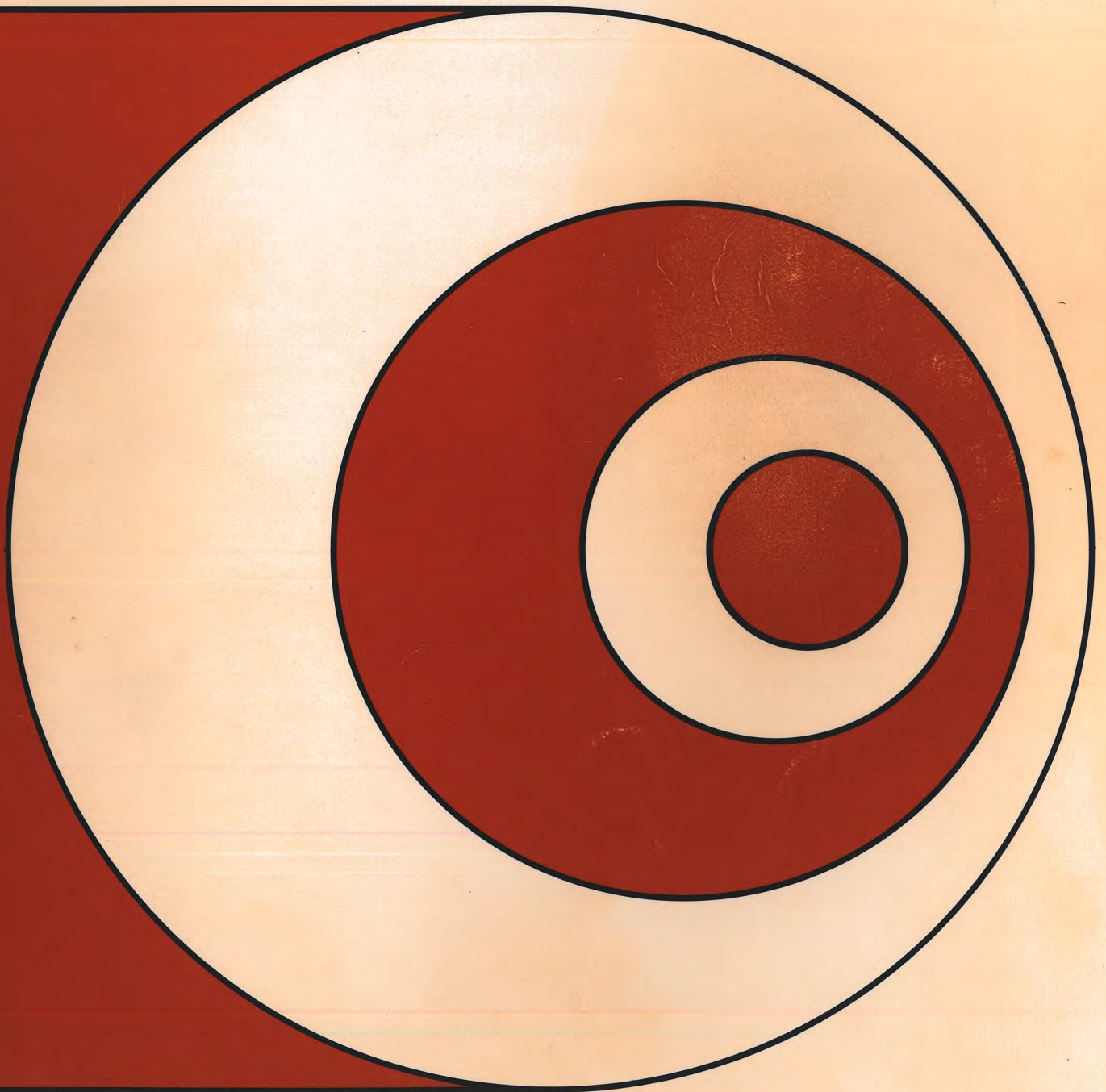


SIGNETICS TWIN

TESTWARE INSTRUMENT

OPERATOR'S GUIDE



SIGNETICS TWIN
TESTWARE INSTRUMENT

OPERATOR'S GUIDE

signetics

a subsidiary of U.S. Philips Corporation

Signetics Corporation
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700

DOCUMENT NO. TW09003000

PRICE - \$5.00

COPYRIGHT 1976, SIGNETICS CORP. ALL RIGHTS RESERVED.

SIGNETICS CORP. CLAIMS TRADEMARK RIGHTS TO THE NAMES TWIN AND TWICE

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE
1	INTRODUCTION	1-1
	1.0 INTRODUCTION	1-1
	1.1 TWIN OVERVIEW	1-1
	1.2 ABOUT THIS BOOK	1-5
	1.3 MANUAL CONTENTS	1-6
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	2-6
	2.2.4 USER-SUPPLIED PERIPHERALS	2-6
	2.3 SOFTWARE	2-6
	2.3.1 SDOS	2-6
	2.3.1.1 THE DEBUG MONITOR	2-7
	2.3.1.2 PROM PROGRAMMING	2-7
	2.3.2 THE EDITOR	2-7
	2.3.3 THE ASSEMBLER	2-7
	2.3.4 SYSTEMS READINESS TEST	2-8
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

CHAPTER	TITLE	PAGE
	3.2.1 POWER REQUIREMENTS	3-5
	3.2.2 INTERCONNECTION	3-5
3.3	SYSTEM CONTROLS AND INDICATORS	3-6
	3.3.1 DEVELOPMENT COMPUTER	3-6
	3.3.2 DUAL FLOPPY DISK UNIT	3-8
	3.3.3 CRT TERMINAL	3-8
	3.3.4 PRINTER	3-8
3.4	OPERATION	3-10
	3.4.1 MANUAL RESET	3-12
4	SIGNETICS DISK OPERATING SYSTEM	4-1
4.0	INTRODUCTION	4-1
4.1	ENTERING SDOS COMMANDS	4-2
4.2	SPECIAL KEYS	4-3
4.3	FILES, DEVICES, AND CHANNELS	4-4
4.4	THE SDOS COMMANDS	4-7
	4.4.1 THE SDOS COMMAND DESCRIPTIONS	4-7
	4.4.1.1 SDOS COMMAND COMPLETION	4-9
	4.4.2 SYSTEM CONTROL COMMANDS	4-9
	4.4.3 SYSTEM OPTIONS	4-14
	4.4.4 SYSTEM UTILITIES	4-16
	4.4.5 OBJECT PROGRAM UTILITIES	4-24
4.5	RESIDENT SDOS AND OVERLAY AREAS	4-27
	4.5.1 RESIDENT SDOS	4-27
	4.5.2 SDOS OVERLAYS	4-27
4.6	COMMAND FILES	4-28
	4.6.1 COMMAND FILES UTILITIES	4-30
5	THE TEXT EDITOR	5-1
5.0	INTRODUCTION	5-1
5.1	THE EDIT COMMAND	5-2
5.2	EDIT EXAMPLE	5-3
5.3	EDITOR COMMAND DESCRIPTIONS	5-12
	5.3.1 EDITOR COMMAND LINE	5-12
	5.3.2 EDITOR COMMAND DESCRIPTION CONVENTIONS	5-14
	5.3.3 INSERTION	5-16
	5.3.4 DELETION	5-18
	5.3.5 ALTERATION	5-19
	5.3.6 SEARCH	5-21
	5.3.7 I/O	5-22

CHAPTER	TITLE	PAGE
	5.3.8 LINE POINTER COMMANDS	5-25
	5.3.9 UTILITIES	5-25
	5.3.10 MACROS	5-31
5.4	EDITOR MESSAGES	5-31
6	THE 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	6-2
	6.4 ASSEMBLER ERRORS	6-5
	6.5 LOADING AN ASSEMBLED PROGRAM	6-6
	6.6 THE ASSEMBLER TAB FEATURE	6-7
7	THE PROM PROGRAMMER	7-1
	7.0 INTRODUCTION	7-1
	7.1 USING THE PROM PROGRAMMERS	7-1
8	THE DEBUGGER	8-1
	8.0 INTRODUCTION	8-1
	8.1 THE DEBUG PACKAGE	8-2
	8.2 THE DEBUG COMMANDS	8-4
	8.3 SAMPLE DEBUG SESSION	8-4
	8.4 DEBUG COMMANDS	8-13
	8.5 TWICE DEBUG CABLE	8-26
APPENDIX		
A	SDOS COMMAND SUMMARY	A-1
B	TEXT EDITOR COMMAND SUMMARY	B-1
C	ABSOLUTE OBJECT FORMAT	C-1
D	SMS TAPE FORMAT	D-1
E	SYSTEM READINESS TEST	E-1
F	SYSTEM UTILITY COMMAND FILES	F-1

FIGURE	TITLE	PAGE
1-1	ELEMENTARY PARTITIONING OF 2650 MICROCOMPUTER SYSTEM LOGIC	1-3
1-2	TWIN SLAVE CPU EMULATES USERS SYSTEM CPU	1-4
2-1a	A DISKETTE	2-5
2-1b	A DISKETTE	2-5
3-1	DEVELOPMENT COMPUTER PC BOARD LAYOUT	3-2
3-2	DEVELOPMENT COMPUTER (TOP VIEW)	3-3
3-3	COMPUTER FRONT PANEL	3-7
3-4	COMPUTER REAR PANEL	3-9
3-5	INSERTING A DISKETTE	3-11
5-1	A SAMPLE SOURCE PROGRAM	5-3
5-2	ENTERING TEXT AND DISPLAYING THE BUFFER	5-5
5-3	FIND< SUBSTITUTE AND REPLACE COMMANDS	5-6
5-4	DISPLAYING THE BUFFER AND FILING	5-7
5-5	DOUBLE PRECISION ADD AND SUBSTRACT	5-8
5-6	ADDING DATA TO AN EXISTING FILE	5-10
5-7	INSERTING LINES INTO THE BUFFER	5-13
6-1	SAMPLE PROGRAM	6-3
6-2	SAMPLE ASSEMBLY LISTING	6-4
8-1	SAMPLE PROGRAM	8-5
8-2	LOADING OBJECT CODE AND DEBUGGER INITIALIZING SLAVE REGISTERS	8-6
8-3	SINGLE STEP TRACE ALL MODE	8-8
8-4	TRACE ALL MODE	8-11
8-5	USING BREAKPOINTS	8-10
8-6	USING THE TRACE ALL JUMPS MODE	8-12
8-7	CLEARING BREAKPOINTS	8-14
8-8	TERMINATING A DEBUG SESSION	8-14
TABLE	TITLE	
4-1	DEVICE NAMES	4-5
4-2	SDOS ERROR MESSAGES	4-33
4-3	SDOS SYSTEM PROGRAM IDENTIFIERS	4-35
8-1	DEBUG COMMANDS	8-3
8-2	TRACE TABLE MNEMONICS	8-24

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 minicomputer 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 the TWIN user.

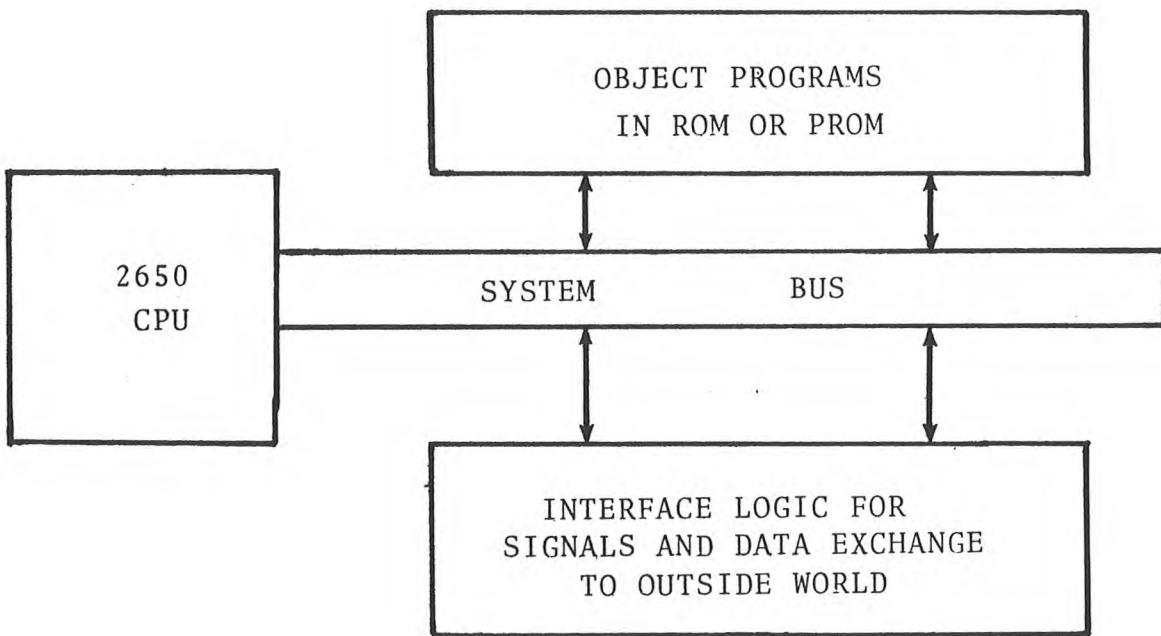


FIGURE 1-1
ELEMENTARY PARTITIONING OF 2650
MICROCOMPUTER SYSTEM LOGIC

/// TWIN SYSTEM

/// USER SYSTEM

⊗ DIRECT SIMULATION

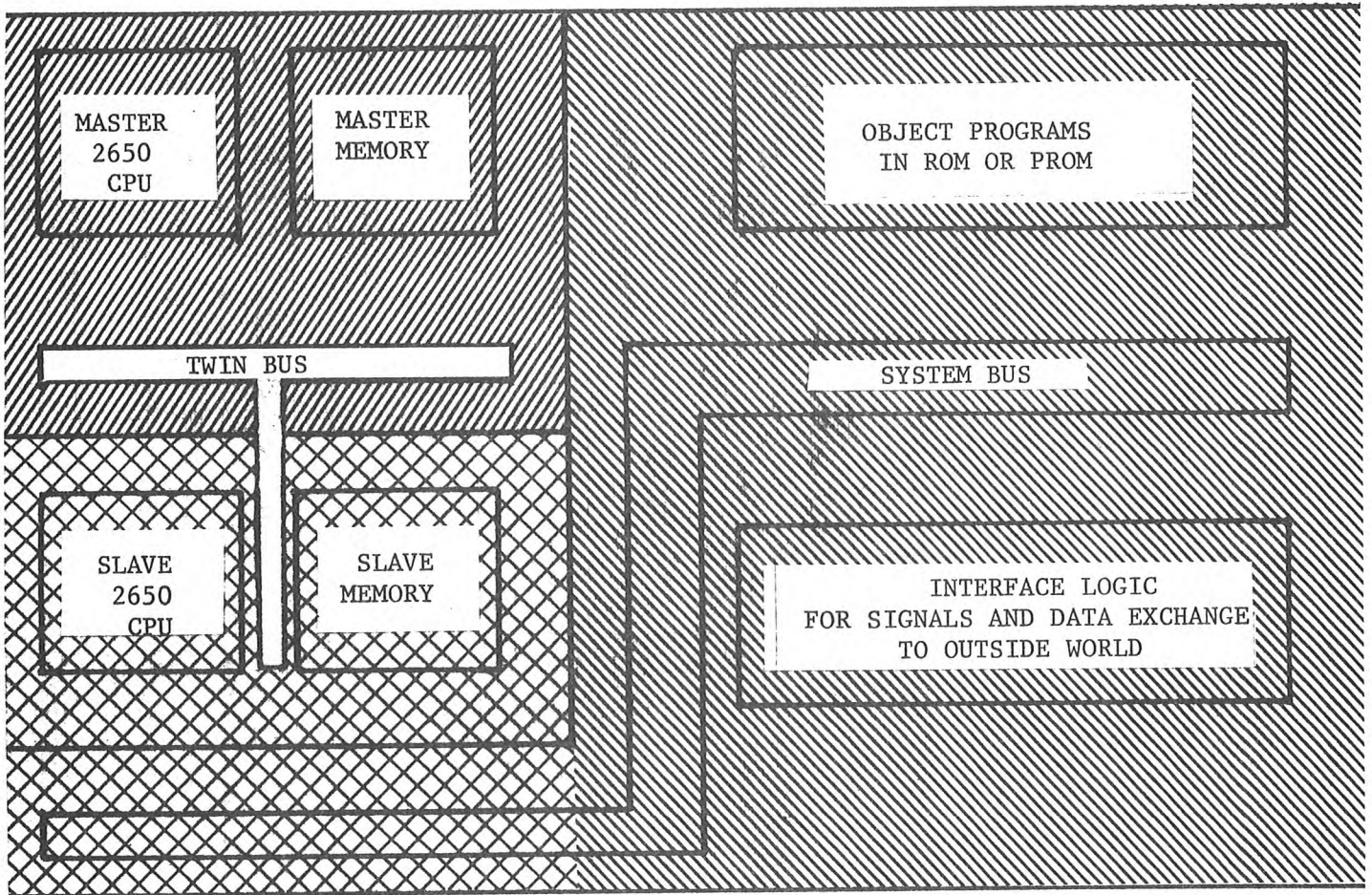


FIGURE 1-2

TWIN SLAVE CPU

EMULATES USERS 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 Assembler and how it is used to create object programs from assembly language programs. Chapter 7 gives procedures for programming type 1702A and type 82S115 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. This system is used to create and edit assembly language source programs, to assemble source programs into object code, and to execute object programs. User's object programs may be executed out of TWIN memory, or by using the TWICE interconnecting cable assembly, object programs may be executed out of external memory that is part of an end product. 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 and 16K bytes of slave memory, and a dual drive floppy disk unit; peripherals include a CRT terminal and a line printer. Options available include additional floppy disk units, additional memory, 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, Assembler 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 become 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 the System Reference Manual.

