*ATA1 files will remain available to you.

### Form 2

The interpretation of FILENAME is based on whether it is a new file or an existing file. If FILENAME is an existing file, FILENAME is edited to itself as in the previous example of EDIT DATA1 DATA1. If FILENAME is a new file, then FILENAME designates the PRIMARY OUTPUT file, and there is no PRIMARY INPUT file. Since there is no PRIMARY INPUT file, you may not input from the default file, so an ALTERNATE INPUT file must be specified if disk reads are requested.

### Form 3

There is no PRIMARY INPUT file and no PRIMARY OUTPUT file. If you desire to input or output data, ALTERNATE INPUT or ALTERNATE OUTPUT files must be specified in the command.

In all cases, the Editor will respond with an identifying message and then present its prompt character, the asterisk \*, to indicate it is ready to accept commands.

You may not start the Editor while a COMMAND FILE is active under SDOS. The EDIT request will be rejected if an attempt is made to do so.

#### NOTE

While the Editor is executing, the special SDOS keys, ESC and SPACE BAR, retain their special functions. Consult Section 4.4 for an explanation of their use.

## 5.2 SAMPLE EDIT SESSION

Let us go through an example of editing. Suppose you have conceived and coded the 2650 program in Figure 5-1 and wish to create a new file, DADDSB, which will contain the source program data. Start the Editor program by typing:

```
>EDIT DADDSB/0 (r)
```

This will load Edit into slave memory and begin execution. The Editor will display:

```
**EDIT VER X.Y**  
**NEW FILE**  
*
```

The last \* is the Text Editor prompt character, which indicates that the Editor is ready to accept commands. Figures 5-2 through 5-7 are hard copy equivalents of the Edit sessions that will be described.

```
*DOUBLE PRECISION ADD. A IN R0, R1. B IN R2,R3  
*ON RETURN, A+B IS IN R2,R3  
*  
DADD  STRR,R1  DAR1  
      ADDR,R3  DAR1  
      PPSL     WC  
      ADDZ     R2  
      STRZ     R2  
      CPSL     WC  
      RETC,Un  
DAR1  RES     1  
*
```

Figure 5-1. EDIT Sample Double Precision Add

The first command entered, line 1 of Figure 5-2, is the TAB command (the Set TAB Character Command). The command, TAB ., sets the period '.' as the TAB character. This gives the '.' a special meaning, that when '.' is entered, the Editor is requested to fill the buffer with spaces until the next TAB stop. This feature will be discussed later.

The Editor has two basic modes.

1. EDIT - any of the EDIT functions may be performed.
2. INPUT - text may be entered, but no edits performed.

If you desire to enter more than one or two lines of data, it is desirable to enter the input mode. Since you desire to enter all of the source program at one time, the input mode should be entered. To enter the input mode, press I and then RETURN (line 2 of Figure 5-2). The Editor acknowledges this command by displaying "INPUT:" to remind you of its mode (line 3 of Figure 5-2). You may then enter the source program (lines 4-14 of Figure 5-2). As can be seen, errors have occurred (lines 9 and 13 of Figure 5-2). To change from the input mode back to the edit mode, enter a null line by pressing RETURN twice in succession (line 15 of Figure 5-2).

```
1 *TAB .
2 *I
3 INPUT:
4 * DOUBLE PRECISION ADD. A IN R0,R1. B IN R2,R3
5 * ON RETURN, A+B IS IN R2,R3
6 *
7 DADD.STRR,R1.DAR1
8 .ADDR.R3.DAR1
9 .PPSL.WD
10 .ADDZ.R2
11 .STRZ.R2
12 .CPSL.WC
13 .RETDMYNN
14 DAR1.RES.1
15
16 *B
17 *TY 55
18 * DOUBLE PRECISION ADD. A IN R0,R1. B IN R2,R3
19 * ON RETURN, A+B IS IN R2,R3
20 *
21 DADD STRR,R1 DAR1
22 ADDR,R3 DAR1
23 PPSL WD
24 ADDZ R2
25 STRZ R2
26 CPSL WC
27 RETDMYNN
28 DAR1 RES 1
29 **EOF**
```

Figure 5-2. Entering Text and Displaying the Buffer

The effect of entering the TAB character can be seen by examining lines 9 and 23 in the display of the buffer (see Figure 5-2). Entering '.' at the start of line 9 resulted in spaces being entered up to the next TAB stop, which is located in column 8. Entering the second '.' as the sixth character in line 9 resulted in spaces being entered up to the next TAB stop, which is located in column 16. The user may change either the TAB character or TAB stops by using the TAB and TABS commands. The default TAB character is CTRL-I and the default TAB stops are 8, 16, 24, 32, 40, 48, 56, and 64.

To view the text that has been entered, it is necessary to move the line pointer to the top line of the buffer. This is accomplished by entering B, the Move Pointer to Beginning of Buffer command (line 16 of Figure 5-2). On line 17 of Figure 5-2, the command to display 55 lines of the buffer is entered (55 is an arbitrarily large number which will allow the entire buffer to be displayed). The Editor displays the buffer (lines 18-28 of Figure 5-2) and then displays **\*\*EOF\*\*** to indicate it has reached the bottom of the buffer. Note that the tabs entered in the input mode are present as spaces in the buffer.

Upon examination of Figure 5-2, it is clear that two changes are necessary to the text currently residing in the buffer. Line 23 should have WD altered to WC and line 27 should have RETDMYNN altered to RETC,UN.

To find these lines, type F (the FIND command), a space, then \$WD\$, where the data between the \$s is the data you wish to find (line 1 of Figure 5-3). In this case, the \$ is the delimiting character, which means that the \$s tell the Editor where the data starts and where the data ends. The Editor finds the first line in the buffer that contains WD, moves the line pointer to the beginning of the line, and displays the line (line 2 of Figure 5-3). To alter the WD to WC, enter S (the SUBSTITUTE command), a space, then \$WD\$WC\$ (line 3 of Figure 5-3). The first \$ says this is the start of the string to be deleted. WD is the string to be deleted. The second \$ is the end of the string to be deleted, and the beginning of the string to substitute for the deleted string. The final \$ indicates the end of the string to substitute.

The Editor performs the substitution and displays the line as altered (line 4 of Figure 5-3). To change RETDMYNN to RETC,UN, find the line by entering F (FIND), space, the \$RET\$ to locate this string (line 5 of Figure 5-3). The Editor prints the line on which it locates RET (line 6 of Figure 5-3). In this case, you want to replace the line with the correct information. This is done by pressing R (the REPLACE command), space, and then entering the information desired, namely .RETC,UN (line 7 of Figure 5-3). This command replaces the current line with the line following the R and space. The Editor displays the replacement line after it has performed the replace function (line 8 of Figure 5-3).

```
1 *F $WD$
2     PPSL      WD
3 *S $WD$WC$
4     PPSL      WD
5 *F $RET$
6     RETDMYNN
7 *R .RET,UN
8     RETC,UN
```

Figure 5-3. FIND, SUBSTITUTE and REPLACE Commands



To insure that the changes were performed correctly, go to the top of the buffer and display its contents (see Figure 5-4).

Since you are satisfied that the buffer contains the correct information, you want to store the information on the disk. This is accomplished using the FILE command (line 15 of Figure 5-4), which writes the contents of the buffer to the PRIMARY OUTPUT file and then transfers the rest of the PRIMARY INPUT file, if one exists, to the PRIMARY OUTPUT file. Following the final transfer, the Editor is exited and SDOS displays its prompt character. In this case, the buffer will be copied to disk file DADDSB/0. There is no input file, so DADDSB/0 will be closed, the Editor will be exited (line 16 of Figure 5-4) and SDOS will display its prompt character (line 18 of Figure 5-4).

```
1 *B
2 *TY 55
3 * DOUBLE PRECISION ADD.  A IN R0,R1.  B IN R2,R3
4 * ON RETURN, A+B IS IN R2,R3
5 *
6 DADD      STRR,R1 DARI
7          ADDR,R3 DARI
8          PPSL   WC
9          ADDZ   R2
10         STRZ   R2
11         CPSL   WC
12         RETC,UN
13 DARI     RES   1
14 ** EOF **
15 *FILE
16 *SLJ* EOJ
17
18 >
```

Figure 5-4. Displaying the Buffer and Filing

Now suppose you wish to expand DADDSB/O to include not only a double precision add, but a double precision subtract, as in Figure 5-5.

```
* DOUBLE PRECISION ADD.  A IN ,R0,R1.  B IN R2,R3
* ON RETURN, A+B IS IN R2,R3
*
DADD      STRR,R1 DAR1
*
          ADDR,R3 DAR1
          PPSL      WC
          ADDZ      R2
          STRZ      R2
          CPSL      WC
          RETC,UN
DAR1      RES      1
* DOUBLE PRECISION SUBTRACT, A IN R2,R3.  B IN R0,R1
* ON RETURN, A-B IS IN R2,R3
*
DSUB      STRR,R0 DSRO
          STRR,R1 DSR1
          SUBR,R3 DSR1
          PPSL      WC
          SUBR,R2 DSRO
          CPSL      WC
          RETC,UN
DSRO      RES      1
DSR1      RES      1
          END      DADD
```

Figure 5-5. EDIT Sample Double Precision Add and Subtract

To edit the additional information into the file DADDSB/0, do the following tasks.

Start the Editor by entering:

```
> EDIT DADDSB/0 (r)
```

While this command is identical to the command entered earlier, it now has a different interpretation. In the first example, DADDSB/0 was a new file.

When a new filename is the sole argument to an EDIT command, the file is treated as the PRIMARY OUTPUT file and there is no PRIMARY INPUT file. This is as it should be, since if you are in the process of creating a new file which will contain unique information, there is no need for a PRIMARY INPUT file. In this case, DADDSB/0 is an existing file which contains the double precision addition routine, so this EDIT command requests that DADDSB/0 be edited to itself.

When the Editor displays its prompt character \* you can proceed. Since the new text is to be appended to the existing text in DADDSB, you must read the existing file into the buffer. This is accomplished by entering G (the GET command), space, and then 20, an arbitrarily large number that will result in DADDSB/0, which we know to be approximately 10 lines long, being read into the buffer. (See line 1 of Figure 5-6). The Editor reads the PRIMARY INPUT file, which is the default filename in the GET command, until it inputs the specified number of lines or until it reaches the end-of-file. In this case, the end-of-file is reached first, so the message \*\*EOF\*\* is displayed (line 2 of Figure 5-6).

```

1 *G 20
2 ** EOF **
3 *B
4 *TY 55
5 * DOUBLE PRECISION ADD. A IN R0, R1. B IN R2,R3
6 * ON RETURN, A+B IS IN R2,R3
7 *
8 DADD STRR,R1 DAR1
9 ADDR,R3 DAR1
10 PPSL WC
11 ADDZ R2
12 STRZ R2
13 CPSL WC
14 RETC,UN
15 DAR1 RES . 1
16 ** EOF **
17 *END
18 ** EOF **
19 *TAB .
20 *I
21 INPUT:
22 * DOUBLE PRECISION SUBTRACT. A IN R2,R3. B IN R0,R1
23 * ON RETURN, A-B IS IN R2,R3
24 *
25 DSUB.STRR,R0.DSRO
26 .STRR,R1.DSR1
27 .SUBR,R3.DSR1
28 .PPSL.WC
29 .SUBR,R2.DSRO
30 .CPSL.WC
31 .RETC,UN
32 DSRO.RES.1
33 DSR1.RES.1
34 .END.DADD
35
36 *B
37 *TY 25
38 * DOUBLE PRECISION ADD. A IN R0,R1. B IN R2,R3
39 * ON RETURN, A+B IS IN R2,R3
40 *
41 DADD STRR,R1 DAR1
42 ADDR,R3 DAR1
43 PPSL WC
44 ADDZ R2
45 STRZ R2
46 CPSL WC
47 RETC,UN
48 DAR1 RES 1
49 * DOUBLE PRECISION SUBTRACT. A IN R2,R3. B IN R0,R1
50 * ON RETURN, A-B IS IN R2,R3
51 *
52 DSUB STRR,R0 DSRO
53 STRR,R1 DSR1
54 SUBR,R3 DSRO
55 PPSL WC
56 SUBR,R3 DSRO
57 CPSL WC
58 RETC,UN
59 DSRO RES 1
60 DSR1 RES 1
61 END 1
62 **EOF**

```

Figure 5-6. Adding Data to an Existing File

Where was the data inserted in the buffer? The answer is that the data was inserted above the line pointer as in the INPUT mode example. To view the buffer, move the pointer to the beginning of the buffer (line 3 of Figure 5-6). Display the buffer by entering TY 55 (line 4 of Figure 5-6). This command displays the Buffer and prints \*\*EOF\*\* to indicate the bottom of the buffer (lines 5-16 of Figure 5-6). Note that \*\*EOF\*\* has two uses, one to indicate the end of the buffer and one to indicate the end of the file.

To enter the double precision subtract routine after the add routine, you must go to the bottom of the buffer to perform the insertion. Do this by entering END. This command moves the line pointer to a location below the last line of text (lines 17 - 18 of Figure 5-6). The TAB character is specified as a . in line 19. Enter the input mode by entering I (line 20 of Figure 5-6). Enter the source data (lines 22 - 35 of Figure 5-6). The effect of the TAB character can be seen in lines 52 to 61 of Figure 5-6, when the entire buffer is displayed by the commands on lines 36 and 37.

Suppose you desired to make the source listing a little more readable. For example, suppose you want to add an \* line between lines 48 and 49 and between lines 59 and 59 of Figure 5-6. To do these tasks, you must first position the line pointer to point to the line that begins with "\* DOUBLE PRECISION SUBTRACT". This can be accomplished by moving the line pointer down the buffer. Enter D (the Move Line Pointer DOWN the Buffer Command), space, 10 (line 1 of the Figure 5-7). This moves the line pointer 10 lines down. The Editor displays the line that the line pointer now points to in line 2 of Figure 5-7. The \* line is desired between the lines "DAR1 RES 1" and the "\* DOUBLE PRECISION SUBTRACT". The INSERT command inserts the line specified above the current line. Therefore, go down the buffer one more line. This is accomplished by entering D (r) , since the default value for the number of lines to move is 1 (line 3 of Figure 5-7). To enter the \* line, enter I (the INSERT line command unless I is immediately followed by a RETURN, in which case the user enters the INPUT mode), space, \* (r) (line 5 Figure 5-7). To enter the second \* line between the two temporary variables, DSRO and DSR1, and the subtract routine, go to the bottom of the buffer. This is accomplished by entering END (line 6 of Figure 5-7). The Editor indicates the line pointer's position at the bottom of the buffer by displaying \*\* EOF \*\* (line 7 of Figure 5-7). Move the pointer buffer to the line where you wish to enter the \* by entering U (the Move Line Pointer UP the buffer command), 3 (r) , (line 8 of Figure 5-7). This command moves the line pointer up three lines and displays the line (line 9 of Figure 5-7). To enter the \* line, enter I (INSERT), space, \* (r) (line 10 of Figure 5-7).

After displaying the buffer (lines 11 - 39 of Figure 5-7) and insuring that the text you desire is present, the data may be stored on file DADDSB/0 by the use of the FILE command (line 40 of Figure 5-7). The contents of the buffer are written to the temporary Primary Output file, \*ADDSB/0. The remainder of the PRIMARY INPUT file, DADDSB/0, is copied to the temporary output file. Since all the data has been read from the temporary input file (lines 1-2 of Figure 5-6, \*\* EOF \*\*), no additional data is written to the temporary Output file. The file DADDSB/0 is deleted and \*ADDSB/0 is renamed DADDSB/0. The Editor exits and SDOS displays its prompt character.

```

1 *D 10
2 DAR1 .RES 1
3 *D
4 * DOUBLE PRECISION SUBTRACT. A IN R2,R3. B IN R0,R1
5 *I *
6 *END
7 ** EOF **
8 *U 3
9 DSRO RES 1
10 *I *
11 *B
12 *TY 55
13 * DOUBLE PRECISION ADD. A IN R0,R1. B IN R2,R3
14 * ON RETURN, A+B IS IN R2,R3
15 *
16 DADD STRR,R1 DAR1
17 ADDR,R3 DAR1
18 PPSL WC
19 ADDZ R2
20 STRZ R2
21 CPSL WC
22 RETC,UN
23 DAR1 REC 1
24 *
25 * DOUBLE PRECISION SUBTRACT. A IN R2,R3. B IN R0,R1
26 * ON RETURN, A-B IS IN R2,R3
27 *
28 DSUB STRR,R0 DSRO
29 STRR,R1 DSR1
30 SUBR,R3 DSR1
31 PPSL WC
32 SUBR,R2 DSRO
33 CPSL WC
34 RETC,UN
35 *
36 DSRO RES 1
37 DSR1 RES 1
38 END DADD
39 ** EOF **
40 *FILE
41 *SLJ* EOJ
42
43 >

```

Figure 5-7. Inserting Lines into the Buffer

### 5.3 EDITOR COMMAND DESCRIPTIONS

This section provides detailed descriptions of all the Text Editor commands. As a prelude to these descriptions, the Editor command line, the conventions and terms used in the descriptions, and certain limitations are described.

#### 5.3.1 Editor Command Line

When the editor presents its prompt character, \*, it is ready to accept commands. All Editor commands are of the form:

command parameterlist

where:

command identifies the particular action desired. The underlined portion denotes the minimum number of characters to identify the command.

parameterlist identifies necessary variables for the command. The parameter list may be null.

#### NOTE

Comma (,) is not a valid parameter delimiter in the Editor.

A command line consists of one or more commands terminated by RETURN. If you desire to specify two or more commands in one command line, the commands must be separated by the command delimiter semicolon (;).

Example:

\*F \$BADLINES\$:K 1 (r)

would find the next line in the buffer with the string BADLINE in it and then delete that line.

A command line may not exceed 128 characters. If the line does exceed 128 characters, \*\*TRUNCATED\*\* is displayed on the console and the entire command line is rejected.



### 5.3.2 Editor Command Description Conventions

There are several conventions employed in the description of the Editor commands, and two features of the Editor that require explanation.

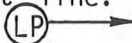
Conventions used in the command descriptions are:

- 1) N refers to two possible entries. These are:
  - a) an absolute number (n)
  - b) a line range (p-q).

For example, KILL N refers to two possible types of command lines, KILL n or KILL p-q. Thus you could KILL the next n lines or KILL lines p through q (inclusive) in the buffer. The default of N is 1 with the exception of:

- a) COPY command
- b) GET and PUT commands when an alternate file is specified.

In these cases, N must be specified. In addition, N may be directly appended to the command when it is used. For example, KILL N may be written as Kn or Kp-q. The arguments n, p and q must be integers in the range 1 to 32,767, inclusive.

- 2) The letters B, E, and C may be used in the "p-q" form of expressing line ranges. They refer to the Beginning, Ending, and Current lines in the workspace. If used, these letters may not be directly appended to the command. A space after the command is required.
- 3) The Editor maintains a line pointer to the line in the buffer that it is currently considering. The line is known as the current line. The line pointer will be designated in this discussion as: 

For example, the buffer appears as follows:

	ADDR,R2	TEMP
	STRR,R2	TEMP2
	ADDI,R2	ERR1

Inserts always occur above the current line.

- 4) \$ is used to represent the delimiting character for a string of text. The delimiter cannot be a space and cannot appear in the string being delimited. '\$' was used in the previous Edit example.
- 5) Parameters that are optional for a command are enclosed in parentheses.

The two features of the TWIN Editor are:

- 1) The Editor will type the line pointed to by the line pointer at the completion of most commands. This feature keeps the user apprised of his position in the buffer. For a complete discussion of how to manipulate this feature, consult the BRIEF command.

2) The Editor has two special command delimiters:

- a) The semicolon: which allows you to stack commands on a command line.
- b) < and > which execute a command line repetitively.

These characters may be entered as text only while in INPUT mode or with the / prefix. The slash / overrides the command delimiter meaning for the current command.

### 5.3.3 Insertion

You may insert source lines into the buffer with the INSERT and INPUT commands.

**INSERT string**

This command will insert the data line STRING before the current line in the buffer. This allows the user to enter single lines into the buffer. The position of the line pointer is not changed.

