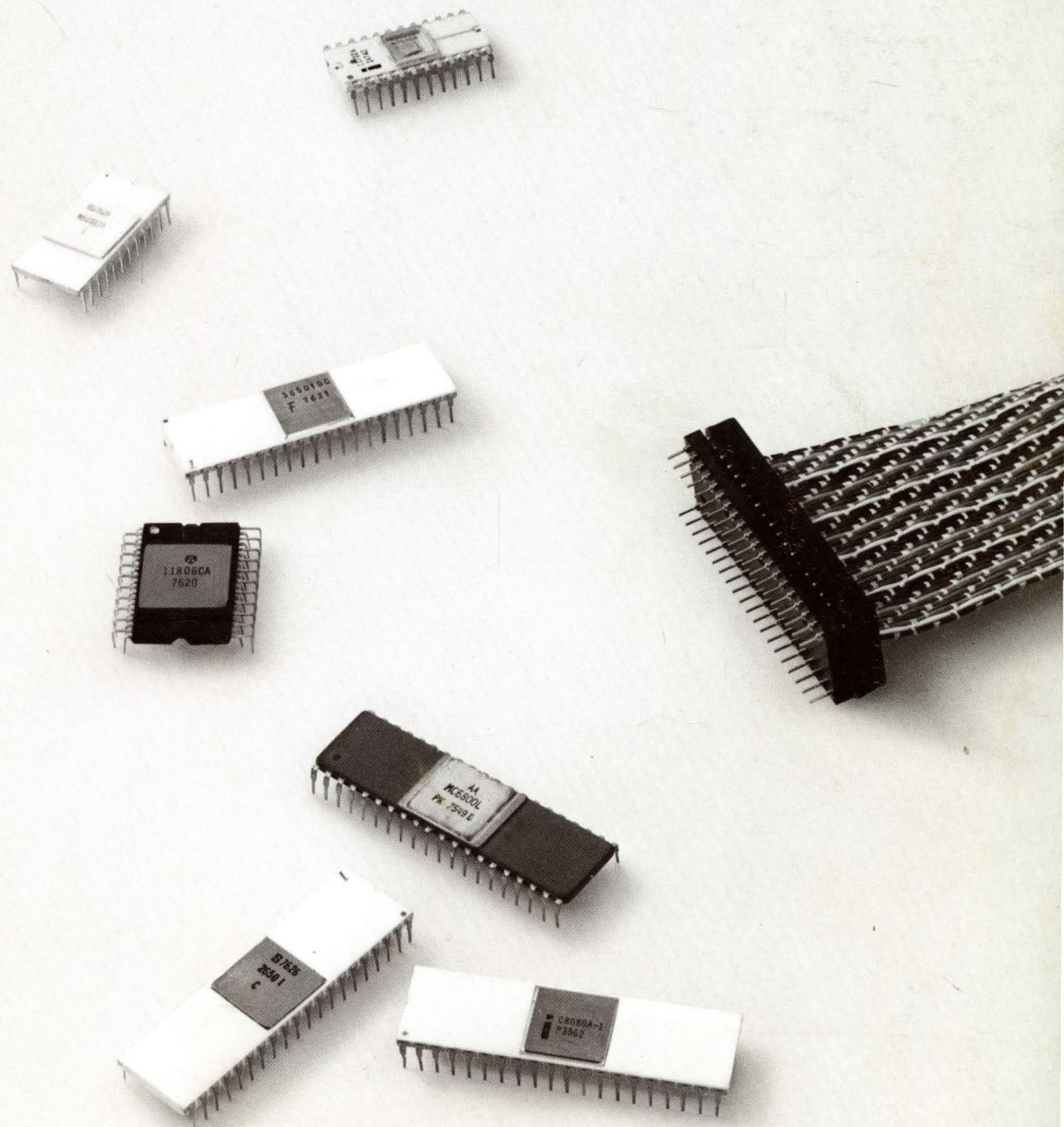


MILLENNIUM INFORMATION SYSTEMS, INC.  
UNIVERSAL ONE

Microcomputer  
Development System

Operator's Guide



**UNIVERSAL ONE**

**Microcomputer  
Development System**

**Operator's Guide**

Document No. UO1-0000-21  
Issued Nov. 1976

**MILLENNIUM INFORMATION SYSTEMS, INC.**

420 MATHEW STREET, SANTA CLARA, CALIFORNIA 95050, 408-243-6652

**DOCUMENT NO. UO1-0000-21**

**PRICE — \$35.00**

**Copyright 1976, Millennium Information Systems, Inc. All rights reserved.**

**Millennium Information Systems, Inc. claims trademark rights to the names  
UNIVERSAL ONE and UNIVERSAL EMULATOR**

# *Table of Contents*

<b>Paragraph</b>		<b>Page</b>
<b>CHAPTER 1 INTRODUCTION</b>		
1.1	Objectives of a Microcomputer Development System . . . . .	1-1
1.2	Universal One Overview . . . . .	1-1
1.3	About This Book . . . . .	1-4
<b>CHAPTER 2 SYSTEM DESCRIPTION</b>		
2.1	Introduction . . . . .	2-1
2.2	Hardware . . . . .	2-1
2.2.1	Development Computer . . . . .	2-1
2.2.2	Emulation Hardware . . . . .	2-4
2.2.3	Dual Floppy Disk Subsystem . . . . .	2-4
2.2.4	Peripherals . . . . .	2-5
2.2.5	User-Supplied Peripherals . . . . .	2-7
2.3	Software . . . . .	2-7
2.3.1	UDOS (Universal Disk Operating System) . . . . .	2-7
2.3.2	The Debugger . . . . .	2-7
2.3.3	PROM Programming . . . . .	2-8
2.3.4	The Editor . . . . .	2-8
2.3.5	The Assembler . . . . .	2-8
2.3.6	Systems Readiness Test . . . . .	2-8
<b>CHAPTER 3 SYSTEM INSTALLATION AND OPERATION</b>		
3.1	Introduction . . . . .	3-1
3.2	Unpacking . . . . .	3-1
3.2.1	Unpacking the Universal One Development Computer . . . . .	3-1
3.2.2	Unpacking the CRT Terminal . . . . .	3-1
3.2.3	Unpacking the Floppy Disk Unit . . . . .	3-2
3.2.4	Unpacking the Line Printer . . . . .	3-2
3.2.5	Installing the Emulation Cable. . . . .	3-4
3.3	Physical Installation . . . . .	3-4
3.3.1	Power and Cable Interconnections . . . . .	3-4
3.3.2	Controls and Indicators . . . . .	3-7
3.3.3	Development Computer . . . . .	3-7



## Table of Contents (Cont.)

<b>Paragraph</b>		<b>Page</b>
3.3.4	Dual Floppy Disk Unit . . . . .	3-8
3.3.5	CRT Terminal . . . . .	3-9
3.3.6	Line Printer . . . . .	3-9
3.4	Operation . . . . .	3-10
3.4.1	Formatting and Verifying New Diskettes . . . . .	3-10
3.4.2	System Startup Procedure . . . . .	3-11
3.4.3	Manual Reset . . . . .	3-12
<b>CHAPTER 4    UNIVERSAL DISK OPERATING SYSTEM</b>		
4.1	Introduction . . . . .	4-1
4.2	UDOS Overview . . . . .	4-1
4.2.1	Resident UDOS Modules . . . . .	4-2
4.2.2	UDOS Overlays . . . . .	4-2
4.3	Files, Devices, and Channels . . . . .	4-3
4.4	Entering UDOS Commands . . . . .	4-5
4.5	Special Keys . . . . .	4-6
4.5.1	Delete Key . . . . .	4-6
4.5.2	Escape Key . . . . .	4-7
4.5.3	Space Bar . . . . .	4-7
4.5.4	CTRL-Z Command . . . . .	4-7
4.6	The UDOS Commands . . . . .	4-8
4.6.1	The UDOS Command Structure . . . . .	4-8
4.6.2	UDOS Command Completion . . . . .	4-9
4.6.3	System Control Commands . . . . .	4-10
4.6.4	System Option Commands . . . . .	4-15
4.6.5	System Utilities Commands . . . . .	4-17
4.6.6	Object Program Utility Commands . . . . .	4-25
4.6.7	Command Files . . . . .	4-29
4.6.8	Command File Utilities . . . . .	4-30
4.6.9	UDOS Error Messages . . . . .	4-32

Table of Contents (Cont.)

<b>Paragraph</b>		<b>Page</b>
<b>CHAPTER 5 THE DEBUGGER</b>		
5.1	Introduction . . . . .	5-1
5.2	The Debug Program . . . . .	5-1
5.3	Invoking the Debugger . . . . .	5-2
5.4	Sample Debug Session (Using a 2650 Slave) . . . . .	5-3
5.5	Debug Commands . . . . .	5-7
<b>CHAPTER 6 THE EDITOR</b>		
6.1	Introduction . . . . .	6-1
6.2	Editor Overview . . . . .	6-1
6.3	UDOS Command Edit . . . . .	6-2
6.4	Edit Example . . . . .	6-3
6.5	Editor Command Descriptions . . . . .	6-10
6.5.1	Editor Command Line . . . . .	6-10
6.5.2	Editor Command Description Conventions . . . . .	6-10
6.5.3	Insertion Commands . . . . .	6-13
6.5.4	Deletion Commands . . . . .	6-14
6.5.5	Alteration Commands . . . . .	6-15
6.5.6	Search Commands . . . . .	6-17
6.5.7	I/O Commands . . . . .	6-18
6.5.8	Buffer Line Pointer Commands . . . . .	6-21
6.5.9	Utilities . . . . .	6-21
6.5.10	MARCROS . . . . .	6-26
6.6	Editor Messages . . . . .	6-27
<b>CHAPTER 7 THE ASSEMBLER</b>		
7.1	Introduction . . . . .	7-1
7.2	Assembler Overview . . . . .	7-1
7.3	Using the Assembler . . . . .	7-1
7.4	Loading An Assembled Program . . . . .	7-2
7.5	Sample Assembly Listing . . . . .	7-3

Table of Contents (Cont.)