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. Two system programs, the Editor and the Assembler, plus a small Debug trace 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, after the user has a completely assembled

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

and debugged program. A front-panel switch turns off PROM programmer power so that devices cannot be damaged during insertion and removal.

DEBUG HARDWARE

The Debug circuitry is the 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 microprocessor 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 an eight 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 disc operating system reserves track 0 for the disc directory, and tracks one through four are normally automatically reserved for a 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 4.4.4).

2.2 PERIPHERALS

Peripherals compatible with the system include a CRT terminal with a full ASCII keyboard, a line printer, an ASR-33 Teletypewriter, and a paper tape reader. In addition, the GPI/O 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 offline.

A DISKETTE

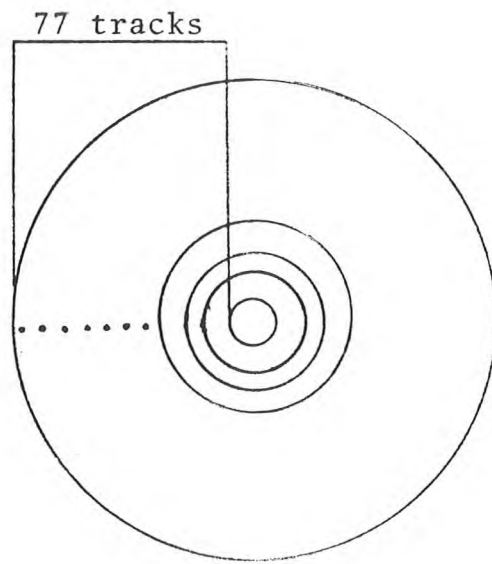


FIGURE 2-2a

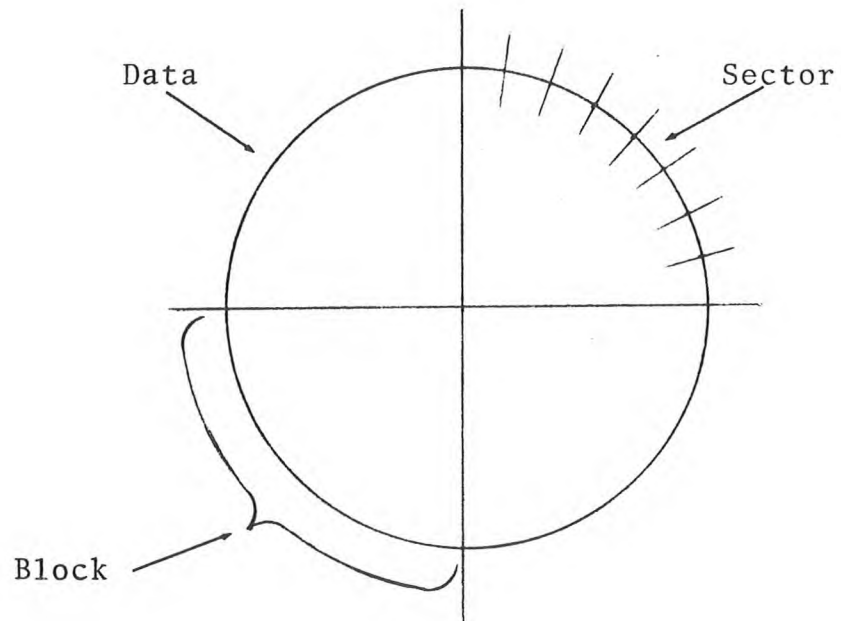


FIGURE 2-2b

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 GPI/O card, or any 8-bit parallel device to one of the four parallel ports on the GPI/O card. If these peripherals are to interface to the operating system, the addition of a software driver to control the device is required. This driver is added to the TWIN software using the method described in the TWIN System Reference Manual.

2.3 SOFTWARE

The TWIN development system software consists of SDOS, the Signetics Disk Operating System, and its associated commands. Three SDOS commands 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.

There are two other features of SDOS that deserve mention. There are the Debug Monitor and the PROM programming capability.

2.3.1.1 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 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 section of the Debugger is placed in slave memory to make slave CPU registers available to the Debugger for examination and modification.

2.2.1.2 PROM PROGRAMMING

SDOS provides a series of commands that allow PROMs to be read, written and compared with slave memory. All these commands apply to the PROM sockets located in the front panel.

2.3.2 THE EDITOR

After a source program is conceived and designed, it is then input to the TWIN system through the use of 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.

2.3.3. THE ASSEMBLER

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 the Assembler, which stores the object code it has assembled from the source program on mass storage.

The Assembler runs in slave memory using the slave CPU. The Assembler uses the available part of slave memory for I/O buffers and to create its symbol tables. SDOS handles all the Assembler's I/O requests.

2.3.4 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.

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

FIGURE 3-1
DEVELOPMENT COMPUTER PC BOARD LAYOUT

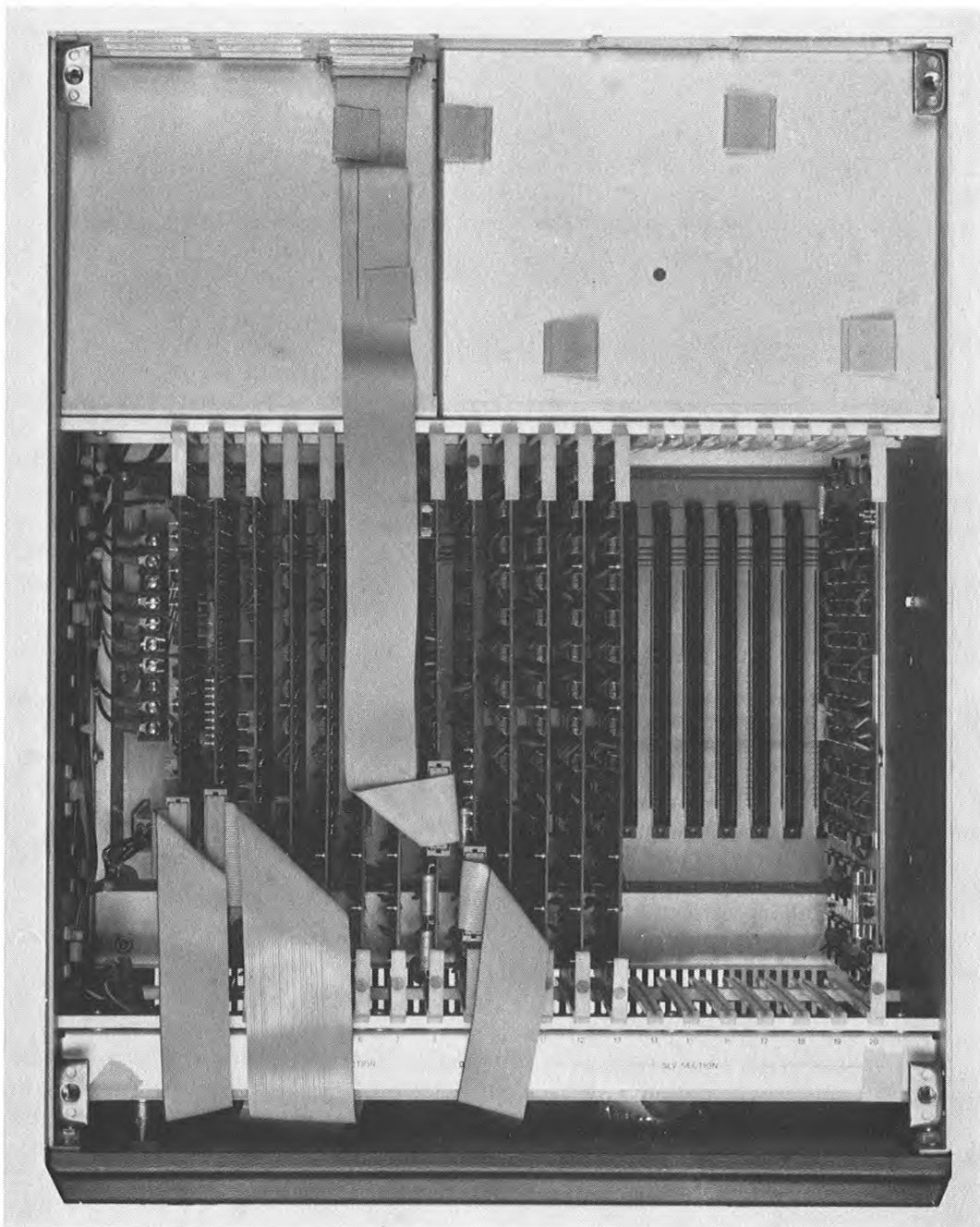


FIGURE 3-2
DEVELOPMENT COMPUTER (TOP VIEW)

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

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 amperes at 115 VAC, 60 Hz 1.6 amperes at 230 VAC, 50 Hz
CRT Terminal:	2 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.
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 re-load 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.
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 sockets.
6. PROM programming sockets. The leftmost socket (PROM1) is used for programming type 1702A MOS PROMs. The center socket (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.

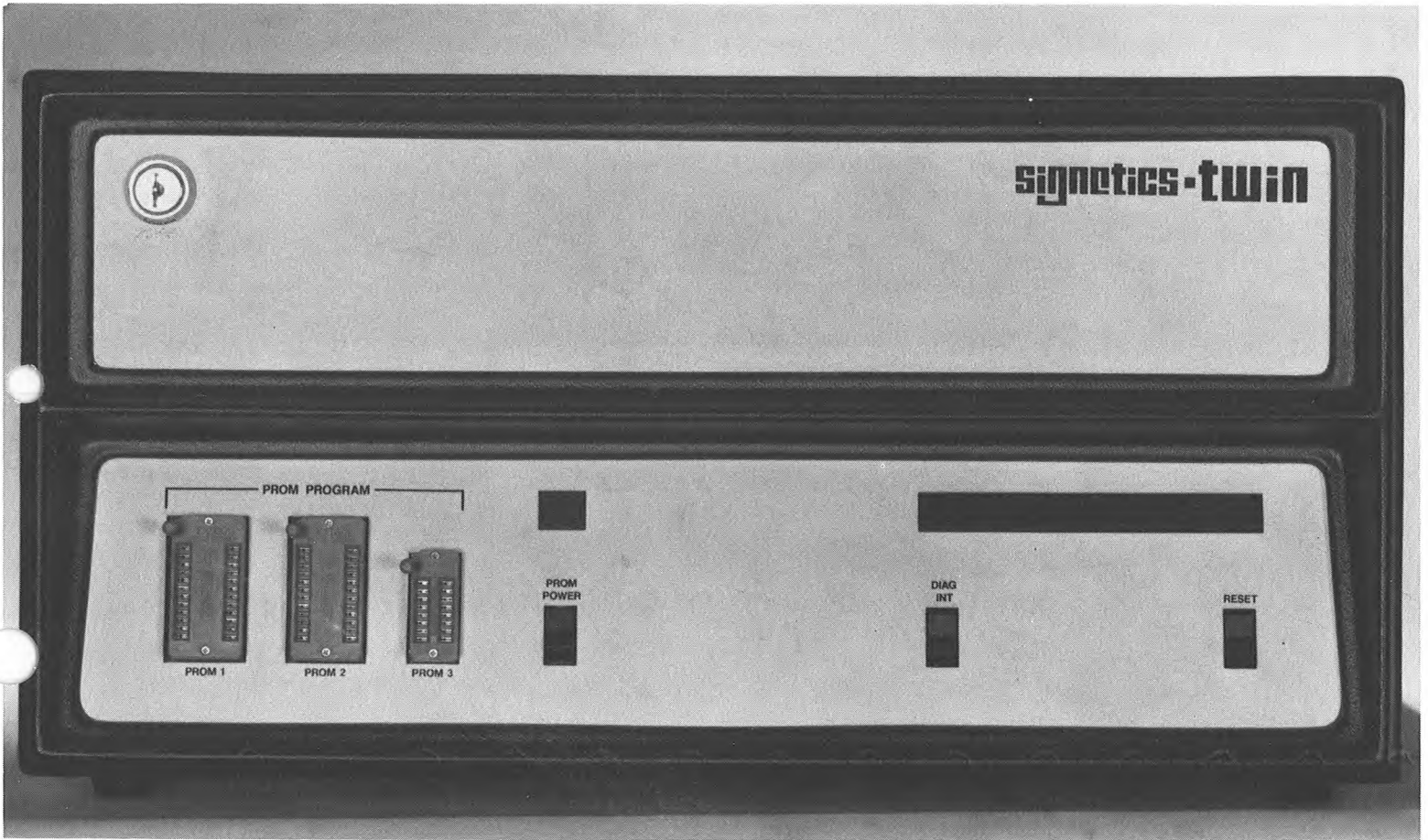


FIGURE 3-3
COMPUTER FRONT PANEL

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.
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.

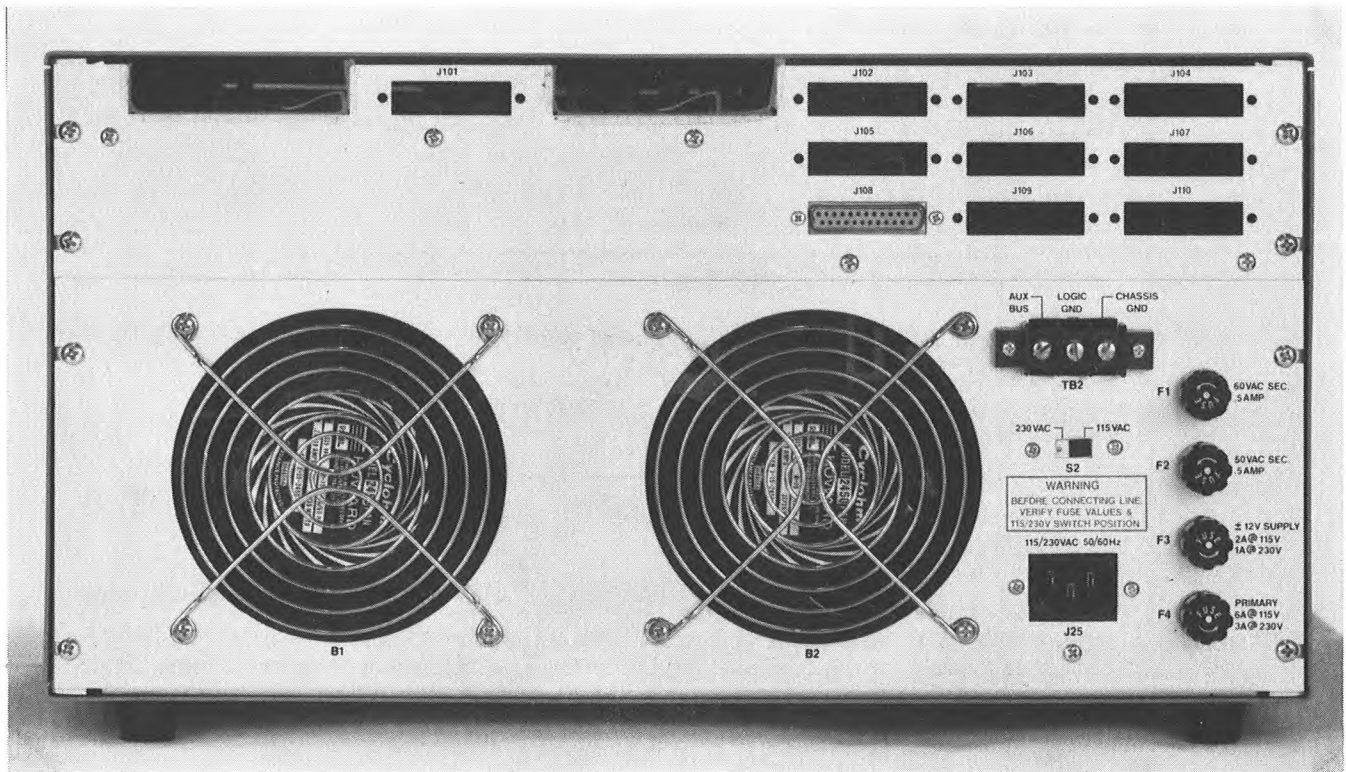


FIGURE 3-4
COMPUTER REAR PANEL

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.
- 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. When SDOS has been loaded, a welcoming message will be displayed on the terminal:

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

The > is the SDOS prompt character which informs the user that SDOS is ready to accept commands.