For example, if the buffer appears as follows:

Ⓕ → ADDR,RO DAR2  
ADDR,R1 DAR3

and the command

\*INSERT STRR,RO DAR5

is entered, the buffer is altered to

Ⓕ → ADDR,RO DAR2  
STRR,RO DAR5  
ADDR,R1 DAR3

If you enter a null input string by depressing Ⓕ after the single delimiting space, the editor enters the INPUT mode, described below.

**INP**UT

The editor may be placed in the INPUT mode by entering:

INPUT Ⓕ

The editor responds with

INPUT:

to indicate that it has entered the INPUT mode.

In the INPUT mode, you may enter any number of text lines. These lines will be entered into the buffer in front of the current line. The INPUT mode is terminated by entering a null line. The position of the line pointer is not changed. For example, if the buffer appears as follows:

Ⓕ → ADDR,RO DAR2  
ADDR,R1 DAR3

and the sequence

```
*INPUT (r)
INPUT:
STRR,R0   DAR5 (r)
ADDR,R2   DAR4 (r)
ADDZ,R2   (r)
(r)
```

is entered, the buffer is altered to

```
ADDR,R0   DAR2
STRR,R0   DAR5
ADDR,R2   DAR4
ADDZ,R2
(LP) → ADDR,R1   DAR3
```

In the INPUT mode, no text line may exceed 128 characters. If more than 128 characters are input before RETURN is pressed, **\*\*TRUNCATED\*\*** is printed on the console, and only the first 128 characters entered are placed in the buffer.

### 5.3.4 Deletion

You may delete lines in the buffer using the KILL command.

**KILL n**

This command has two forms:

- 1) delete the next n lines beginning with current line or
- 2) delete line numbers p through q in the buffer.

If no argument is specified, the current line is deleted.

For example, if the buffer appears as follows:

(LP) → LINE 1  
LINE 2  
LINE 3  
LINE 4  
LINE 5  
RETC,UN

and the following command is performed

\*K 4

the buffer will be changed to

(LP) → LINE 5  
RETC,UN

The command K 1-4 could have been used to produce the same effect.

The KILL command moves the line pointer in the following manner:

- 1) If K n is used, and the line pointer is positioned on line x, the line pointer is repositioned to point at what was line n + x before the deletion took place.

In the above example, K 4 is used as the command, and the line pointer is positioned at line 1 in the buffer. Therefore, the line pointer is repositioned to point at what was the fifth line, LINE 5.

- 2) If K p-q is used, there are two possible positionings of the line pointer.
  - a) If the pointer points at a line between line p and line q, the line pointer will be repositioned at what was line q + 1.
  - b) If the line pointer points at a line that is not between line p and line q, the position of the line pointer is not changed.

### 5.3.5 Alteration

You may alter lines in the buffer through the use of the SUBSTITUTE and REPLACE commands. Both commands operate on the line pointed to by the line pointer.

**SUBSTITUTE \$string1\$string2\$**

The SUBSTITUTE command finds the first occurrence of STRING1 in the current line and replaces STRING1 with STRING2. The \$ delimits the start and end of the string definition. To perform a substitution with a \$ in either STRING1 or STRING2, any other character may be used as the delimiter, but the same character must be used for all three delimiter positions in the command. The final delimiter need not be entered unless the : is used to append another command to this one.

If STRING1 is not found in the current line, **\*\*NOT FOUND\*\*** is displayed on the console.

STRING2 may contain TAB characters (See Section 5.3.9). Conversion of the TAB characters to spaces in the buffer depends on the column in which the substitution occurs. The substitution of spaces for TAB characters is always in accord with the current TAB positions.

The AGAIN command may be used to alter additional occurrences of STRING1 in the current line only. The line pointer is not advanced if the string is **\*\* NOT FOUND \*\***.

If a substitution causes a line to exceed 128 characters, the message **\*\*TRUNCATED\*\*** is displayed on the console and the line is truncated by truncating characters from the text which is being inserted.

Example 1:

If the current line is

```
ADDR,R3   DRSV
```

the command

```
S $DR$DA$
```

would alter the line to

```
ADDA,R3   DRSV
```

Example 2:

If the current line is a 127 character line

```
(63 A's)      (62 C's)
```

```
(LP) → AA.....AABBCC.....CC
```

and the command

```
S $BB$12345$
```

is performed, the message **\*\*TRUNCATED\*\*** is displayed on the console and the current line is altered to 128 characters:

```
      (63 A's)      (62 C's)
AA.....AA123CC.....CC
```

The result of any substitution does not change the position of the line pointer.

Example 3:

In Example 1, `SUBSTITUTE $DR$DA$`, the character '\$' is used as a delimiter. The first '\$' indicates the completion of the first string and the beginning of the string to substitute. The third and final '\$' indicates the completion of the string to substitute.

Suppose, however, this line appeared in the buffer:

```
WINES BY $RIDGE$
```

If you want to replace \$RIDGE\$ with +RIDGE+, you cannot use this command:

```
S $$RIDGE$$+RIDGE+$
```

Since \$ is being used as the delimiter, it may not be inserted in either STRING1 or STRING2. Use another character as the delimiter. Here we choose the slash.

```
S /$RIDGE$/+RIDGE+/  
alters the line to the desired:
```

```
WINES BY +RIDGE+
```

**REPLACE string**

The REPLACE command is used to replace the current line with STRING. STRING may not be a null line.

Example 1:

If the current line is:

```
(LP) → ADDR,R2 DAR1
```



the command:

```
*R STRR,R2 DAR1
```

will result in the current line being altered to:

ⓁP → STRR,R2 DAR1

The position of the line pointer is not altered.

### 5.3.6 String Search

You may search the buffer for a specified string using the FIND command.

**FIND \$string\$**

This command searches the buffer, starting at the current line, for the first line that contains STRING. If STRING is found, the line pointer is repositioned to point to the line in which string occurs. The \$ delimits the start and end of the string to find. To locate a string which itself contains \$, any other character may be used as the start and end delimiter. The final \$ need not be entered unless the : is used to append another command to this one.

If STRING is not found the message \*NOT FOUND\* is displayed, and the line pointer is left unchanged.

If the FIND command is invoked by use of the AGAIN command (see Section 5.3.9), the search starts at the current line plus one.

Example:

If the buffer appears as follows:

```

  (LP) → LINE 1
          LINE 2
          LINE 3
          LINE 4
          LINE 5
```

and the command:

F \$4\$

is executed, the line pointer will be moved to this position:

```

          LINE 1
          LINE 2
          LINE 3
  (LP) → LINE 4
          LINE 5
```

Note that the command

F \$1\$

will display \*NOT FOUND\* on the console since the specified string is in a line above the line pointer.

### 5.3.7 I/O Commands

You may bring information into or send information out of the buffer using the GET, PUT, and LIST commands. You may move data between files using the COPY command.

Before discussing the I/O commands, two concepts require explanation.

- 1) The Editor maintains 'pointers' into the PRIMARY INPUT and PRIMARY (or temporary) OUTPUT files. These pointers indicate the position of the next line to be read from the PRIMARY INPUT file (the PI pointer) and the position of the next line to be written in the OUTPUT file (the PO pointer). Initially, both pointers point to the first line in the respective files.

The PI pointer will only be affected by GET commands that use the default filename option. The PO pointer will only be affected by PUT or COPY commands that use the default filename option.

- 2) When a file is closed, the file is only affected if it was being written. The SDOS buffer containing the file data is written to the file. The end-of-file mark is used by SDOS to determine the logical end of a file. Therefore, any data that existed after the end-of-file mark is no longer considered part of the file. Note that problems will arise if a data file is mistakenly end-filed in the middle of the data file, as all data following the end-of-file mark will be lost permanently.

**GET n (filename)**

The command reads N lines of data into the buffer. FILENAME specifies the file that will be accessed to provide the data. If FILENAME is omitted, data will be input from the PRIMARY INPUT file. The data that is input is inserted above the current line pointer. The position of the line pointer is not changed.

If you specify the PRIMARY INPUT file as FILENAME, the pointer into the PRIMARY INPUT file will not be altered. But, if you specify no FILENAME, the PI pointer is advanced.

The PRIMARY OUTPUT file may not be used as FILENAME.

Example 1:

If the buffer appears as follows:

ⓁP →	PPSL	WC
	RETC,UN	
	DAR1 RES	1

and file A contains the five lines

```
          ADDZ   R2
          STRZ   R2
          CPSL   WC
LAB       RETC, UN
          RES    1
```

performing the command:

```
GET 1-3 A
```

alters the buffer to :

```
          PPSL
          ADDZ   R2
          STRZ   R2
          CPSL   WC
(LP) →   RETC, UN
          DAR1  RES    1
```

Example 2:

If 6 lines have been read from the PRIMARY INPUT file, ASYM, with a GET 6 command, a

```
GET 2
```

command would read the 7th and 8th lines and move the PI pointer to the ninth line.

However, if the command was not GET 2 but:

```
GET 2 ASYM
```

The 1st and 2nd lines would be read into the buffer. The GET 2 ASYM command would not affect the pointer into the file ASYM. Any succeeding GET N command would begin with the 7th line.

<p><b>PUT n (filename)</b> <b>PUTK n (filename)</b></p>
---

These commands write N lines of data from the buffer to an output file. FILENAME specifies the file where the data will be written. FILENAME may not be specified as the PRIMARY INPUT file or the PRIMARY OUTPUT file. If FILENAME is specified, the data will be output to the beginning of the file and the file will be closed when the write is complete. Thus, if FILENAME already contains data, the old data will be lost.

The default FILENAME is the PRIMARY OUTPUT file. The data will be written beginning at the PO pointer and the PO pointer will be moved to the next empty line in the file. This allows you to write the buffer to your output file before GETting the next set of lines to edit for large files.

For PUTK, the lines written to the output file are deleted from the buffer. If PUTK is specified there, are two possibilities for line pointer position:

- 1) The line pointer points to a line which will be deleted. After the PUTK command, the line pointer is repositioned to the line immediately following the deleted text.
- 2) The line pointer points to a line which will not be deleted. In this case, the position of the line pointer is not altered.

Example:

If the buffer appears as follows:

```

  (LP) → LINE 1
        LINE 2
        LINE 3
        LINE 4
        LINE 5
```

and the command

```
*PUTK 2
```

is executed, the second and third lines will be written to the PRIMARY OUTPUT file and deleted from the buffer, leaving the buffer as follows:

```

  (LP) → LINE 1
        LINE 4
        LINE 5
```

**LIST n**

This command lists N lines of data on the line printer. The current line pointer position is not changed. The default value of N is 1.

**COPY n infile (outfile)**

This command copies N lines from INFILE to OUTFILE. If OUTFILE is not specified, the data is copied from INFILE to the PRIMARY OUTPUT file. OUTFILE may not be the PRIMARY INPUT file. You may specify the PRIMARY INPUT file as the INFILE without disturbing the pointer into the PRIMARY INPUT file.

When OUTFILE is specified, the data is copied from INFILE to the beginning of the file and OUTFILE is then closed. Thus if OUTFILE already contains data, the old data will be lost. If the PRIMARY OUTPUT file is used by default, the data is copied from INFILE to the PRIMARY INPUT file beginning at the PO pointer.

The COPY command does not use the buffer to transfer data, and it will not alter the buffer or the current line pointer.

### 5.3.8 Line Pointer Commands

You may alter the position of the line pointer by using the **BEGIN**, **END**, **DOWN**, and **UP** commands. You may print the current line pointer position with the **N** command.

**BEGIN**

This command positions the line pointer at the first line of the buffer.

**END**

This command positions the line pointer past the last line of the buffer. **\*\*EOF\*\*** is displayed on the console to indicate that the line pointer is at the end of the workspace.

**DOWN n**

This command moves the line pointer *n* lines down the buffer. The default value of *n* is 1. If the current line is *x* and *n+x* is greater than the number of lines in the buffer, the effect is the same as the **END** command.

**UP n**

This command moves the line pointer *n* lines up the buffer. The default value of *n* is 1. If the current line is *x* and *1-x* is less than 1, the line pointer is set to point at the first line in the buffer, having the same effect as the **BEGIN** command.

**N**

This command displays on the console the number of the line pointed to by the current line pointer.



### 5.3.9 Utility Commands

You may perform a variety of service or program maintenance functions using the AGAIN, BRIEF, FILE, QUIT, SDOS, TAB, TABS, TYPE, ?, / commands and the iterate command function, m <command> .

#### AGAIN

This command performs the previous 'repeatable' command. Commands that are not repeatable are:

AGAIN  
BRIEF  
FILE  
INPUT  
MACRO  
QUIT  
TAB  
TABS

The SUBSTITUTE command is repeatable within the current line.

#### NOTE

If a non-repeatable command was the last command specified, and the AGAIN command is entered, the AGAIN command will look back to discover the last 'repeatable' command, which will then be performed.

Example:

If the buffer appears as follows,

(LP) → LINE 1  
LINE 2  
LINE 3  
LINE 4  
LINE 5  
LINE 6

and the command

\*K2

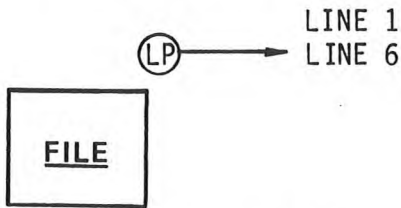
is performed, the buffer is altered to

(LP) → LINE 1  
LINE 4  
LINE 5  
LINE 6

If the next command performed is

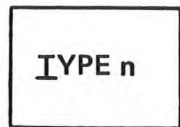
\*A

the buffer is altered to



The FILE command is used to close an edit session. All the data in the buffer is transferred to the PRIMARY OUTPUT file. The data is inserted beginning at the PO pointer, and the PO pointer is then repositioned to the end of the inserted text. The rest of the PRIMARY INPUT file, the portion from the PI pointer to the end of the PRIMARY INPUT file, is then moved to the PRIMARY OUTPUT file beginning at the PO pointer. Both files are then closed.

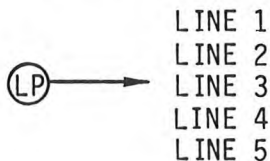
The Edit session is terminated, \*SLJ\* EOJ is displayed on the console, and control returns to SDOS.



This command displays N lines on the console. The current line pointer is left unchanged. If no value is specified for N, the current line is displayed.

Example:

If the buffer appears as follows:



the command

\*TY 2-4

results in the following display on the console

LINE 2  
LINE 3  
LINE 4

## QUIT

The QUIT command is used to abort an edit session and throw away all edits performed in the session thus far. The PRIMARY INPUT and PRIMARY OUTPUT files are closed and then the Edit session is terminated. If the PRIMARY OUTPUT file is a new file, this file is deleted before the editor exits.

\*SLJ\* E0J is displayed on the console, and control returns to SDOS.

## TAB character

This command defines the single character CHARACTER as the tab character. The tab character may not be the :, <, or > characters. The default value of the tab character is CONTROL-I, which is produced by depressing the I key while the CONTROL key is depressed.

Example:

To set the tab character to a different value enter:

```
*TAB C
*I
INPUT*
USING C AS THE TAB CHARACTER IS NOT A GOOD IDEA
r
*B:T
USING AS THE TAB HARA TER IS NOT A GOOD IDEA
```

In this example, all occurrences of the character C have been interpreted as tabs.

## TABS C1 C2 C3...

This command sets the tab positions to the specified columns Cn. When the TAB character is entered from the console, the Editor replaces the TAB character in the buffer with spaces up to the next TAB position. The additional spaces are not reflected on the console at input, but they can be seen if the buffer is listed out.

The default TAB positions are:

8, 16, 24, 32, 40, 48, 56 and 64.

Example:

The default TAB positions produce this result,

```

*TAB C
*I
INPUT:
CHARACTER C IS THE TAB CHARACTER
Ⓡ
*B:T
           HARA   TER   IS THE TAB   HARA   TER

```

The TAB positions could be altered to produce this result,

```

*TABS 1 6 11 16 21 31 36
*I
INPUT:
CHARACTER C IS THE TAB CHARACTER
Ⓡ
*B:T
           HARA   TER   IS THE TAB   HARA TER

```

**m<editor commands>**

This form of the command line causes the commands inside the angle brackets to be repeated M times. If M is omitted, the commands inside the brackets are performed once. The angle brackets must be paired.

Iterated commands may be nested to a depth of 16 levels.

Example:

If the buffer appears as follows:

```

Ⓡ → STRZ      DAR2
      PPSL      WD
      ADDR,R2  DAR1
      ADDR,R3  DAR1
      CPSL      WD

```

the command

**\* 2<F\$WD\$:S \$WD\$WC\$>**

results in the buffer being altered to:

```

Ⓡ → STRZ      DAR2
      PPSL      WC
      ADDR,R2  DAR1
      ADDR,R3  DAR1
      CPSL      WC

```

## SDOS

This command suspends the Editor and returns control to SDOS. The Editor may be continued using the SDOS CONTINUE command.

## ?

Entering a ? character causes the Editor's I/O status to be displayed on the console. The following information is displayed.

PI	= PRIMARY INPUT Filename
LINE	= Next line to "GET" from the PI file
PO	= PRIMARY OUTPUT Filename
LINE	= Next line to "PUT" to the PO file
LAST AI	= Last Alternate Input file name referenced
LAST AO	= Last Alternate Output file name referenced

## /

If the / character is the first character in an EDIT command, the special characters <, >, and : do not perform their usual functions.

Example:

The command

```
F $LEFTANGLE, <$
```

will be rejected because the angle brackets do not balance. However, the command

```
/ F $LEFTANGLE, <$
```

finds the string 'LEFTANGLE, <'.

## BRIEF

The BRIEF command changes the current state of the BRIEF switch. The BRIEF switch is an ON/OFF switch which controls Editor output to the console as follows:

OFF - The line pointed to by the current line point is displayed on the console after completion of the END, UP, DOWN, FIND, SUBSTITUTE, and REPLACE commands.

ON - The line pointed to by the current line pointer is not displayed.

The default switch setting is OFF. Appending a period . to one of the commands listed above when BRIEF is OFF will suppress the display for the duration of that command.

Example: 1

If the buffer appears as follows:

```
ⓁⓅ → LINE 1  
      LINE 2  
      LINE 3
```

and the command:

```
*D 1
```

is performed, the Editor moves the line pointer down the buffer to LINE 2 and displays on the console:

```
LINE 2
```

You may issue a BRIEF command, changing the BRIEF switch to ON. This state will suppress the display of the current line. Now if these commands are entered:

```
*BRIEF  
*D 1
```

the Editor performs the DOWN command to move the line pointer down the buffer to LINE 3 but does not display the line.

Example 2:

In the previous example, assume BRIEF is off, entering

```
*D.1
```

suppress display of the line

```
LINE 2
```

at completion of the DOWN command.

### 5.3.10 Macros

You may define or execute a macro with the MACRO command.

**MACROm=command line**

This is the MACRO definition command. M is an integer which identifies the macro, and must be greater than 0 and less than 128.

COMMANDLINE may be any of the Editor commands discussed previously in this chapter. However, COMMANDLINE must not contain a macro execution or definition command; this will result in an error when the macro is executed.

If a MACROm already exists, and MACROm=COMMANDLINE is performed, COMMANDLINE replaces the old MACROm.

**MACROm**

To execute MACROm, simply enter the macro number as Mm. The effect is equivalent to having entering all of the commands in the previously defined COMMANDLINE.



## 5.4 EDITOR MESSAGES

This section provides a list of all Editor messages and an explanation of their meaning.

### Fatal Errors

<u>MESSAGE</u>	<u>MEANING</u>
** SDOS STAT= XX **	XX is the SDOS SRB status byte returned to the editor when an unusual request or event has occurred. The meaning of the status byte can be found in Appendix G of this manual. The Editor is aborted when the error is on one of the PRIMARY files.

### Non-Fatal Errors

<u>MESSAGE</u>	<u>MEANING</u>
** WSP FULL **	The buffer is full.
** NOT FOUND **	The given string could not be found.
** DISK FULL **	The parameter n is in error.
** RANGE? **	The parameter n is an error or an attempt was made to reference lines which are not in the workspace.
** MODE **	An attempt was made to execute a macro from within a macro string; this is not allowed.
** NEST **	The nesting brackets < and > do not balance.
** COMMAND? **	An unknown command was encountered in the command line.
** BREAK **	The ESCAPE Console Key was depressed to terminate execution of a file I/O function.
** PROCEDURE ERROR **	Editor usage is in error.
** NO PI **	For this editing session there is no PRIMARY INPUT file; the user may not do "GET's" without specifying an Alternate Input file.
** NO PO **	For this editing session there is no PRIMARY OUTPUT file; the user may not do "PUT's" without specifying an Alternate Output File.
** READ FILE? **	An attempt was made to read from a non-existent file or an illegal input device.
** (INPUT) **	The editor response is in reference to an input attempt.
** (OUTPUT) **	The editor response is in reference to an output attempt.