<b>Paragraph</b>		<b>Page</b>
<b>CHAPTER 8 PROM PROGRAMMER</b>		
8.1	Introduction . . . . .	8-1
8.2	PROM Programming Hardware and Software . . . . .	8-1
8.3	Using the PROM Programmer . . . . .	8-1
8.4	PROM Programmer Commands . . . . .	8-2
<b>CHAPTER 9 SUPERVISOR CALL INTERFACE</b>		
9.1	Introduction . . . . .	9-1
9.2	General Description of Supervisor Calls . . . . .	9-1
9.3	Service Request Block (SRB) . . . . .	9-2
9.3.1	SRB Bytes . . . . .	9-2
9.3.2	SVC Function Descriptions . . . . .	9-3
APPENDIX A	UDOS COMMAND SUMMARY	
APPENDIX B	DEBUGGER COMMAND SUMMARY	
APPENDIX C	EDITOR COMMAND SUMMARY	
APPENDIX D	SVC FUNCTION CODES	
APPENDIX E	SRB STATUS CODES	
APPENDIX F	SMS TAPE FORMAT	
APPENDIX G	SYSTEM READINESS TEST	
APPENDIX H	SYSTEM UTILITY COMMAND FILES	

## *List of Illustrations*

<b>Figure</b>		<b>Page</b>
1-1	The UNIVERSAL ONE Microcomputer Development System . . . . .	1-0
1-2	General Block Diagram of a Microprocessor-based Product . . . . .	1-3
2-1	Overall Block Diagram of the UNIVERSAL ONE . . . . .	2-2
2-2	Data Organization on a Diskette . . . . .	2-6
3-1	Development Computer Printed Circuit Board Layout . . . . .	3-2
3-2	Development Computer (Top View) . . . . .	3-3
3-3	Envelope Dimensions of System Units . . . . .	3-5
3-4	Typical Cabling Diagram of UNIVERSAL ONE System Installation . . . . .	3-6
3-5	Front Panel of the Development Computer . . . . .	3-8
3-6	Rear Panel of the Development Computer . . . . .	3-9
3-7	Rear Panel of the Floppy Disk Drive . . . . .	3-10
3-8	Inserting a Diskette . . . . .	3-11
5-1	Displays During Sample Debugging Session . . . . .	5-4
5-2	Typical Displays During Various Debugging Modes . . . . .	5-8
6-1	A Sample Source Program . . . . .	6-3
6-2	Entering Text and Display the Buffer . . . . .	6-5
6-3	Use of FIND, SUBSTITUTE and REPLACE Commands . . . . .	6-6
6-4	Displaying the Buffer and Filing . . . . .	6-6
6-5	Sample Double Precision Add and Subtract Programs . . . . .	6-7
6-6	Adding Data to an Existing File . . . . .	6-9
6-7	Inserting Lines Into the Buffer . . . . .	6-11
7-1	Sample 2650 Slave Program Listing Ready for Assembly . . . . .	7-4
7-2	Sample 2650 Slave Assembly Listing . . . . .	7-5

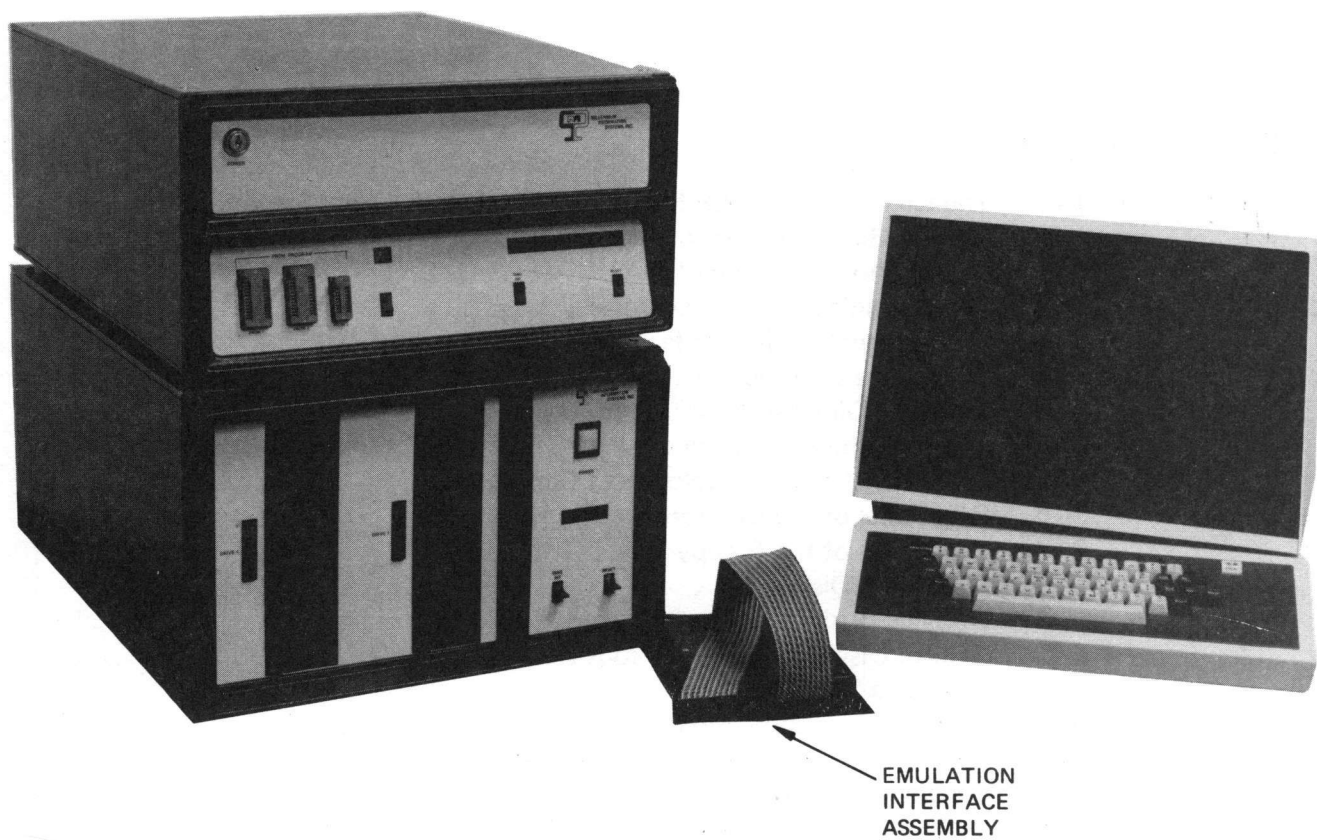


Figure 1-1. The UNIVERSAL ONE Microcomputer Development System  
(Shown with Optional CRT Terminal.)



# *Chapter 1*

## *Introduction*

### **1.1 OBJECTIVES OF A MICROCOMPUTER DEVELOPMENT SYSTEM.**

In the development of any product that includes a microprocessor, there are aspects which have no parallel either in random 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 random logic hardware, or logic which should be implemented with programmed instructions; that is what makes microprocessor product development unique. A successful microcomputer development system, such as UNIVERSAL ONE, must therefore support digital logic development and program creation with equal ease.

If in addition, the product or series of products can benefit from the use of different microprocessors, a single universal hardware and software development system, of the configuration of UNIVERSAL ONE, has several easily recognizable advantages:

1. It eliminates the cost of another development system each time a new microprocessor is to be implemented.
2. It provides a common system for all development, thus eliminating heavy investments in personnel training and software for multiple systems.
3. It frees the designer to consider any microprocessor-solely on the basis of its capabilities and cost-effectiveness, rather than because the designer is locked into a microprocessor from a previous product commitment.

### **1.2 UNIVERSAL ONE OVERVIEW.**

**1.2.1 Master/Slave Concept.** UNIVERSAL ONE, achieves the required universality by dividing its operations into two functional areas. Those tasks that are related to the development system itself are assigned to a master central processing unit, and those that are prototype-related are assigned to a second, slave CPU. As many as four different slaves may be installed simultaneously and individually invoked with a corresponding slave diskette. This multiple architecture enables the hardware to support a particular microprocessor with the addition of a printed circuit card containing the corresponding slave CPU. Since the master processor need not be changed to accommodate new slave units, all of the operating system software remain the same.

The master CPU is responsible for all system services that are not prototype-dependent, such as:

- File management — the storage and retrieval of data and programs.
- Text editor — maintains text files in the floppy disk.
- System input/output — the normal I/O activities between the standard system peripherals, such as the floppy disk, line printer, and system control console.
- System utilities, including programming of PROMs for the final version of the prototype.
- Debug functions — the master executes the debug software and controls the slave through a separate debugging hardware module.

The slave CPU's functions include:

- Program assembly — each slave may be used as a resident assembler of prototype programs.
- Prototype program execution — the prototype program is loaded into the slave memory and executed by the slave.
- Prototype I/O — any special input/output required in the prototype can be performed by the slave, without involving the master CPU.
- In-circuit emulation — a cable extends from the slave to the CPU socket in the prototype.

**1.2.2 System Functions.** UNIVERSAL ONE may at first look like any other general purpose minicomputer system; because there is a terminal 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, UNIVERSAL ONE 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 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 for the purpose of detecting conceptual errors — that is, instruction sequences which, though they are syntactically correct, do not accurately represent the intended logic or data flow.

So complete is this parallel between UNIVERSAL ONE and a general purpose minicomputer, that there is nothing preventing UNIVERSAL ONE from being used like a minicomputer — 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 can be included as a utility within a large user-written program. 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.

But UNIVERSAL ONE is much more than a general purpose minicomputer. The typical microprocessor user program created on UNIVERSAL ONE 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 microprocessor-based logic, in a configuration that may not even remotely resemble a computer. The only constant that may be ascribed to these products is that they will contain a 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 1-2 therefore generally identifies the ultimate configuration which any microprocessor-based product will have.

**1.2.3 System Capabilities.** Every part of the end product illustrated in Figure 1-2 may be developed using UNIVERSAL ONE.