If the welcoming message does not appear within 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

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

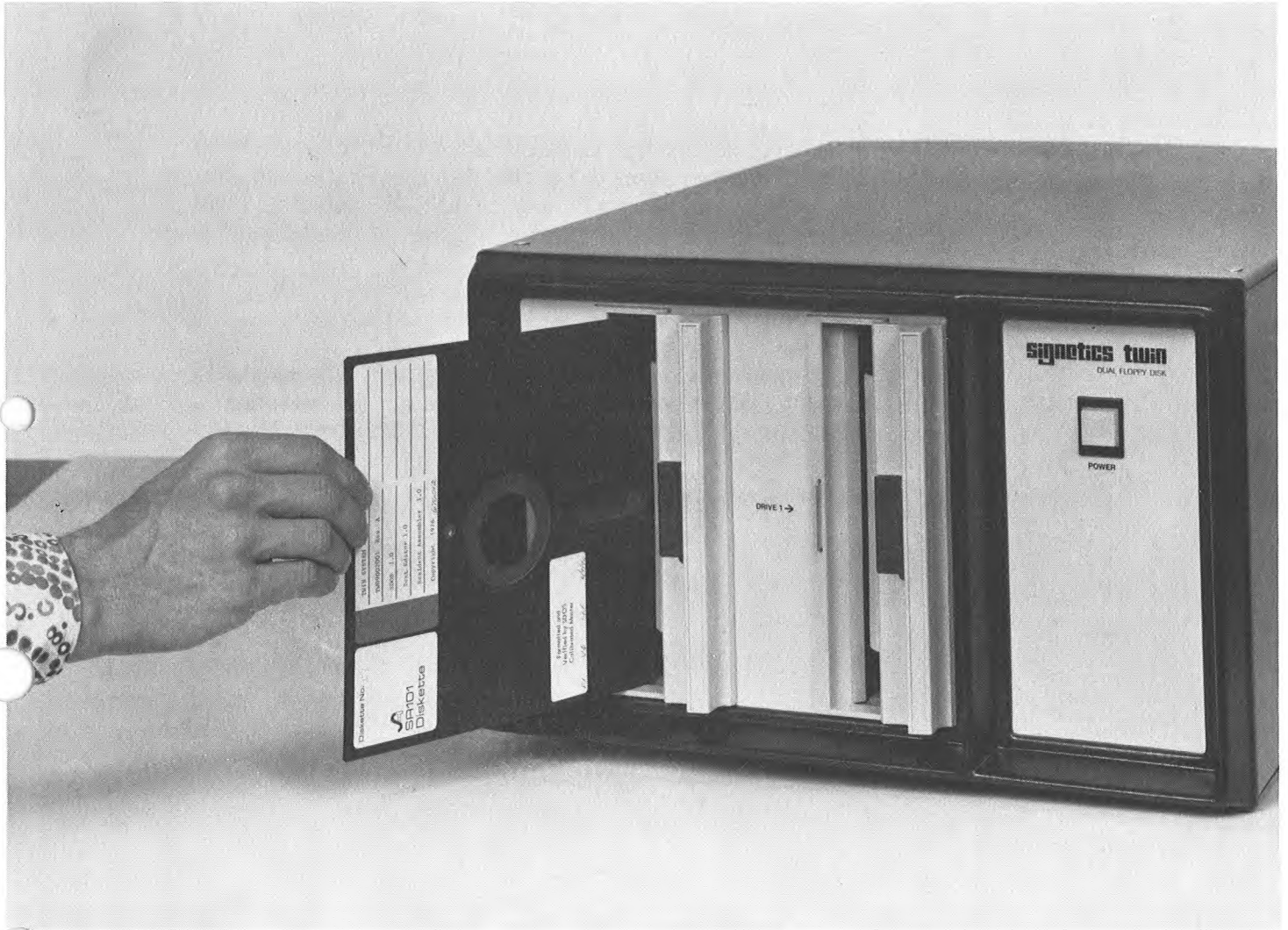


FIGURE 3-5
INSERTING A DISKETTE

service assistance. 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. Refer to the TWIN System Reference Manual for information on changing the baud rate on the Master CPU card.

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.

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 for 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.

Before a description of SDOS is presented, several concepts, pertaining to diskettes and the disk drives, require explanation:

- 1) Before any new diskette is used, THE NEW DISKETTE MUST BE FORMATTED AND VERIFIED. All diskettes purchased from Signetics are pre-formatted and verified and therefore do not require these operations before use. If a diskette must be formatted, follow the procedures outlined in the discussion of the SDOS commands FORMAT and VERIFY (Section 4.4.4).
- 2) On every diskette, there is a write protect slot (see Figure 3-5). If this slot is covered, the diskette is write enabled, meaning that data may be written onto the diskette. Data may also be read from the diskette. If this slot is not covered, the diskette is write protected, meaning that data may not be written onto the diskette, but may only be read from the diskette. If an attempt is made to write to a write-protected disk, the appropriate SDOS error message will be displayed.
- 3) In a typical TWIN system, there are two disk drives.

Drive 0 is usually the system drive. The system drive is the drive which SDOS accesses when it must load a system program. The system drive is also the drive used when a drive number is not specified with a file name. The diskette loaded on the system drive is known as the system diskette and must contain the system programs. 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,

since it may be used as a scratch area, is not write protected.

4.1 ENTERING SDOS COMMANDS

When the prompt character, > , is displayed, the user is allowed to enter commands to SDOS. These commands will all have a similar format. The format is:

> command parameters (r)

where:

command is the name of the command to be executed;

parameters is the required or optional list of parameters for the specified command; and

(r) is the RETURN key

The command is always separated from its parameters by one or more spaces or by a comma.

For example, if the user entered the underlined portion of the following line:

> LDIR 0(r)

LDIR would be the command to be executed and 0 would be the parameter for LDIR. When one presses the RETURN key, SDOS is told that a command is waiting to be interpreted. SDOS identifies the command and loads the appropriate program into master memory. Control is passed to the loaded program to perform the requested function. In the LDIR 0 example, the command LDIR, which is the List Directory command, is identified by SDOS, and results in the List Directory program being loaded and executed. The parameter, 0, specifies the drive whose directory will be listed. The listing will be displayed on the console.

If one desired a listing that included the system files, the following entry should be made:

> LDIR 0 .(r)

where:

LDIR is the command, and

0 and . (which requests that system files be included in the directory listing) are the parameters.

Note that a space separates or delimits the two parameters. When two or more parameters are present in a command line, they must be separated by spaces, or by a comma (,). Since the comma is also a delimiting character, the following command line is interpreted by SDOS in the same way as the above example:

```
> LDIR, 0,.(R)
```

The space and comma can be used as delimiters in the same command line.

4.2 SPECIAL KEYS

SDOS pays special attention to certain keys in order to facilitate the entry of command lines and operator control of the system. These keys are DELETE (or RUBOUT), ESCAPE, and the space bar. SDOS also recognizes CTRL-Z as a special character.

Suppose the operator was entering the command LDIR 0,. discussed in the previous section, miskeyed, and instead entered:

```
>LDK
```

To remove the incorrect character, 'K', from the buffer, the DELETE key is used. One depression of the DELETE key deletes the last character in the buffer, and echoes that character to the console. While the console displays:

```
>LDKK
```

the buffer contains LD. The entry of the command line can then be completed as if the K was never entered.

Suppose the error in the entry was of this nature:

```
>;DIR 0
```

and as the operator prepares to enter the delimiting character, notices that ";" was entered instead of "L". Rather than pressing DELETE six times to reach the incorrect character, the operator may delete the entire line through the use of the ESCAPE key.

Pressing the ESCAPE key during entry of a command line can result in different SDOS responses depending on the current system mode. The possible system input modes are:

- a) Input is being performed for an SDOS command line;

- b) Input is being performed for the Text Editor;
- c) Input is being performed for a user application program.

No matter which of these modes the system is operating in, the current input line will be deleted.

If command input for SDOS is being performed, which is the case in the ;DIR 0 example, the system will delete the current command line and then respond with a double prompt, >>. An exception to this rule occurs when the EXAM command is being performed. If the ESCAPE key is pressed while EXAM is being performed, the memory locations which were altered prior to the key depression will remain altered.

If the Text Editor is running, the response will be the Editor prompt character (*), except if the Editor is in the INPUT mode, in which case no prompt character will be displayed.

The system response when a user application program is running will depend on what the user has programmed as a response.

The system response to a depression of the ESCAPE key when an SDOS or user program is executing, as distinguished from console input being performed, is discussed in Section 4.4.2, System Control Commands.

The space bar (key) allows the user to control system output to the console. Suppose the user has completed entering the LDIR 0, . (R) command and the system is listing the directory on the console. Depressing the space bar once will temporarily pause output to the console and allow the user to examine the directory before it "scrolls" off the top of the CRT. Depress the space bar once again and the listing will resume.

CTRL-Z, which is formed by holding the CTRL (CONTROL) key down while pressing Z, is treated as an end-of-file character when an ASCII read is being performed from the console or other system input device.

4.3 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.

A file is a 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 the 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. Disk files can be accessed through their logical beginning address, a map that indicates where the data in the file is located on the disk, and a logical ending address.

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

TABLE 4-1 DEVICE NAMES	
DEVICE NAME	DEVICE
CONI	CONSOLE INPUT
CONO	CONSOLE OUTPUT
LPT1	LINE PRINTER 1
LPT2	LINE PRINTER 2
HSPT	HIGH SPEED PAPER TAPE READER
TTYR	TELETYPE READER

For example, the command:

```
COPY TTYR LPT1
```

would copy the information from the teletype paper tape reader to the line printer.

<p style="text-align: center;"><u>NOTE</u></p> <p>ALTHOUGH SDOS SOFTWARE SUPPORTS A HIGH SPEED PAPER TAPE READER, THIS PERIPHERAL IS NOT CURRENTLY AVAILABLE FOR THE TWIN SYSTEM.</p>

Files may also be viewed as devices. Files can be specified as input or output devices. To refer to a file as a device, the operator 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 may be necessary to specify the drive on which the file is located. SDOS can automatically create the necessary new files or search other diskette directories.

A filename has the following properties:

- 1) The filename 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)
These special characters: !, ", #, %, &, ', (,), *, ;, =, and ?.
- 3) The filename may not begin with a numeric character.
- 4) The filename must not be one of the reserved names which identify physical devices: CONO, CONI, LPT1, LPT2, HSPT or TTYR.
- 5) The filename is unique to the diskette containing the file.

Every diskette has a system area, called the directory, where system information is kept on all the files on the diskette. This information includes the filename, disk sectors used, beginning and ending disk addresses, etc. The directory also includes system information which prevents bad disk sectors from being allocated for file usage.

SDOS is only aware of diskettes that are loaded in the available disk drives. For this reason, diskettes are not referred to by diskette name; rather, they are referred to by drive number. 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 was 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 specify the drive, append the drive number to the file name. This is done by following the filename with a '/' to separate the filename and drive and then inserting the drive number. To copy DATA1 on drive 1 to the line printer, the following command would be performed:

```
COPY DATA1/1 LPT1
```

If no drive number is appended to a filename, SDOS normally assumes that the file resides on the system drive, and will search the system drive directory for the file. See the SEARCH command for an alternative mode.

Channels are used by the program running on the slave CPU. The user can assign a channel to a device using the ASSIGN command. When this is accomplished, the slave is able to perform input or output to the device through the channel. The devices specified in the assignment may be physical devices or files.

4.4 THE SDOS COMMANDS

This section provides a description of all SDOS commands with the following exceptions:

- * Commands that are primarily used in conjunction with the COMMAND FILE facility are described in Section 4.6.
- * The EDIT command is described in Chapter 5.
- * The ASM command is described in Chapter 6.
- * Commands that are used for PROM programming and verification are described in Chapter 7.
- * Commands that are associated with the Debug function are described in Chapter 8.

4.4.1 THE SDOS COMMAND DESCRIPTIONS

All SDOS command line specifications are enclosed in this fashion:

```
command parameter
```

The command line descriptions are followed with a description of their function. Most descriptions proceed as follows:

- 1) The command line is presented. Parameters that are optional are enclosed in parentheses. Three periods (...) indicate that the preceding parameter may be repeated as many times as the limitations of the command allow. The minimum characters required to initiate the command are underlined. For example:

```
COPY INPUT (...INPUT) OUTPUT
```

COP INPUT OUTPUT, where INPUT and OUTPUT are two filenames, is the minimum COPY command that will be executed. Additional INPUT files may be specified as in COP INPUT1 INPUT2 INPUT3 OUTPUT.

- 2) The first sentence provides a brief description of the command's function.
- 3) The parameters associated with the command are discussed. The effect of parameters on execution and their default values, if any, are described.

- 4) If further discussion of the command is necessary, the reasoning behind the command, its logic flow, or possible problems will be analyzed in the next paragraph.
- 5) The error messages that the command might display are listed. The format and a list of SDOS error messages is presented in Section 4.7.

In the command line specification, several terms and conventions are used. The terms and conventions involved are NAME, CH, DEVICE, ADDRESS or Ai, FILENAME, D and L.

NAME refers to a program name. For example, ABORT NAME requests that the program NAME be aborted. If the program VAIL was to be aborted, ABORT VAIL would be used.

CH refers to a channel number. Channel numbers may be in the range 0 - 7. For example, if channel 2 were to be assigned to the line printer, 2 would replace CH and LPT1 would replace DEVICE in the ASSIGN CH DEVICE command. This would result in ASSIGN 2 LPT1 being executed.

DEVICE refers to any of the system devices or to any disk files. For example, if channel 3 were to be assigned to the disk file SRCCD/1, 3 would replace CH and SRCCD/1 would replace DEVICE in the ASSIGN CH DEVICE command. This would result in ASSIGN 3 SRCCD/1 being executed.

ADDRESS

or Ai refers to a hexadecimal address constant between 0 and FFFF. For example, MODULE FILENAME, A1, A2, A3 could be replaced with MODULE LDFLE, 100, 2FFF, 80.

FILENAME refers to a disk filename. To edit the file DTA1/1 using the Editor, the user would replace EDIT FILENAME with EDIT DTA1/1.

- D refers to the disk drive number. To duplicate the diskette on drive 0 on the diskette on drive 1, 0 would replace D1 and 1 would replace D2 in the DUP D1 D2 command. This would yield a DUP 0 1 command.
- L refers to a line number. To list the 8th through 14th lines of a file name DTA1/1 on the line printer, the user would replace PRINT FILENAME (L1L2) with PRINT DTA1/1 8 14.

4.4.1.1 SDOS COMMAND COMPLETION

Most SDOS commands indicate that they have completed their function by displaying an End-of-Job message. The form of this message is *id* EOJ where 'id' is the SDOS system program identifier (see Table 4-3) and EOJ is the end of job message. Completion of any user-entered command causes the SDOS prompt character '>' to be displayed.

4.4.2 SYSTEM CONTROL COMMANDS

The user may control system or slave programs through these special keys:

ESC
SPACE BAR

The ESC ESC sequence is used to suspend system or slave programs and to return control to SDOS. The SPACE BAR key is used to control SDOS displays.

The user may also control the execution of system or slave programs and control the slave channels with these commands:

SUSPEND
CONT
ABORT
ASSIGN
CLOSE

SUSPEND halts program execution. CONT restarts suspended programs. ABORT terminates program or command execution. ASSIGN forms a connection between a slave channel and a device. CLOSE terminates the logical connection formed in an ASSIGN command.

ESC
or
ESC ESC

A single depression of the ESCAPE key has two possible interpretations:

- a) If input was being performed to SDOS, the Editor or an application program, refer to Section 4.2 for a discussion of the actions taken.
- b) If an SDOS or user program is executing, a single depression of the ESCAPE key will result in that program being temporarily suspended, unless the program is one of the following four SDOS programs:

LDIR
TRACE
STATUS
DUMP

If one of these four programs is executing, a depression of the ESCAPE key will terminate its execution. To restart any of the other SDOS programs or the user program, either press RETURN or enter a valid SDOS command.

When the ESCAPE key is hit, SDOS will respond with a double prompter to record the fact, unless a command line is being input to the Editor or to a user application program.