MESSAGE

MEANING

** PI **	The editor response occurred in reference to the Primary or Alternate Input or Output, as applicable.
** PO **	
** AI **	
** AO **	
** NEW FILE **	A new file was created. This is a informational message only.
** (LPT1) **	The editor response occurred in reference to the line printer.
** ASSIGN PROBLEM **	The editor was unable to assign a channel to a given device.
** PI=NEW FILE? **	An attempt was made to "EDIT INFILENAME OUTFILENAME" where INFILENAME and OUTFILENAME were not the same file and INFILENAME was non-existent.
** EOF **	1. An end-of-file was reached on input or output. 2. The line pointer is positioned at the end of the workspace.
** NO FILES SPECIFIED **	The user initiated the editor without specifying any primary files; for this editing session the user may not do "GET's" or "PUT's" without specifying an Alternate file.
** TRUNCATED **	1. A command line exceeded 128 characters and was rejected. 2. An INPUT line exceeded 128 characters and was truncated to the first 128 characters entered. 3. A SUBSTITUTE caused the line to exceed 128 characters and the line was truncated to 128 characters. (See Example in Section 5.3.5)
** NUMBER **	The line number or range entered was in error.
** ERROR **	An error condition not listed above was encountered.



## CHAPTER 6

### THE ABSOLUTE ASSEMBLER

#### 6.0 INTRODUCTION

The Absolute Assembler, ASM, is a system program used to translate 2650 source program code into 2650 object code that is executable by the TWIN system. ASM performs three major tasks:

- 1) It will assemble the user specified source file and generate hex format object code which is written to a user specified object file. Hex format object code is described in Appendix C.
- 2) It will create a listing which includes every assembled source instruction, the instruction address generated for the source instruction, the object code generated for the source instruction, and all assembly errors. This listing is written to a user-specified device or file. The assembler directive PRT may be used to suppress the listing and list only the errors. For details on 2650 assembly language syntax, instruction codes and other related material, consult the TWIN 2650 Assembly Language Manual.
- 3) It will display errors on the console, if not overridden by a command parameter.

#### 6.1 PRE-ASSEMBLY TASKS

The user must insure that two conditions exist before ASM may be used:

- 1) The source program is present on a floppy disk file that is on a currently loaded disk.
- 2) SDOS is ready to accept commands. SDOS presents its prompt character > when it is ready for commands.

## 6.2 THE ASM COMMAND

To execute the Assembler, enter the following SDOS command:

```
ASM{sourcefilename} [listfilename] [objectfilename] [WIDE] [NOERR]
```

where:

**SOURCEFILENAME** is the name of the disk file where the source code resides.

**LISTFILENAME** is the name of the disk file or output device to which the hex format object code is to be written.

**WIDE** the output line is to be 120 print positions wide; default is 72 print positions. The parameter may be abbreviated to W.

Note: If the listing is directed to the TWIN Printer, the N/C (Normal/Compacted) print switch on the printer should be set to the position compatible with the output line width:

N for default,  
C for WIDE

**NOERR** indicates that errors should not be displayed on the console. The parameter may be abbreviated to N.

For example, if the double precision add/subtract subroutine shown in Figure 6-1 were to be assembled, the following tasks need to be performed.

- a) Six EQU assembler directives must be entered into the source file. These EQU's are necessary to define the meaning of R0, R1, R2, R3, UN and WC to the assembler. See the 2650 Assembly Language Manual for details.
- b) The command

```
> ASM DADDSB/0,LPT1,DADOBJ/0
```

will write the hex object code produced on file DADOBJ/0, and produce the listing shown in Figure 6-2 on the line printer.

```

1 * DOUBLE PRECISION ADD.  A IN R0,R1.  B IN R2,R3.
2 * ON RETURN, A+B IS IN R2,R3.
3 *
4 *
5 DADD  STRR,R1 DAR1
6      ADDR,R3 DAR1
7      PPSL  WC
8      ADDZ  R2
9      STRZ  R2
10     CPSL  WC
11     CPSL  WC
12     RETC,UN
13 DAR1  RES  1
14 *
15 * DOUBLE PRECISION SUBTRACT.  A IN R2,R3.  B IN R0,R1
16 * ON RETURN, A-B IS IN R2,R3
17 *
18 DSUB  STRR,R0 DSR0
19      STRR,R1 DSR1
20      SUBR,R3 DSR1
21      PPSL  WC
22      SUBR,R2 DSR0
23      CPSL  WC
24      RETC,UN
25 *
26 DSP0  RES  1
27 DSR1  RES  1
28      END DADD
29      END  DADD

```

Figure 6-1. Sample Program

LINE ADDR OBJECT E SOURCE

```

0001          * DOUBLE PRECISION ADD.  A IN R0,R1.  B IN R2,R3.
0002          * ON RETURN, A+B IS IN R2,R3.
0003          *
0004 0000      R0    EQU    0
0005 0001      R1    EQU    1
0006 0002      R2    EQU    2
0007 0003      R3    EQU    3
0008 0003      UN    EQU    3
0009 0008      WC    EQU    8
0010          *
0011 0000 C90B   DADD   STRR,R1 DARR1
0012 0002 8B09   ADDR,R3 DARR1
0013 0004 7708   PPSL   WC
0014 0006 82     ADDZ   R2
0015 0007 C2     STRZ   R2
0016 0008 7508   CPSL   WC
0017 000A 7508   CPSL   WC
0018 000C 17     RETC,UN
0019 000D      DARR1 RES    1
0020          *
0021          * DOUBLE PRECISION SUBTRACT.  A IN R2,R3.  B IN R0
0022          * ON RETURN, A-B IS IN R2,R3
0023          *
0024 000E C80B   DSUB   STRR,R0 DSR0
0025 0010 C90A   STRR,R1 DSR1
0026 0012 AB08   SUBR,R3 DSR1
0027 0014 7708   PPSL   WC
0028 0016 AA03   SUBR,R2 DSR0
0029 0018 7508   CPSL   WC
0030 001A 17     RETC,UN
0031          *
0032 001B      DSR0 RES    1
0033 001C      DSR1 RES    1
0034 0000      END DADD

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 6-2. Absolute Assembler Output



### 6.3 POST-ASSEMBLY TASKS

When ASM has completed its task, SDOS will display its prompt character > to indicate it is ready for commands. Errors will have been displayed on the console unless the N option was entered, in which case the error display will have been suppressed.

If ASM produced a listing, the listing will contain the two heading lines in Figure 6-2, the assembled source code, and the final line which indicates the number of assembly errors. The columns in the second line of Figure 6-2 have these meanings:

- LINE -- This column contains the number of the assembled source code line. It is provided for the programmer's convenience as an aid when using the Editor to correct source code lines that are in error.
- ADDR -- This column contains the address of the assembly location counter and indicates the address at which the first byte of object code is to be loaded.
- OBJECT -- This column contains the data bytes, (two hex characters per byte) which are to be stored in sequential locations starting with the address in the Address Column.
- E -- This column contains the error codes for the line of source code represented in the Source column. The possible error codes are discussed in section 6.4.
- SOURCE -- This column reproduces the source code as it was read by the Assembler.

## 6.4 ASSEMBLER ERRORS

ASM provides an indication of errors in the source code by printing an alphabetic character in the error field of the listing. For convenience, the erroneous lines of code are also displayed on the console output device, unless the NOERR option is invoked with the ASM command. The error codes and their interpretation are:

- L -- Label Error. The label contains too many characters, contains invalid characters, has been previously defined or is an invalid symbol.
- O -- Op-code Error. The op-code mnemonic has not been recognized as a valid mnemonic.
- R -- Register Field Error. The register field expression could not be evaluated, or when evaluated, was less than 0 or greater than 3, or the register field was not found.
- S -- Syntax Error. The instruction has violated some syntax rule.
- U -- Undefined Symbol. A symbol which appears in the argument field has not been previously defined.
- A -- Argument Error. The argument has been coded in such a way that it cannot be resolved to a unique value.
- P -- Paging Error. A memory access instruction has attempted to address across a page boundary.
- W -- Warning. The Assembler has detected a syntactically correct but unusual construction. The error will be counted but will not inhibit the production of the object module.

In addition, ASM will display the following run-time error messages on the console if it detects an error while trying to execute the ASM command:

### MISSING INPUT FILE PARAMETER

The input file was not specified.  
ASM (r) is not a valid command.

### UNACCEPTABLE INPUT DEVICE

The input file is not on a valid input device.  
ASM LPT1 is not a valid command.

### INPUT FILE ASSIGN ERROR - SRB STAT=XX

An I/O assign file error has occurred on the input file.  
The SRB Status codes are listed in Appendix G of this manual.

When the assembler has completed its analysis of the parameters and has determined that they are acceptable it displays the following message:

ASSEM VER 2.X1

## 6.5 LOADING AN ASSEMBLED PROGRAM

To load an object file assembled by the TWIN assembler, use the following procedure:

- 1) Insure that the hex object file exists on a disk loaded in one of the disk drives and that SDOS is ready to accept commands.
- 2) Read the hex object file by entering the SDOS command

```
>RHEX OBJECTFILE
```

where OBJECTFILE is the name of the file that contains the hex object code.

When the loading process is complete, the SDOS prompt character > will be displayed.

Hex object code programs created on paper tape outside the TWIN (for example, by the 2650 cross-assembler) can be read into slave memory by the RHEX command or to a disk file by using the SDOS COPY command (Section 4.7.6). Note that a CTRL-Z character is required by the COPY command at the end of the tape in order to terminate the COPY and close the file.

A binary load module can then be made from slave memory by using the MODULE command (Section 4.9.3). A binary load module created in this way can be loaded by the LOAD command.

## 6.6 THE ASSEMBLER TAB FEATURE

The TWIN assembler contains a tab feature which is useful in conserving disk space. This is of particular value when large source files are being assembled.

The assembler will interpret the CTRL-I character (ASCII 09<sub>H</sub>) in a source line as a tab character and cause the listing produced by the assembler to tab to the next tab position. These positions are at columns 8, 16, 24, 32, 40, 48, 56, and 64. Disk space is conserved since spaces are then not required for readability.

In order for the CTRL-I character to appear in the source file, the Editor TAB command must be used to define an alternate TAB character. Any CTRL-Is entered will then be passed to the source file. A disadvantage of this technique is that readability of the source file will be poor, since the CTRL-I is a non-printable character.



## CHAPTER 7

### PROM PROGRAMMING

#### 7.0 INTRODUCTION

TWIN provides facilities for programming and creating programming tapes for PROMS. Two types of hardware and software support are provided:

- . PROM programming sockets on the TWIN front panel (see Figure 3-3).
- . A Universal PROM Programming Interface to the DATA I/O Prom Programmer, Models 7 and 9.

#### 7.1 ON BOARD TWIN PROM PROGRAMMING

TWIN hardware provides PROM programming support for the 82S115 bipolar fusible link PROM and the 1702A MOS erasable PROM. These PROMs are programmed via the sockets provided on the TWIN front panel along with the appropriate PROM programming logic cards. Socket usage is as shown in Table 7-1 below.

Table 7-1. PROM Socket Usage

Socket	# Pins	PROM
PROM #1	24	1702A
PROM #2	24	82S115
PROM #3	16	Unused

Some precautions should be taken in using these PROM sockets:

1. PROM power is always OFF whenever inserting or removing PROMs from their sockets. Power to the socket is controlled by the PROM PWR switch on the front panel. The PPWR indicator above the switch is lighted when power is on.
2. PROMs are inserted in the correct sockets. Use of the wrong socket can permanently damage the PROM.
3. PROMs are inserted correctly:
  - a. Push the socket lever up (it should normally be in this position).

- b. Insert the PROM into the socket.
  - c. Push the lever down to clamp the PROM in the socket.
4. Align Pin 1 of the PROM with Pin 1 of the socket, adjacent to the lever.

Three SDOS commands provide the software interface to on-board TWIN PROM programming. They are:

<u>Command</u>	<u>Description</u>
RPROM	Reads contents of a PROM into slave memory.
WPROM	Writes a specified portion of slave memory to a PROM.
CPROM	Compares the contents of slave memory with the contents of a PROM.

## 7.1.1 RPROM

**RPROM [slave address] [PROM name] [start address] [end address] [complement]**

### PURPOSE

The RPROM command reads the contents of the specified PROM from the TWIN front panel into slave memory.

### EXPLANATION

Data is read from the PROM from START ADDRESS through END ADDRESS and stored into slave memory starting at SLAVE ADDRESS. If SLAVE ADDRESS is not specified, the data will be stored beginning at address 0. Default value for START ADDRESS is 0 and for END ADDRESS is 01FF.

Valid PROM NAMES are:

1702 (do not enter 1702A)  
82S115

The default PROM NAME is 1702.

The COMPLEMENT flag specifies whether the data read should be complemented before it is stored in slave memory. The COMPLEMENT flag may have the values:

0 - no complement  
1 - complement data

The default is 0, no complement.

### \*PRM\* Error Responses

7-Device write error  
29-PROM power failure  
30-Invalid parameter  
35-Invalid start address  
36-Invalid end address



## 7.1.2 WPROM

**WPROM [slave address] [PROM name] [start address] [end address] [complement]**

### PURPOSE

The WPROM command programs the specified PROM from data in slave memory.

### EXPLANATION

Data is written to the PROM at START ADDRESS through END ADDRESS from slave memory starting at SLAVE ADDRESS. If no SLAVE ADDRESS is specified, data is read starting at slave memory address 0. If START is not specified, the PROM is programmed beginning at address 0. If END is not specified, the PROM is programmed through address 01FF.

Valid PROM NAMES are:

1702 (do not enter 1702A)  
82S115

The default PROM NAME is 1702.

The COMPLEMENT flag specifies whether the data in slave memory is to be complemented before it is written to the PROM. The COMPLEMENT flag may have the values:

0 - no complement  
1 - complement data

The default is 0, no complement.

After each memory byte is written, the PROM address is read back and compared with the byte written from slave memory. If the bytes are unequal, after several retries at the verify, the PROM address and its contents are displayed in the system console. Table 7-2 gives number of retries and write attempts for each PROM.

Table 7-2. PROM Programming Retries

PROM	# VERIFY RETRIES	# WRITE ATTEMPTS BEFORE NEXT RETRY
1702A	16	5
82S115	8	0

\*PRM\* Error Responses

- 7-Device write error
- 29-PROM power failure
- 30-Invalid parameter
- 35-Invalid start address
- 36-Invalid end address

### 7.1.3 CPROM

**CPROM [slave address] [PROM name] [start address] [end address] [complement]**

#### PURPOSE

The CPROM command compares the contents of the specified PROM with the contents of slave memory.

#### EXPLANATION

Data on the specified PROM from START ADDRESS through END ADDRESS is compared with slave memory starting at SLAVE ADDRESS. If no SLAVE ADDRESS is specified, the compare begins at slave address 0. If START ADDRESS is not specified the PROM is read starting at address 0. If no END ADDRESS is specified, the PROM is read through address 01FF.

Valid PROM NAMES are:

1702 (do not enter 1702A)  
82S115

The COMPLEMENT flag specifies whether the data in slave memory is to be complemented before the compare. The COMPLEMENT flag may have the values:

0 - no complement  
1 - complement data

The default is 0, no complement.

If the data read from the PROM and slave memory are unequal the slave memory location, its contents, and the PROM address contents are displayed on the system console.

#### \*PRM\* Error Responses

7-Device write error  
29-PROM power failure  
30-Invalid parameter  
35-Invalid start address  
36-Invalid end address

## 7.2 UNIVERSAL PROM PROGRAMMING INTERFACE

The Universal PROM Programmer Interface drive is designed for the Data I/O Models 7 and 9 or compatible models to support either the Basic I/O (055-0000) or the Remote Control interfaces (055-0092) that are supplied by Data I/O Corporation as part of the Serial I/O package. The basic difference between the two options lies in the mode of data transfer and in the operator actions required to program, verify or read a PROM device. Refer to the Data I/O Programmer's Instruction Manual for specifics.

### HARDWARE

The interface driver is capable of supporting data transfer operations between the TWIN system and the stand-alone PROM programmer unit, which is equipped with an RS232 I/O port. Hardware requirements are as follows:

#### Data I/O Model 7, 9

1. Serial I/O Interface (950-0045) with either Basic or Remote Control I/O software option. Jumper JP3: Position "B" - No Echo.
2. Data format: Basic I/O - Binary. Translator format specification:  
055-0000  
Remote control I/O - ASCII Hex. Translator format  
specification: 055-0092.
3. Baud Rate: 300 - Standard; 600 - Maximum

#### TWIN System

1. General purpose I/O card on Master side. Switch settings (see TWIN System Reference Manual):  
A4, A6, A7 - ON  
A3, A5 - OFF  
MSTR/SLV - MASTER  
HIGH BAUD/TTY - HIGH BAUD  
  
I/O connector - P2
2. Baud Rate: 300 - Standard; 600 - Maximum with jumper in J29
3. NOTE: When the Basic I/O Interface is used, Pins 35 and 38 of the UART (D5) have to be disconnected from GND and tied to Vcc to achieve a "Binary with no parity" data transmission format.

#### Data I/O Model 19

Configuration: 990-1902, Computer Remote Control  
Software Package: Rev. C (and up)  
Baud Rate: 600

#### TWIN System

Set baud rate on GPIO card to 600 (jumper in J29)

## Interconnection

The full duplex RS-232-C interconnection cable is shown in Figure 7-1.

### NOTE

The RS232 interface as used by DATA I-0 does not conform to the RS232C standard.

## SOFTWARE

SDOS must be configured with the optional RS232 driver. SDOS is shipped with this configuration. However, if you have reconfigured it for some other optional driver, you must run the MAK3ORS procedure before using this PROM programming interface. MAK3ORS is described in Chapter 4 of this manual.

Because the program occupies both overlay areas and data transfers take place via slave memory, no other SDOS jobs can be executed while the PROM command is executing. The first part of the program, in Overlay Area 1, performs general purpose tasks common to every programmer model. The second overlay area, containing the actual driver, is dedicated to the specific tasks for a given programmer model. The software is designed to allow for future support of other standard PROM programming units.

A single general purpose command is used to control the interface between the TWIN and the DATA I/O unit.

<u>Command</u>	<u>Description</u>
PROM	Provides an interface between the TWIN system and the DATA I/O PROM programmer.

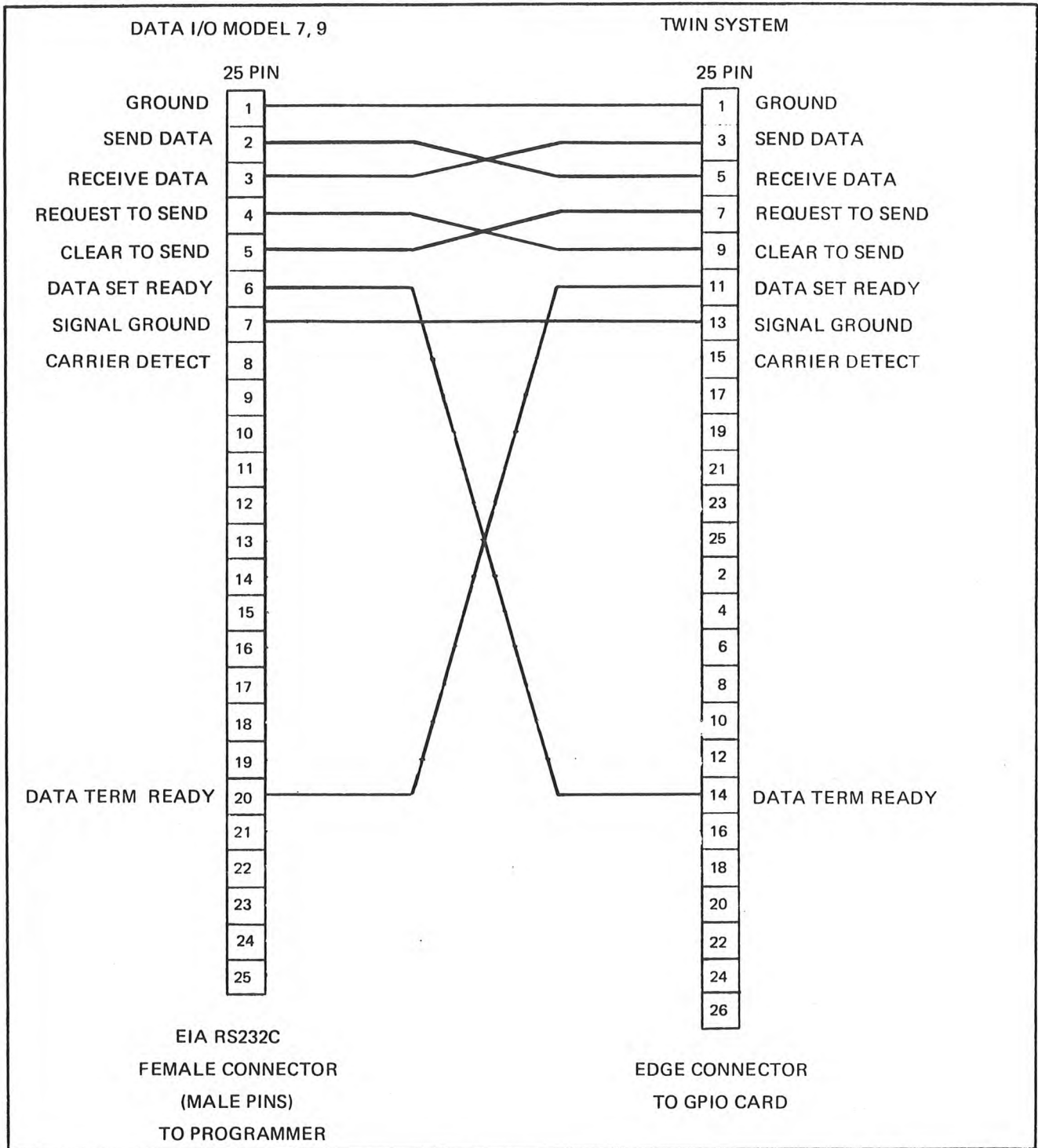


Figure 7-1. TWIN Data I/O Interconnection Cable

## 7.2.1 PROM

**PROM [M] [OP] [A1] [A2] [A3] [N1] [C]**

### PURPOSE

The PROM command provides a software interface between data in slave memory and the DATA I/O PROM programmer.

### EXPLANATION

The parameters are defined as follows:

M specifies which programmer will be used and it must always be given; no default value is established.

M = DB - Data I/O Model 7 and 9 with Basic I/O interface.  
M = DR - Data I/O Model 7 and 9 with Remote Control interface.

OP determines the type of operation to be performed:

OP = R - Read (list)  
OP = W - Write (program)  
OP = V - Verify (compare)

The default value is V, Verify.

A1, A2, and A3 represent address values in Hex format, where:

A1 = first address in slave memory to read data into, program from or compare with. Legal values are from 0-FFFF. Default = 0.

A2 = first address in PROM to read, program or compare. Legal values are 0-3FFF. Default = 0.

A3 = last address of PROM to read, program or compare. Legal values are 0-3FFF. Default = 3FFF.

NI controls which nibble (4 bits) of the data byte in the TWIN common memory is being processed: the lower nibble, higher nibble or both. This parameter is intended to be used with 4-bit PROM's only.

NI = 0, both nibbles (entire byte)  
NI = 1, lower (first) nibble  
NI = 2, upper (second) nibble

The default is 0, both nibbles.



C determines whether the data in PROM or slave memory should be complemented.

C = 0, data is not complemented  
C = 1, data is complemented

The default value is 0, not complemented.

Due to the relatively large number of parameters, their actual value is displayed on the system console if any of them defaults. The operator starts or aborts the program execution in response to the question "EXECUTE?" by typing a character "Y" for yes, or "N" for no, respectively, followed by a carriage return (CR). Note that any input other than Y or N is an error.

Each request for operator assistance is displayed on the system console.

#### Basic I/O (M=DB)

When this software option is installed in the DATA I/O programmer unit, the command line parameters have virtually no control over its operation, therefore, several restrictions apply:

1. PROM starting address (A2) must be 0.
2. The value of the last PROM address (A3) determines the number of data bytes to be transmitted and it always should be less than or equal to the actual PROM size.

#### **CAUTION**

If A3 is greater than the actual PROM size, READ operations will cause the system to hang up. To recover from this condition, enter ESCAPE.

3. The driver is capable of transmitting data in straight binary format between the programmer memory and the TWIN system. All other PROM programmer operations have to be selected manually by the operator at the DATA I/O programmer unit itself.

#### Remote Control I/O (M=DR)

This interface performs all necessary operations between TWIN and programmer automatically, such as load PROM content, blank check, etc. After completing the data transmission in the Write mode (OP=W), you choose either to program a device or terminate the process by typing the character Y or N on the system console (command request "PROGRAM?"), respectively. If the answer is "Y" and the PROM device is programmed successfully, the above command request will be displayed again. This enables you to program several devices without re-entering the command line.

After a task is completed or an "N" is typed for a command request, the programmer will be reset to manual mode.



Since the time it takes to transmit 1K bytes of data is relatively long with a low baud rate, a BELL signal is sent to the system console for each data record indicating proper operation.

#### \*PRM\* Error Responses

- 7-Device write error, Programmer did not accept data
- 30-Invalid parameter,  $N > 2$  or  $C > 1$
- 32-Too many parameters, Number of parameters  $> 7$
- 34-Invalid address, A1, A2 or A3 is not in Hexadecimal form or is more than four characters in length.
- 36-Invalid end address,  $A3 > 3FFF$
- 37-Invalid transfer address,  $A1 + (A3-A2) > 64K$
- 52-Invalid device, M parameter invalid
- 54-Invalid mode, OP parameter invalid
- 62-Device not operational, GPIO is not installed or properly selected

#### Special Messages

Messages requesting operator assistance are self-explanatory.

In case of an unsuccessful "Verify" operation, the following message will be displayed on the system console for Remote Control I/O only, and then processing is aborted:

\*PRM\* ADDRESS = xxxx MEM = xx PROM = xx

Where: ADDRESS = The first memory address that failed  
MEM = Data in slave memory  
PROM = Data in PROM device

When a PROM device programming is unsuccessful the following message is displayed (the PROM content and address can be displayed by performing a "verify" operation):

FAIL \*PRM\* EOJ