The process of creating an executable object program was discussed first, since this is the most obvious capability of a system that looks like a general purpose minicomputer. But the similarities between UNIVERSAL ONE and a general purpose

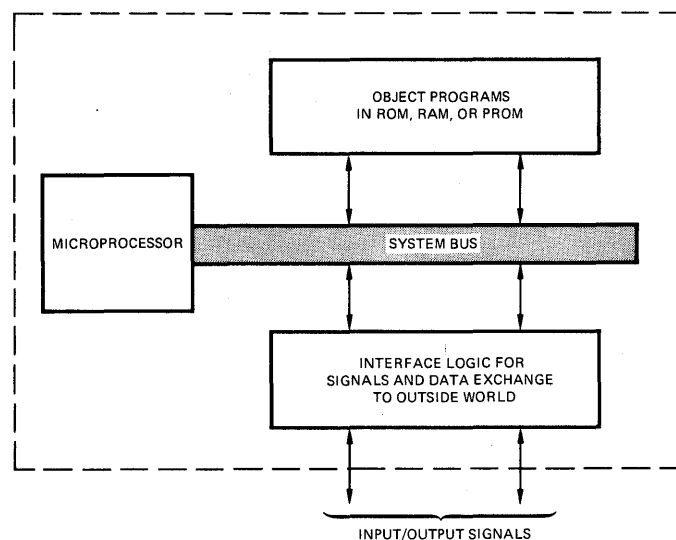


Figure 1-2. General Block Diagram of a Microprocessor-based Product

minicomputer end at this superficial level. Consider some of the additional features which UNIVERSAL ONE provides to serve as a total microprocessor-based product development aid.

To begin with the UNIVERSAL ONE provides at least two CPUs. The master CPU performs monitoring and disk operating system functions; functions required by UNIVERSAL ONE, but absent in the product being developed. The slave microprocessor takes the place of the 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 a separate memory which remains available to simulate prototype memory, for user application programs. When appropriate, the master CPU accesses slave processor memory. The master and slave memories can be initially viewed as separated for system use and for application or user use.

Because object programs are likely to be stored in PROM or ROM devices, UNIVERSAL ONE allows you to create the PROM, or to define the ROM mask.

Simulation of the I/O logic shown in Figure 1-2 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 memory, as opposed to random logic. UNIVERSAL ONE resolves the open-endedness of this additional logic by providing the emulation cable. Any external logic may communicate with the slave microprocessor and its slave memory via this cable. Moreover, external logic beyond the emulation cable may, itself, contain program memory.

Thus, UNIVERSAL ONE becomes a total microprocessor-based product development system. Every aspect of a microprocessor-based product may be simulated and designed using UNIVERSAL ONE. Object programs which, while being created, are executed by a microprocessor which is identical to the end product microprocessor. While object programs were executed by UNIVERSAL ONE, during their creation, they interacted via the emulation cable with additional logic which, package-for-package, will be identical to the eventual end product. Therefore, when going from emulation to end product, the only changes will be in physical fabrication.

### **1.3 ABOUT THIS BOOK.**

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

Additionally, there is a UNIVERSAL ONE System Reference Manual, document number UO1-0000-11 which provides a detailed description of the UNIVERSAL ONE system hardware and its various components.

Each microprocessor supported on UNIVERSAL ONE system is provided with a manual supplement describing the software and hardware peculiar to that microprocessor.

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 about the Universal Disk Operating System and describes procedures for using it; Chapter 5 describes the emulation and the debug capabilities of the UNIVERSAL ONE system and explains all Debugger commands; Chapter 6 describes the Text Editor and gives procedures for using the Editor to create and modify files; Chapter 7 describes how the Assembler is used to create object programs from assembly language programs; Chapter 8 gives procedures for programming type 1702A and type 82S115 PROMs from assembled user programs; and Chapter 9 describes Supervisor Calls (SVCs), with which any slave CPU program can communicate with system peripherals.



# *Chapter* **2** *System Description*

## **2.1 INTRODUCTION.**

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

## **2.2 HARDWARE.**

The UNIVERSAL ONE is a complete microcomputer development system. 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 UNIVERSAL ONE memory, or by using the emulation interconnecting cable assembly, object programs may be executed out of external memory that is part of an end product. Thus UNIVERSAL ONE can simulate an end product, or interface directly to it, and has the ability to support every phase of product development. A UNIVERSAL ONE system consists of a development computer with 16K bytes of master memory and 16K to 65K bytes of slave (or common master/slave) memory, and a dual drive floppy disk unit; peripherals include a 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.

### **2.2.1 Development Computer.**

The development computer consists of a mainframe enclosure and printed circuit boards. In addition to the basic CPU and memory boards, optional boards can be added to implement a particular development function. The following paragraphs describe major functions of the development computer hardware and Figure 2-1 shows a block diagram of it.

**Master and Slave CPUs.** The UNIVERSAL ONE operating system (UDOS) and the Text Editor run under the master CPU. The Assembler, other system programs, and user programs all run on the slave CPU.

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

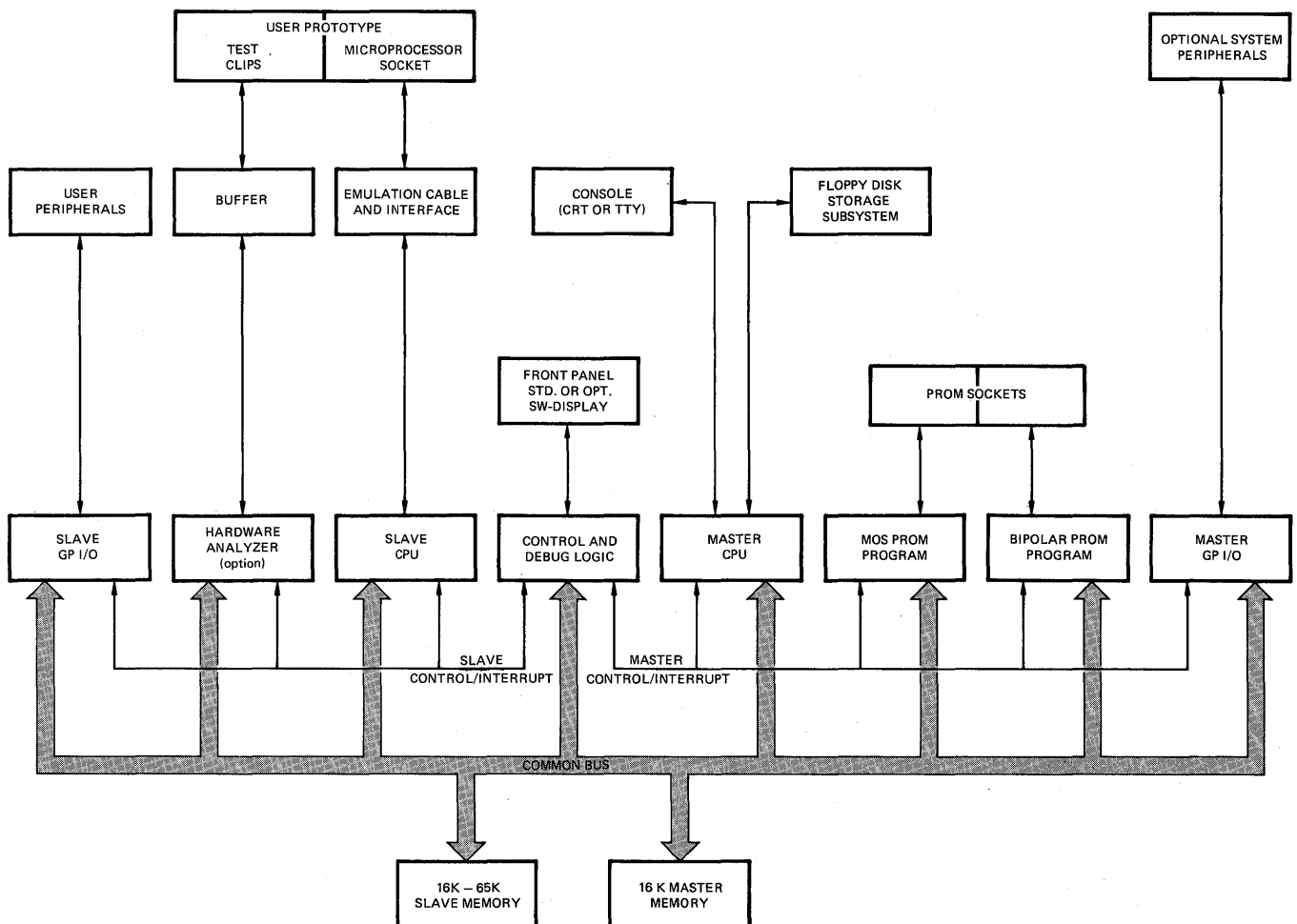


Figure 2-1. Overall Block Diagram of the UNIVERSAL ONE

**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 Chapter 9.

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 hardware environment making full use of the users prototype resources, including I/O logic, power supplies, etc.

**Dual Memories.** The system includes two separate memories: one is the slave memory of up to 65K bytes. This memory is accessible by both master and slave CPUs. Two system programs, 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 by the slave CPU from this memory. (The Editor is run out of slave memory by master CPU.)

The other memory is the master memory from which the operating system and the Debug monitor are run by 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 one 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 used for supervisor calls (SVCs).

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

**PROM Programming.** The development computer contains two optional PROM programming boards and three front-panel PROM sockets. Unique 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 and debugged program. A front panel switch turns off PROM programmer power, so that devices cannot be damaged during insertion and removal.

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

### **2.2.2 Emulation Hardware.**

The emulation hardware consists of a cable and driver/receiver circuits that allow in-circuit emulation of user programs in user developed hardware. The user's micro-processor is removed and replaced by a cable plugged directly into the socket. The other end of the cable is attached to the UNIVERSAL ONE slave CPU circuit board, which contains multiplexing and other logic to support and discriminate between the UNIVERSAL ONE operating modes. The slave CPU thus can become the CPU for the user system.