Two consecutive depressions of the ESCAPE key will result in all active programs in the system being suspended. No program suspended by this double depression of the ESCAPE key will resume execution unless the user issues a CONT (Continue Execution) command.

SPACE BAR

The space bar key is discussed in Section 4.2.

SSPEND NAME
or
SSPEND *
or
SSPEND /

This command suspends the execution of active programs. The DEBUG program may not be suspended.

SUSPEND NAME suspends the specified program. SUSPEND * suspends all active programs. SUSPEND / suspends the slave program.

The primary use for this command is in conjunction with the COMMAND FILE capability discussed in Section 4.6. Inserting this command in a COMMAND FILE will suspend system operation to allow some required user action, such as inserting a special diskette into one of the drives.

SUS ERROR RESPONSES

- 24 - JOB NOT ACTIVE
- 26 - JOB ALREADY SUSPENDED
- 31 - PARAMETER REQUIRED

CONT NAME
or
CONT *
or
CONT /

This command continues the execution of a suspended program.

CONT NAME causes the specified program to be continued. CONT * causes all suspended programs to be continued. CONT / continues the slave program.

A program may be suspended in one of two ways. 1) If the ESCAPE key is depressed twice in succession, SDOS will have suspended all programs. 2) The user may suspend programs through the use of the SUSPEND command.

CON ERROR RESPONSES

- 24 - JOB NOT ACTIVE
- 25 - JOB NOT SUSPENDED
- 31 - PARAMETER REQUIRED

ABORT NAME
or
ABORT *
or
ABORT /

This command causes an active SDOS or user program to be aborted.

ABORT NAME causes the specified program to be aborted. ABORT * causes all active programs to be aborted. ABORT / causes the slave program to be aborted.

All ABORT commands close the channels used by the programs that are aborted.

ABT ERROR RESPONSES

- 24 - JOB NOT ACTIVE
- 31 - PARAMETER REQUIRED

CLOSE CH (...CH)

This command causes the specified channels to be closed. The channel numbers must be in the range 0-7.

The logical connection between channel and device that was created in the ASSIGN command is severed, and the channel and device are no longer logically related. If the channel was assigned to a disk output file, the data remaining in the SDOS deblocking buffer will be output to the file before it is closed.

CLS ERROR RESPONSES

- 2 - DIRECTORY WRITE ERROR
- 7 - DEVICE WRITE ERROR
- 19 - INVALID CHANNEL NUMBER
- 31 - PARAMETER REQUIRED
- 62 - DEVICE NOT OPERATIONAL
- 64 - INVALID DISKETTE

ASSIGN CH DEVICE (...CH DEVICE)

This command causes the connection of the logical slave channel CH to the specified DEVICE. CH must be in the range 0-7. DEVICE must be one of the system device names or the name of a disk file.

The ASSIGN command views every disk file 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 filename. If the filename is not found, the file is created in the directory.

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

The ASSIGN command applies to the user channels only.

ASN ERROR RESPONSES

- 1 - DIRECTORY READ ERROR
- 9 - INVALID DRIVE NUMBER
- 12 - INVALID FILE NAME
- 19 - INVALID CHANNEL NUMBER
- 20 - CHANNEL IN USE
- 21 - CHANNEL ASSIGN FAILURE
- 31 - PARAMETER REQUIRED

4.4.3 SYSTEM OPTIONS

The user may set the value of various system options with these commands:

SEARCH
SYSTEM
DEVICE

SEARCH allows the user to invoke the automatic file searching system. SYSTEM allows the user to designate the system drive. DEVICE informs SDOS of device status.

SEARCH <u>ON</u> (N) or SEARCH <u>OFF</u>

This command turns the automatic file searching flag, SEARCH, ON or OFF. The default value of SEARCH is OFF. N specifies the number of drives in the user system. The default value of N is 2. If N is given, it must be greater than the number of the system drive.

If automatic file searching is not being performed, i.e., SEARCH is OFF, then when the user specifies a filename, SDOS only searches the directory of the specified drive for the file. (If no drive is specified, the default value is the system drive.)

If automatic file searching is being performed, i.e., SEARCH is ON, then when the user specifies a filename but not a drive number, SDOS will search, in circular fashion, N directories, beginning with the system diskette, for that filename. If the filename is not found, it will be created on the first diskette which can contain a file. If that diskette is write protected, a directory write error will result.

This feature is very useful when drive 0 is a write protected system diskette and all user files are on drive 1.

SCH ERROR RESPONSES

- 30 - INVALID PARAMETER
- 31 - PARAMETER REQUIRED

SYSTEM D

This command designates drive D as the system disk drive.

This command allows the user to designate any disk drive attached to the system as the system drive.

The default value for the system drive is 0.

SDOS ERROR RESPONSES

9-INVALID DRIVE NUMBER

DEVICE DEVICE U
or
DEVICE DEVICE D

This command informs SDOS of the availability of a peripheral device. The argument DEVICE must be one of the system device names (see Section 4.3).

If U is specified as the second argument, the system is informed that the device is UP, or available for use. If D is specified as the second argument, the system is informed that the device is DOWN, or not available for use. Either U or D must be specified.

DEV ERROR RESPONSES

- 30 - INVALID PARAMETER
- 31 - PARAMETER REQUIRED
- 52 - INVALID DEVICE

4.4.4. SYSTEM UTILITIES

The user can perform disk and file maintenance and move data around the TWIN system with these commands:

FORMAT
VERIFY
RENAME
DUP
LDIR
DELETE
COPY
PRINT

FORMAT initializes the diskette for use by the TWIN system. VERIFY determines if bad blocks exist on the disk and catalogs the location of the bad blocks. RENAME changes the name of a disk file or changes a disk identification. DUP duplicates diskettes. LDIR lists the directory of a specified diskette. DELETE removes files from the disk. COPY copies data from one part of the system to another. PRINT outputs the contents of a disk file on an appropriate device.

FORMAT D (IDENT)

This command causes the diskette on drive D to be formatted. The ASCII character string IDENT is used to identify the diskette. D may not be the designated system drive. IDENT is truncated if it is longer than 48 characters.

The formatting process is primarily performed by the floppy disk controller and involves writing clock bits, sync patterns, the track and sector number, a data pattern and a CRC character on every sector of the diskette. During the formatting process, the directory is preset to indicate that tracks 1 through 4 are in use. This serves to reserve those tracks for SDOS. If a bad sector is detected on tracks 0 through 4 (the directory and SDOS area) the formatting process is aborted. If the diskette will not be used for storage of system software, the area reserved for SDOS may be freed for use by use of the DELETE command. This, however, will prevent ever using this diskette as a system disk.

When the formatting process is completed, the ASCII character string IDENT is written to the diskette and serves as the diskette identification. This identification is displayed when the LDIR command is used to list the diskette directory. Note that if IDENT is not specified, a string of blanks will be used to identify the diskette.

VIRGIN DISKETTES MUST BE FORMATTED AND VERIFIED BEFORE THEY CAN BE USED BY SDOS.

FMT ERROR RESPONSES

- 2 - DIRECTORY WRITE ERROR
- 9 - INVALID DRIVE NUMBER
- 17 - OUTPUT DEVICE ASSIGN FAILURE
- 18 - DEVICE IN USE
- 47 - SYSTEM AREA BAD

VERIFY D

This command causes the diskette on drive D to be verified.

The verification process consists of reading every sector on the diskette and noting all the errors that occur. If, when a sector is read, an error occurs, bits corresponding to all the logical blocks on the track which contains the bad sector are set in a Bad Block Bit Map. In addition, the track and sector number of the defective sector are output to the console. When all the sectors have been read, the Bad Block Bit Map is written on the diskette. Whenever files are created and disk space allocation for the file is performed, reference will be made to the Bad Block Bit Map and the defective blocks will not be allocated.

If a defective sector is detected on any of tracks 0 through 4 (the SDOS system area) during the verification process, the process will be aborted and an appropriate message will be 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
- 47 - SYSTEM AREA BAD

RENAME OLDFILE/D NEWFILE
or
RENAME D IDENT

The RENAME function has two forms. The first form renames the file OLDFILE to NEWFILE. This form requires that a drive number be specified with OLDFILE. If a drive number is specified with NEWFILE, it must be the same as the drive number specified with OLDFILE.

The second form reidentifies the diskette on drive D with the character string IDENT. When the string IDENT is used in the second form, IDENT will be truncated if it is longer than 48 characters.

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

DUP D1 D2 (IDENT)

This command causes the diskette on drive D1 to be copied to the diskette on drive D2. Diskette D2 is identified by the character string IDENT. D1 may not be the same as D2, and D2 may not be the system drive. IDENT will be truncated if it is longer than 48 characters.

D1 is copied to D2 by copying all the files on D1 to D2. In the event of a disk read or write error during a file copy, the output file will be deleted on D2, a warning message will be displayed, and the DUP process will continue with the next file.

The diskette on drive D2 should be verified before the DUP command is executed. This is done to establish the Bad Block Bit Map for the diskette.

***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
- 16 - INPUT DEVICE ASSIGN FAILURE
- 17 - OUTPUT DEVICE ASSIGN FAILURE
- 21 - CHANNEL ASSIGN FAILURE

LDIR (D) (.) (/) (DEVICE)

This command lists the contents of the directory of the diskette on drive D on DEVICE. If D is not specified, the directory of the system diskette will be listed. If '.' is specified, the SDOS system files will be included in the directory listing. If '/' is specified, diskette space allocation information will be listed for each file in the directory. ~~And~~ A summary of the total diskette utilization will follow at the end of the directory listing. If DEVICE is not specified, the listing will be displayed on the console.

DIR ERROR RESPONSES

- 1 - DIRECTORY READ ERROR
- 7 - DEVICE WRITE ERROR
- 10 - OVERLAY LOAD FAILURE
- 15 - INVALID OUTPUT DEVICE
- 17 - OUTPUT DEVICE ASSIGN FAILURE

DELETE FILENAME/D (...FILENAME/D)

This command deletes all the filenames specified in its parameter list. Each filename must have a drive number associated with it. Each file specified in the parameter list will be deleted from the directory of the disk on which it resides, and the sector blocks allocated to the file will be released for reallocation.

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

COPY INPUT (...INPUT) OUTPUT

This command copies INPUT data to an OUTPUT file or device. INPUT is a disk file or an input device. OUTPUT is a disk file or an output device.

If COPY INPUT OUTPUT is specified, data is copied from the specified INPUT device or file to the specified OUTPUT device or file until an end-of-file condition is encountered on the INPUT. If more than one INPUT is specified, the data is copied to the OUTPUT file in the following manner:

- 1) The first INPUT is copied to OUTPUT until the end-of-file condition is reached.
- 2) The second INPUT is then concatenated behind the first INPUT by copying its data to OUTPUT directly after the first INPUT.
- 3) The third INPUT is then copied after the second, etc.

The copy process is completed when the last INPUT is written to OUTPUT, and its end-of-file condition is reached. The OUTPUT file is then closed.

None of the INPUT files or devices may be the OUTPUT file or device.

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

COP ERROR RESPONSES

- 6 - INPUT READ ERROR
- 7 - OUTPUT WRITE ERROR OR EOD
- 13 - INPUT FILE NOT FOUND
- 14 - INVALID INPUT DEVICE
- 15 - INVALID OUTPUT DEVICE
- 16 - INPUT DEVICE ASSIGN FAILURE
- 17 - OUTPUT DEVICE ASSIGN FAILURE
- 30 - PARAMETER ERROR

PRINT FILENAME (DEVICE)(L1 L2)
or
PRINTL FILENAME (DEVICE)(L1 L2)

This command causes lines from FILENAME to be written to a specified output DEVICE. If DEVICE is not specified, the data is printed on LPT1. If L1 and L2 are specified, they must be greater than or equal to 1 and less than 32,768. L2 must be greater than or equal to L1.

If a line range is specified (L1 L2), only the lines from L1 through L2 will be printed. If only L1 is specified, the lines from the first line through L1 will be printed. If no line range is specified, the entire file will be printed.

If the PRINTL form is used, the lines will be numbered.

***PRN* ERROR RESPONSES**

- 6 - INPUT READ ERROR
- 7 - OUTPUT WRITE ERROR OR END OF DEVICE
- 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

4.4.5 OBJECT PROGRAM UTILITIES

The user may manipulate object program files with these commands:

MODULE
RHEX
WHEX
CSMS
WSMS

MODULE writes a binary load module from slave memory. RHEX reads a hexadecimal object file into slave memory. WHEX writes a hexadecimal object file from slave memory. CSMS translates an SMS file and then compares the file with slave memory. WSMS writes a block of slave memory in SMS format. SMS format is used by Signetics for the generation of PROMs and is described in Appendix D. Hexadecimal format is described in Appendix C.

MODULE FILENAME A1, A2, A3 (IDENT)

This command writes a binary load module to FILENAME. A2 must be greater than or equal to A1. IDENT is an optional character string used to identify the module. IDENT will be truncated after the first 20 characters entered.

The contents of slave memory from A1 to A2 will be output to the disk file FILENAME. The load module will be preceded by a 'header' which will contain the memory bounds A1 and A2 and the starting address of the program, A3.

MOD ERROR RESPONSES

- 7 - DEVICE WRITE ERROR
- 10 - OVERLAY LOAD FAILURE
- 12 - INVALID FILENAME
- 32 - TOO MANY PARAMETERS
- 34 - INVALID ADDRESS

RHEX (/BIAS) (DEVICE)

This command reads an absolute hexadecimal object file into slave memory. BIAS is used to alter the initial load address for the file. The default value of BIAS is 0. DEVICE is used to specify the input device or disk file where the object code resides. The default value of DEVICE is TTYR, the teletype paper tape reader.

The absolute hexadecimal file is read into memory from the specified input DEVICE. The initial load address is altered by BIAS which is a signed hexadecimal address constant. If no sign is specified, the default value is assumed to be +.

The program start address given at the end of the object file will be ignored by SDOS. It must be entered by the operator 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

WHEX A1 A2 ... (,,A1 A2) (A3) (DEVICE)

This command outputs an absolute hexadecimal file from slave memory. The pairs A1,A2 are hexadecimal address constants that indicate the memory to be written. A3 is an optional starting address. DEVICE is an optional output device or file. If DEVICE is not given, the default value is CONO, the console output device. If DEVICE is specified, then the starting address vector, A3, must be specified.

This command writes, in hexadecimal format, the data from A1 to A2 for each A1, A2 pair present in the parameter list. Note that 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

CSMS (ADDRESS) (DEVICE)

This command reads a file that is written in SMS format from DEVICE, translates the data to binary, and compares the data with slave memory. 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 is present. The default value of DEVICE is TTYR. 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, the SMS value, and the contents of the memory location will be displayed on the console.

***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

WSMS (ADDRESS) (DEVICE)

This command outputs a 512-byte block of slave memory in SMS format. 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 where the SMS data is to be written. The default value of DEVICE is CONO.

***SMS* ERROR RESPONSES**

- 7 - DEVICE WRITE ERROR
- 15 - INVALID OUTPUT DEVICE
- 21 - CHANNEL ASSIGN FAILURE
- 30 - INVALID PARAMETER
- 35 - INVALID ADDRESS

4.5 RESIDENT SDOS AND OVERLAY AREAS

SDOS is split into two sections: Resident SDOS which is always present in the master memory, and the SDOS overlays, which are loaded into the master memory automatically from the 'system' diskette upon invoking certain SDOS commands.

4.5.1 RESIDENT SDOS

The portion of SDOS which is resident in master memory includes the JOB DISPATCHER, the SVC PROCESSOR, all of the peripheral DEVICE HANDLERS, and four SDOS commands.

The JOB DISPATCHER performs all the command line interpretation and schedules all pending jobs. The SVC PROCESSOR handles any Supervisor Calls (SVCs) that are made by system and user programs. SVCs are described in detail in the System Reference Manual. The DEVICE HANDLERS are the programs which interface directly with the system hardware to perform I/O functions. The four SDOS commands that are resident are: GO, LOAD, SYSTEM and XEQ.

4.5.2 SDOS OVERLAYS

The SDOS overlays consist of all SDOS commands except the four memory-resident commands and the Editor and Assembler which run using the slave memory.

Master memory contains two overlay areas into which the SDOS overlays are loaded and executed. The overlay areas are referred to as Overlay Area 1 and Overlay Area 2. Some SDOS overlays 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 & 2</u>
COPY	RHEX	ABORT
DEBUG	RSMS	PATCH
DUP	VERIFY	ASSIGN
FORMAT	WHEX	RENAME
PRINT	WSMS	EKPT
RPROM	WPROM	RESET
CPROM		CLBP
		DSTAT
		CLOSE
		SET
		CONT
		SLAVE
		KILL
		SUSPEND
		DELETE
		TRACE
		DEVICE
		TYPE
		DUMP
		STATUS
		EXAM

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.