```

SAMPLE RUNS

REMOTE CONTROL I/O

> PROM DR W 0 0 3FF
OP=W
A1=0
A2=0
A3=3FF
NI=0
C=0
EXECUTE?
Y
ON DATA I/O: HOLDING DOWN I/O KEY, PRESS EXECUTE
PROGRAM?
Y
PROGRAM?
N
*PRM* EOJ

ASSUME: TWIN COMMON MEMORY AT ADDRESS H'0'=00, PROM=FF

> PROM DR V 0 0 3FF 0 0
ON DATA I/O: HOLDING DOWN I/O KEY, PRESS EXECUTE
*PRM* ADDRESS=0000 MEM=00 PROM=FF
*PRM* EOJ

BASIC I/O

> PROM DB W 0 0 3FF 0 0
ON DATA I/O
SELECT LOAD
HOLDING DOWN I/O KEY, PRESS EXECUTE
ON SYSTEM CONSOLE: RETURN

*PRM* EOJ

> PROM DB R 0 0 3FF 0 0
ON DATA I/O
SELECT PROGRAM
ON SYSTEM CONSOLE: RETURN
ON DATA I/O: HOLDING DOWN I/O KEY, PRESS EXECUTE

*PRM* EOJ

> PROM DB V 0 0 3FF 0 0
ON DATA I/O
SELECT VERIFY
IF VERIFICATION FAILS, "V" INDICATOR ON DATA I/O BLINKS
HOLDING DOWN I/O KEY, PRESS EXECUTE
ON SYSTEM CONSOLE: RETURN

*PRM* EOJ

```

Figure 7-2. Sample Session Universal PROM Programmer

SAMPLE SESSION USING COMPUTER REMOTE MODELS 17 AND 19

PROM DR W  $\emptyset \emptyset$  3FF

OP=W

A1= $\emptyset$

A2= $\emptyset$

A3=3FF

NI= $\emptyset$

C= $\emptyset$

EXECUTE?

Y

ON DATA I/O: SELECT F1, PRESS START  
PROGRAM?

Y

PROGRAM?

N

\*PRM\* EOJ

## CHAPTER 8

### THE DEBUGGER

#### 8.0 INTRODUCTION

The Debugger is a combination of system software and unique hardware features which help the user debug programs in four ways:

- 1) It displays memory and register contents, as well as Debug status, and allows these values to be modified.
- 2) It controls program execution and allows the user to request control at specified locations using breakpoints.
- 3) It traces program execution and displays relevant machine states.
- 4) It allows debugging in the user's prototype system.

To accomplish these functions, the Debugger monitors the user program progress and state and saves necessary information. The monitoring process requires that from time to time, the Debugger must take control of the system. For this reason, user programs will run approximately 14% slower when they are under Debug control.

The Debugger uses breakpoints to control user program execution. A breakpoint is a location in the user program where the user wishes to have the Debugger take control of the system.

The Debugger can do a trace to observe program execution. The entire program or portions can be traced. As each instruction is executed, various parameters that indicate the system state are displayed.

The Debugger is also used to debug user developed hardware. The TWICE cable allows the user to connect the slave CPU hardware directly to the user developed system where in-circuit-emulation, ICE, may be performed.

There are three important facts that require explanation before discussing the Debugger:

- 1) The special SDOS keys, ESC and SPACEBAR retain their meanings while the Debugger is executing. Their use is discussed in Section 4.4. Note in particular the impact of the ESC key on the EXAM command.
- 2) If it is necessary to change the ICE mode for a Debug session, the change must be made before the Debugger is invoked. To change the ICE mode, execute the ICE command, which is described in Section 4.6.3, System Options ICE.

3) Executable programs are stored in two formats:

- a) Hex format. Two hex characters are stored for each byte of object code produced. The Absolute Assembler ASM creates hex format files. RHEX is the SDOS command used to read hex format files into slave memory. Hex format is described in Appendix C.
- b) Binary format. One byte of data is stored for each byte of object code. Both the relocatable assembler, RASM, and the SDOS command, MODULE, create binary format files. LOAD is the SDOS command used to read binary format files into slave memory.

If you are familiar with Debuggers and their commands, Sections 8.1, THE DEBUG PACKAGE; 8.2, THE DEBUG COMMAND; 8.4 and 8.5, DEBUG COMMANDS; and 8.6, the TWICE CABLE are recommended.

If you are not familiar with Debuggers, the above sections plus Section 8.3, SAMPLE DEBUG SESSION, are recommended.

### **8.1 THE DEBUG PACKAGE**

The Debugger is a subsystem of the SDOS system that is enhanced through some TWIN hardware features that allow the Debugger to control slave CPU execution. When the Debugger is executing, the user has a subset of the SDOS commands at this disposal.

When the user invokes the Debugger, a Debug control program is loaded into a Master Memory Overlay Area 1. In addition, a small utility program package which resides in slave memory (see System Description, Software) is used. This package, which is 256H bytes long, is used to save and restore the slave CPU registers when using GO and Breakpoints, and serves as the interface between the Master and Slave CPU's.

After the Debug control program has been loaded, the SDOS prompt character > is issued to the console. Whenever this prompt is displayed, the Debugger is ready to accept commands. The commands available to the Debug user are listed in Table 8-1. Note that several of the primary functions of the Debugger, such as examining and altering memory (the EXAM command) and execution control (the GO and XEQ commands), are SDOS commands. The other SDOS commands are not available when in Debug.

To start a user program while under Debug, load your program into slave memory via the SDOS 'LOAD' or 'RHEX' command. Start the DEBUG package, by entering the DEBUG command. Select the desired DEBUG options, such as Trace or Breakpoint.

Table 8-1. COMMANDS AVAILABLE IN DEBUG

<u>SYSTEM CONTROL*</u>
ABORT
GO
LOAD
XEQ
<u>FILE MAINTENANCE*</u>
ASSIGN
CLOSE
DELETE
<u>SYSTEM OPTIONS*</u>
SYSTEM
<u>BREAKPOINT</u>
BKPT
CLBP
<u>STATUS</u>
DSTAT
STATUS*
TRACE
<u>SYSTEM</u>
RESET
SET
<u>MEMORY*</u>
DUMP
EXAM
PATCH

\* These commands are also available when not under Debug control.

Care must be taken not to overlay the utility program package when LOADING your program.

When the SDOS prompt character is not displayed on the console and you want control, the following procedure should be followed:

- 1) Depress the 'ESC' key twice. If the trace mode is active, a single depression is sufficient.
- 2) When the SDOS prompt character appears, enter the desired commands.
- 3) To continue your program, as after a breakpoint, type 'GO'. The program will continue from the point at which it was interrupted.

When the user program is stopped, the SDOS prompt character is displayed and the system becomes available for input commands, according to the following conditions:

- 1) Console control was requested by depressing the ESC key.
- 2) The user program has encountered a breakpoint.
- 3) The user program has executed a HALT instruction.
- 4) The user program has executed one instruction when in the TRACE STEP mode.
- 5) The user program has reached a normal end of job condition.

The only way for you to terminate the Debugger is to use the SDOS ABORT command. This may be accomplished by entering ABORT DEBUG or ABORT \*. In either case, both DEBUG and the user program are terminated.



## 8.2 THE DEBUG COMMAND

**DEBUG [device]**

This command causes the Debug package to be loaded. DEVICE is the output device or disk file to which the Debug output displays will be written. The DEVICE default value is CONO.

### CAUTION

The format of DEBUG is changed from SDOS 2.0. The DEBUG command now uses only one parameter.

### 8.3 SAMPLE DEBUG SESSION

Let's monitor the sample program shown in Figure 8-1 in order to examine some of the Debug features. The sample program was assembled into hex object code format using the absolute assembler ASM. It was written to a disk file named DEMO, and has a starting address of 3000<sub>H</sub>.

Because the TWIN system is in 2650 slave mode OFF by default, an initial ICE command to set ICE mode is not required.

To load the hex object code from file DEMO into slave memory, enter the SDOS command

```
>RHEX DEMO
```

Note that if the file DEMO contained a binary load module produced from ASM output by the MODULE command or by the relocatable assembler, RASM, we would use LOAD rather than RHEX to load the file.

Now load the Debug package by entering

```
>DEBUG
```

Now the sample program object code resides in slave memory and the Debug control program is located in Master Overlay Area 1 ready to accept Debug commands.

Our sample program uses Registers 0 and 1. If we wish to assign these registers specific values, we use the SET command. To set Register 0 to 0 and Register 1 to 1, enter

```
>SET R0 0 1
```

R0 specifies the register in which to put the first data value of 0. The second data value of 1 is loaded into Register 1. Use the DSTAT command to verify that these values were properly loaded into the registers.

```
>DSTAT  
P=0000 _____ R=00 01 00 00 00 00 00 00  
 1           2           3     4     5     6
```

The one line display immediately below the command provides:

1. Slave CPU program counter at the time the last slave CPU instruction was executed;
2. Breakpoints currently active in the Debugger. Since we have not set any breakpoints, no information is displayed;
3. Register 0 value;
4. Values of Registers 1, 2, and 3 of bank 0;
5. Values of Registers 1, 2, and 3 of bank 1;
6. PSU and PSL values.

TWIN ASSEMBLER VER 2. X1

PAGE 0001

LINE ADDR OBJECT E SOURCE

```
0001 0000      R0  EQU  0
0002 0001      R1  EQU  1
0003           *
0004 0000           ORG  H'3000'
0005 3000 81      TOP  ADDZ  R1      ADD REGISTER 1 TO REG
0006 3001 8401    LOOP ADDI, R0 1      INCREMENT R0
0007 3003 E400           COMI, R0 0     COMPARE R0 WITH 0
0008 3005 987A           BCFR, 0 LOOP  IF COMPARE FAILED, BR
0009 3007 1877           BCTR, 0 TOP    IF COMPARE SUCCEEDED,
0010 3000           END  TOP
```

TOTAL ASSEMBLY ERRORS = 0000

Figure 8-1. Sample Program to Debug

Suppose we wish to trace the execution of DEMO. To enable the trace function and trace all instructions one step at a time, enter the TRACE command specifying the ALL and Step options:

```
> TR A,,S
```

The All mode results in output of a line of trace information to the console for every instruction executed by the slave CPU. The single Step mode returns control to the operator after each slave CPU instruction is executed. Since we wish to trace the entire program rather than instructions in a specific address range, we entered three commas to indicate that those two parameters are missing.

The Debug environment is now defined, and we are ready to execute the program. Because the object code was initially loaded with the RHEX command, the starting address, 3000, must be specified at this time also. To begin program execution, type:

```
> GO 3000
```

After the GO command is executed, the Debugger, now in single step mode, assumes control and produces the trace display below.

```
LOC INST      MNEMON      XR U   OPAD IADD IV EADD R0 R1 R2 R3 R4 R5 R6 PU PL
3000 81        ADDZ,01                01 01 00 00 00 00 00 00 40
```

The trace display headings are defined as follows:

- LOC is the location of the last instruction executed.
- INST is the value of the last instruction executed.
- MNEMON is the instruction mnemonic, including the register or condition code value, if required.
- XR is the index register, if any, for the instruction.
- U If U is +, auto increment indexing is performed for an absolute addressing instruction, or a forward address is calculated for a relative addressing instruction.  
If U is -, auto decrement indexing is performed for an absolute addressing instruction, or a backward address is calculated for a relative addressing instruction.
- OPAD is the operand value or operand address.
- IADD is the indirect address value.
- IV is the index register value.
- EADD is the calculated effective address for the last instruction.

R0 is the value of R0.  
R1,R2,R3 are the values of R1, R2, and R3 in bank 0.  
R4,R5,R6 are the values of R1, R2, and R3 in bank 1.  
PU is the value of the Program Status Word Upper.  
PL is the value of the Program Status Word Lower.

Note that if you are executing this sample session as you read, only the values shown for R0 and R1 will be exactly as shown in this section. This is because we did not specify data for the other registers, and they could contain any random data at this point.

To single step through the next instruction, simply enter the GO command without an address parameter:

```
> G
```

Another line of trace information will be displayed. To continue single stepping in this manner, we must type the GO command after each trace display line.

Suppose, instead, we wish to trace all instructions executed, but without single stepping through each instruction, a very time consuming process. We can change the trace mode by reentering it with new options. To trace all instructions with continuous execution, type

```
> TR A
```

When the next GO command is executed, the Debugger takes control of the slave CPU after every slave CPU instruction is executed, but after the line of trace information is displayed, control is returned back to the slave CPU, and not to us. The resultant output after typing GO is shown in the complete sample session in Figure 8-2.

We can see that the program is properly executing the loop at location 3001, but we are concerned about what will happen when location 3007 is executed. This is accomplished by means of a breakpoint set at location 3007. First, however, the current trace must be cancelled. To cancel the current trace, press the ESCAPE key, then procede to set the breakpoint at location 3007 using the BKPT command.

```
> G 3000
.
.
.
3001 8401 A (ESCAPE key depressed to cancel trace)
>> BKPT 3007
```

Breakpoints are used to control execution by commanding the Debugger to take control whenever the specified address is referenced.