Presently 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 UNIVERSAL ONE system. This is the normal non-emulation 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.

The emulation cable contains an in-line printed circuit assembly (interface assembly) which provides isolation for the UNIVERSAL ONE 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 Chapter 3 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 emulation cable.

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

### **2.2.3 Dual Floppy Disk Subsystem.**

The floppy disk subsystem is the mass storage medium for the system. The disk subsystem consists of two disk drives, a microprocessor controller, power supplies, and cabinet. The two disk drives are designated as drive 0 and drive 1, and the disk subsystem communicates directly with the master CPU card in the development computer through an interconnecting cable.

Drive 0 is usually the system drive. That is, the diskette with the system programs is normally placed in this drive. The system drive is automatically accessed 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 normally contains all system programs, including UDOS, Editor and the Debugger peculiar to a specific slave on the four outside tracks. (The system diskette can be write protected to ensure that the system programs

are not altered.) Alternatively, any drive can be designated as the system drive by UDOS commands.

Drive 1 usually contains a second diskette utilized primarily for storing user files, for modifying user files, or as a scratch data area, and may or may not contain the system programs. (If it does, the programs usually are duplicate of those on the system diskette.) This diskette, since it may be used as a scratch area, is not write protected.

**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 2–2a and 2–2b. On each diskette, there are 77 concentric tracks (Figure 2–2a), which can contain data. In Figure 2–2b, a typical 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 system programs.

In order for the disk drive to be able to read or write a diskette, the diskette must have certain initial information on it. The process of placing this information on the diskette is called formatting. Diskettes must be formatted before use. (See paragraph 3–16).

### 2.2.4 Peripherals.

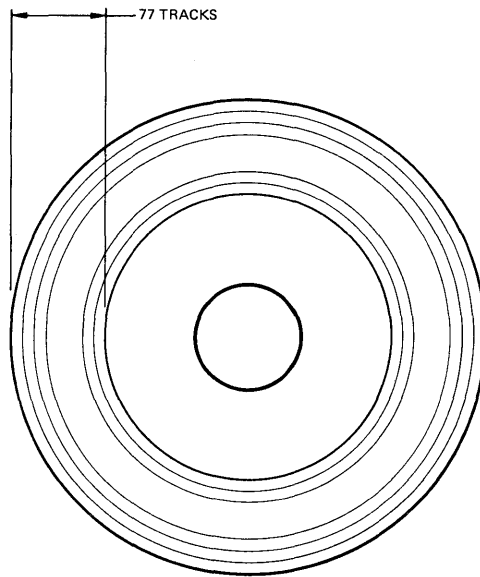
Optional peripherals compatible with the system include a CRT terminal with a full ASCII keyboard, a line printer, and a paper tape reader. In addition, an optional general purpose I/O card supports any RS-232-C compatible device and contains four 8-bit parallel I/O ports which allow the user to interface TTL compatible peripherals to the UNIVERSAL ONE.

**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.

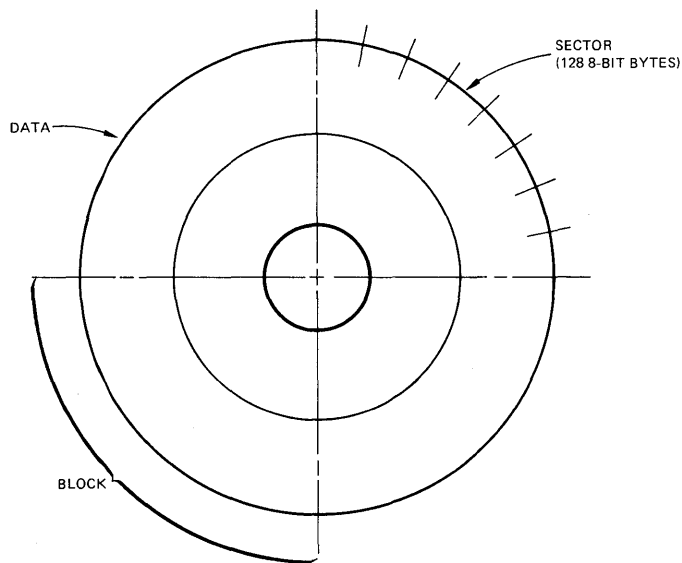
**ASR-33 Teletypewriter.** A standard ASR-33 with a 20 mA current loop or RS-232-C interface can be used as an alternate console I/O device. In addition, the TTY can be used to provide hard copy and to punch paper tapes for file storage off line. (Note: because of its wide availability, Millennium Information Systems does not offer the TTY as an option, it should also be noted that the system will also drive a Silent 700 terminal).

**Line Printer.** A Centronics 306C line printer is available as an option for hard copy output. The line printer is connected through a cable to the floppy disk subsystem,





2-2a Track Layout



2-2b Layout of a Typical Data Track

Figure 2-2. Data Organization on a Diskette

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.

### **2.2.5 User-Supplied Peripherals.**

Any RS-232-C compatible peripheral can be connected to the serial I/O port of the General Purpose I/O card, or any 8-bit parallel device to one of the four parallel ports on the General Purpose I/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 UNIVERSAL ONE software using the method described in the UNIVERSAL ONE System Reference Manual.

## **2.3 SOFTWARE.**

The UNIVERSAL ONE development system software consists of an overall operating system, called UDOS (Universal Disk Operating System), and several more or less independent subordinate programs for specific functions: the Debugger, the Editor, the Assembler, the optional PROM Programmer, and a Systems Readiness Test program. Together, these are referred to as the system programs (as opposed to user or application programs generated by the user, stored in the slave memory and run entirely by the slave CPU, without involving the master CPU).

During operation UDOS resides in the master memory, but other programs reside in the slave memory (except a portion of the Debugger is in the master memory). The subordinate system programs are invoked by UDOS commands.

### **2.3.1 UDOS (Universal Disk Operating System).**

UDOS provides the user with a variety of commands that allow the user to exercise the flexibility of the UNIVERSAL ONE system. UDOS 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 are described in more detail in Chapter 4.

There are two other features of UDOS that deserve mention. These are the Debug Monitor and the PROM programming capability. These are described in Chapter 5 and 8, respectively.

### **2.3.2 The Debugger.**

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.

All Debugger I/O functions are performed by UDOS. 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. The Debug monitor executes in master memory.

### **2.3.3 PROM Programming.**

UDOS 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.

### **2.3.4 The Editor.**

After a source program is conceived and designed, it is input to the UNIVERSAL ONE 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 master CPU. The 16K segment of slave memory in which the Editor is located is also available as a text buffer for the data being operated on by the Editor (all 16K is available, less space occupied by Editor program). UDOS performs all the Editor's I/O requests.

### **2.3.5 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. UDOS handles all the Assembler's I/O requests.

### **2.3.6 Systems Readiness Test.**

The systems Readiness Test allows the user to insure that the UNIVERSAL ONE system is operational. This test is described in Appendix G.

Table 2-1. Performance Specifications and Leading Particulars

CHARACTERISTIC	VALUE												
DEVELOPMENT COMPUTER  MASTER CPU SLAVE CPUs I/O INTERFACES Control Console Others SPARE CIRCUIT CARD POSITIONS MASTER MEMORY SLAVE (COMMON) MEMORY POWER REQUIREMENTS SIZE WEIGHT OPERATING TEMPERATURE HUMIDITY	Type 2650 Up to four – types 8080A, 2650, 6800 Combined TTY and limited RS-232C interface, with EIA XMIT and EIA RCV pin positions interchanged for direct connection to a terminal. High speed paper tape reader, card reader – requires General Purpose I/O card. 15 spare positions can be used selectively for PROM programmers, General Purpose I/O cards, slave memory expansion, and slave CPU expansion. 16,384x8 bits RAM and 256x8 bits PROM 16384x8 bits RAM (can be expanded to 65,536x8 bits or 32,768x16 bits) 3.5 amps at 115 V, $\pm 10\%$ , 60 Hz; 2.0 amps at 230 V, $\pm 10\%$ , 50 Hz 44 cm W x 22 cm H x .59 cm L (17.5 x 8.75 x 23.23 inches) 66 lbs. 0°–55° C (32°–130° F) to 90% relative, non-condensing												
DUAL FLOPPY DISK  CAPACITY Per diskette Per track Per sector ACCESS TIME POWER REQUIREMENTS SIZE WEIGHT OPERATING TEMPERATURE HUMIDITY	<table border="0" style="width: 100%;"> <thead> <tr> <th style="width: 60%;"></th> <th style="width: 20%;">BITS</th> <th style="width: 20%;">BYTES</th> </tr> </thead> <tbody> <tr> <td>77x32x128x8 bits =</td> <td>2,523,136</td> <td>315,392</td> </tr> <tr> <td>32x128x8 bits =</td> <td>32,768</td> <td>4,096</td> </tr> <tr> <td>128x8 bits =</td> <td>1,024</td> <td>128</td> </tr> </tbody> </table> 10 msec/track 3.0 amps at 115 V, $\pm 10\%$ , 60 Hz; 1.5 amps at 230 V, $\pm 10\%$ , 50 Hz 44 cm W x 27 cm H x 60 cm L (17.5 x 10.5 x 23.62 inches) 85 lbs. 10°–38° C (50°–100° F) 20 – 80% relative, non-condensing		BITS	BYTES	77x32x128x8 bits =	2,523,136	315,392	32x128x8 bits =	32,768	4,096	128x8 bits =	1,024	128
	BITS	BYTES											
77x32x128x8 bits =	2,523,136	315,392											
32x128x8 bits =	32,768	4,096											
128x8 bits =	1,024	128											

# *Chapter 3 System Installation and Operation*

## **3.1 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.

## **3.2 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.

### **3.2.1 Unpacking the Universal One Development Computer.**

To unpack the 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 suggested position for each board is shown in Figure 3-1. The board connectors are offset to prevent them from being installed backwards. Push each board firmly into its motherboard 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 logic 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.