For example, suppose a paper tape was being read into slave memory. This would be accomplished using the RHEX command:

```
>R
```

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

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

When the (r) was entered, the RHEX command would be continued and the DEL command started. Note that RHEX executes in overlay area 1, while DELETE operates in overlay area 2, which allows the concurrent execution of these programs.

4.6 COMMAND FILES

SDOS provide 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 (we shall assign the name COM1) is used as an SDOS command:

```
>COM1(r)
```

SDOS first determines that COM1 is not one of the basic SDOS commands. SDOS then searches the system directory for the file COM1. When SDOS locates COM1, it treats the first line in COM1 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 COM1.

For example, suppose the Editor was used to create the following file named LISTALL:

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

If LISTALL is entered as an SDOS command, SDOS will locate LISTALL 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.

SDOS also allows for parameters to be entered in the command line with the COMMAND FILE filename. This is accomplished by allowing parameters following the COMMAND FILE filename to replace parameters beginning with a \$ in the COMMAND FILE file. For example, if LISTALL was changed to:

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

and the command:

```
>LISTALL . /
```

was entered the '.' (the first parameter) would replace all the \$1s in the LISTALL file and the '/' (the second parameter) would replace all the \$2s in the LISTALL file. This would result in the following command stream being performed:

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

If the command:

```
> LISTALL /
```

were executed, the '/' would replace all the \$1s in the LISTALL file, resulting in this 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.

In general, if COM is a COMMAND FILE and has \$1, \$2, \$3, ... \$n as parameters in the file, performing:

```
COM X1 X2 X3 ... Xi
```

will result in:

```
X1 replacing the $1s in the COM file
X2 replacing the $2s in the COM file
X3 replacing the $3s in the COM file
.
.
.
Xi replacing the $ns in the COM 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 4.6.1)

COMMAND FILES cannot be nested, but they can be chained. That is, if the last SDOS command in a COMMAND FILE is the name of another COMMAND FILE, the COMMAND FILE in progress will be terminated and the next COMMAND FILE will be started. Parameters can be passed from one COMMAND FILE to another in the same way they are passed to SDOS commands.

A maximum of six disk files instead of the normal eight can be assigned to a slave program while a COMMAND FILE is in progress.

4.6.1 COMMAND FILE UTILITIES

The user may control actions taken during command file execution with these commands:

```
KILL
TYPE
*
```

KILL ON
or
KILL OFF

This command causes the SDOS switch KILL to be set ON or OFF.

If the KILL switch is ON, a COMMAND FILE will be aborted if the current SDOS command being executed by the COMMAND FILE processor encounters an error. If the KILL switch is OFF, the COMMAND FILE processor will continue with the next SDOS command in the file.

The default value of the KILL switch is ON.

KIL ERROR RESPONSES

- 30 - INVALID PARAMETER
- 31 - PARAMETER REQUIRED

TYPE ON
or
TYPE OFF

This command causes the SDOS switch TYPE to be set ON or OFF.

If the TYPE switch is ON, SDOS command lines executed by the COMMAND FILE processor will be output to the system console. If the TYPE switch is OFF, SDOS command lines or 'EOJ' messages will not be displayed on the console. Error messages from SDOS programs are displayed regardless of the TYPE setting.

The default value of the TYPE switch is ON.

TYP ERROR RESPONSES

- 30 - INVALID PARAMETER
- 31 - PARAMETER REQUIRED

*** COMMENT**

This command is used to insert comments into the job flow. The * must be followed by a space or a carriage return. The ASCII string which follows the space cannot be longer than 77 characters. This command is effectively ignored by SDOS.

The primary use of the * command is in COMMAND FILES where it can be used to display comments around SDOS commands.

4.7 SDOS ERROR MESSAGES

All SDOS error messages are of the following form:

* id * ERROR #

where id is the SDOS system program identifier, Table 4-3, and error # is the SDOS error number, Table 4-2. For example,

WHX ERROR 30

is issued by the program WHEX, indicated by the SDOS program identifier, *WHX*, and informs the operator that an invalid parameter was received, indicated by the SDOS error number, 30.

TABLE 4-2
SDOS ERROR MESSAGES

- 1 - DIRECTORY READ ERROR
- 2 - DIRECTORY WRITE ERROR
- 3 - COMMAND FILE NOT FOUND
- 4 - COMMAND FILE INPUT ERROR
- 5 - PROCEDURE BUSY
- 6 - DEVICE READ ERROR
- 7 - DEVICE WRITE ERROR OR END-OF-DEVICE
- 8 - DRIVE NOT SPECIFIED
- 9 - INVALID DRIVE
- 10 - OVERLAY LOAD FAILURE
- 11 - OVERLAY AREA IN USE
- 12 - INVALID FILE NAME
- 13 - INPUT FILE NOT FOUND
- 14 - INVALID INPUT DEVICE
- 15 - INVALID OUTPUT DEVICE
- 16 - INPUT DEVICE ASSIGN FAILURE
- 17 - OUTPUT DEVICE ASSIGN FAILURE
- 18 - DEVICE IN USE
- 19 - INVALID CHANNEL NUMBER
- 20 - CHANNEL IN USE
- 21 - CHANNEL ASSIGN FAILURE
- 22 - COMMAND LINE BUFFER OVERFLOW
- 23 - INVALID COMMAND
- 24 - JOB NOT ACTIVE
- 25 - JOB NOT SUSPENDED
- 26 - JOB ALREADY SUSPENDED
- 27 - JOB EXECUTING
- 28 - JOB UNDER DEBUG CONTROL
- 29 - PROM POWER FAILURE
- 30 - INVALID PARAMETER
- 31 - PARAMETER REQUIRED
- 32 - TOO MANY PARAMETERS
- 33 - BIAS PARAMETER ERROR
- 34 - INVALID ADDRESS
- 35 - INVALID START ADDRESS
- 36 - INVALID END ADDRESS
- 37 - INVALID GO ADDRESS
- 38 - INVALID DEBUG SLAVE PROGRAM ADDRESS
- 39 - INVALID HEX CHARACTER
- 40 - INVALID RHEX INPUT FORMAT

- 41 - INVALID BREAKPOINT ACCESS MODE
- 42 - INVALID REGISTER PARAMETER
- 43 - INVALID DATA PARAMETER
- 44 - INVALID TRACE MODE PARAMETER
- 45 - INVALID SLAVE SRB ADDRESS
- 46 - SLAVE HALTED
- 47 - SYSTEM AREA BAD
- 48 - LOAD FILE NOT FOUND
- 49 - LOAD FILE ASSIGN FAILURE
- 50 - FILE NOT A LOAD MODULE
- 51 - INVALID LOAD REQUEST
- 52 - INVALID DEVICE
- 53 - INVALID SLAVE CPU
- 54 - INVALID MODE
- 55 - INVALID MEMORY
- 56 - INVALID DEVICE ADDRESS
- 57 - FILE NAME IN USE
- 58 - DEVICE ASSIGN FAILURE
- 59 - MEMORY WRITE ERROR
- 60 - END OF MEDIA
- 61 - FILE IN USE
- 62 - DEVICE NOT OPERATIONAL
- 63 - DIRECTORY FULL
- 64 - INVALID FILE NAME
- 65 - INVALID DISKETTE
- 66 - MASTER MEMORY PARITY ERROR
- 67 - SLAVE MEMORY PARITY ERROR

TABLE 4-3

SDOS SYSTEM PROGRAM IDENTIFIERS

ABT	ABORT OVERLAY
ASN	ASSIGN OVERLAY
CLS	CLOSE OVERLAY
CON	CONT OVERLAY
COP	COPY OVERLAY
DEB	DEBUG OVERLAY
DEL	DELETE OVERLAY
DEV	DEVICE OVERLAY
DIR	LDIR OVERLAY
DMP	DUMP OVERLAY
DOS	SDOS RESIDENT PROGRAM
DUP	DUP OVERLAY
EXM	EXAM OVERLAY
FMT	FORMAT OVERLAY
KIL	KILL OVERLAY
MOD	MODULE OVERLAY
PAT	PATCH OVERLAY
PRM	PROM OVERLAY
PRN	PRINT OVERLAY
REN	RENAME OVERLAY
RHX	RHEX OVERLAY
SCH	SEARCH OVERLAY
SLJ	PROGRAM RUNNING UNDER SLAVE CPU
SLV	SLAVE OVERLAY
SMS	SMS OVERLAY
SUS	SUSPEND OVERLAY
TYP	TYPE OVERLAY
VER	VERIFY OVERLAY
WHX	WHEX OVERLAY

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, e.g., 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 16K system.

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

Buffer: 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 may start the Editor by utilizing the SDOS EDIT command. This command has three forms:

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

If form 1 is used, 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. This is accomplished by automatically creating a temporary work file to be used as OUTFILENAME. 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)
```

was performed, DATA1 would be the input file, and *ATA1 would be the output file. After you complete your edit session, DATA1 would be deleted, then *ATA1 would be 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.

If form 2 is used, the interpretation is based on whether FILENAME 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 form 3 is used, 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, *, to indicate it is ready for 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 4.2 for an explanation of their use.

5.2 EDIT EXAMPLE

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/O (r)
```

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

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

The * 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
A SAMPLE SOURCE PROGRAM

The first command entered, line 1 of Figure 5-2, is the TAB command (The Set TAB Character Command). The command, TAB ., sets '.' as the TAB character. This gives the '.' a special meaning, which is 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. These are an EDIT mode, where you may perform any of the editing functions, and an INPUT mode, where you may only enter source text.

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).

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 first TAB stop, which is 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

```

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

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, then \$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, 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   WC
5 *F $RET$
6   RETDMYNN
7 *R .RETC,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 DAR1
7       ADDR,R3 DAR1
8       PPSL   WC
9       ADDZ   R2
10      STRZ   R2
11      CPSL   WC
12      RETC,UN
13 DAR1  RES   1
14 ** EOF **
15 *FILE
16 *SLJ* EOJ
17
18 >
```

FIGURE 5-4

DISPLAYING THE BUFFER AND FILING

Suppose you wished to expand DADDSB/0 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

DOUBLE PRECISION ADD AND SUBTRACT

To edit the additional information into the file DDSB/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/O is an existing file which contains the double precision addition routine, so this EDIT command requests that DADDSB/O 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/O, 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).

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 58 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


```

1 *GET 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 .CPSI.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

```

FIGURE 5-6

ADDING DATA TO AN EXISTING FILE

```
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 DSR1
55      PPSL      WC
56      SUBR,R2 DSRO
57      CPSL      WC
58      RETC,UN
59 DSRO RES      1
60 DSR1 RES      1
61      END      DADD
62 **EOF**
```

FIGURE 5-6

ADDING DATA TO AN EXISTING FILE

(CONTINUED)

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 DAR1 RES 1 lines and the * DOUBLE PRECISION SUBTRACT line. 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, $\text{\textcircled{R}}$, 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, * , $\text{\textcircled{R}}$ (line 5 of 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 up the buffer to the line where you wish to enter the * by entering U (the Move Line Pointer UP the buffer command), space, 3 $\text{\textcircled{R}}$, (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, * , $\text{\textcircled{R}}$ (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/O by the use of the command FILE (line 40 of Figure 5-7). The contents of the buffer are written to the PRIMARY OUTPUT file, *ADDSB/O. The remainder of the PRIMARY INPUT file, DADDSB/O, is copied to the PRIMARY OUTPUT file. Since all the data has been read from the PRIMARY INPUT file (lines 1-2 of Figure 5-6, **EOF**), no additional data is written to the PRIMARY OUTPUT file. The file DADDSB/O is deleted. *ADDSB/O is renamed DADDSB/O. The Editor is exited and SDOS displays its prompt character.

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 will be described.

5.3.1 EDITOR COMMAND LINE

When the Editor presents its prompt character, * , it is ready to accept

```

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 RES 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

commands. All Editor commands are of the form:

command parameterlist

where:

command identifies the particular action desired

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

There must be a space between command and parameters with one exception, described in Section 5.3.2.

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 (:).

For example:

```
* F $BADLINES$: K 1 Ⓟ
```

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**** will be displayed on the console and the entire command line will be 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.

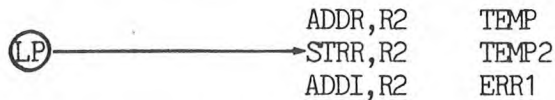
Conventions used in the command descriptions are:

- a) The symbol N refers to two possible entries. These are an absolute number (n) or a line range (p-q). For example, KILL N refers to two possible types of command line, KILL n or KILL p-q. Thus, you could KILL the next n lines or KILL lines p through q (inclusive) in the buffer. N assumes a default of n=1 if N is omitted, with the exception of the COPY command and, when an alternate file is specified, the GET and PUT commands. 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.

In the "p-q" form, the letters B, E, and C may be used, where applicable, to 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.

- b) 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:



- c) The '\$' character 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 Edit example, and its use was discussed on pages 5-4 and 5-6.
- d) The minimum characters required to initiate the command are underlined.
- e) 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 three special commands delimiters, : (which allows the user to stack commands on a command line), <, and > (which execute a command line repetitively, see 5.3.9). To enter these characters into the buffer, they must be entered while in the INPUT mode or when a / prefix is used. (See Section 5.3.2 for INPUT, Section 5.3.9 for /).

5.3.3 INSERTION

The user may insert source lines into the buffer by the use of the commands INSERT and INPUT.

INSERT string

This command will insert the 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  
LP → ADDR,R1 DAR3
```

and the command

```
*INSERT STRR,RO DAR5
```

was performed, the buffer would be altered to

```
      ADDR,RO DAR2  
      STRR,RO DAR5  
LP → ADDR,R1 DAR3
```

If the user enters a null string, by depressing (r) after the single delimiting space, the editor will enter the INPUT mode, which is described below.

INPUT

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

```
INPUT (r)
```

The editor will respond with

```
INPUT:
```

to indicate that it has entered the input mode.

In the INPUT mode, the user may enter any number of lines. These lines will be entered into the buffer before 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
(LP) → ADDR,R1   DAR3
  
```

and the sequence

```

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

was performed, the buffer would be altered to

```

      ADDR,RO   DAR2
      STRR,RO   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**** will be printed on the console, and only the first 128 characters entered will be placed in the buffer.

5.3.4 DELETION

The user may delete lines in the buffer using the command, KILL.

KILL N

This command has two forms, 1) delete the next n lines beginning with current line, or 2) delete lines p through q in the buffer. If no argument is specified, the current line is deleted.

For example, if the buffer appears as follows:

```
ⓁP → 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

```
ⓁP → 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 q, the line pointer is repositioned to point at what was line n+q 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 line 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

The user may alter lines in the buffer through the use of the commands SUBSTITUTE and REPLACE. 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.

For example, if the current line is

```
ADDR,R3   DRSV
```

the command

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

would alter the line to

```
ADDA,R3   DRSV
```

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.

If a substitution causes a line to exceed 128 characters, the message ****TRUNCATED**** will be displayed on the console and the line will be truncated by truncating characters from the text which is being inserted. For example, if the current line is the 127 character line

```
63 A's      62 C's
```

```
(LP) → AA.....AABBCC.....CC
```

and the command

```
S $BB$BBBBB$
```

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

63 A's 62 C's
AA.....AABBCC.....CC

No matter what the result of any substitution, the position of the line pointer is not changed.

In the example command line, SUBSTITUTE \$DR\$DA\$, the character '\$' is used as a delimiter. The first '\$' indicates the beginning of the string to be substituted for. The second '\$' 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 desire to replace \$RIDGE\$ with +RIDGE+, you cannot use this command:

```
S  $$RIDGE$$+RIDGE+$  
   string1 string2
```

Since \$ is being used as the delimiter, it may not be inserted in either string1 or string2.

```
S /$RIDGE$/+RIDGE+/  
   string1 string2
```

would alter the line to the desired:

WINES BY +RIDGE+

In this command, the / is used as the delimiting character.

REPLACE string

The command REPLACE is used to replace the current line with string. For example, if the current line is:

ⓁⓅ → ADDR,R2 DAR1

the command:

*R STRR,R2 DAR1

will result in the current line being altered to:

ⓁⓅ → STRR,R2 DAR1

The position of the line pointer is not altered. A blank line is not allowed as string. For example:

RⓇ

is not a valid command.

If the REPLACE command is used in a command line that contains more than one command, the command delimiting characters : , > , or < will indicate the end of string. For example:

K4:R GOODBYE:I THEN

would delete the next four lines, then the current line would be replaced by the line GOODBYE. The INSERT command would then be executed.