```

> RHEX DEMO
*RHX* EOJ

> DEBUG

> SET RO 0 1

> DSTAT
P=3000                      R=00 01 77 00 D5 55 75 00 40

> TR A,,,S

> GO 3000
LOC INST  MNEMON  XR U  OPAD IADD IV EADD R0 R1 R2 R3 R4 R5 R6 PU PL
3000 81    ADDZ,01
> G
3001 8401  ADDI,00      01                02 01 77 00 D5 55 75 00 40
> G
3003 E400  COMI,00      00                02 01 77 00 D5 55 75 00 40
> G
3005 987A  BCFR,00    - 3001          =3001 02 01 77 00 D5 55 75 00 40

> TR A

> G 3000
3000 81    ADDZ,01                03 01 77 00 D5 55 75 00 40
3001 8401  ADDI,00      01                04 01 77 00 D5 55 75 00 40
3003 E400  COMI,00      00                04 01 77 00 D5 55 75 00 40
3005 987A  BCFR,00    - 3001          =3001 04 01 77 00 D5 55 75 00 40
3001 8401  ADDI,00      01                05 01 77 00 D5 55 75 00 40
3003 E400  COMI,00      00                05 01 77 00 D5 55 75 00 40
3005 987A  BCFR,00    - 3001          =3001 05 01 77 00 D5 55 75 00 40
3001 8401  ADDI,00      01                06 01 77 00 D5 55 75 00 40
3003 E400  COMI,00      00                06 01 77 00 D5 55 75 00 40
3005 987A  BCFR,00    - 3001          =3001 06 01 77 00 D5 55 75 00 40
3001 8401  ADDI,00      01

>>BKPT 3007

> TRACE OFF

> G
3007 1877  BCTR,00    - 3000          =3000 00 01 77 00 D5 55 75 00 21
3007 BREAK

> SET R1 FA

> DSTAT
P=3007  BP=3007 WR          R=00 FA 77 00 D5 55 75 00 21

> TRA J

> G 3000
3005 987A  BCFR,00    - 3001          =3001 FB FA 77 00 D5 55 75 00 80
3005 987A  BCFR,00    - 3001          =3001 FC FA 77 00 D5 55 75 00 80
3005 987A  BCFR,00    - 3001          =3001 FD FA 77 00 D5 55 75 00 80
3005 987A  BCFR,00    - 3001          =3001 FE FA 77 00 D5 55 75 00 80
3005 987A  BCFR,00    - 3001          =3001 FF FA 77 00 D5 55 75 00 80
3005 987A  BCFR,00    - 3001          =3001 00 FA 77 00 D5 55 75 00 21
3007 1877  BCTR,00    - 3000          =3000 00 FA 77 00 D5 55 75 00 21
3007 BREAK

> DSTAT
P=3007  BP=3007 WR          R=00 FA 77 00 D5 55 75 00 21

> CLBP

> DSTAT
P=3007                      R=00 FA 77 00 D5 55 75 00 21

> ABORT DEBUG
>

```

Figure 8-2. Sample Debug Session

We will also turn off the trace before resuming program execution, since we would have to wait for a larger number of trace lines to be displayed before 3007 is executed. Execution is resumed from the point at which it was interrupted by the ESCAPE key above.

```
> TRACE OFF
> G
3007 1877 BCTR,00 - 3000 =3000 00 01 00 00 00 00 00 00 21
3007 BREAK
>
```

The Debugger monitors the slave program execution, and when the instruction at 3007 is executed, a line of trace information is displayed and a breakpoint message is output to indicate that the breakpoint was encountered. Note that the effective address to which control will be transferred, EADD, is 3000. The prompt character indicates that control has been returned to us.

Now we wish to monitor the execution of all branch instructions. First, however, let's set Register 1 to FA to cut down the number of times the loop will execute, and verify that Register 1 is correct. Then, change the trace option to specify trace of jump instructions only. The proper sequence of commands to accomplish these tasks is

```
> SET R1 FA
> DSTAT
P=3008 BP=3007 WR R=00 FA 00 00 00 00 00 00 21
> TRA J
```

This time, DSTAT contains information about active breakpoints. BP=3007 specifies that a breakpoint is set at address 3007, and the WR indicates that either a write or a read to location 3007 will cause a break.

When the slave program is restarted with the GO command, the Debugger displays the trace information for all branch instructions executed. The generated trace output is shown in Figure 8-2. Again, execution stops when the breakpoint at 3007 is encountered; the BREAK message informs us of this:

```
> G 3000
:
:
:
3007 BREAK
```

To clear a breakpoint, enter the CLBP command:

```
> CLBP 3007
> DSTAT
P=3007 R=00 FA 00 00 00 00 00 00 21
```

We are finished with our sample Debug session. The Debugger must be exited using the SDOS command ABORT:

> ABORT DEBUG



## 8.4 DEBUG SDOS COMMANDS

This section lists commands that are used with the Debugger. Eight commands are primarily used with the Debugger, but may be used under SDOS. These commands are:

<u>COMMAND NAME</u>	<u>DESCRIPTION</u>
GO	Is used to start user programs.
LOAD	Is used to read binary load files into slave memory.
XEQ	Combination LOAD and GO.
DUMP	Displays contents of slave memory.
EXAM	Examines and/or alters slave memory.
PATCH	Alters slave memory.
STATUS	Display status of slave CPU, I/O and job being executed by it.

#### 8.4.1 GO

**GO [address]**

##### PURPOSE

The GO command causes control to be passed to ADDRESS in slave memory.

##### EXPLANATION

This command causes execution of a user program to begin or resume after a breakpoint occurred.

If ADDRESS is present, control is passed directly to that location in the slave memory. If ADDRESS is not present, either control is passed to the start address of a previously LOADED module or execution continues from the last point stopped in the Debugger.

##### \*DOS\* Error Responses

37-Invalid go address

## 8.4.2 LOAD

```
LOAD {filename [/disk drive]} [parameter 1,...]
```

### PURPOSE

The LOAD command loads the binary load module FILENAME into the slave memory.

### EXPLANATION

FILENAME will be loaded into the slave memory starting at the location specified at the time the load module was created. The program may be executed simply by typing GO. LOAD may only be executed with ICE mode OFF.

The PARAMETER fields may be used when your slave program calls for input data to be entered in a command line. Parameters entered in this manner can be retrieved with the SVC Get Parameter.

The binary load module must have been created by the MODULE command or the relocatable assembler RASM and link editor LINK. RASM and LINK are described in a separate manual named TWIN 2650 Relocatable Assembler Manual, publication number TW09007000.

### \*DOS\* Error Responses

- 6-Device read error
- 14-Invalid input device
- 48-Load file not found
- 49-Load file assign failure or missing file name
- 50-File not a load module
- 51-Invalid load request: a) ICE mode is on  
b) Slave job is executing

### 8.4.3 XEQ

**XEQ**{file name [/disk drive]} [parameter 1,...]

#### PURPOSE

The XEQ command causes a binary load module to be loaded into slave memory and executed.

#### EXPLANATION

The XEQ command is equivalent to the two commands LOAD FILENAME followed by GO.

FILENAME must have been created by either the MODULE command, or the relocatable assembler, RASM.

#### \*DOS\* Error Responses

- 6-Device read error
- 14-Invalid input device
- 48-Load file not found
- 49-Load file assign failure
- 50-File not a load module
- 51-Invalid load request

## 8.4.4 DUMP

**DUMP {A1} [A2] [device]**

### PURPOSE

The DUMP command copies the contents of the specified locations to the system console or other device.

### EXPLANATION

The DUMP command causes the contents of slave memory to be displayed on DEVICE, beginning with address A1. The display consists of two hexadecimal characters representing the contents of each byte displayed. If A2 is not specified, then only 16 bytes of data are displayed. If DEVICE is not specified, the data will be displayed on the system console.

Addresses A1 and A2, if specified, are adjusted in the following manner. The low order hexadecimal character is replaced with 0. For example, 3F3E is altered to 3F30. Then, A2 is replaced by A2 + 10<sub>H</sub>. This has the effect of lowering A1 to the next lowest multiple of 10<sub>H</sub> and raising A2 to the next highest multiple of 10<sub>H</sub>. The contents of memory from adjusted A1 to A2 is then displayed.

If ICE mode 2 is selected, user prototype memory is dumped.

Example:

To dump the contents of slave memory locations 3F3E through 4001, enter the command

```
> DUMP 3F3E 4001
```

The data from memory locations 3F30 through 4010 is displayed as shown:

```
-----
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
-----
3F30=FF FF 00 FF FF FF FF FF FF FF FF FF FF 00 FF FF FF .....
3F40=FF FF FF FF FF FF 00 FF FF FF FF FF FF FF FF FF .....
3F50=00 FF FF FF FF FF FF FF FF FF FF FF 00 FF FF FF FF .....
3F60=FF FF FF FF 00 FF FF FF FF FF FF FF FF FF FF 00 FF .....
3F70=FF FF FF FF FF FF FF FF 00 FF FF FF FF FF FF FF .....
3F80=00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
*4000=00 FF FF FF FF FF FF FF FF FF 00 FF FF FF FF FF .....
-----
>
```

\*DMP\* Error Responses

- 07 - Device write error
- 17 - Output device assign failure
- 31 - Parameter required
- 35 - Invalid starting address (A1)
- 36 - Invalid ending address (A2)

## 8.4.5 EXAM

**EXAM {address}**

### PURPOSE

The EXAM command displays the data byte at the specified address and allows the data to be altered.

### EXPLANATION

The EXAM command causes the contents of the slave memory location ADDRESS to be displayed on the console. You then have several options. You may

- a) display the next sequential byte;
- b) display the current location and its contents;
- c) replace the current memory byte with entered data and display the next sequential memory byte;
- d) terminate the EXAM command.

After the initial memory byte is displayed, press any of these keys to initiate the corresponding function.

<u>Key</u>	<u>Resultant EXAM Action</u>
SPACE	Display the next memory location.
LINEFEED or RUBOUT	Go to the next line and then display the current location and its associated data byte.
HEX DATA PAIR	Replace the current memory location with the hex data pair, then display the next sequential byte.
RETURN	Terminate the EXAM command, leaving any altered memory locations in the altered state.
ESC	Same as RETURN.

The display of memory bytes will automatically go to the next line and display the location and its data byte whenever the location to be displayed is a multiple of 10H.

If ICE mode 2 is selected, user prototype memory is examined and altered.

## Examples:

If locations 3000-3003 contained 00, 01, 02, 03 respectively, the EXAM command could be used as follows. User interaction is underlined.

```
> EXAM 3000  
3000=00_01_02_03 (r)  
>
```

When the space bar was entered, the next sequential byte was displayed. When return was entered, the command was terminated.

To increment each location, this sequence could be used:

```
> EXAM 3000  
3000=00-01 01-02 02-03 03-04 (r)  
>
```

The dash is provided by the EXAM command when a hex data pair is entered.

## \*EXM\* Error Responses

- 31-Parameter required
- 35-Invalid start address
- 39-Invalid hex character
- 59-Memory write error
- 68-Attempt to write past top of available memory



## 8.4.6 PATCH

```
PATCH {address} {hex-string}
```

### PURPOSE

The PATCH command alters slave memory with the specified hexadecimal data.

### EXPLANATION

The PATCH command allows you to alter slave or user prototype memory. ADDRESS is a hexadecimal address constant. HEX-STRING is a string of hexadecimal digits from 2 to 72 digits in length.

The contents of slave memory starting at ADDRESS is replaced with the value HEX-STRING. This replacement is performed on a byte-to-byte basis.

If ICE mode 2 is selected, user prototype memory is patched.

Example:

To patch three slave memory locations starting at address 3000, enter

```
>PATCH 3000 3F001E
```

The data at address 3000 is replaced with 3F, 3001 with 00 and 3002 with 1E.

\*PAT\* Error Responses

- 31-Parameter required
- 34-Invalid address
- 39-Invalid hex character
- 59-Memory write error
- 68-Attempt to write past top of available memory



## 8.5 DEBUG COMMANDS

Six commands are unique to the Debugger and can only be used after the DEBUG command has been executed. These commands are:

<u>Command Name</u>	<u>Description</u>
BKPT	Sets hardware breakpoints.
CLBP	Clears breakpoints.
RESET	Resets the slave CPU.
SET	Sets the slave CPU registers and PSW.
DSTAT	Displays slave CPU Debugger status.
TRACE	Allows for trace of slave CPU execution.

All errors reported by these commands are tagged as \*DEB\* errors.

## 8.5.1 BKPT

**BKPT {address} [option]**

### PURPOSE

The BKPT command causes a hardware breakpoint to be set for the slave CPU at ADDRESS.

### EXPLANATION

The BKPT command suspends program execution after a read or write operation is performed at the specified ADDRESS.

OPTION may either WRITE or READ. If WRITE is specified the break occurs only when there is an attempt to write to the specified address. If READ is specified, the break occurs only when there is an attempt to read the specified address. The default OPTION causes the break to occur whenever a read or write to the specified address is attempted.

When the breakpoint ADDRESS is accessed during program execution, a trace line is displayed on the debug output device, followed by a breakpoint message on the system console.

Up to two breakpoints may be active in the system. However, a breakpoint address identical to one already entered may be set; for example, to change the OPTION.

### LOCAL Error Response

TOO MANY BREAKPOINTS - Two breakpoints are already active.

### \*DEB\* Error Responses

30-Invalid Parameter  
31-Address required  
32-Too many parameters  
34-Invalid address

## 8.5.2 CLBP

**CLBP [address]**

### PURPOSE

The CLBP command clears a breakpoint set by the BKPT command.

### EXPLANATION

The CLBP command is used to clear breakpoints set at specified addresses in slave memory.

If ADDRESS is specified, the breakpoint at the specified address is cleared. If ADDRESS is not specified, all breakpoints are cleared.

### LOCAL Error Response

BREAK POINT NOT ACTIVE - The specified ADDRESS is not an active break point address

### \*DEB\* Error Responses

32-Too many parameters  
34-Invalid address

### 8.5.3 RESET

**RESET**

#### PURPOSE

The RESET command allows the slave CPU hardware to be set to a known beginning state.

#### EXPLANATION

The RESET command causes a RESET pulse to be applied to the slave processor. A subsequent GO command causes execution to begin at address 0 if no address is specified.

The RESET command has no immediate visible effect.

#### \*DEB\* Error Responses

None

## 8.5.4 SET

<u>SET</u> { Rm D1 [...Di] } PSU D1 [D2] PSL D
--

### PURPOSE

The SET command allows you to reassign hexadecimal values to the slave CPU registers or program status word.

### EXPLANATION

The SET command causes the specified slave CPU registers to be set to the hexadecimal data constants  $D_j$ . The  $D_j$  must be hexadecimal numbers in the range 0 to FF.

SET Rm D<sub>1</sub>... causes the slave CPU general registers beginning with register Rm to be set to the values specified. m is an integer 0 through 6, so that R1 identifies register 0, etc. Rm is set to the first data constant D1, Rm+1 is set to D2, and so forth. Only the registers for which values are specified are changed.

SET PSU D1 causes the PSU of the slave CPU to be set to D1. Set PSU D1 D2 causes the PSU to be set to D1 and the PSL to be set to D2.

Set PSL D causes the PSL of the slave CPU to be set to the value D.

The registers of the slave CPU are designated by the following notation. PSU and PSL may not be referred to as R7 and R8.

<u>Rm</u>	<u>Register</u>
R0	Register 0
R1	Bank 0 Register 1
R2	Bank 0 Register 2
R3	Bank 0 Register 3
R4	Bank 1 Register 1
R5	Bank 1 Register 2
R6	Bank 1 Register 3
PSU	Program Status Upper
PSL	Program Status Lower

If an error is detected anywhere in the command line, none of the parameters are processed.

Examples:

> SET R2 4F 23 51

sets Register 2 in Bank 0 to the value 4F, Register 3 in Bank 0 to the value 23, and Register 1 of Bank 1 to the value 51.

It is also possible to set the PSU and PSL in this manner:

> SET R6 FF A0 B0

sets Register 3 of bank 1 to FF, the PSU to A0, and the PSL to B0.

\*DEB\* Error Responses

- 30-Invalid register parameter or parameter missing
- 32-Too many parameters
- 43-Invalid data parameter
- 58-System failure in accessing slave CPU



## 8.5.5 DSTAT



### PURPOSE

The DSTAT command displays the current status of the debugging session.

### EXPLANATION

The DSTAT command causes the Debug status to be displayed in a single line on the Debug output device. The slave CPU's last instruction address, the active breakpoints, and the slave CPU's register contents are displayed. The format of the DSTAT display is as follows:

> DSTAT