### **3.2.2 Unpacking the CRT Terminal.**

Open the carton and remove the packing material from the top of the unit. Lift the terminal and the keyboard out of the carton and set it on a bench top. No further action is required until the system is ready for interconnection and operation.



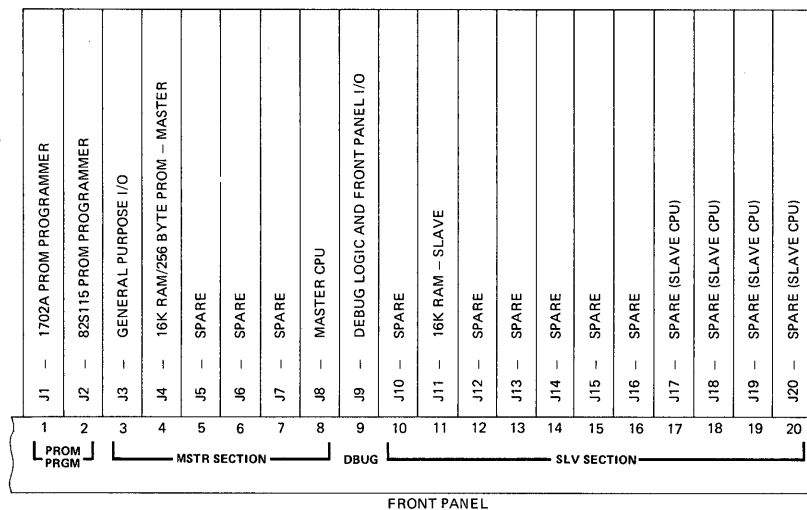


Figure 3-1. Development Computer Printed Circuit Board Layout

### 3.2.3 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 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.

### 3.2.4 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

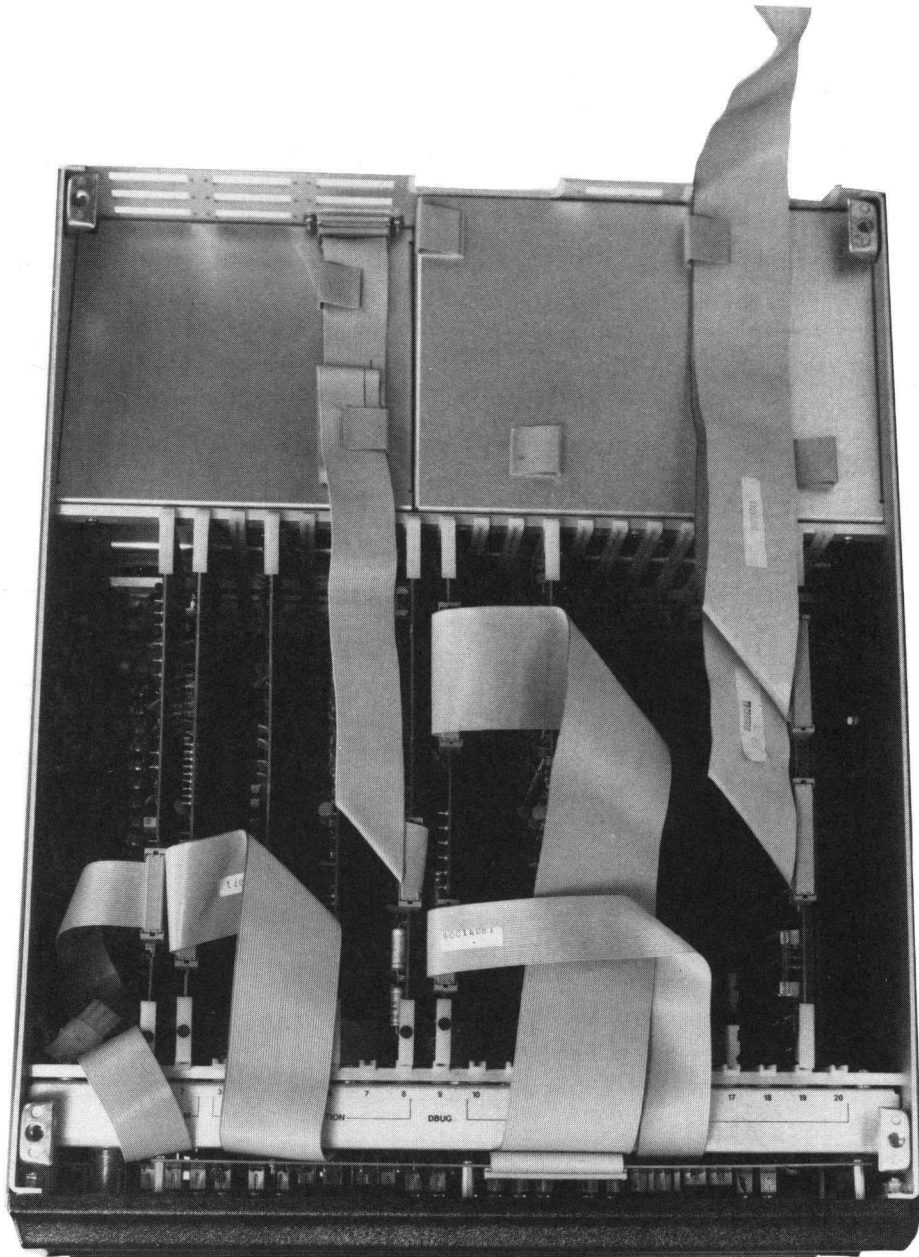


Figure 3—2. Development Computer (Top View)

detailed instructions packed with the printer. These instructions also provide the necessary information on paper installation procedures.

### **3.2.5 Installing the Emulation Cable.**

It is recommended that the emulation 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 emulator 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 emulation cable connector to the socket on the user prototype system, making sure that pin 1 aligns correctly. The UNIVERSAL ONE system is now ready for emulation operation.

## **3.3 PHYSICAL INSTALLATION.**

The units should be placed on a convenient flat surface, close enough to each other for the interconnecting cables to reach. (Figure 3-3 shows the envelope dimensions of each unit of the UNIVERSAL ONE system.)

Since the CRT terminal and the UNIVERSAL ONE 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.

### **3.3.1 Power and Cable Interconnections.**

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. Ensure that all units are wired for the primary input voltage used. Each system unit has a separate power cord and requires a separate outlet for primary power. Current requirements are as listed in Table 2-1.

Refer to Figure 3-4 and make the system interconnections as follows:

1. Connect the dual floppy disk unit to the development computer by routing the 40 lead ribbon cable (90014221) from the rear of the disk unit through the center cableway on the rear of the computer to P3 of the Master CPU card. Ensure 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.