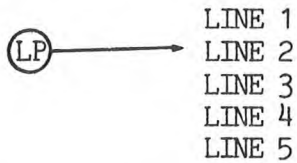
5.3.6 SEARCH

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

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

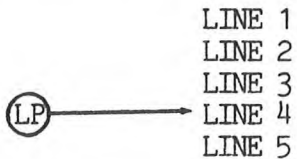
If the buffer appears as follows:



and the command:

F \$4\$

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



In the sample command F \$4\$, the \$ is used as a delimiting character. The first \$ indicates the beginning of the string to be found; the second \$ indicates the end of the string to be found.

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.

If the FIND command is invoked by use of the AGAIN command (see section 5.3.8), the search starts at the current line plus one.

5.3.7 I/O

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

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

- 1) The Editor maintains 'pointers' into the PRIMARY INPUT and 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 PRIMARY 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. An end-of-file mark which is a CTRL-Z, is written to the file. The end-of-file mark is used by SDOS to determine the logical end on 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.

GET N (FILENAME)

This 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.

For example, if the buffer appears as follows:

		PPSL	WC
(LP) →		RETC,UN	
	DAR1	RES	1

and file A contains the five lines

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

performing the command:

GET 1-3 A

will cause the buffer to be altered to:

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

Other features of the GET command include:

- 1) If the user specifies the PRIMARY INPUT file as FILENAME the pointer into the PRIMARY INPUT file will not be altered. For example, 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. If, however, 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.

- 2) The PRIMARY OUTPUT file may not be used as FILENAME.

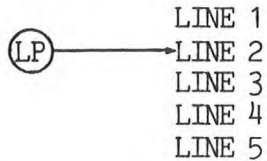
<u>PUT</u> N (FILENAME)
<u>PUTK</u> N (FILENAME)

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 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. If FILENAME is not specified, output will be to the PRIMARY OUTPUT file. The data will be written beginning at the PO pointer and the PO pointer will then be moved at the next empty line in the file.

If the command is PUTK, the lines written to the output file are deleted from the buffer. If PUT is specified there are two possibilities:

- a) The line pointer points to a line which will be deleted. In this case, the line pointer is repositioned to the line immediately following the deleted text.

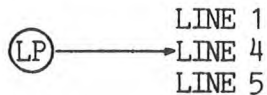
For example, if the buffer appears as follows:



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:



- b) The line pointer points to a line which will not be deleted. In this case, the position of the line pointer is not altered.

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. If the PRIMARY OUTPUT file is used by default, the data is copied from INFILE to the PRIMARY OUTPUT 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

The user may alter the position of the line pointer by using the commands BEGIN, END, DOWN, and UP. The user may have the line pointer position printed using the command, N.

BEGIN

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

END

This command positions the line pointer at the last line plus one of the buffer, that is, past any lines already entered into the buffer. ****EOF**** is displayed on the console.

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 q and n+q 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 q and q-n is less than 1, the line pointer is set to point at the first line.

N

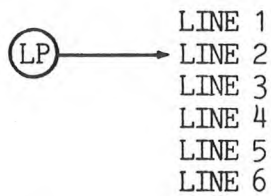
This command displays on the console the number of the line pointed to by the current line pointer.

5.3.9 UTILITIES

The user may perform a variety of functions, including repeating previous commands, listing portions of the buffer, setting the tabs, and terminating an edit session, using the commands AGAIN, BRIEF, FILE, QUIT, SDOS, TAB, TABS, TYPE, ?, / and the iterate command function, m < command >.

AGIN

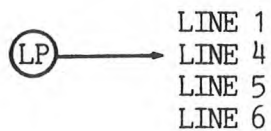
This command performs the previous 'repeatable' command. For example, if the buffer appears as follows,



and the command

*K2

were performed, the buffer would be altered to,



If the next command performed was

*A

the buffer would be altered to



Commands that are not repeatable are:

AGAIN
BRIEF
FILE
INPUT
MACRO
QUIT
TAB
TABS

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.

FILE

This command transfers all the data in the buffer 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 will then be terminated, *SLJ* EOJ will be displayed on the console, and control will return to SDOS.

TYPE N

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. For example, if the buffer appears as follows:

```

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

the command

```
*TY 2-4
```

would result in the following display on the console

```

      LINE 2
      LINE 3
      LINE 4
```

QUIT

This command closes the PRIMARY INPUT and PRIMARY OUTPUT files and then terminates the Edit session. If the PRIMARY OUTPUT file is a new file, this file is deleted before the Editor is exited. *SLJ* EOJ is displayed on the console, and control returns to SDOS.

TAB CHAR

This command defines the single character CHAR 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.

An example of setting the tab character to a different value is:

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

```
TABS C1 C2 C3 ...
```

This command sets the tab positions to the given columns. 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 default TAB positions are 8, 16, 24, 32, 40, 48, 56 and 64.

For example, the default TAB positions would 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 25 31 36
*I
INPUT:
CHARACTER C IS THE TAB CHARACTER
Ⓡ
*B:T
HARA TER IS THE TAB HARA TER
```


m<commands>

This form of the command line will cause the commands inside the angle brackets to be repeated m times. If m is omitted, the commands inside the brackets are performed once. For example, if the buffer appears as follows:

(LP) → STRZ DAR2
 PPSL WD
 ADDR,R2 DAR1
 ADDR,R3 DAR1
 CPSL WD

the command

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

would result in the buffer being altered to:

(LP) → STRZ DAR2
 PPSL WC
 ADDR,R2 DAR1
 ADDR,R3 DAR1
 CPSL WC

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

SDOS

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

?

This command displays the Editor's I/O status. Entering the following command:

?

results in the following information being displayed on the console.

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 referenced
LAST AO = Last Alternate Output file referenced

/

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

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

would be rejected because the angle brackets do not balance. The command

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

would find the string 'LEFTANGLE,<'.

BRIEF

This command changes the state of the BRIEF switch from OFF to ON or from ON to OFF. Under the initial BRIEF state, OFF, the Editor will type the line pointed to by the current line pointer after completing the commands END, UP, DOWN, FIND, SUBSTITUTE, and REPLACE. For example, if the buffer appears as follows:

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

and the command:

```
*D 1
```

is performed, the Editor will move the line pointer down the buffer to LINE 2 and display on the console:

```
LINE 2
```

The user may issue a BRIEF command to change the BRIEF switch to ON. This state will suppress the display of the current line. For example, if these commands were entered:

```
*BRIEF  
*D 1
```

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

If the BRIEF switch is OFF, the user may still suppress the display if he appends a (.) to the command. In the previous example, if the line pointed at

LINE 1 and the BRIEF switch was OFF, this command:

*D.1

would suppress the display of the current line following completion of the D command.

If the BRIEF switch is ON, meaning display is suppressed, the user may display the current line by appending the (.) to the command.

5.3.10 MACROS

The user may define or execute a macro through the use of the MACRO command.

MACRO m=COMMANDLINE

This command is the MACRO definition command. m is an integer which identifies the macro, and must be greater than 0 and less than 128. 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 will replace the old MACROm.

MACRO m

This command executes MACRO m. The effect is equivalent to having entered the command line COMMANDLINE used when the MACRO was defined.

5.4 EDITOR MESSAGES

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

**** 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 string 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.

**** 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 the System Reference Manual.

**** NO PI ****

For this editing session there is no PRIMARY INPUT file; the user may not do "GET's" with out 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.

**** PI ****

**** PO ****

**** AI ****

**** AO ****

The editor response occurred in reference to the Primary or Alternate Input or Output, as applicable.

****NEW FILE ****

A new file was created.

**** (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 INFILNAME OUTFILNAME" where INFILNAME and OUTFILNAME were not the same file and INFILNAME was non-existent.

**** EOF ****

An end-of-file was reached on input or output or the end of workspace text was reached.

**** 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 ****

A command line exceeded 128 characters and was rejected.

An INPUT line exceeded 128 characters and was truncated to the first 128 characters entered.

A SUBSTITUTE caused the line to exceed 128 characters and the line was truncated to 128 characters.

(See Example in Sec. 5.3.5)

CHAPTER 6

THE ASSEMBLER

6.0 INTRODUCTION

The Assembler is the system program used to translate 2650 source program code into 2650 object code that is executable by the TWIN system. The Assembler 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 the Assembler 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, the user enters the following SDOS command

```
ASM SOURCEFILENAME (LISTFILENAME) (OBJECTFILENAME)(WIDE) (NOERR)
```

where:

SOURCEFILENAME is the name of the disk where the source code resides

LISTFILENAME is the name of the disk file or device where the assembly listing is to be written.

OBJECTFILENAME is the name of the disk file or output device where 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.

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.

NOERR indicates that errors should not be displayed on the console.

For example, if the double precision add/subtract subroutine entered in Chapter 5 (see Figure 6-1) was to be assembled, the following tasks would have to be performed.

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

```
ASM DADDSB/O,LPT1,DADOBJ/O
```

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

6.3 POST-ASSEMBLY TASKS

When the Assembler 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 the Assembler produced a listing, the listing will contain the two heading lines in Figure 6-2, the assembled source code, and the final line which

```

* 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 SUBSTRACT.  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
      DPSL  WC
      RETC,UN
*
DSRO  RES    1
DSR1  RES    1
      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 C909  DADD   STRR,R1  DRR1
0012 0002 8B07          ADDR,R3  DRR1
0013 0004 7708          PPSL   WC
0014 0006 82          ADDZ   R2
0015 0007 C2          STRZ   R2
0016 0008 7508          CPSL   WC
0017 000A 17          RETC,UN
0018 000B      DRR1  RES    1
0019          *
0020          * DOUBLE PRECISION SUBTRACT. A IN R2,R3. B IN R0,R1
0021          * ON RETURN, A-B IS IN R2,R3.
0022          *
0023 000C C80E  DSUB   STRR,R0  DSR0
0024 000E C90A          STRR,R1  DSR1
0025 0010 AB0E          SUBR,R3  DSR1
0026 0012 7708          PPSL   WC
0027 0014 AA03          SUBR,R2  DSR0
0028 0016 7508          CPSL   WC
0029 0018 17          RETC,UN
0030          *
0031 0019      DSR0  RES    1
0032 001A      DSR1  RES    1
0033 0000      END    DADD

```

TOTAL ASSEMBLY ERRORS = 0000

FIGURE 6-2

SAMPLE ASSEMBLY LISTING

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. This column is provided for the programmer's convenience as a 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

The assembler 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 reproduced 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. There is a symbol in the argument field which 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, the assembler 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

The SRB Status codes are listed in the System Reference 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 1.0

6.5 LOADING AN ASSEMBLED PROGRAM

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

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

2) Enter the SDOS command

RHEX OBJECTFILE

where OBJECTFILE is the name of the file that contains the 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 4.4.4). 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 file can be made from slave memory by using the MODULE 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 hex 09) 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

THE PROM PROGRAMMER

7.0 INTRODUCTION

TWIN provides facilities for programming and creating programming tapes for PROMs. The current hardware and software support two types: the 82S115, bipolar fusible link PROM, and the 1702A, MOS erasable PROM.

The system software contains three commands which utilize the PROM programming sockets on the TWIN front panel (see Figure 3-3): R PROM, C PROM and W PROM. The 24-pin socket labeled PROM 1 is used for 1702A PROMs. The 24-pin socket labeled PROM 2 is used for 82S115 PROMs. The third socket is reserved for future use.

7.1 USING THE PROM PROGRAMMERS

This section describes the SDOS commands applicable to the PROM programmers and certain basic precautions that should be taken while using the PROM programmers.

The user should insure that:

- 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 is likely to cause permanent damage to the PROM.
- 3) PROMs are inserted in the correct fashion. Leave the socket lever up normally. Push down on the lever to clamp the PROM in the socket.
- 4) Align pin 1 of the PROM with pin 1 of the socket. Pin 1 is adjacent to the lever.

The SDOS commands that apply to the PROM programmers are R PROM, W PROM and C PROM. R PROM is used to read the contents of a PROM into slave memory. W PROM is used to write a portion of slave memory to the PROM programmer. C PROM is used to compare the contents of slave memory with the contents of a PROM.

RPROM (A1) (N) (A2) (A3) (C)

This command is used to read the contents of the PROM inserted in socket N into slave memory. A1 is the first location in slave memory to be stored into. The default value of A1 is 0. N is the PROM port to be read. If N is equal to 1 the 1702A port is specified. If N is equal to 2 the 82S115 port is specified. The default value of N is 1. A2 is the address to begin reading from on the PROM. The default value of A2 is 0. A3 is the last address to read from on the PROM. The default value of A3 is 00FF. C determines whether the data from the PROM should be complemented. If C is equal to 1, the data is complemented before it is stored in memory. If C is equal to 0, the data is not complemented. The default value of C is 0.

PRM ERROR RESPONSES

- 7 - DEVICE WRITE ERROR
- 29 - PROM CWER FAILURE
- 30 - INVALID PARAMETER
- 35 - ILLEGAL START ADDRESS
- 36 - ILLEGAL END ADDRESS

WPROM (A1) (N) (A2) (A3) (C)

This command causes the PROM on port N to be programmed with the contents of slave memory. A1 is the address of the first slave memory byte to be programmed in the PROM. The default value of A1 is 0. N is the number of the PROM programmer port. N equal to 1 corresponds to the 1702A port and N equal to 2 corresponds to the 82S115 port. The default value of N is 1. A2 is the initial PROM location and A3 is the last PROM location to program. The default value of A2 is 0. The default value of A3 is 00FF. C indicates whether the data should be complemented before it is programmed in the PROM. If C is equal to 1, the data will be complemented. If C is equal to 0, the data will not be complemented. The default value of C is 0.

After each memory byte has been written, the PROM is read. The byte read from the PROM is compared with the byte written. If the bytes are not equal, a certain number of retries are attempted. If the comparison still fails after these retries, the PROM address and the contents of the PROM are displayed on the console. The maximum number of retries is sixteen (16) for the 1702A and eight (8) for the 82S115. If an unsuccessful compare occurs on the 1702A, the PROM is rewritten five (5) times before the next comparison.

PRM ERROR RESPONSES

- 7 - DEVICE WRITE ERROR
- 29 - PROM POWER FAILURE
- 30 - INVALID PARAMETER
- 35 - INVALID START ADDRESS
- 36 - INVALID END ADDRESS

CPROM (A1) (N) (A2) (A3) (C)

This command causes the contents of the PROM on port N to be compared with the contents of slave memory. A1 is the location of the first slave memory byte to be used in the comparison. The default value of A1 is 0. N is the number of the PROM programmer port. N equal to 1 corresponds to the 1702A port and N equal to 2 corresponds to the 82S115 port. The default value of N is 1. A2 is the initial PROM location to be compared with slave memory. The default value of A2 is 0. A3 is the last PROM location to be compared with slave memory. The default value of A3 is 00FF. C indicates whether the slave memory data should be complemented before it is compared with the contents of PROM. If C is equal to 1, the slave memory data will be complemented before the compare occurs; if C is equal to 0, the data will not be complemented. The default value of C is 0.

If the value read from the PROM and the slave memory data are not equal, the memory location, its contents, and the PROM contents are displayed on the console.

PRM ERROR RESPONSES

- 7 - DEVICE WRITE ERROR
- 29 - PROM POWER FAILURE
- 30 - INVALID PARAMETER
- 35 - INVALID START ADDRESS
- 36 - INVALID END ADDRESS

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 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 4.2. Note in particular the impact of the ESC key on the EXAM command.

- 2) If it is necessary to change the slave mode for a Debug session, the change must be made before the Debugger is invoked. To change the slave mode, execute the SLAVE command, which is described in Section 8.4.
- 3) Executable programs are stored in two formats:
 - a) Hex format. Two hex characters are stored for each byte of object code produced. The Assembler creates hex format files. RHEX is the SDOS command used to read hex format files. Hex format is described in Appendix C.
 - b) Binary format. One byte of data is stored for each byte of object code. The SDOS command, MODULE, creates binary format files. LOAD is the SDOS command used to read binary format files.

If you are familiar with Debuggers and their commands, Sections 8.1, THE DEBUG PACKAGE, 8.2, THE DEBUG COMMAND, 8.4, DEBUG COMMANDS, and 8.5, 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 his disposal.

When the user invokes the Debugger, the debug package is loaded into a Master area (Overlay Area 1). In addition, a small trace package is loaded into the slave memory (see Debug command). This package, which is 100 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 package 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 utilizing Debug, load the user program into slave memory via the SDOS 'LOAD' or 'RHEX' command. Start the DEBUG package, using the DEBUG command. Select the desired DEBUG functions (Trace, Breakpoint, etc.).

TABLE 8-1
DEBUG COMMANDS

SYSTEM CONTROL	FILE MAINTENANCE	SYSTEM OPTIONS	
ABORT GO XEQ LOAD	DELETE ASSIGN CLOSE	SYSTEM	
BREAKPOINT	STATUS	SYSTEM	MEMORY
* BKPT * CLBP	* DSTAT STATUS * TRACE	* RESET * SET	DUMP EXAM PATCH

* These commands are available only after the DEBUG command has been issued.