```
P=OBOA  BP=0900  WR  OA00  WR  R=FF  00 00 02  04 05 06  00 08  
  1      2      3    2    3    4      5      6      7
```

- 1 gives the location of the last instruction executed by the slave CPU.
- 2 gives the address of the active breakpoints.
- 3 informs the user what conditions are necessary for the break to occur. Note that up to two breakpoint addresses and conditions may be listed.

If a W is present, a break will occur every time a write is attempted to the associated location.

If an R is present, a break will occur every time a read is attempted to the associated location.

If a WR is present, a break will occur every time a read or write is attempted to the associated location.

- 4 gives the contents of R0.
- 5 gives the contents of Registers 1, 2 and 3 in Bank 0.
- 6 gives the contents of Registers 1, 2 and 3 in Bank 1.
- 7 gives the contents of the Program Status Word Upper.
- 8 gives the contents of the Program Status Word Lower.

\*DEB\* Error Responses

- 7-Device write error
- 58-System failure in accessing slave CPU

## 8.5.6 TRACE

<code>TRACE {   OFF   ALL [A1 A2] [STEP]   JMP [A1 A2] [STEP] }</code>
--

### PURPOSE

The TRACE command allows you to monitor program execution.

### EXPLANATION

The TRACE command determines the Debug trace mode. TRACE options have the following meaning:

- OFF - The trace is turned off. No trace information is displayed
- ALL - Trace information is displayed for all instructions executed by the slave CPU. ALL is the default TRACE option.
- JMP - Trace information is displayed for branch instructions whether or not the branch is taken.

The trace display is printed on the debug output device as specified by the DEBUG command, or to CONO if none was specified.

When A1 and A2 are specified, the TRACE function will be performed as specified by the option, but trace information is displayed only for the instructions executed between A1 and A2. A1 and A2 are hexadecimal address constants in the range 0 - FFFF. A2 must be equal to or larger than A1. The default value for A1 is 0. The default value for A2 is FFFF.

The STEP option specifies single step mode. After each trace line is displayed, control is returned to the system console, and the GO command must be entered to execute the next instruction. Note that STEP does not have meaning with the OFF option.

### CAUTION

The STEP option has been moved to the last parameter of the TRACE command. To specify STEP with no address parameters, enter three commas (,,) to denote the missing A1 and A2 parameters.

The format of the trace display is shown in Table 8-2. All values are displayed in hex.

\*DEB\* Error Responses

- 31-Parameter required
- 35-Invalid start address
- 36-Invalid end address or  $A1 > A2$
- 44-Invalid trace mode parameter

Table 8-2. TRACE Display Format

LOC INST MNEMON XR U OPAD IADD IV EADD R0 R1 R2 R3 R4 R5 R6 PU PL

where:

LOC	gives the location of the last instruction executed.
INST	gives the value of the instruction executed.
MNEMON	gives the instruction mnemonic including the register or condition code value, if required.
XR	is the index register, if any, for the instruction.
U	If U is a +, auto increment indexing is used for absolute addressing instructions, OR, a forward address is calculated for a relative addressing instruction. If U is a -, auto decrement indexing is used for absolute addressing instructions, OR, a backward address is calculated for a relative addressing instruction.
OPAD	gives the value or address of the operand.
IADD	is the indirect address value.
IV	is the index register value.
EADD	gives the effective address that has been calculated for this instruction.
R0	gives the value of Register 0.
R1	gives the value of Register 1 in Bank 0.
R2	gives the value of Register 2 in Bank 0.
R3	gives the value of Register 3 in Bank 0.
R4	gives the value of Register 1 in Bank 1.
R5	gives the value of Register 2 in Bank 1.
R6	gives the value of Register 3 in Bank 1.
PU	gives the value of the Program Status Word Upper.
PL	gives the value of the Program Status Word Lower.

## 8.6 TWICE DEBUG CABLE

The TWICE debug cable is used to connect the slave CPU board to the user's system. This will allow the TWIN's slave CPU to operate the user prototype system.

The TWICE cable contains an in-line printed circuit assembly which provides isolation for the TWIN system from the user system. The cable is approximately 10 feet long and has two connectors on one end (this end is attached to the slave CPU board) and a 40-pin plug on the other end (which is inserted into the user system). Refer to Section 3.1.5 for detailed installation instructions.

The cable may remain installed even though not in use as long as care is taken not to short out the 40-pin plug. A 1 amp fuse on the slave CPU board protects the +5V power to the TWICE cable.

The ICE command controls what signals are passed over the TWICE cable to the user prototype system.

APPENDIX A  
SDOS AND DEBUG COMMAND SUMMARY

The short form required to invoke the command is underlined.

Square brackets denote optional parameters. Braces denote available choices for required parameters.

	<u>Page</u>
<u>ABORT</u> { <u>program name</u> } { * / }	4-21
<u>ASSIGN</u> {channel number} { <u>device name</u> file name [/disk drive] } [channel number { <u>device name</u> file name [/disk drive] } ]...	4-26
<u>ASM</u> {sourcefilename} [listfilename] [objectfilename] [ <u>WIDE</u> ] [ <u>NOERR</u> ]	6-2
<u>BKPT</u> {address} [option]	8-24
<u>CLBP</u> [address]	8-25
<u>CLOSE</u> {channel number} [channel number]...	4-27
<u>CMPT</u> {file name 1 [/disk drive]} {file name 2 [/disk drive]} [output device name [output file name [/disk drive]]] [mode]	4-40
<u>CONT</u> { <u>program name</u> } { * / }	4-20
<u>COPY</u> {input device name input file name [/disk drive]} [input device name input file name [/disk drive]] ... {output device name output name [/disk drive]}	4-37
<u>CROM</u> [slave address] [PROM name] [start address] [end address] [complement]	7-6
<u>CSMS</u> [address] [file name [/disk drive]] [device]	4-60
<u>DEBUG</u> [device]	8-5
<u>DELETE</u> {file name/disk drive} [file name/disk drive]...	4-39
<u>DEVICE</u> {device name} { <u>U</u> } { <u>D</u> }	4-24
<u>DFIL</u> {file name [/disk drive]} [output device name [output file name [/disk drive]]] [start byte] [end byte]	4-43
<u>DSTAT</u>	8-29
<u>DUMP</u> {A1} {A2} [device]	8-17
<u>DUP</u> {disk drive 1} {disk drive 2} [diskette identifier]	4-32
<u>EDIT</u> [infilename] [outfilename]	5-2

	<u>Page</u>
<u>EXAM</u> {address}	8-19
<u>FILL</u> {start address} {end address} {hex-string}	4-48
<u>FORMAT</u> {disk drive} [ident]	4-29
<u>GO</u> [address]	8-14
<u>ICE</u> {mode}	4-25
<u>KILL</u> { <sup>ON</sup> OFF}	4-65
<u>LDIR</u> [disk drive] [.] [/] [device name file name/disk drive]	4-34
<u>LOAD</u> {filename [/disk drive]} [parameter 1,...]	8-15
<u>MODULE</u> {file name [/disk drive]} {address 1} {address 2} {address 3} [module identifier]	4-58
<u>MOVE</u> {source-destination} {start source address} {end source address} {destination address}	4-47
<u>PATCH</u> {address} {hex-string}	8-21
{ <u>PRINT</u> <u>PRINTL</u> } {file name [/disk drive]} [device file name [/disk drive]] [begin line number} {end line number} end line number	4-38
<u>PROM</u> [M] [OP] [A1] [A2] [A3] [N1] [C]	7-10
<u>READ</u> {memory address 2650 read instruction type} [output option]	4-49
<u>RENAME</u> {old file name/disk drive} {new file name} or <u>RENAME</u> {disk drive} {diskette identifier}	4-35
<u>RESET</u>	8-26
<u>RHEX</u> [/bias amount] [device file name [/disk drive]]	4-57
<u>RPROM</u> [slave address] [PROM name] [start address] [end address] [complement]	7-3
<u>SET</u> { <sup>Rm D1 [...Di]</sup> PSU D1 [D2] PSL D	8-27
<u>STATUS</u>	8-22
<u>SUSPEND</u> { <sup>*</sup> / program name	4-19
<u>SYSTEM</u> {drive number}	4-23



	<u>Page</u>
<u>TRACE</u> { <u>OFF</u> <u>ALL</u> [A1 A2] [STEP] <u>JMP</u> [A1 A2] [STEP] }	8-31
<u>TYPE</u> { <u>ON</u> <u>OFF</u> }	4-66
<u>UPR</u> [address]	4-54
<u>VERIFY</u> {disk drive}	4-31
<u>WHEX</u> {address 1} {address 2} ,, [ {address 1} {address 2} ] ..., [ {address 3} { <sup>device</sup> file name [/disk drive] } ]	4-56
<u>WPROM</u> [slave address] [PROM name] [start address] [end address] [complement]	7-4
<u>WRITE</u> { <sup>memory address</sup> 2650 write instruction type } [option] [hex-string]	4-51
<u>WSMS</u> [address] [ <sup>file name [/disk drive]</sup> device ]	4-59
<u>XEQ</u> {filename [/disk drive]} [parameter 1,...]	8-16
* (The Asterisk) [comment]	4-64

D

) O

O

O

) O

O

O

O

O

**APPENDIX B**  
**TEXT EDITOR COMMAND SUMMARY**

The short form required to invoke a command is underlined.

Parentheses denote optional parameters.

<u>COMMAND</u>	<u>PAGE</u>
<u>AGAIN</u>	5-29
<u>BEGIN</u>	5-28
<u>BRIEF</u>	5-33
<u>COPY</u> n infile (outfile)	5-26
<u>DOWN</u> n	5-28
<u>END</u>	5-28
X <u>FILE</u>	5-30
<u>FIND</u> \$string\$	5-23
<u>GET</u> n (filename)	5-24
<u>INPUT</u>	5-17
<u>INSERT</u> string	5-17
<u>KILL</u> n	5-19
<u>LIST</u> N	5-26
<u>MACRO</u> m=command line	5-35
<u>MACRO</u> m	5-35
<u>N</u>	5-28
<u>PUT</u> n (filename)	5-25
<u>PUTK</u> n (filename)	5-25
<u>QUIT</u>	5-31
<u>REPLACE</u> string	5-21

<u>COMMAND</u>	<u>PAGE</u>
<u>SDOS</u>	5-33
<u>SUBSTITUTE</u> \$string1\$string2\$	5-20
<u>TAB</u> character	5-31
<u>TABS</u> C1 C2 C3 ...	5-31
<u>TYPE</u> n	5-30
<u>UP</u> n	5-28
m <editor commands>	5-32
<u>?</u>	5-33
<u>/</u>	5-33

## APPENDIX C

### HEXADECIMAL OBJECT FORMAT

Absolute hexadecimal object code is formatted into blocks. Within a block, only hexadecimal characters are permitted, with the exception of the colon which indicates the start of a block.

Each block contains the following elements:

1. A start of block character. This is always a colon (:).
2. An address field. This is a four hex character field that indicates where the data is to be stored.
3. A count field. This is a two hex-character field in the range 00 to 1E. This indicates the number of actual data bytes in the block, which is half the number of hex characters in the data field. A block length of zero indicates an End-of-File (EOF) block. The address field of an EOF block contains the start address of the loaded program.
4. A Block Check Character (BCC) for the address and count fields. This is a two hex character field. The BCC is 8 bits formed from the actual bytes, not the ASCII characters (e.g., if the count field was 1E, the two byte ASCII value 31 45 would not be used, the value 1E would be used). The bytes, in this case the two bytes from the address field and the byte from the count field, are in turn exclusive ORed to the BCC byte and then the BCC byte is rotated left one bit.

This field prevents storing data at an invalid memory address.

5. The data field. This field contains two times the number of characters specified in the count field. Two bytes in this field are composed into one byte of data to be stored into memory. For example, if the first two characters on the tape were '1E', (ASCII Values 31 and 45) 1E is stored into memory.
6. A Block Check Character for the data field. This character is formed in the same way as the BCC for the address and count fields, only the data used to compute the BCC is the data in the data field.

Each block is independent. For example, paper tape can be positioned prior to any block and a load started. The loading of absolute object code will be halted by:

- a) A Block Control Character error on the address and count fields,
- b) A Block Control Character error on the data field,
- c) An incorrect block length,
- d) A non-hex character within the block.

Interblock characters must be non-printing ASCII control characters. For example, a CR (Carriage Return)/LF (Line Feed) combination is used within the inter-block gap to reset the TTY or terminal after each block.

:05000A3C0455B024FFF01F015040030

2 3 4 5 6 7

- 2 - Start block character (colon)
- 3 - Starting address for block (H'0500')
- 4 - Number of bytes in block (H'0A'=10)
- 5 - BCC bytes for fields 3 and 4 (H'3C")
- 6 - Data, two characters per byte
- 7 - BCC byte for field 6 (H'30')

**APPENDIX D**  
**SMS TAPE FORMAT**

An SMS tape consists of a block of data, preceded by a TAPE ON character (CTRL-R or 12<sub>H</sub>) and followed by a TAPE OFF character (CTRL-T or 14<sub>H</sub>). When the TAPE ON character is read, the address counter is set to zero. This means that the next data byte will be stored at location 0. When the TAPE OFF character is read, the tape has been read and no more data is stored.

The data in between is represented as follows:

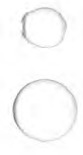
1. Each data word is represented by one or two hexadecimal characters
2. Each data word is followed by an apostrophe (27<sub>H</sub>). When the apostrophe is read, the data word composed from the previous hexadecimal characters is stored at the location pointed to by the address counter. The address counter is then incremented.

All characters are punched in the standard 8-channel ASCII teletype code. Parity is not checked.

**EXAMPLE OF SMS FORMAT**

↑	<u>01</u>	<u>'FA'</u>	<u>'FA'</u>	<u>'00'</u>	<u>'01'</u>	↓
1	2	3			4	

- 1 The tape on character. Resets location counter to 0.
- 2 An individual data byte, 01'. 01 is the data to store, ' indicates end of the data byte.
- 3 The complete data field for this tape.
- 4 The tape off character. Indicates end of data.





APPENDIX E  
SYSTEM READINESS TEST

READY [/disk drive] [LPT1 CONO]
------------------------------------

PURPOSE

READY is a command file which provides a quick check of the TWIN system in the SDOS environment.

EXPLANATION

**CAUTION**

PROM power switch must be on prior to executing this command.

READY is a command file which provides a quick check of the TWIN system in the SDOS environment by exercising each device and the majority of system commands.

DISK DRIVE identifies the disk drive of the diskette containing READY. This diskette must be writable (TAB in place over slot) and have space for one file which will be written and then deleted. The printed output must be directed either to CONO (basic system) or to LPT1 (super system). Each command executed is displayed on the console. At the end of its execution, READY invokes the Editor, which prints:

```
**EDITOR VERSION n.n**  
*
```

where:

n.n is the current version of the Editor.

At this point enter the string: QUIT. The 'End of Ready Test' message notifies the user that the test has completed.

Error Responses

Two types of error messages may occur:

1. Memory type errors in the form:

```
ERROR ADDRESS XXXX  
DATA WAS = XX  
DATA S/B = XX
```

2. Standard SDOS error message.

```

*
* [READY] TEST FOR SDOS 3.0
*
* TO EXECUTE TYPE
*   READY $1 $2
*   $1 - DRIVE NUMBER OF DISK WITH READY FILE
*   $2 - LIST OUTPUT DEVICE
*
* TYPE ON
*
*   TURN PROM POWER ON
*
*   MEMORY TEST
*
* TYPE OFF
RPRM 0 825115 0 1FF
CPROM 0 825115 0 1FF
RPRM 0 1702 0 FF
CPROM 0 1702 0 FF
*
DEBUG
ABORT DEBUG
RHEX READY/$1
GO 00
UPR
TYPE ON
*
COPY READY/$1 ***HELP/$1
PRINT ***HELP/$1 $2
DEL ***HELP/$1
LDIR / $1 $2
*
DEBUG
BKPT 0 W
BKPT 1000 R
DSTAT
CLBP 0
DSTAT
CLBP 1000
DSTAT
ABORT DEBUG
*
ICE 2
ICE 1
ICE OFF
*
* TYPE 'QUIT' WHEN EDIT ASK FOR INPUT
EDIT
*
*
* TURN PROM POWER OFF
* END OF READY TEST
*
ABORT *

```

Figure E-1. System Readiness Test

```
:000003061F007018
:00401B370053004B00600068004A1A020100000013014710010000000005B33
:005B1E51434F4E4F0D020100000013015A020100000013016DD4F775FF04FFCC018440
:00791ED920CC0188C2CE0187CE0187000187ED018790000C0190E400900486109A6777
:00971D64CE01875252520E6188CC01560E618CCC0157D4F60C01840401E404980288
:00041EEED4F3CC01840C6180CC018A0601059FCE0185CDE185850159798601EE0187A8
:00021D71986F059F0601CE01850DE185EC018ABC0108850159738601EE0187986969
:00EF1C871F00ABC144F05050508430E43A1A020407450F0530E53A1685071793
:01001C1CC001893F00F2CC017601CD01770C01868C01893F00F2CC016ACD016885
:01271CAC0C01853F00F2CC0168CD01690C018A3F00F2CC017DC017ED4F5D4F4D7
:01431B33000189174D454D4F52592053495A452049532020204B004552524F8
:015E1E4D52204144445245535205858580044415441205741533D585820532F42F6
:017C1BCF3D5858000FF55A0FF0000000000020302034203831323136323062
:01970046323432383332000001
:007000C100
```

Figure E-1. System Readiness Test (Continued)



**APPENDIX F**  
**TWIN SUPERVISOR CALLS**

NOTE

The information in this Appendix supersedes Appendix B of the TWIN System Reference Manual.

Supervisor calls are used by the slave CPU to gain access to SDOS resident service such as input and output from system peripherals or getting parameters from a program command line.

Supervisor Call Description

The master CPU has as one of its functions the monitoring of the slave CPU. Included in the monitoring function is providing I/O from system peripherals to the slave CPU. Because the slave CPU does not have direct access to system peripherals or operating system functions, the master CPU services such I/O or monitor requests.

The slave CPU obtains service from the master CPU by issuing a supervisor call, SVC. The supervisor call is implemented as an output instruction sequence in the user program. This output instruction is through a specific device address, which is linked by a pointer to a buffer area containing service request parameters. This buffer area is called the service request block, SRB. The Pointers to SRBs must be at specific memory locations in slave memory. Table F-1 shows the one-to-one correspondence between device addresses and SRB pointers. A total of six SVCs may be defined at any one time.

Table F-1. SVC References

SVC	Device Address	SRB Pointer Location
1	F7	40
2	F6	42
3	F5	44
4	F4	46
5	F3	48
6	F2	4A

The sequence of events for issuing an SVC is:

1. Define an SRB pointer in the location corresponding to the SVC and device address through which you want to issue the SVC.

2. Define the SRB parameters according to the SVC function you are going to request.
3. Do a:

WRTE,r Device Address

instruction, where Device Address corresponds to your SRB pointer location as defined in Table F-1. In this instruction, r is a "don't care" value. The act of writing to the appropriate device address, rather than the data written, to it is what actually issues the SVC.

### Service Request Block (SRB)

The SRB contains the parameters that are needed to perform the function requested by an SVC. Each SRB contains eight bytes of data. The following table indicates the SRB contents in the order they must appear in your program.

Table F-2. Contents of the Service Request Block

Byte Name	Byte	Contents
SFC	1	SVC function code
SCH	2	Channel number
STAT	3	SRB status
SDAT	4	Single byte data
BCNT	5	I/O byte count
BMAX	6	I/O buffer length
BPTR	7-8	I/O buffer pointer

### Description of SRB Bytes

This subsection contains a description of each byte in the service request block.

**SVC FUNCTION CODE - Byte 1.** The SVC function code specifies the I/O or monitor function to be performed. The functions are described in the subsection titled "SVC Function Codes", and listed in Table F-3 in that section.

**CHANNEL NUMBER - Byte 2.** A logical channel number must be assigned for each SVC function code that requests I/O service. The channel number must be in the range 0 to 7. When a channel is assigned to a physical device or file, the channel stays connected to that device or file until a CLOSE command is issued on the channel or the job is aborted.

The console devices CONO and CONI, as well as the flexible disk, are sharable devices which can be assigned to more than one channel. The other devices are non-sharable and can be assigned to only one channel at a time. A user program can have a maximum of seven channels assigned to files.

SRB STATUS - Byte 3. The operating system stores an SRB status code in this byte. When a "Read and Proceed" or a "Write and Proceed" SVC function is requested, the operating system will write 7F (I/O in progress) in this byte. When the I/O operation is completed, one of the other SRB status codes will be stored in this byte. Appendix G lists the SRB status codes in hexadecimal, and a short description of each.

SINGLE BYTE DATA - Byte 4. This byte is used by the operating system to return single byte data requested by a non-I/O SVC function. For an I/O SVC function, the physical status of the device being accessed is stored in this byte.

I/O BYTE COUNT - Byte 5. The actual number of bytes of data input or output is stored in this byte. For line oriented ASCII I/O operations, the count is the actual number of characters plus the carriage return. For binary I/O the count is the actual number of bytes. Byte 5, I/O Byte Count, is also used with Byte 4, Single Byte Data, to return double byte data requested by a non-I/O SVC function.

I/O BUFFER LENGTH - Byte 6. In this SRB byte, you specify the maximum number of bytes for I/O that you expect, for both ASCII and binary I/O.

I/O BUFFER POINTER - Bytes 7 and 8. These bytes point to the address of the I/O buffer. The location of this buffer must be in the first 16K page of program memory. This buffer is used to transfer data to or from your program.

Table F-3. SVC Function Codes (Hexadecimal)

Code	Function
10	Assign channel to device or channel
01	Read ASCII and wait
81	Read ASCII and proceed
02	Write ASCII and wait
82	Write ASCII and proceed
41	Read binary and wait
C1	Read binary and proceed
42	Write binary and wait
C2	Write binary and wait
03	Close device or file on channel
04	Rewind file on channel
05	Delete file on channel
06	Rename file on channel
11	Get time (milliseconds)
12	Get overlay addresses
13	Get parameter (procedure parameter buffer)
1C	Get parameter (emulation parameter buffer)
15	Get device status
16	Get device type
17	Load overlay
18	Execute overlay
19	Suspend execution
1D	Get top of slave memory
1A	Exit
1F	Abort



## SVC Function Codes

Following are descriptions of the SVC functions available to user programs. You enter the SVC function code in byte 1 of the SRB prior to issuing an SVC. Note that the SVC function codes are hexadecimal values.

### ASSIGN CHANNEL - Code 10

A user program may perform I/O through up to eight logical channels, numbered 0-7. Any logical channel can be assigned to any physical device attached to the system. If you assign a channel to a floppy disk file which does not exist, the file will be created and a "new file" status is returned to your SRB.

Devices must be assigned to channels prior to performing I/O, and the assign is in effect until a CLOSE or ABORT is performed.

Set up the SRB as follows:

```
SFC - 10
SCH - number (0-7) of the concerned channel
BPTR - pointer to first byte of device or file
       name. The device or file name is an ASCII
       string terminated by a carriage return
       (ODH).
```

Data returned are:

```
STAT - status of the assign
SDAT - device physical status
```

### READ/WRITE ASCII - Codes 01, 81, 02, 82

A line is defined as a string of ASCII characters terminated by an end of line (EOL) character, which is the normal ASCII carriage return (OD<sub>H</sub>).

For ASCII READ/WRITE AND PROCEED, codes 81 and 82, the master CPU initiates the requested I/O and returns control to the user program immediately; it does not wait for completion of the I/O operation. The SRB status, STAT, is set to 7F<sub>H</sub> to indicate "I/O in Progress". Your program must make specific tests on STAT for a change status to indicate completion of I/O.

For ASCII READ/WRITE AND WAIT, codes 01 and 02, the Master CPU initiates the I/O and then waits for it to complete before returning control to your program.

All READ and WRITE operations are performed over a logical channel which has been defined with either the ASSIGN SVC or the SDOS ASSIGN command.

SRB setup:

```
SFC - 01 For READ ASCII and WAIT
      - 81 for READ ASCII and PROCEED
      - 02 for WRITE ASCII and WAIT
      - 82 for WRITE ASCII and PROCEED
```

- SCH - Number of the previously assigned channel
- BMAX - Maximum number of data bytes to read or write. BMAX must be in the range  
 $1 \leq \text{BMAX} \leq 255$
- BPTR - Address of the input or output buffer area

The actual number of data bytes transferred may be less than or equal to BMAX.

If the data stream does not contain a carriage return (CR) character at character BMAX + 1, then BMAX number of characters are transferred and a CR is appended to the end of data stream. For this reason the buffer area to which BPTR points must be of size BMAX + 1 for read operations. The next I/O operation begins with character BMAX + 2. Note that the BMAX + 1st character is not reprocessed.

Data returned are:

- STAT - termination status
- SDAT - device status
- BCNT - actual number of bytes moved into the input buffer area or out of the output buffer area.

#### READ/WRITE BINARY - Codes 41, C1, 42, C2

A block of binary data bytes can be input or output through a channel which has been previously assigned with either the ASSIGN SVC or the SDOS ASSIGN command. The binary data transfer is terminated when a specified number of bytes has been transferred; the carriage return character has no special meaning in binary I/O operations.

BINARY READ/WRITE AND PROCEED, codes C1 and C2, is handled by the master CPU as explained above for ASCII I/O AND PROCEED operations.

BINARY READ/WRITE AND WAIT, codes 41 and 42, is handled by the master CPU as explained for ASCII I/O AND WAIT operations.

SRB setup:

- SFC - 41 for READ BINARY AND WAIT  
- C1 for READ BINARY AND PROCEED  
- 42 for WRITE BINARY AND WAIT  
- C2 for WRITE BINARY AND PROCEED
- SCH - number of the previously assigned channel
- BMAX - number of binary data bytes to transfer;  
BMAX must be in the range  
 $0 \leq \text{BMAX} \leq 256$   
If 0 is specified, 256 bytes are transferred.

BPTR - address of the input or output buffer area

The actual number of data bytes transferred may be less than or equal to BMAX.

Data returned are:

STAT - termination status  
SDAT - device status  
BCNT - actual number of bytes transferred

#### CLOSE - Code 03

The close function disconnects the given channel from the device or file to which it was assigned. When the channel is assigned to a file on a flexible disk the last buffer of data stored in the system memory buffer is output to the file and the disk directory is updated to indicate the length of the file.

SRB setup:

SFC - 03  
SCH - number of channel to close

Data returned are:

STAT - termination status  
SDAT - device status

#### REWIND - Code 04

Rewind applies only to floppy disk files. It has the effect of positioning the file pointer to the beginning of the file. If a device other than a floppy disk is assigned to the specified channel the rewind is treated as a NOP.

When the file is rewound, it is treated as if it had just been assigned. If the first operation for the rewound file is read, the data is input from the file in the normal manner. If the first operation from the rewound file is a write, the file is treated as if it were a new file.

SRB setup:

SFC - 04  
SCH - channel assigned to disk file to be rewound

Data returned is:

STAT - termination status

## DELETE - Code 05

The delete function causes the file assigned to the given channel to be deleted from the directory of the diskette, and the channel is disconnected from the file. If a device is assigned to the channel, the delete function will be treated the same as the CLOSE function.

SRB setup:

SFC - 05  
SCH - channel assigned to disk file to be deleted

Data returned is:

STAT - termination status

## RENAME - Code 06

RENAME applies only to floppy disk files. The file being renamed must have first been assigned or rewound; that is, it must not be in an I/O process. If a device other than a floppy disk is assigned to the channel, the function is treated as a NOP.

SRB setup:

SFC - 06  
SCH - channel assigned to the disk file to be renamed  
BPTR - pointer to the first byte of the new file name. The new file name is an ASCII string terminated by an EOL.

Data returned is:

STAT - termination status

## GET PARAMETER - Codes 13, 1C

Parameters in the command line invoking a user program (LOAD or XEQ SDOS commands) are stored in a master memory in either the system parameter buffer or the procedure buffer and are accessible to the user program via an SVC. Since the user program can be invoked directly from the system console or from within a procedure file, an SVC is provided for each case.

Parameters are delimited in the command line by a space, comma, or EOL. A parameter can be omitted from an ordered sequence of parameters in the command line by entering two consecutive commas (,,). An EOL terminates the command line.

Each parameter is stored in the system parameter buffer as an ASCII string terminated with an EOL. Omitted parameters are stored as a single EOL. The parameter is identified by a number corresponding to its position in the command line; the GET request is keyed on this number.

The GET PARAMETER function transfers the ASCII string and EOL corresponding to the number of the parameter requested from the system parameter buffer to a specified buffer area in the user program. When a parameter number greater than the number of parameters in the command line is requested, a -1 is put in the first byte of your buffer.

SRB setup:

- SFC - 13 for GET PARAMETER from PROCEDURE command  
1C for GET PARAMETER from system level command line
- SDAT - number of parameter to get. If the high bit is set, all remaining parameters will be fetched. SDAT is incremented by the SVC processor so that it need only be set for the first call when several successive parameters are being fetched.
- BMAX - maximum expected length of parameter(s) ASCII string
- BPTR - pointer to buffer area to which parameter(s) is to be transferred

Data returned is:

- STAT - termination status. A short read status, code 06, is returned when the number requested is greater than the number of parameters in the command line.

#### LOAD OVERLAY - Code 17

User program overlays stored on a diskette may be loaded by the user program executing under the slave CPU. Each overlay must be stored on the disk as a binary load module, created either with the SDOS MODULE command or by the TWIN-Resident Relocatable Assembler, RASM (see 2650 Relocatable Assembler Manual).

Execution of the loaded overlay is not started and control remains with the requesting user program.

SRB setup:

- SFC - 17
- BPTR - pointer to buffer containing the name of the overlay to load. The overlay name is an ASCII string terminated with an EOL.

Data returned is:

- STAT - status of the load

### EXECUTE OVERLAY - Code 18

This function is called and performed in the same way as the LOAD OVERLAY function except that execution of the overlay is started after being loaded. The EXECUTE OVERLAY function also provides the capability of chaining separate programs.

SRB setup:

SFC - 18  
BPTR - pointer to buffer containing the name of the overlay to load and execute. The overlay name is an ASCII string terminated with an EOL.

Data returned is:

STAT - status of the load and execute. If no error occurred during the load, the overlay is executed.

### SUSPEND EXECUTION - Code 19

The SUSPEND EXECUTION causes suspension of the requesting program at the place where the SVC was issued. The action is similar to an I/O and WAIT operation. The program can be restarted by entering the SDOS command "CONT /". Execution resumes at the next location after the SUSPEND EXECUTION SVC.

SRB setup:

SFC - 19

No data is returned.

### GET OVERLAY ADDRESS - Code 12

This SVC gets the memory bounds of the last overlay loaded into slave memory. Address data is returned to a buffer in the user program.

SRB setup:

SFC - 12  
BPTR - pointer to 6-byte data buffer for data returned by the SVC processor.

Data returned are:

STAT - status

The data buffer contains 6 bytes of data as follows:

Bytes 0, 1 - overlay begin address  
Bytes 2, 3 - overlay end address  
Bytes 4, 5 - overlay execution address



EXIT - Code 1A

EXIT terminates user program execution and gives control to the TWIN operating system. Assigned channels are not closed.

SRB setup:

SFC - 1A

ABORT - Code 1F

ABORT terminates user program execution and gives control to the TWIN operating system. All assigned channels are closed.

SRB setup:

SFC - 1F

GET DEVICE TYPE - Code 14

This SVC request returns to the SRB device type code and device system identification number of the device attached to the specified channel. Tables F-4 and F-5 show Device Identification and Device Type codes respectively.

Table F-4. Device Identification and Type

Name	Description	I.D. Number	Type Code
CONI	Console input	1	1
CONO	Console output	2	2
LPT1	Line printer	3	2
DISK FILE	File	-1	43
TTYR	TTY P/T reader	5	1
HSPT	High speed P/T reader	6	1
R232	RS-232 port	6	43
LPT2	Optional line printer	6	2

Table F-5. Device Type Code

Type Code	Description
1	ASCII Read only
2	ASCII Write only
3	ASCII Read/Write
41	Binary Read only
42	Binary Write only
43	Binary Read/Write



The device type specifies the type of I/O performed on the device under normal usage. A user program can read from input device in either ASCII or binary mode and can write to any output device in either ASCII or binary mode.

CONI, CONO, and the floppy disk are sharable devices which can be assigned to multiple channels simultaneously. The remaining devices can be assigned to only one channel at a time. A maximum of seven channels can be assigned to floppy disk files.

SRB setup:

SFC - 14  
SCH - channel assigned to the device whose type is requested

Data returned are:

STAT - status of the SVC request  
SDAT - device number  
BCNT - device type

#### GET DEVICE STATUS - Code 15

The physical status of the device attached to the specified channel is returned to the SRB. The physical status returned is that associated with the last operation performed by the device.

SRB setup:

SFC - 15  
SCH - channel assigned to the device whose status is to be returned

Data returned are:

STAT - status of the SVC request  
SDAT - device physical status. A zero is returned if no physical status is available

#### GET LAST CONSOLE INPUT CHARACTER - Code 16

This function returns the last character entered at the system console to the SRB data byte. If this SVC is issued within a loop while performing extensive calculations or I/O, it provides the user program with a way to respond to operator action at the system console.

SRB setup:

SFC - 16

Data returned are:

STAT - status of the SVC request  
SDAT - last console character input

GET TOP OF SLAVE MEMORY - Code 1D

The top of slave memory as determined at boot time is returned to the data bytes of the SRB.

SRB setup:

SFC - 1D

Data returned are:

STAT - status of SVC request  
SDAT - top of memory address, high byte  
BCNT - top of memory address, low byte

GET TIME - Code 11

The TWIN has a real time clock which records time since system start-up in milliseconds. Currently, the system clock is always off, so that a value of zero is always returned. The SVC has been provided for future system enhancement and expansion.

SRB setup:

SFC - 11

Data returned are:

STAT - status of the SVC request  
SDAT - high byte of system time  
BCNT - low byte of system time

**APPENDIX G**  
**SRB STATUS CODES**

CODE	MEANING
00	Function complete/no error
01	Channel assigned to new file
02	Illegal channel number
03	Channel not assigned
04	Channel busy
05	Illegal function code
06	No EOL on ASCII read
07	No EOL on ASCII write
08	Illegal drive number
09	File in use
0A	Device not operational
0B	Device not available
0C	Device not ready
0D	Device in use
0E	Directory read error
0F	Directory write error
10	Directory full
11	Device read error
12	Device write error
13	Invalid address, attempt to clobber utility programs or memory wraparound
14	Unused
15	File name in use
16	Illegal file name
17	File in read/write progress
18	Channel already assigned
19	Incorrect diskette
7F	I/O in progress
FF	End of file or end of device

These status codes are returned to system slave jobs and displayed on the system console or to user slave programs when I/O errors occur in byte 3 of the SRB.



## APPENDIX H

### ADDING A DEVICE DRIVER TO SDOS

The following technique for adding a driver to SDOS assumes the reader is familiar with the two separate handler and interrupt service portions of a driver and serves to show how to incorporate them (merge them into) the SDOS load module. The user must configure his driver such that it occupies the memory locations reserved for an optional device driver.

The discussion below describes how to add the High Speed Paper Tape Reader driver (device Name HSPT) to SDOS. A copy of the source listing used is at the end of this description.

The merging scheme is very simple. The user will assemble his logic as a separate object module (in Hexadecimal Format), and then perform the following steps using SDOS functions to generate a new SDOS module.

1. LOAD SDOS/0

Load the SDOS Load module into common memory.

2. RHEX FILENAME (of the object module)

Merge the new driver logic into common memory with SDOS.

3. MODULE SDOS/1 80 3FFF 100 (comment)

Make a new SDOS module on a different diskette (advisable until the new driver is debugged).

Follow the sequence of steps below to actually merge the driver logic and its related device tables into SDOS. The HSPT listing shows clearly what is needed in the way of linkages and tables for the driver; briefly the needs are these:

1. Linkages to SDOS routines SAVR, RESR, IOC3, SDCB, and DISP (DISPATCHER). The user should be familiar with the functions of these routines (briefly annotated on the listing); the linkage addresses are shown.
2. FCB (File Control Block) linkage address to the first of the 22 SDOS FCB's. There is no "dedicated" FCB for a particular handler, but the driver is provided with an "Active FCB Index" in the driver-dedicated DCB described below (entry DFCB) when I/O is initiated. Upon entry to the "Handler" the Active FCB Index is in register R1.
3. Shown next in the listing is the creation of linkages to the new driver's DCB (Device Control Block). Note that no data need be merged into the DCB, but linkages must be provided.

4. Now that a device index has been chosen, the corresponding Device Definition Table (DDT) can be set up to define the new device, by name and characteristics, in the SDOS data base. The user must use the DDT entry point linkage addresses shown in the listing (DHAN, DTST, etc.) and use the new device index for an offset in the "ORG's" in place of the PTDV index (HSPT).

The data which must be provided in the entries for this table should reflect the new device as shown in the listing. Note that the DHAN entry is an "unconditional absolute branch" to the device "handler" portion of the new driver.

5. Add the transfer vector for the "interrupt service" portion of the driver. An appropriate slot must be chosen among those in the available hardware interrupt locations which start at location 1000E<sub>H</sub>. See the TWIN I/O USER GUIDE.
6. Now all that is left is to add the driver logic itself to SDOS. The origin chosen must be within the available master memory as shown (3E80<sub>H</sub>).

```

1      TITL   HSPT VER 3.0 APRIL 1978
2 *
3 *
4 *
5 *      *****
6 *      *
7 *      *
8 *      * HIGH SPEED PAPER TAPE *
9 *      * READER DRIVER *
10 *     *
11 *     *
12 *     *****
13 *
14 *
15 *
16 *
17 *
18 * REGISTER EQUATES
19 *
20 R0 EQU 0 REGISTER 0
21 R1 EQU 1 REGISTER 1
22 R2 EQU 2 REGISTER 2
23 R3 EQU 3 REGISTER 3
24 *
25 * CONDITION CODES
26 *
27 P EQU 1 POSITIVE RESULT
28 Z EQU 0 ZERO RESULT
29 N EQU 2 NEGATIVE RESULT
30 LT EQU 2 LESS THAN
31 EQ EQU 0 EQUAL TO
32 GT EQU 1 GREATER THAN
33 UN EQU 3 UNCONDITIONAL
34 *
35 * PSW LOWER EQUATES
36 *
37 CCI EQU H'00'
38 CCO EQU H'40'
39 RS EQU H'10'
40 WC EQU H'08'
41 OVF EQU H'04'
42 COM EQU H'02'
43 C EQU H'01'
44 *
45 * PSW UPPER EQUATES
46 *
47 SENS EQU H'00'
48 FLAG EQU H'40'
49 II EQU H'20'
50 SP2 EQU H'04'
51 SP1 EQU H'02'
52 SP0 EQU H'01'
53 *
54 * HIGH SPEED PAPER TAPE READER INPUT AND CONTROL PORTS
55 *
56 PCPT EQU H'D0' HSPT CONTROL PORT
57 PDPT EQU H'D1' HSPT DATA PORT
58 BSPT EQU H'EE' COMMON MEMORY BANK SELECT
59 *
60 *

```

Figure H-1. Adding a Driver to SDOS Sample

```

61 *
62 *
63 *
64 *****
65 *
66 *   GENERAL CONTROL CHARACTERS *
67 *
68 *****
69 *
70 EOL EQU H'0D' END OF LINE
71 CR EQU H'0D' CARRIAGE RETURN
72 LF EQU H'0A' LINE FEED
73 CTLZ EQU H'1A' CONTROL Z
74 RUB EQU H'7F' RUBOUT
75 ESC EQU H'1B' ESCAPE
76 SPAC EQU H'20' SPACE
77 SLSH EQU A'/'
78 NULL EQU 0 NULL
79 XON EQU H'11' XON FOR TTY READER
80 XOF EQU H'13' XOFF FOR TTY READER
81 *
82 *   JCB STATES
83 *
84 JS0 EQU 0 JOB IDLE (NO JOB)
85 JS1 EQU 1 JOB LOADED
86 JS2 EQU 2 JOB READY TO START
87 JS3 EQU 3 JOB EXECUTING
88 JS4 EQU 4 JOB IN I/O WAIT
89 JS5 EQU 5 JOB I/O COMPLETE
90 JS6 EQU 6 JOB SUSPENDED
91 JS7 EQU 7 JOB BEING ABORTED
92 JS8 EQU 8 JOB SELF PAUSED
93 *
94 *   DCB STATES
95 *
96 DRDY EQU 0 DEVICE READY
97 DBSY EQU 1 DEVICE BUSY
98 DVDN EQU 2 DEVICE DOWN
99 *
100 *   DSB STATES
101 *
102 DSBU EQU 0 DRIVE STATE UNKNOWN
103 DSBS EQU 1 DRIVE DSB SET
104 DSBD EQU 2 DRIVE DOWN
105 *
106 *   FCB STATES
107 *
108 FREE EQU 0 FREE
109 FRSS EQU 1 ASSIGNED
110 FOPN EQU 2 OPEN
111 FBSY EQU 3 BUSY
112 FRDY EQU 4 READY
113 FEOF EQU 5 EOF/EOD
114 FEOD EQU FEOF
115 FRBT EQU 6 ABORT
116 *
117 *
118 *
119 *
120 *

```

Figure H-1. Adding a Driver to SDOS Sample (Continued)



```

121 *
122 *
123 *
124 *****
125 *
126 *   SRB STATUS CODES *
127 *   *
128 *****
129 *
130 SE00 EQU 0 FUNCTION COMPLETE / NO ERROR
131 SE01 EQU 1 NEW FILE
132 SE02 EQU 2 ILLEGAL CHANNEL NUMBER
133 SE03 EQU 3 CHANNEL NOT ASSIGNED
134 SE04 EQU 4 CHANNEL BUSY
135 SE05 EQU 5 ILLEGAL FUNCTION CODE
136 SE06 EQU 6 SHORT READ
137 SE07 EQU 7 SHORT WRITE
138 SE08 EQU 8 ILLEGAL DRIVE NUMBER
139 SE09 EQU 9 FILE IN USE
140 SE10 EQU 10 DEVICE NOT OPERATIONAL
141 SE11 EQU 11 DEVICE NOT AVAILABLE
142 SE12 EQU 12 DEVICE NOT READY
143 SE13 EQU 13 DEVICE IN USE
144 SE14 EQU 14 DIRECTORY READ ERROR
145 SE15 EQU 15 DIRECTORY WRITE ERROR
146 SE16 EQU 16 DIRECTORY FULL
147 SE17 EQU 17 DEVICE READ ERROR
148 SE18 EQU 18 DEVICE WRITE ERROR
149 SE19 EQU -1 NOT ASSIGNED
150 SE20 EQU -1 NOT ASSIGNED
151 SE21 EQU 21 FILE NAME IN USE
152 SE22 EQU 22 ILLEGAL FILE NAME
153 SE23 EQU 23 FILE IN R/W PROGRESS
154 SE24 EQU 24 CHANNEL ALREADY ASSIGNED
155 SE25 EQU 25 INCORRECT DISKETTE
156 SE7F EQU H'7F' I/O IN PROGRESS
157 SEFF EQU H'FF' END OF FILE / END OF DEVICE
158 *
159 *****
160 *
161 *   SVC FUNCTION OP CODES *
162 *   *
163 *****
164 *
165 R00P EQU H'01' READ
166 WTOP EQU H'02' WRITE
167 CLOP EQU H'03' CLOSE
168 RWOP EQU H'04' REWIND
169 DEOP EQU H'05' DELETE
170 RNOP EQU H'06' RENAME
171 DIOP EQU H'07' DIRECT I/O
172 ASOP EQU H'10' ASSIGN
173 RBOP EQU H'61' READ BLOCK
174 WBOP EQU H'62' WRITE BLOCK
175 *
176 *
177 *
178 *
179 *
180 *

```

Figure H-1. Adding a Driver to SDOS Sample (Continued)