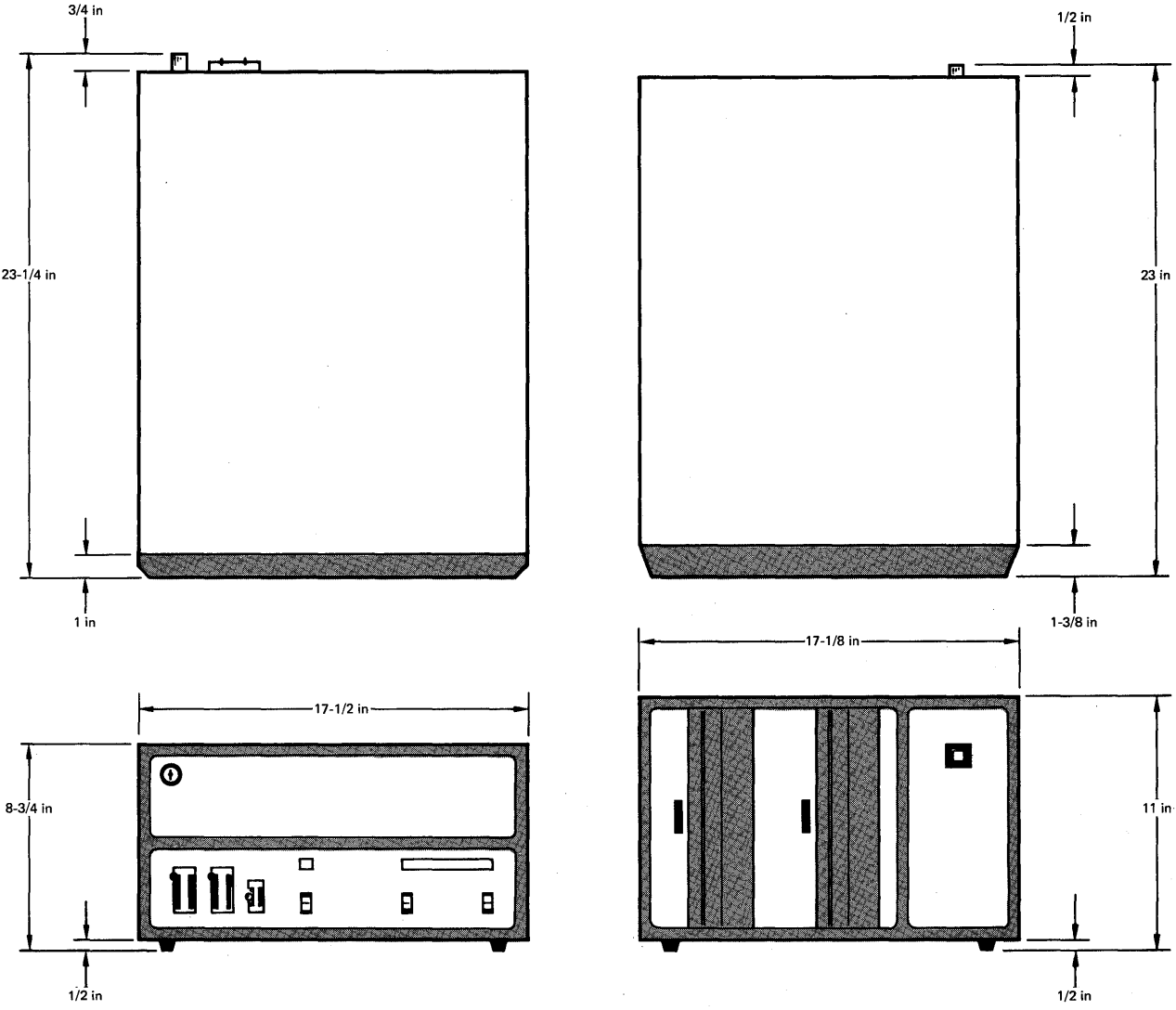
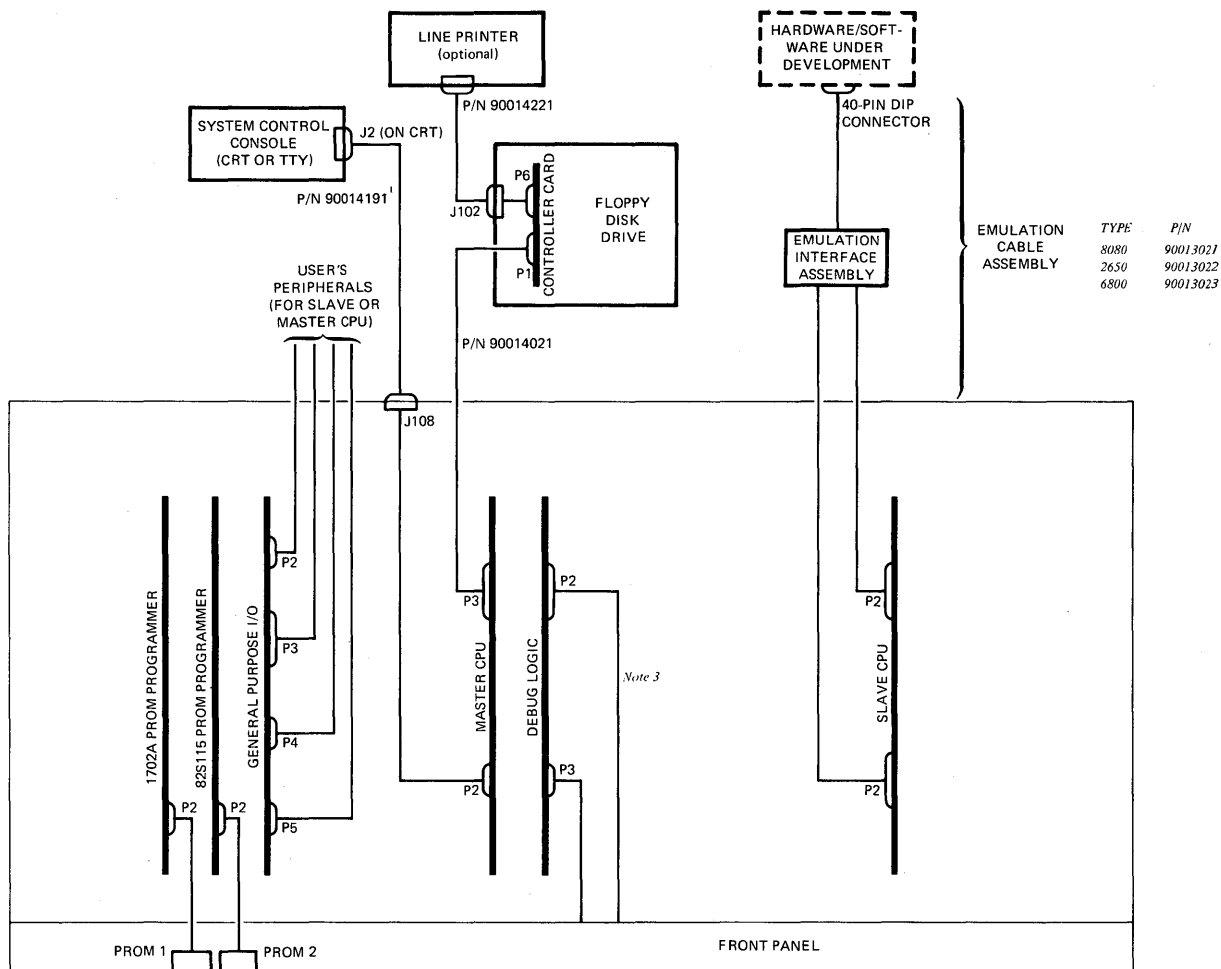


Figure 3-3. Envelope Dimensions of System Units



**Notes:**

1. If a control console other than the CRT Terminal supplied by Millennium Information Systems is used, this cable must be supplied by the user. See Table 3-1 for pin assignments of connector J108 and P3 on master CPU.

2. All units shown (except emulation interface assembly) are individually connected to primary

power source (115 or 230 VAc). Power cables not shown.

3. P2 to front panel cable used only when the optional full display front panel is used on the development computer.

Figure 3-4. Typical Cabling Diagram of UNIVERSAL ONE System Installation

### NOTE

If the operator's control console used is a user supplied device (instead of the CRT terminal from Millennium Information Systems), the user must also supply the interconnecting cable. See Note in Figure 3-4 for J108 interfacing information.

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.

### 3.3.2 Controls and Indicators.

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

### 3.3.3 Development Computer.

Referring to Figure 3-5, 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 backlighted display has the following legends:
  - PWR lights when primary power is applied.
  - MSTR lights when the master CPU has control.
  - SLV lights when the slave CPU has control.
  - RUN lights when the system is running.
3. The DIAG INT switch initiates a reload of UDOS 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 POWER 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 PROGRAM sockets. The left most socket (PROM 1) is used for programming type 1702A MOS PROMs. The center socket (PROM 2) is used for programming type 82S115 bipolar PROMs. The rightmost socket (PROM 3) is reserved for future use. All three sockets are zero insertion force sockets.

Referring to Figure 3-6, the following items are located on the rear panel.

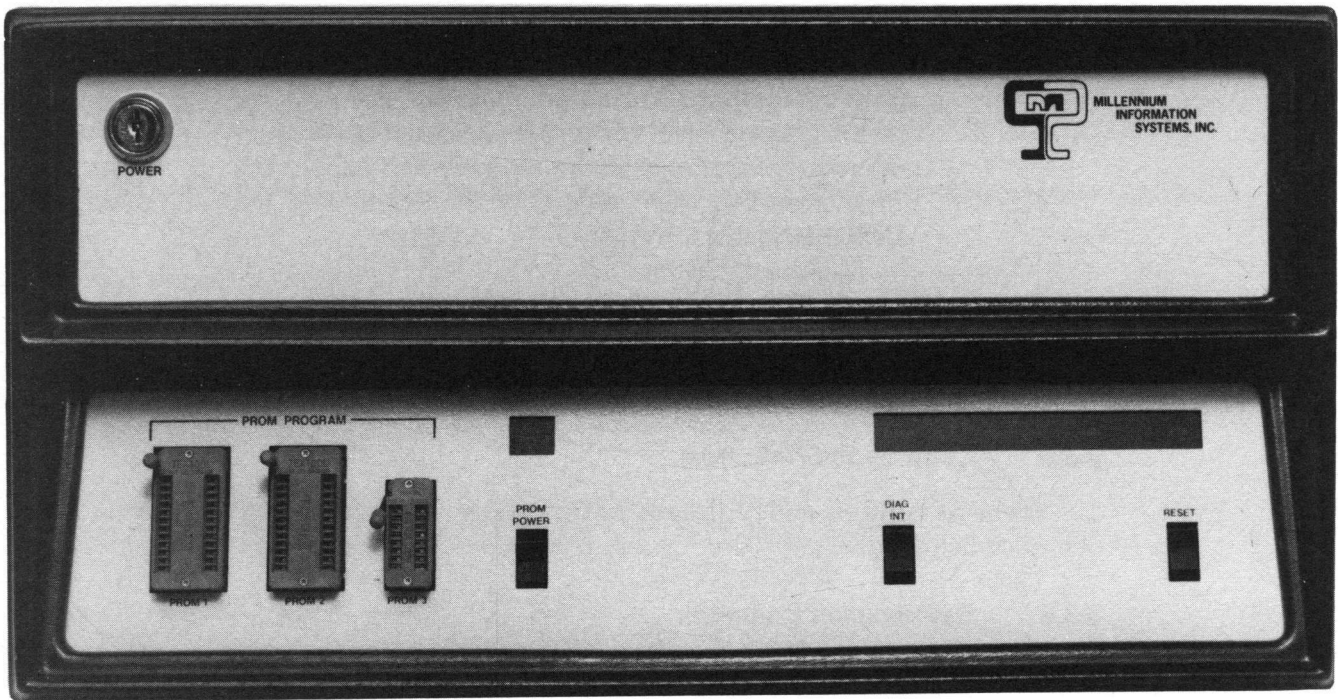


Figure 3-5. Front Panel of the Development Computer

7. The 115/230 VAC 50/60 Hz connector J25 is for primary power; use the power cable supplied with the unit.
8. The 230 VAC/115 VAC slide switch S2 selects the internal voltage taps for 115V or 230V operation. Ensure the F3 and F4 contain the proper fuses for the selected voltage.
9. The barrier terminal strip TB2 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 logic ground to chassis ground, connect the terminals so marked together.
10. 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.
11. J108 is a female connector used to connect the CRT terminal or the teletype to the computer.