The user must take care not to overlay the trace package. When the user issues the DEBUG command, the trace package is loaded into slave memory, which could be overlaid by a user program if the LOAD or XEQ command were executed later.

When the SDOS prompt character is not displayed on the console and the operator desires control, the following procedure should be utilized:

- 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) When it is necessary to continue the user program, typing the 'GO' command will continue the user program from the point it was interrupted.

The user program will be stopped, which will result in the SDOS prompt character being displayed and the system becoming available for input commands, under the following conditions:

- 1) The user requested console control 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 in the TRACE STEP mode.
- 5) The user program has reached a normal end of job condition.

The only way for the user to terminate the Debugger is to use the SDOS 'ABORT' command. This may be accomplished by ABORT DEBUG or ABORT *. In either case, both DEBUG and the user program are terminated.

8.2 THE DEBUG COMMAND

The user loads and starts the Debug subsystem by issuing the following command:

```
DEBUG (ADDRESS) (DEVICE)
```

This command causes the Debug package to be loaded. ADDRESS is the address in slave memory where the trace package is loaded. The default value of ADDRESS is the top of memory (as specified in the SLAVE command) minus the 100 bytes necessary for the trace package. DEVICE is the output device or disk file where the Debug output displays will be written. The default value of DEVICE is CONO, the console output device.

8.3 SAMPLE DEBUG SESSION

Let's monitor the program in Figure 8-1 with the Debugger so that we may examine some of the Debug features. The sample program has been assembled into hex object code which was written to a disk file named DEMO. The starting location for the program is 3000.

The system is in the 2650 slave mode 0 by default, so an initial SLAVE command is not required.

To load the hex code from the file DEMO, enter the SDOS command RHEX DEMO (line 1 of the Figure 8-2). This command loads the object code on the disk file DEMO into slave memory. (If the file DEMO contained a binary load module produced from the assembler output by the use of the MODULE command, the command LOAD DEMO would be used). To load the DEBUG package, enter the SDOS command DEBUG (line 3 of Figure 8-2). Both the object code from the sample program in Figure 8-1 and the DEBUG trace package now reside in slave memory. The DEBUG package is located in a Master CPU area.

The sample program uses the Registers 0 and 1. If we wish to give these registers specific values, the SET command must be utilized. Suppose we wish to enter the value 0 in Register 0 and 1 in Register 1. To do this, enter SET R0 01. (line 6 of Figure 8-2). SET specifies the Set Register command, and R0 specifies the first register to store into. If we desire to view the Debug status before beginning execution, the DSTAT command must be employed. Entering DSTAT (line 8 of Figure 8-2) causes the information on line 9 of Figure 8-2 to be displayed. This is a one line display which provides the location of the last instruction executed in the slave CPU, the active breakpoints, and the contents of the registers in the slave CPU. The area indicated by ① displays the

TOP	ADDZ	R1	ADD REGISTER 1 REGISTER 0
LOOP	ADDI,RO	1	INCREMENT RO
	COMI,RO	0	COMPARE RO WITH 0
	BCFR,0	LOOP	IF COMPARE FAILED, BRANCH TO LOOP
	BCTR,0	TOP	IF COMPARE SUCCEEDED, BRANCH TO TOP

FIGURE 8-1

SAMPLE PROGRAM

```
1 > RHEX DEMO
2 *RHX* EOJ
3
4 > DEBUG
5
6 > SET RO 0 1
7
8 > DSTAT
9 P=0000
```

①

②

R=00 01 00 00 00 00 00 00
③ ④ ⑤ ⑥

FIGURE 8-2

LOADING OBJECT CODE AND DEBUGGER
INITIALIZING SLAVE REGISTERS

program counter at the time the last slave CPU instruction was executed. P=0000 is the value of the program counter before any slave instruction is executed. The area indicated by 2 displays the breakpoints currently active in the Debugger. Since we have not set any breakpoints, no information is displayed. 3 contains the value of Register 0. 4 contains the values of Registers 1, 2, and 3 of Bank 0. 5 contains the values of Registers 1, 2 and 3 of Bank 1. 6 contains the PSU and PSL values.

Suppose we wished to trace the execution of this program. This is accomplished by turning the TRACE function on, as shown on line 1 of Figure 8-3. TR A S is the TRACE (TR) command which requests that all (A) instructions be traced and that the single step (S) mode be employed. The All mode results in the TRACE display being written to the console for every instruction executed by the slave CPU, and the single step mode returns control to the operator after each slave CPU instruction that is executed.

To start the execution of the program, the command, GO 3000 is entered (line 3 of Figure 8-3). Because the object code was initially loaded with the RHEX command, a starting address (3000) must be given with the GO command. (If the LOAD command is used to initially load the object code, the start address is automatically entered into the system.) After this instruction is executed, the Debugger, which is in the single step trace mode, assumes control and produces the TRACE display (Lines 4 and 5 of Figure 8-3). The headings in line 4 have the following meanings, where all values are in hex:

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.

```

1 > TR A S
2
3 > GO 3000
4 LOC INST MNEMON XR U OPAD IADD IV EADD R0 R1 R2 R3 R4 R5 R6 PU PL
5 3000 81 ADZ ,01 01 01 00 00 00 00 00 00 00 40
6
7 > G
8 3001 8401 ADI ,00 01 02 01 00 00 00 00 00 00 40
9
10 > G
11 3003 E400 CMI ,00 00 02 01 00 00 00 00 00 00 40
12
13 > G
14 3005 987A BFR ,00 - 3001 =3001 02 01 00 00 00 00 00 00 40

```

FIGURE 8-3
SINGLE STEP TRACE ALL MODE

Line 5 informs us that location 3000 was the last location executed; 81 was the hex value of that location; ADZ,01 was the instruction mnemonic (note that ADZ is a shortened form of ADDZ. See Table 8-2 for mnemonic list), and the next nine entries indicate the register contents.

We can single step through the next instruction by entering the SDOS command G (the GO command, line 7 of Figure 8-3). As can be seen in lines 7 - 8, as well as lines 10 - 11 and 13 - 14 of Figure 8-3, the Debugger performs a single step and then displays the TRACE information.

Suppose we did not wish to single step, but still wished to trace all the instructions executed. This could be accomplished by altering the TRACE mode. TR A (line 1 of Figure 8-4) requests that all instructions be traced, but does not request the single step mode. When the next GO command is executed (line 3 of Figure 8-4), the Debugger takes control of the slave CPU after every slave CPU instruction is executed, but after it displays the TRACE information, control is not returned to the user, but to the slave CPU. This results in the lines from 4 to 14 being displayed, one line at a time, as each instruction is executed in the slave CPU. If we are interested in whether the logic of the instruction at 3007 is correct (3007 will not be executed until Register 0 overflows and reverts to 0), we would have to wait for large number of TRACE lines to be displayed. To cancel the current trace, the ESCAPE key is pressed, which terminates the current TRACE (the effect can be noted on line 14 of Figure 8-4) and displays this prompt, >>, (line 15 of Figure 8-4) to indicate readiness to accept commands.

Suppose we desire not to view any TRACE's until the instruction at 3007 is executed. This could be accomplished by the actions shown in Figure 8-5. First we set a breakpoint by the command in line 1 of Figure 8-5. BKPT 3007 requests that a breakpoint be set at location 3007 of slave memory. Breakpoints are used to control execution by commanding the Debugger to take control whenever the address that is a breakpoint is referenced. Since we don't wish to see all the executed instructions traced, the command of line 3 of Figure 8-5 turns the TRACE mode off.

Execution is resumed using the GO command (line 5 of Figure 8-5). The Debugger monitors the slave program execution, and when the instruction at 3007 is executed, the display on lines 6 and 7 of Figure 8-5 is produced. Line 6 is the standard TRACE display of the last instruction executed. Line 7 indicates that the program execution stopped because a breakpoint was encountered. In line 6, note that the EADD, which is the address where control will be transferred, is 3000. The prompt character, '>', at line 9 indicates that control has been returned to the operator.

Suppose we wished to monitor the execution of all the branch instructions. This could be accomplished using the commands in Figure 8-6. First, let's set Register 1 to FA (line 1 of Figure 8-6). Then, via the DSTAT command, we can view

```

1 > TR A
2
3 > G 3000
4 3000 81      ADZ ,01      - 01      03 01 00 00 00 00 00 00 40
5 3001 8401    ADI ,00      01      04 01 00 00 00 00 00 00 40
6 3003 E400    CMI ,00      00      04 01 00 00 00 00 00 00 40
7 3005 987A    BFR ,00      - 3001    =3001 04 01 00 00 00 00 00 00 40
8 3001 8401    ADI ,00      01      05 01 00 00 00 00 00 00 40
9 3003 E400    CMI ,00      00      05 01 00 00 00 00 00 00 40
10 3005 987A   BFR ,00      - 3001    =3001 05 01 00 00 00 00 00 00 40
11 3001 8401    ADI ,00      01      06 01 00 00 00 00 00 00 40
12 3003 E400    CMI ,00      00      06 01 00 00 00 00 00 00 40
13 3005 987A    BFR ,00      - 3001    =3001 06 01 00 00 00 00 00 00 40
14 3001 8401    A
15 >>

```

FIGURE 8-4
TRACE ALL MODE

```
1 >BKPT 3007
2
3 >TRACE OFF
4
5 >G
6 3007 1877 BTR ,00 - 3000 =3000 00 01 00 00 00 00 00 21
7 3007 BREAK
8
9 >
```

FIGURE 8-5
USING BREAKPOINTS


```

1 > SET R1 FA
2
3 > DSTAT
4 P=3008 BP=3007 WR          R=00 FA 00 00 00 00 00 00 21
5
6 > TRA J
7
8 > G 3000
9 3005 987A   BFR ,00   - 3001   =3001 FB FA 00 00 00 00 00 00 80
10 3005 987A  BFR ,00   - 3001   =3001 FC FA 00 00 00 00 00 00 80
11 3005 987A  BFR ,00   - 3001   =3001 FD FA 00 00 00 00 00 00 80
12 3005 987A  BFR ,00   - 3001   =3001 FE FA 00 00 00 00 00 00 80
13 3005 987A  BFR ,00   - 3001   =3001 FF FA 00 00 00 00 00 00 80
14 3005 987A  BFR ,00   - 3001   =3001 00 FA 00 00 00 00 00 00 21
15 3007 1877  BTR ,00   - 3000   =3000 00 FA 00 00 00 00 00 00 21
16 3007 BREAK

```

FIGURE 8-6

USING THE TRACE ALL JUMPS MODE

the current DEBUG status (line 3 and line 4 of Figure 8-6). Note that the presence of the breakpoint at 3007 is indicated in this display. The WR in the section BP=3007 WR, refers to the fact that either a read or a write to location 3007 will cause a break. Line 6 of Figure 8-6, TRA J, is the TRACE (TRA) command which requests that only branch (J) instructions be displayed.

When the slave program is continued through the GO command (line 8 of Figure 8-6), the Debugger displays the TRACE information for all branch instructions executed, whether the branch was performed or not (lines 9 - 15 of Figure 8-6). The Debugger informs us that a break has taken place in line 16 of Figure 8-6.

If we desire to clear a breakpoint, the command in line 4 of Figure 8-7 must be executed. CLBP 3007 requests that the breakpoint at location 3007 be cleared. By viewing the DSTAT displays in lines 2 and 7 of Figure 8-7, the effect of the CLBP operation is clear.

When we are finished with a Debug session, the Debugger must be exited using the SDOS command, ABORT. Consult Figure 8-8 for an example of exiting the Debugger.

8.4 DEBUG 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:

- GO
- LOAD
- XEQ
- DUMP
- EXAM
- PATCH
- STATUS
- SLAVE

GO is used to start user programs. LOAD is used to read binary load files into the slave memory. XEQ is a combination of the LOAD and GO programs. DUMP displays the contents of slave memory on a specified device. EXAM allows the user to examine or alter slave memory. PATCH allows the user to alter slave memory. SLAVE identifies the slave CPU and sets the TWICE debug mode. STATUS displays the status of the slave CPU and the job being executed by it.

```
1 > DSTAT
2 P=3007 BP=3007 WR R=00 FA 00 00 00 00 00 00 21
3
4 > CLBP 3007
5
6 > DSTAT
7 P=3007 R=00 FA 00 00 00 00 00 00 21
```

FIGURE 8-7
CLEARING BREAKPOINTS

```
1 > ABORT DEBUG
```

FIGURE 8-8
TERMINATING A DEBUG SESSION

There are six commands that are unique to the Debugger and can only be used after the DEBUG command has been executed. These commands are:

BKPT
CLBP
RESET
SET
DSTAT
TRACE

BKPT and CLBP are used to set and clear breakpoints. RESET generates a RESET pulse to the slave processor. SET allows the user to set slave CPU registers. DSTAT provides information on the Debug status. TRACE allows the user to trace slave CPU execution.

GO (ADDRESS)

This command causes control to be passed to a location in slave memory.

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

LOAD FILENAME

This command loads the binary load module FILENAME into slave memory. This load module must have been created by the MODULE command.

FILENAME will be loaded into the slave memory starting at the location specified at the time the load module was created. Control is not passed to the load module as in the XEQ command.

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

XEQ FILENAME

This command causes the binary load module FILENAME which was created using the MODULE command to be loaded into slave memory and executed. This command is the equivalent of executing LOAD FILENAME followed by the GO command.

SDOS 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

DUMP A1 (A2) (DEVICE)

This 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 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 + hexadecimal 10. This has the effect of lowering A1 to the next lowest multiple of 10_{16} and raising A2 to the next highest multiple of 10_{16} . The contents of memory from A1 to A2 is then displayed. For example, if DUMP 3F3E-4001 was entered, the DUMP program would display the data from 3F30 to 4010. Sixteen bytes are displayed on each line, preceded by the address of the first byte on that line.

DMP ERROR RESPONSES

- 17 - OUTPUT DEVICE ASSIGN FAILURE
- 31 - PARAMETER REQUIRED
- 35 - INVALID STARTING ADDRESS (A1)
- 36 - INVALID ENDING ADDRESS (A2)

EXAM ADDRESS

This command causes the contents of the slave memory location ADDRESS to be displayed on the console. The user then has several options. The user 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, the user can press any of these keys to initiate the corresponding function:

SPACE	Display the next sequential byte.
LINEFEED or DELETE (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.

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 10 .

The ESC Key has a different interpretation when the EXAM command is being used. Consult Section 4.2 for details.