```

181 *
182 *
183 *
184 *****
185 * *
186 * DEVICE CONTROL BLOCK INDICES *
187 * *
188 *****
189 *
190 CIDV EQU 0 CONSOLE INPUT
191 CODV EQU 1 CONSOLE OUTPUT
192 LPDV EQU 2 LINE PRINTER
193 XXXX EQU 3 AVAILABLE
194 ERDV EQU 4 ERROR PSEUDO DEVICE
195 TTYR EQU 5 TTY READER
196 PSDV EQU 6 RS-232
197 LPT2 EQU 6 SECOND LINE PRINTER
198 PTDV EQU 6 PAPER TAPE READER
199 USR1 EQU 7 USER 1 INDEX
200 USR2 EQU 8 USER 2 INDEX
201 USR3 EQU 9 USER 3 INDEX
202 USR4 EQU 10 USER 4 INDEX
203 USR5 EQU 11 USER 5 INDEX
204 USR6 EQU 12 USER 6 INDEX
205 USR7 EQU 13 USER 7 INDEX
206 USR8 EQU 14 USER 8 INDEX
207 FDCV EQU 15 FLOPPY DISK
208 *
209 * (NOTE, USER INDICES ARE 7-14)
210 *
211 *****
212 * *
213 * LINKS TO SDO'S ROUTINES AND DATA BASE *
214 * *
215 *****
216 * *
217 * SDO'S ROUTINES *
218 * *
219 *****
220 *
221 SAVR EQU H'2003' SAVE REGISTERS
222 RESR EQU H'2006' RESTORE REGISTERS
223 IOC3 EQU H'136C' SET CHANNEL I/O COMPLETE
224 SDCB EQU H'136A' SET-UP DCB FROM THE FCB
225 DISP EQU H'200F' DISPATCHER
226 *
227 *****
228 * *
229 * FCB ENTRIES (1ST FCB) *
230 * *
231 *****
232 *
233 FCS EQU H'2150' FCB STATE
234 FJCB EQU FCS+22 JOB INDEX
235 FBANK EQU FJCB+22 BANK
236 FCH EQU FBANK+22 CHANNEL NUMBER
237 FCOO EQU FCH+22 SVC FUNCTION CODE
238 FPDS EQU FCOO+22 PHYSICAL DEVICE STATUS
239 FSTA EQU FPDS+22 I/O COMPLETE STATUS
240 FCNT EQU FSTA+22 I/O BUFFER BYTE COUNT

```

Figure H-1. Adding a Driver to SDOS Sample (Continued)

```

241 *
242 *
243 *
244 *
245 *
246 *****
247 * *
248 * DEFINE THE HSPT DCB *
249 * *
250 *****
251 *
252 DCS EQU H'2346' 1ST DCB ADDRESS (DEVICE INDEX 0)
253 PDCS EQU DCS+PTDV DEVICE STATE FOR HSPT
254 PTTR EQU PDCS+16 DEVICE STATUS
255 PTDX EQU PTTR+16 I/O BUFFER INDEX
256 PTNT EQU PTDX+16 I/O BUFFER COUNT
257 PTCO EQU PTNT+16 I/O BUFFER ECHO COUNT
258 PTCB EQU PTCO+16 ACTIVE FCB INDEX
259 PTOD EQU PTCB+16 SVC FUNCTION CODE
260 PTNK EQU PTOD+16 BANK SWITCH
261 PTOR EQU PTNK+16+PTDV I/O BUFFER POINTER (DOUBLE-BYTE)
262 *
263 *
264 * *** ENTRIES TO DDT AND INTERRUPT VECTOR SHOULD BE PATCHED
265 * *** TO SDOS 3.0 DATA BASE WHEN DRIVER IS MERGED INTO SDOS.
266 *
267 *****
268 * *
269 * DEFINE THE HSPT DDT *
270 * *
271 *****
272 *
273 *
274 *
275 DHAN EQU H'2436' 1ST 'DHAN' ENTRY (THREE-BYTE)
276 * ORG DHAN+PTDV+PTDV+PTDV
277 * BCTA.UN HPTH ADD XFER VECTOR FOR HSPT HANDLER
278 *
279 *
280 *
281 DTST EQU H'2416' 1ST 'DTST' ENTRY
282 * ORG DTST+PTDV HSPT DEVICE AVAILABILITY
283 * DATA H'0' SET HSPT UP AND NON-SHAREABLE
284 *
285 *
286 *
287 DTTP EQU H'2426' 1ST 'DTTP' ENTRY
288 * ORG DTTP+PTDV HSPT DEVICE I/O TYPE
289 * DATA 1 SET 'READ ONLY' IN TABLE
290 *
291 *
292 *
293 DTID EQU H'2406' 1ST 'DTID' ENTRY
294 * ORG DTID+PTDV HSPT DEVICE ID
295 * DATA PTDV+1 SET DEVICE ID
296 *
297 *
298 *
299 DTNM EQU H'2466' 1ST 'DTNM' ENTRY (FOUR-BYTE)
300 * ORG DTNM+PTDV+PTDV+PTDV+PTDV

```

Figure H-1. Adding a Driver to SDOS Sample (Continued)

```

301 *   DATA      A'HSPT'  ADD DEVICE NAME
302 *
303 *
304 *
305 *****
1  306 *
2  307 *   ADD THE HSPT INTERRUPT VECTOR *
3  308 *
4  309 *****
5  310 *
6  311 *   ORG      H'1006'
7  312 *   BCTR,UN  HISR      ADD VECTOR TO VECTOR AREA
8  313 *                               IN AN AVAILABLE SLOT
9  314 *
10 315 *****
11 316 *
12 317 *   HIGH SPEED PAPER TAPE READER *
13 318 *   INTERRUPT SERVICE ROUTINE *
14 319 *
15 320 *****
16 321 *
17 322 *
18 323 *   ORG      H'3E00'  ORG INTO AVAILABLE MEMORY AREA
19 324 *
20 325 *

```

.  
 .  
 .  
**Body of Driver Logic**

**Figure H-1. Adding a Driver to SDOS Sample (Continued)**

**APPENDIX I**  
**RS232 DRIVER DESCRIPTION**

The general purpose I/O board has an RS-232-C type interface. The UART, along with the control signals, represent the DTE (Data Terminal Equipment) in the EIA standard. The device two status byte sends the required signals to the DTE from the DCE (Data Communications Equipment). The interface includes the following signals:

From DTE	From DCE
ON/OFF LINE	RING INDICATOR
ORIGINATE/ANSWER	CARRIER DETECT
SEND RESTRAINT	DATA SET READY
DATA TERMINAL READY	CLEAR TO SEND
REQUEST TO SEND	

These signals allow DCE to be attached to the DTE and make communication between remote sites and the TWIN system possible. The software to support the RS-232-C interface is written to allow operation with or without data communication equipment attached.

The RS232 device is named R232 and is a non-shareable read/write device which supports half-duplex mode. It behaves externally as any device in the system, that is, a device handler is included along with interrupt service routines.

ALLOWABLE SVCs

The allowable SVCs are the following:

<u>HEX CODE</u>	<u>FUNCTION</u>
01	Read ASCII
02	Write ASCII
03	Close Channel
04	Rewind
10	Assign R232 To Channel
21	Read Status of Control Ports
41	Read Binary
42	Write Binary

CONTROL PORTS

The RS232 interface has one data port for data input and output and two control ports. Control is set by outputting to control port 1. Status is read by inputting on control ports 1 and 2. Refer to figures I-1 through I-3 for port bit assignments.

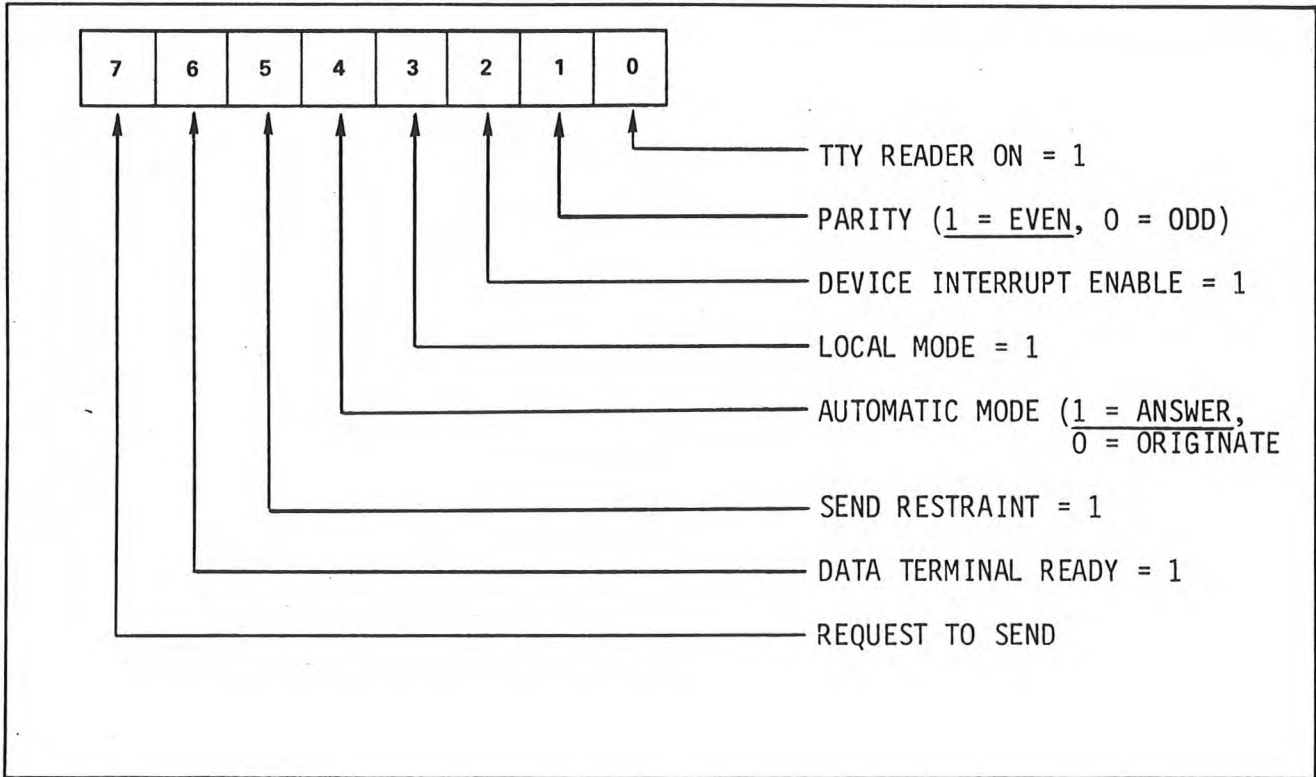


Figure I-1. RS232 Control Port 1 (Output)

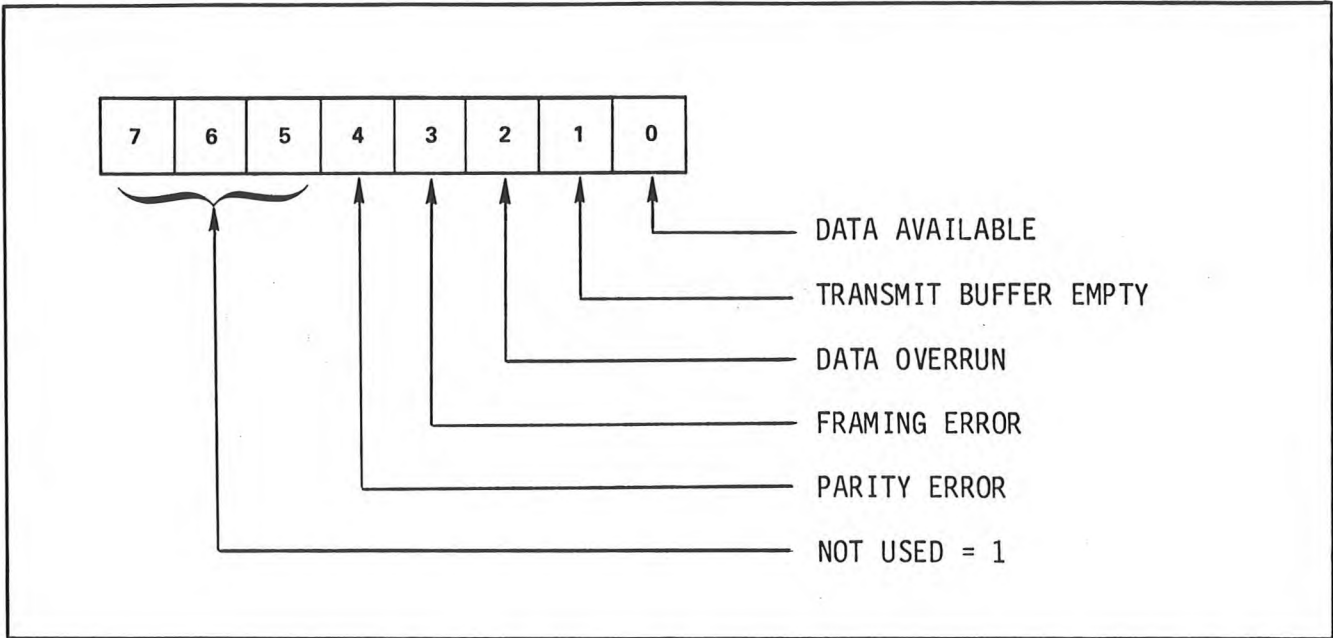


Figure I-2. RS232 Control Port 1 (Input)

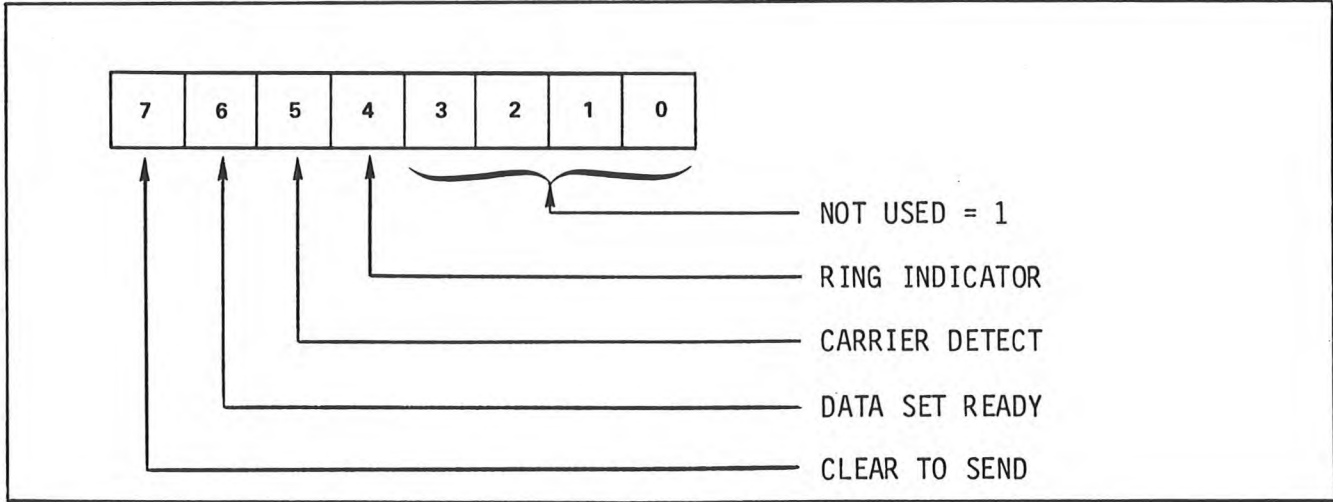


Figure I-3. RS232 Control Port 2 (Input Only)

## NORMAL USAGE

The RS232 device control port takes on the bit assignments as described below for the allowable SVCs. In this mode, the device name R232 may be used as a system device by any of the SDOS commands. Figure I-1 shows the control bit definitions. The bit assignments under normal usage are specified below for each allowable SVC.

### NOTE

The RS232 driver must use the memory space provided in SDOS for an optional driver. Only one optional driver can be configured into the system due to memory limitations.

### Assign

Control is set to DATA TERMINAL READY, ANSWER, DEVICE INTERRUPTS ENABLED and EVEN PARITY. Once a job has successfully assigned a channel to the device, that job is assured of exclusive use of the device and may issue Read or Write SVC's over the channel.

### Read

Control is set to DATA TERMINAL READY, DEVICE INTERRUPTS ENABLED and EVEN PARITY.

### Write

Control is set to REQUEST TO SEND, DATA TERMINAL READY, DEVICE INTERRUPTS ENABLED, and EVEN PARITY. For a Write request, control port 2 is read and the handler delays until the status read from control port 2 signals CLEAR TO SEND. If CLEAR TO SEND is not true and the time-out loop is exceeded, I/O is set complete and the status in the SRB is 'DEVICE NOT OPERATIONAL'.

### Close

When the channel is closed, either by a Close SVC or by an Abort, the control is set to LOCAL MODE and EVEN PARITY.

### Rewind

Control is set as for Assign.



## USER SELECTABLE CONTROL

If you wish to use control assignments other than those defined previously, it is possible to do so by using a Read Status SVC, function code 21H. Prior to any I/O request, issue an SVC 21H with the desired control byte definition loaded into the BMAX byte of the SRB.

## SRB USAGE

Upon completion of the Read or Write SVC, SDAT in the user's SRB contains the status of control port 1. The status of control port 2 is placed in SDAT the user's SRB upon the completion of an Assign, Close or Rewind SVC.

Upon return from a Read Status SVC, BCNT contains control port one status and SDAT contains control port 2 status. See Figure I-4 for a diagram of the SRB for Read Status SVC.

<b>SFC</b> <b>21<sub>H</sub></b>
<b>SCH</b> Channel assigned to RS232
<b>STAT</b> Should be zero if driver received control
<b>SDAT</b> Status of control port 2 returned
<b>BCNT</b> Status of control port 1 returned
<b>BMAX</b> Output control bit settings
<b>BPTR</b>
<b>N/A</b>

**Figure I-4. SRB Usage for Read Status SVC (21<sub>H</sub>)**

## GENERAL DESCRIPTION

At assign time, the terminal is set to on-line and ready. The next request can be either a Read, Write or Close. For a Read request, the Answer mode is set. The DCE is now in a state where the presence of a ring indicator will cause it to take the necessary action (go off hook and establish connection with the communication channel). When this action is complete, the data set ready line will be set. The DCE will keep the received data line to the UART in a marking condition until data set ready is true, i.e., no data is sent across the DCE to the DTE unless both data set ready and data terminal ready are true. When the UART has received a character, the data available goes true causing an interrupt. At this time, error conditions (parity, framing, and data overrun) are valid. The interrupt handler has one full character time to process the data.

The interrupt service routine reads the character and tests for I/O complete, then I/O complete sets the device ready. Interrupts received with the device ready are ignored. If the line protocol is such that the sending station can continue transmission without first receiving an acknowledge, it is likely that messages will be lost. Also system commands, such as COPY, which utilize this device will most likely encounter errors when the device is actually connected to a remote station.

For a Write request, the DTE sets the On-line, Originate and Request to Send lines. The Request to Send causes the DCE to set the Clear to Send line when it is ready to transmit. The DTE will not transmit data until the Clear to Send line goes high. This is one instance where a time out function through the system clock is useful. In the present implementation, the handler will test the Clear to Send line a number of times before returning status of "device not ready". The program issuing the SVC is responsible for determining whether to try again or abort. When Clear to Send goes high, the handler will output one byte of data. When the transmit buffer empty goes true an interrupt will be generated. The interrupt handler will process the interrupt and either set I/O complete or continue to transmit. When I/O is complete the Request to Send line will become low. When the channel is closed, the data terminal ready line is set low and the device is set off-line.



**COMMENT SHEET**

**TITLE:** \_\_\_\_\_

**REVISION:** \_\_\_\_\_

This form is not intended to be used as an order blank. Signetics Corporation solicits your comments about this manual with a view to improving its usefulness in later editions.

Applications for which you use this manual.

Do you find it adequate for your purpose?

What improvements to this manual do you recommend to better serve your purpose?

Note specific errors discovered (please include page number reference).

CUT ON THIS LINE

General comments:

**FROM NAME:** \_\_\_\_\_ **POSITION:** \_\_\_\_\_

**COMPANY NAME:** \_\_\_\_\_

**ADDRESS:** \_\_\_\_\_

**NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.  
FOLD ON DOTTED LINES AND STAPLE**

STAPLE

STAPLE

FOLD

FOLD

FIRST CLASS  
PERMIT NO. 166  
SUNNYVALE, CALIF.

**BUSINESS REPLY MAIL**  
NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

POSTAGE WILL BE PAID BY

**signetics**

a subsidiary of U.S. Philips Corporation

Signetics Corporation  
P.O. Box 9052  
811 East Arques Avenue  
Sunnyvale, California 94086

Bin No. 038 MOS Microprocessor Division



CUT ON THIS LINE

FOLD

FOLD

STAPLE

STAPLE

# Signetics

a subsidiary of U.S. Philips Corporation

Signetics Corporation  
P.O. Box 9052  
811 East Arques Avenue  
Sunnyvale, California 94086  
Telephone 408/739-7700