### 3.3.4 Dual Floppy Disk Unit.

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

The disk unit rear panel has the following items on it (see Figure 3-7):

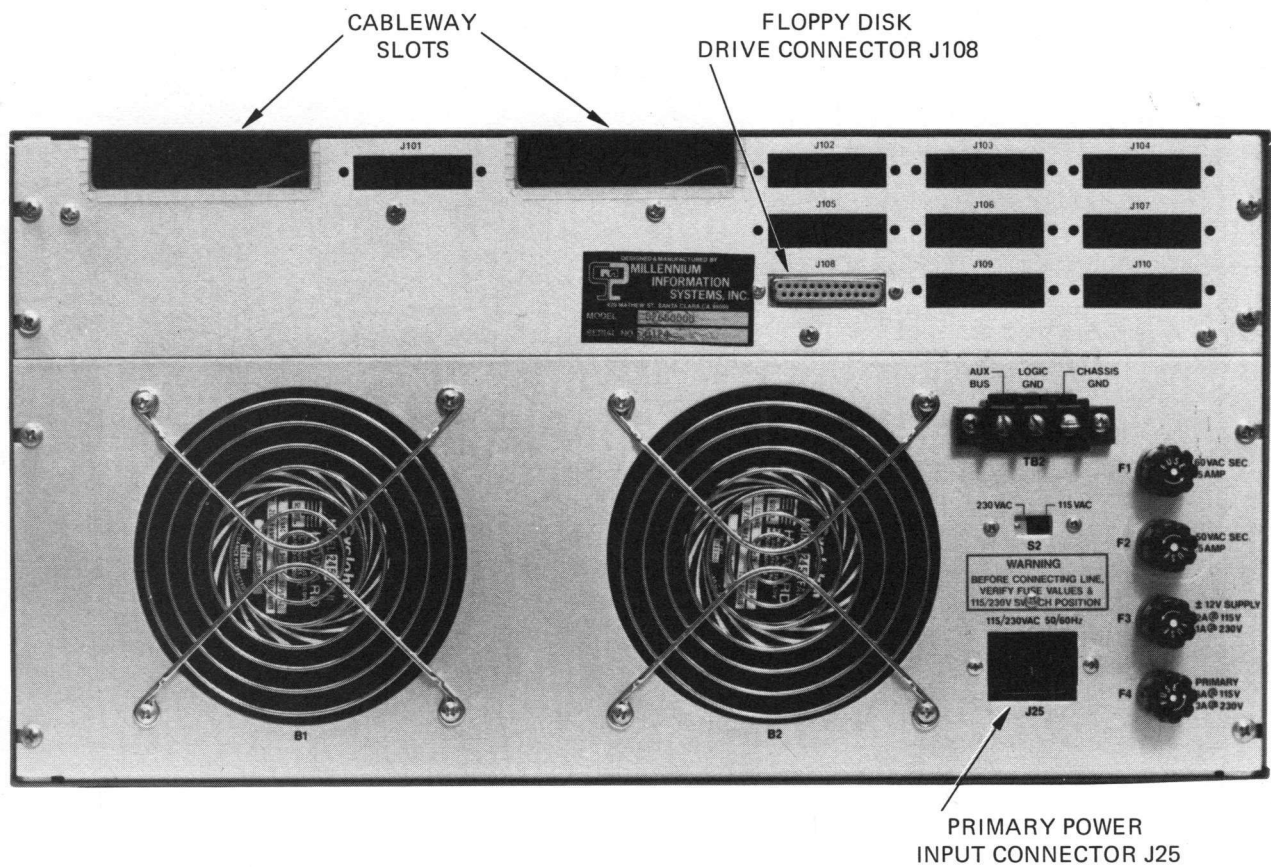


Figure 3-6. Rear Panel of the Development Computer

1. Cableway slot for the cable from the development computer.
2. A 25-pin connector J102 for cable to printer.
3. Spare slot for 25-pin connector J101.
4. Fuses: primary Ac power fuse F1; Ac fuse F2 for disk drives; 5V power supply fuse F3; spare fuse F4.
5. 3-prong receptacle for Ac power cable.

### 3.3.5 CRT Terminal.

The optional 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 a description of all keys and controls, as well as for details of its operation.

### 3.3.6 Line Printer.

The optional line printer is described in the Centronics 306C operator's manual.



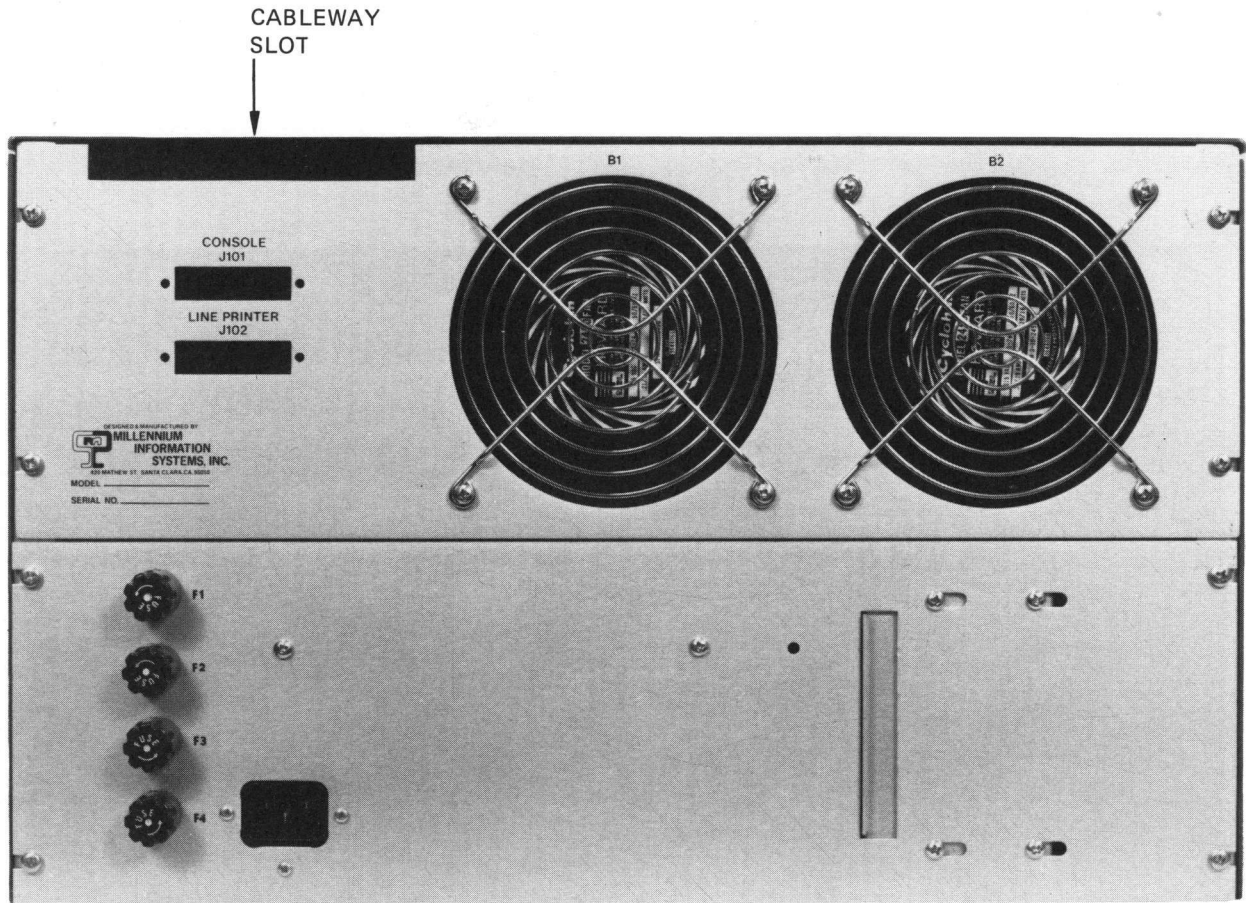


Figure 3-7. Rear Panel of the Floppy Disk Drive

### 3.4 OPERATION.

#### 3.4.1 Formatting and Verifying New Diskettes.

Before any new diskette is used, it must be formatted and verified. If a diskette must be formatted, start up the system according to paragraph 3.4.2 and then follow the procedures outlined in the discussion of the UDOS commands `FORMAT` and `VERIFY` in paragraph 4.6.5.

On every diskette, there is a write protect slot (see Figure 3-8). If this slot is covered, the diskette is write enabled, meaning that data may be written onto or 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 it. If an attempt is made to write to a write-protected disk, the appropriate UDOS error message will be displayed.

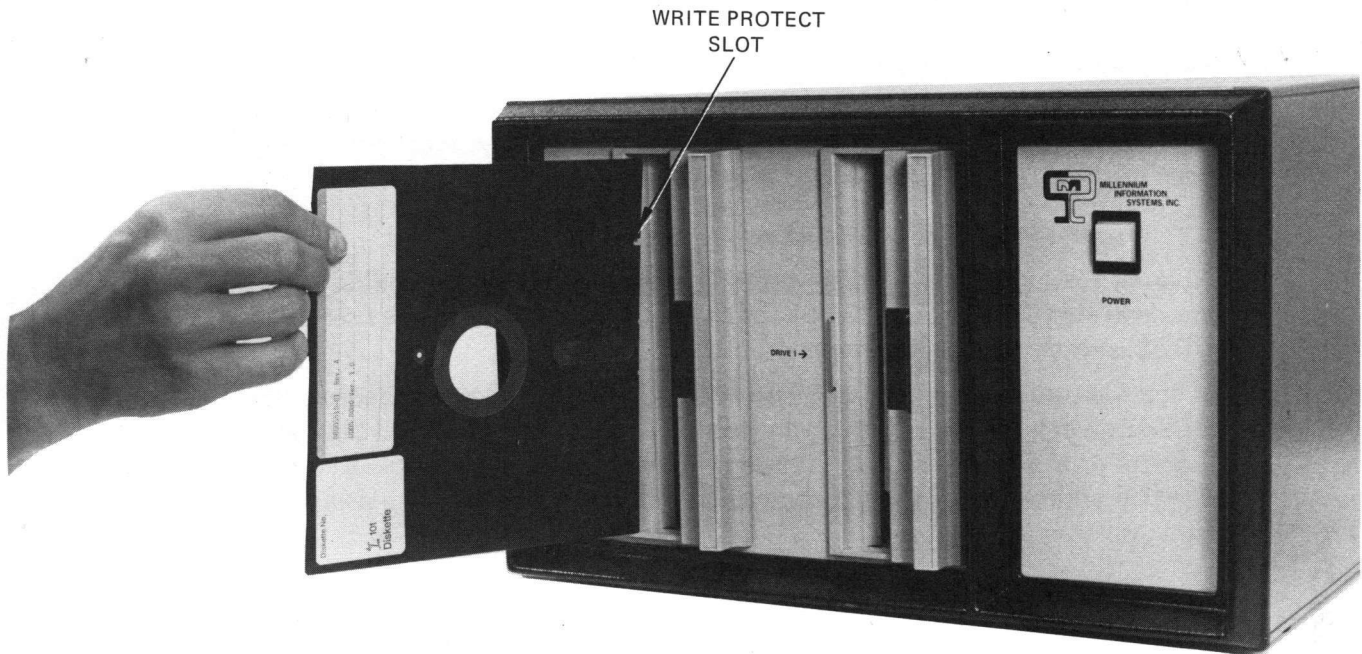


Figure 3-8. Inserting a Diskette

### 3.4.2 System Startup Procedure.

To power up the UNIVERSAL ONE system and load the operating system (UDOS) into memory, perform the following steps:

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

**CAUTION**

Do not turn power on or off the disk unit with diskettes installed and doors closed, because media data might be destroyed.