For example, if locations 3000-3003 contained 00, 01, 02, 03 respectively, the EXAM command could be used as follows (user interaction 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 .....
```

The '-' is provided by the EXAM command when the user enters a hex character.

EXM ERROR RESPONSES

- 31 - PARAMETER REQUIRED
- 35 - INVALID START ADDRESS
- 39 - INVALID HEX CHARACTER

PATCH ADDRESS HEX-STRING

This command allows the user to alter slave memory. ADDRESS is a hexadecimal address constant. HEX-STRING is a string of hexadecimal digits from 1 to 58 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. For example, PATCH 3000 3F001E would replace the data at location 3000 in slave memory with 3F, the data at location 3001 with 00, and the data at location 3002 with 1E.

PAT ERROR RESPONSES

- 31 - PARAMETER REQUIRED
- 34 - INVALID ADDRESS
- 39 - INVALID HEX CHARACTER

STATUS

This command gives the status of the program being executed by the slave CPU.

The name of the program running under the slave CPU, the state of the program, and the channel assignments of the program are output to the system console. The status of any COMMAND FILE currently in progress is displayed. The table below lists the possible values for STATUS information.

SLAVE (CHIP NAME) IS ACTIVE
IDLE

(SLAVE JOB NAME) IS LOADED
EXECUTING
IN I/O WAIT
SUSPENDED
UNDER DEBUG CONTROL

CHAN (N) ASSIGNED TO (DEVICE) (OPEN)
CHAN (N) ASSIGNED TO (DEVICE) (READ)
CHAN (N) ASSIGNED TO (DEVICE) (WRITE)
CHAN (N) ASSIGNED TO (DEVICE) (EOF)

COMMAND FILE (NAME) IS IN PROGRESS
SUSPENDED

SLAVE CHIPNAME (MODE) (MEM) (DEV ADDR)

This command designates the active slave CPU and sets its mode of operation. CHIPNAME is the name of the target slave CPU and is given as a string of up to eight characters. Currently, 2650 is the only CHIPNAME implemented. MODE designates the mode in which the slave CPU will operate. The default value of mode is 0. MEM specifies the memory available to the slave CPU. The default value of MEM is 1. DEV ADDR gives the address of the slave CPU board. The default value of DEV ADDR is determined by CHIPNAME. If CHIPNAME is 2650, the default for DEV ADDR is EO.

The possible values for MODE are:

- 0 - TWIN Mode. Uses TWIN slave memory and I/O.
- 1 - Partial TWICE mode. Uses TWIN slave memory, user prototype I/O and user clock.
- 2 - Full TWICE mode. Uses user prototype memory, I/O, and clock.

In mode 2, the TRACE JUMP option is not available. (See TRACE command description on Page 8-29.)

The possible values for MEM are:

- 1 - slave memory bound 0-16K.
- 5 - slave memory bound 0-32K.

The possible values for DEV ADDR are:

- EO - 2650.

SLV ERROR RESPONSES

- 31 - PARAMETER REQUIRED
- 32 - TOO MANY PARAMETERS
- 53 - INVALID SLAVE CPU
- 54 - INVALID MODE
- 55 - INVALID MEM
- 56 - INVALID DEVICE ADDRESS

BKPT ADDRESS (WRITE) (READ)

This command causes a program breakpoint to be set for the slave. 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. If neither WRITE nor READ are specified the break occurs whenever there is an attempt to read or write to the specified address.

When the breakpoint address is accessed during program execution, a trace line is displayed on the debug output device, and a breakpoint message is displayed at the console.

Up to two breakpoints may be active in the system.

ERROR RESPONSES

TOO MANY BREAKPOINTS - Two breakpoints are already active.

DEB ERROR RESPONSES

- 30 - INVALID PARAMETER
- 34 - INVALID ADDRESS

CLBP (ADDRESS)

This command clears a breakpoint. If ADDRESS is specified, the breakpoint at the specified address is cleared. If ADDRESS is not specified, all breakpoints are cleared.

ERROR RESPONSES

BREAK POINT NOT ACTIVE - The specified address was not an active break point address.

DEB ERROR RESPONSES

- 34 - INVALID ADDRESS

RESET

This command causes a RESET pulse to be applied to the slave processor.

<u>SET Rm A1(...Ai)</u> or <u>SET PSU A1(A2)</u> or <u>SET PSL A</u>
--

This command causes the specified slave CPU registers to be set to the hexadecimal constants A. The limits for A are 0 to FF.

SET Rm A... causes the slave CPU general registers beginning with Rm to be set to the values specified. Rm is set to A1, Rm+1 is set to A2, and so forth. Only the registers for which values are specified are changed. SET PSU A1 causes the PSU of the slave CPU to be set to A1. Set PSU A1 A2 causes the PSU to be set to A1 and the PSL to be set to A2. SET PSL A causes the PSL of the slave CPU to be set to the value A. The registers of the slave CPU are designated by the following notation:

- 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

For example, SET R2 4F 23 51 would set 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.

Note that it is also possible to set the PSU and PSL in this manner. SET R6 FF A0 B0 will set register 3 of bank 1 to FF, the PSU to A0, and the PSL to B0. It is not legal, however, to refer to the PSU as R7 or the PSL as R8.

DEB ERROR RESPONSES

- 30 - INVALID PARAMETER
- 43 - INVALID DATA PARAMETER

DSTAT

This command causes the Debug status to be displayed 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:

```
      1      2      3      2      3      4      5      6      7 8  
P=OBA0  BP=0900  WR   OA00  WR   R=FF  00 00 02  04 05 06  00 08
```

- 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. 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.
- 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.

TRACE OFF

or

TRACE ALL (STEP) (A1 A2)

or

TRACE JMP (STEP) (A1 A2)

This command determines the trace mode for the Debugger. If TRACE OFF is specified, the TRACE mode is disabled, which means that no instruction traces will be displayed on the Debug device. If TRACE ALL is specified, all the instructions executed by the slave CPU will have their trace information displayed on the Debug display device. If TRACE JMP is specified, all branch instructions will have their trace information displayed on the Debug display device. If STEP is specified with the TRACE ALL or TRACE JMP command, control will be returned to the console after every instruction trace is displayed. If the STEP option is used, the GO command must be used to continue the user program after every STEP trace. If A1 and A2 are specified, the TRACE function will be performed as specified, but only the instructions executed between A1 and A2 will have their trace information displayed. 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 format of the TRACE display is as follows:

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 (see Table 8-2) 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 instructions.
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.

All values displayed are in hex.

The TRACE JUMP form is not active in slave mode 2. In slave mode 2, the only information displayed in the TRACE display is LOC and the register and program status word values.

DEB ERROR RESPONSES

- 31 - PARAMETER REQUIRED
- 35 - INVALID START ADDRESS
- 36 - INVALID END ADDRESS
- 44 - INVALID TRACE MODE PARAMETER

TABLE 8-2
TRACE TABLE MNEMONICS

<u>TRACE MNEMONIC</u>	<u>INSTRUCTION MNEMONIC</u>	<u>TRACE MNEMONIC</u>	<u>INSTRUCTION MNEMONIC</u>
LDZ	LODZ	RRR	RRR
LDI	LODI	RRL	RRL
LDR	LODR		
LDA	LODA	BCT*	BCTR BCTA
STZ	STRZ		
STR	STRR	BCF*	BCFR BCFA
STA	STRA		
ADZ	ADDZ	BRN*	BRNR BRNA
ADI	ADDI		
ADR	ADDR		
ADA	ADDA	BIR*	BIRR BIRA
SBZ	SUBZ		
SBI	SUBI	BDR*	BDRR BDRA
SBR	SUBR		
SBA	SUBA		
DAR	DAR	ZBR	ZBRR
ANZ	ANDZ	BXA	BXA
ANI	ANDI	BST*	BSTR BSTA
ANR	ANDR		
ANA	ANDA	BSF*	BSFR BSFA
IOZ	IORZ		
IOI	IORI	BSN*	BSNR BSNA
IOR	IORR		
IOA	IORA		
EOZ	EORZ	ZSR	ZBSR
EOI	EORI		
EOR	EORR	BSX	BSXA
EOA	EORA		
CMZ	COMZ	RTE	RETE
CMI	COMI	RTC	RETC
CMR	COMR	WRD	WRTD
CMA	COMA	RDD	REDD

<u>TRACE MNEMONIC</u>	<u>INSTRUCTION MNEMONIC</u>	<u>TRACE MNEMONIC</u>	<u>INSTRUCTION MEMONIC</u>
HLT	HALT	WRC	WRTC
		RDC	REDC
NOP	NOP	WTE	WRTE
		RDE	REDE
		TMI	TMI
		LSU	LPSU
		LSL	LPSL
		SSU	SPSU
		SSL	SPSL
		CSU	CPSU
		CSL	CPSL
		PSU	PPSU
		PSL	PPSL
		TSU	TPSU
		TSL	TPSL

* Relative Instructions are Recognized by a + or - Direction Indicator in the Trace Line "U" Field.

8.5 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 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 SLAVE command controls what signals are passed over the TWICE cable to the user's prototype system.

APPENDIX A
SDOS COMMAND SUMMARY

The short form required to invoke the command is underlined.

<u>COMMAND</u>	PAGE
<u>ABORT</u> NAME or <u>ABORT</u> * or <u>ABORT</u> /	4-12
<u>ASSIGN</u> CH DEVICE (... CH DEVICE)	4-13
<u>ASM</u> SOURCEFILE (LISTFILE) (OBJECTFILE) (<u>W</u> IDE) (<u>N</u> OERR)	6-2
<u>BKPT</u> ADDRESS (<u>W</u> RITE) (<u>R</u> EAD)	8-20
<u>CLBP</u> (ADDRESS)	8-20
<u>CLOSE</u> CH (... CH)	4-12
<u>CONT</u> NAME or <u>CONT</u> * or <u>CONT</u> /	4-11
<u>COPY</u> INPUT (...INPUT) OUTPUT	4-22
<u>CPROM</u> (A1) (N) (A2) (A3) (C)	7-3
<u>CSMS</u> (ADDRESS) (DEVICE)	4-26
<u>DEBUG</u> (ADDRESS) (DEVICE)	8-4
<u>DELETE</u> FILENAME/D (, ..., FILENAME/D)	4-15
<u>DEVICE</u> DEVICE U or <u>DEVICE</u> DEVICE D	4-21
<u>DSTAT</u>	8-22

<u>DUMP</u> A1 (A2) (DEVICE)	8-16
<u>DUP</u> D1 D2 (IDENT)	4-20
<u>EDIT</u> (INFILENAME) (OUTFILENAME)	5-2
<u>EXAM</u> ADDRESS	8-17
<u>FORMAT</u> D (IDENT)	4-17
<u>GO</u> (ADDRESS)	8-15
<u>KILL</u> ON or <u>KILL</u> OFF	4-31
<u>LDIR</u> (D) (.) (/) (DEVICE)	4-21
<u>LOAD</u> FILENAME	8-15
<u>MODULE</u> FILENAME A1, A2, A3 (IDENT)	4-24
<u>PATCH</u> ADDRESS HEX-STRING	8-18
<u>PRINT</u> FILENAME (DEVICE) (L1 L2) or <u>PRINTL</u> FILENAME (DEVICE) (L1 L2)	4-23
<u>RENAME</u> OLDFILE/D NEWFILE/D or <u>RENAME</u> D IDENT	4-19
<u>RESET</u>	8-20
<u>RHEX</u> (/BIAS) (DEVICE)	4-25
<u>RPROM</u> (A1) (N) (A2) (A3) (C)	7-2
<u>SEARCH</u> ON (N) or <u>SEARCH</u> OFF	4-14
<u>SET</u> Rm A1(...Ai) or <u>SET</u> PSU A1(A2) or <u>SET</u> PSL A	8-21

<u>SLAVE</u> CHIPNAME (MODE)(MEM)(DEV ADDR)	8-19
<u>STATUS</u>	8-18
<u>SUSPEND</u> (NAME)	4-11
or	
<u>SUSPEND</u> *	
or	
<u>SUSPEND</u> /	
<u>SYSTEM</u> D	4-15
<u>TRACE</u> <u>OFF</u>	8-22
or	
<u>TRACE</u> <u>ALL</u> (<u>STEP</u>) (A1 A2)	
or	
<u>TRACE</u> <u>JMP</u> (<u>STEP</u>) (A1 A2)	
<u>TYPE</u> ON	4-31
or	
<u>TYPE</u> OFF	
<u>VERIFY</u> D	4-18
<u>WHEX</u> A1 A2 ... (,,A1 A2) (A3) (DEVICE)	4-28
<u>WPROM</u> (A1) (N) (A2) (A3) (C)	7-2
<u>WSMS</u> (ADDRESS) (DEVICE)	4-26
<u>XEQ</u> FILENAME	8-16
* COMMENT	4-32

APPENDIX B

TEXT EDITOR COMMAND SUMMARY

The short form required to invoke a command is underlined.

<u>COMMAND</u>	PAGE
<u>A</u> GIN	5-26
<u>B</u> EGIN	5-26
<u>B</u> RIEF	5-31
<u>C</u> OPY N INFILE (OUTFILE)	5-25
<u>D</u> OWN n	5-26
<u>E</u> ND	5-26
<u>F</u> ILE	5-28
<u>F</u> IND \$string\$	5-21
<u>G</u> ET N (FILENAME)	5-23
<u>I</u> NPUT	5-16
<u>I</u> NSERT string	5-16
<u>K</u> ILL N	5-18
<u>L</u> IST N	5-25
<u>M</u> ACRO m=COMMANDLINE	5-32
<u>M</u> ACRO m	5-32
<u>N</u>	5-26
<u>P</u> UT N (FILENAME)	5-24
<u>P</u> UTK N (FILENAME)	5-24

<u>QUIT</u>	5-28
<u>REPLACE</u> string	5-21
<u>SDOS</u>	5-30
<u>SUBSTITUTE</u> \$string1\$string2\$	5-19
<u>TAB</u> CHAR	5-28
<u>TABS</u> C1 C2 C3 ...	5-29
<u>TYPE</u> N	5-28
<u>UP</u> n	5-26
m<commands>	5-30
<u>?</u>	5-30
<u>/</u>	5-31

APPENDIX C

ABSOLUTE OBJECT FORMAT

Absolute 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. (e.g., 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 Block Control Character error on the address and count fields
- A Block Control Character error on the data field
- An incorrect block length
- A non-hex character within the block.

Inter-block 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.

:05000A3C0455B024FFF01F05040030

② ③ ④ ⑤ ⑥ ⑦

- 2 – Start of block character (colon)
- 3 – Starting address for block (H'0500')
- 4 – Number of bytes in block (H'0A' = 10)
- 5 – BCC byte 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 hex '12') and followed by a TAPE OFF character (CTRL-T hex '14'). 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:

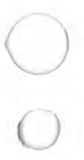
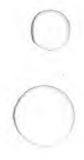
- 1) Each data word is represented by one or two hexadecimal characters.
- 2) Each data word is followed by an apostrophe (hex '27'). When the apostrophe is read, the data word composed from the previous hexadecimal characters is stored at the location pointed to by the address counter. The address counter is then incremented.

All characters are punched in the standard 8-channel ASCII teletype code. Parity is not checked.

EXAMPLE OF SMS FORMAT

↑ 01'FA'FA'00'10' ↓
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 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.

```
                LPT1  
READY D        CONO
```

D 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 2.0**  
                *
```

At this point enter the string: QUIT. The 'End of Ready Test' message notifies the user that the test has completed.

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 messages.


```

TYPE ON
+
+      TURN FROM POWER ON
+
+      MEMORY TEST
+
TYPE OFF
CPROM 0 2 0 1FF
CPROM 0 2 0 1FF
BPROM 0 1 0 FF
CPROM 0 1 0 FF
DEBUG 800
ABORT DEBUG
RHEX READY/#1
GO 00
TYPE ON
*
COPY READY/#1 ***HELP/#1
PRINT ***HELP/#1 #2
DEL ***HELP/#1
LDIR . / #1 #2
*
DEBUG
CMT 0 W
CMT 1000 R
DSTAT
CLBF 0
CLBF 1000
ABORT DEBUG
*
SLAVE 2650 1 1
SLAVE 2650 2 1
SLAVE 2650 0 1
*
* TYPE 'QUIT' WHEN EDIT RSK FOR INPUT
EDIT
*
END OF READY TEST
*
ABORT *
:000003061F007018
:00401B370053004B00500060004A1A02010000001301471001000000000005B33
:00501E51434F4E4F00020100000013015A020100000013016004F775FF04FF00018440
:00791ED920000180C20E01870E8187008187ED01879800003190E400980406109A6777
:009710640E01875252520E6180C001560E6180C0015704F60001848401E40498028B
:00B41EED4F3000184006180C0018A0601059F0E0185C0E185850159798601EE0187A8
:00D21D71986F059F06010E018500E185EC018A0C010B850159730601EE0187986969
:00EF10871F00A0C144F0505050508430E42A1A028407450F8530E53A1685071793
:010010100001893F00F2000176010001770001868001893F00F200016A000168B5
:012710A00001853F00F2000168C001690C018A3F00F200017DC0017ED4F504F4D7
:01431B33000189174D454D4F52592053495A452049532020204B004552524FAB
:015E1E4D52204144445245535320585858580044415441205741533D585820532F42F6
:017018CF3D58580000FF55A0FF0000000000000020202034203831323136323062
:01970846323432383332000001
:007000C100

```


000

0

0

00

APPENDIX F

SYSTEM UTILITY COMMAND FILES

Signetics supplies a command file, COPYSYS, which copies the operating system from one disk to another. The general form of this command is

```
COPYSYS D1 D2
```

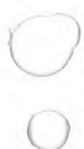
where D1 is the drive to copy from and D2 is the drive to copy to. For example, to copy the operating system from drive 0 to drive 1, enter this command:

```
COPYSYS 0 1
```

When the command is invoked, the following files are copied from D1 to D2:

1. The resident SDOS binary load file
2. All SDOS overlays, including the Assembler and the Text Editor. They are all binary load files.
3. The System Readiness Test
4. The COPYSYS command file.

For the most rapid system response to commands to occur, the operating system should be copied onto a disk before any other files are stored on it. This will allocate the tracks closest to the outside to the system files, and minimize disk lead movement when the overlays for the commands are brought into the overlay areas.



APPENDIX G
STANDARD SYMBOLS

EQUATES is a source file which defines all the standard Assembler symbols. If a programmer wishes to use these symbols, he appends this file to the beginning of his source file, e.g.:

```
EDIT FILENAME  
**EDITOR VERSION 2.0**  
*GET 100 EQUATES  
*FILE
```



```

*****
* 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
* PSM LOWER EQUATES
CC    EQU    H'00'  CONDITIONAL CODES
IDC   EQU    H'20'  INTERDIGIT CARRY
RS    EQU    H'10'  REGISTER BANK
HC    EQU    H'23'  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
* PSM 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

```

000

0

0

0

0

Signetics

a subsidiary of U.S. Philips Corporation

Signetics Corporation
811 East Arques Avenue
Sunnyvale, California 94086
Telephone 408/739-7700