- 2) Power up the floppy disk unit.
- 3) Allow a 5 minute warmup time to let the disk drive electronics reach stable temperature.

- 4) Insert the system diskette into drive 0. The correct method for inserting a diskette is shown in Figure 3-8. Ensure that the label side is toward the POWER switch and that the label is the last part of the diskette inserted into the drive. Close disk drive door.
- 5) Apply power to the line printer.
- 6) Apply power to the development computer. This will cause an automatic read from drive 0, which will load UDOS into master memory. When UDOS has been loaded, a welcoming message will be displayed on the terminal:

```
> UDOS VER 1.0 TYPE
```

Where TYPE will be type of slave CPU enabled, such as 8080, 6800, 2650, etc., and the > is the UDOS prompt character which informs the user that UDOS is ready to accept commands.

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

#### NOTE

The computer will automatically switch the initialization process to drive 1 if only drive 1 contains a diskette.

- 8) 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 UNIVERSAL ONE System Reference Manual for information on changing the baud rate on the Master CPU card.

### 3.4.3 Manual Reset.

If a reinitialization of the system is desired during operation, the user may reload UDOS by pressing the RESET switch on the front panel. The welcoming message and the prompt character will be issued after UDOS has been loaded. (Do not use the manual RESET when using the Editor — data files being created may be lost.)

## System Installation and Operation

Table 3—1. Pin Assignments of Connectors J108 and P2.  
(Development Computer Back Panel and Master CPU)

P2 PIN	J108 PIN	SIGNAL	DESCRIPTION	COMMENTS
1	1	CHS GND	(Not Used)	
2	14	TTX	TTY Current Loop Input -	RCVR IN
3	2	EIA RCV DATA	EIA Serial Input	RCRV IN
4	15	TTRDR +	Tape Reader Control Out	47 ohm to +12
5	3	EIA XMIT DATA	EIA Serial Output	EIA Driver
6	16	TTRDR -	Tape Reader Control Out	Open Collector Driver
7	4	REQ TO SND	Request to Sned	2.2K Pull-up
8	17	TTRCV +	TTY Current Loop Output +	620 ohm to +12V
9	5	CLR TO SND	Clear to Send	2.2K Pull-up
10	18	TTRCV -	TTY Current Loop Output	Open Collector Driver
11	6	DATA SET RDY	Data Set Ready	2.2K Pull-up
12	19	—		
13	7	SIG GND	EIA Ground	
14	20	DATA TERM RDY	Data Terminal Ready	2.2K Pull-up
15	8	CARR DET	(Not Used)	
16	21	—		
17	9	—		
18	22	RING IND	(Not Used)	
19	10	—		
20	23	—		
21	11	ORIG	(Not Used)	
22	24	—		
23	12	LOCAL	(Not Used)	
24	25	SND RESTRAINT	(Not Used)	
25	13	TTX +	TTY Current Loop Input +	620 ohm to +12V
26				

# *Chapter* **4** *Universal Disk Operating System*

## **4.1 INTRODUCTION.**

This chapter describes the Universal Disk Operating System (UDOS) for the UNIVERSAL ONE system. General topics include the use of the keyboard to enter commands or request control of the system, an overview of the UDOS file structure, a catalog of the UDOS commands and their functions, and a study of the command file capability and overlay areas. In addition, summaries of the UDOS commands and UDOS error messages are provided at the end of the chapter.

## **4.2 UDOS OVERVIEW.**

The Universal Disk Operating System (UDOS) executes in master memory and consists of two sections: resident UDOS which is always present in the master memory, and the UDOS overlays, which are loaded into the master memory automatically from the system diskette, whenever certain UDOS commands are invoked. The resident UDOS section, in turn, consists of a PROM portion and a RAM portion.

The PROM resident portion is the UDOS BOOT, which initially loads the resident portion of UDOS from disk into RAM master memory, when the RESET switch is depressed.

The RAM resident portion of UDOS is comprised of the following modules:

- Command Line Processor
- SVC Processor
- Job Dispatcher
- File Manager
- Device Drivers

These modules are described in the following paragraph 4.2.1. In addition, peripheral device I/O buffers are also located in the RAM resident portion.

The section of UDOS located in the disk consists entirely of the overlays, which process most of the UDOS commands; the only commands always resident in RAM are GO, SYSTEM, LOAD and XEQ.

The EDIT and ASM commands are also a part of the UDOS command set, but because the Editor and Assembler programs both are executed out of the slave memory these commands need not be either resident or in an overlay. Instead, they cause the Command Line Processor to load the corresponding program into the slave memory and need not be processed further by the UDOS program modules.

#### 4.2.1 Resident UDOS Modules.

The Command Line Processor operates on commands input from the CRT terminal (i.e. the system control console) or from a command file stored in the disk. It interprets the command(s), prepares a parameter list, and then causes the function to be performed, by transferring control to the appropriate resident procedure, or by loading and executing an overlay.

The SVC Processor operates on internal requests for I/O or a UDOS service function. All of the I/O communication with system peripherals, for system programs running under the slave CPU, are performed by the SVC Processor.

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

The floppy disk File Manager and other Device Drivers control operation of the peripheral devices in the system, all of which are interrupt driven.

#### 4.2.2 UDOS Overlays.

The UDOS overlays consist of all UDOS commands except the four memory-resident commands.

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

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

<u>Overlay Area 1</u>		<u>Overlay Area 2</u>			<u>Overlay Area 1 &amp; 2</u>
COPY	RHEX	ABORT	DELETE	DSTAT	LDIR
DEBUG	RSMS	ASSIGN	DEVICE	SET	MODULE
DUP	VERIFY	BKPT	DUMP	SLAVE	
FORMAT	WHEX	CLBP	EXAM	SUSPEND	
PRINT	WSMS	CLOSE	PATCH	TRACE	
RPPROM	WPROM	CONT	RENAME	TYPE	
CPROM		KILL	RESET	STATUS	

UDOS 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 (described in paragraph 4.6.6):

> R

While the tape is being processed, file maintenance could be performed. Pressing the ESCAPE key would suspend RHEX execution and display the UDOS prompt character >>. The DELETE command (paragraph 4.5.1) could then be entered:

>>DEL FILE1/1 DATA/1 SOURCE/1 (r)

When the (r) (RETURN key) was entered, the RHEX command was 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.

### 4.3 FILES, DEVICES, AND CHANNELS

UDOS 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 UNIVERSAL ONE system, files are stored on 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 UDOS. 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. List of UDOS Device Names.

DEVICE NAME	DEVICE
CONI	CONSOLE INPUT
CONO	CONSOLE OUTPUT
LPT1	LINE PRINTER1
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.

#### NOTE

Although UDOS software supports a high speed paper tape reader, this peripheral is available only on request.

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. UDOS can automatically create the necessary new files or search other diskette directories.

A filename must have 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 must be 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.

UDOS 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 the second 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, UDOS 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 ENTERING UDOS COMMANDS.

When the prompt character > is displayed, the user is allowed to enter commands to UDOS. 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;
- PARAMETER is all of 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 portion of the following line (after the prompt character):

```
> 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, UDOS is told that a command is waiting to be interpreted. UDOS identifies the command, loads the appropriate program into master memory, and control is passed to the loaded program to perform the requested function. In the LDIR 0 example, the command LDIR, (the List Directory command) is identified by UDOS, 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 desires 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 UDOS 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.

## 4.5 SPECIAL KEYS.

UDOS 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. UDOS also recognizes CTRL-Z as a special character.

### 4.5.1 Delete Key.

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.

#### 4.5.2 Escape Key.

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

- 1) Input is being performed for a UDOS command;
- 2) Input is being performed for the Editor;
- 3) 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 UDOS 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 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 a UDOS or user program is executing, as distinguished from console input being performed, is discussed in paragraph 4.6.3, System Control Commands.

#### 4.5.3 Space Bar.

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.

#### 4.5.4 CTRL-Z Command.

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.

## 4.6 THE UDOS COMMANDS.

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

- \* Commands that are primarily used in conjunction with the command file facility are described in paragraph 4.6.8.
- \* Commands that are associated with the Debug function are described in Chapter 5.
- \* The EDIT command is described in Chapter 6.
- \* The ASM command is described in Chapter 7.
- \* Commands that are used for PROM programming and verification are described in Chapter 8.

### 4.6.1 The UDOS Command Structure.

All UDOS commands are structured as follows:

COMMAND PARAMETER

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

- 1) The command 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 are analyzed in the next paragraph.
- 5) The error messages that the command might evoke are listed. The format and a list of UDOS error messages is presented in paragraph 4.6.9 and Table 4-2.

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 file. To edit the file DTA1/1 using the Editor, the user would issue the command EDIT DTA1/1 where DTA1/1 is a specific instance of the general parameter FILENAME. Note that in most commands it is required that the name of the file be followed by /D, where D is the floppy disk drive number.

**D** refers to the disk drive number. To duplicate the diskette on drive 0 on to the diskette on drive 1, 0 would be used for D1 and 1 would be used for 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 DTA/1 on the line printer, the user would replace PRINT FILENAME (L1L2) with PRINT DTA1/1 8 14.

#### **4.6.2 UDOS Command Completion.**

Most UDOS 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 UDOS system program identifier (see Table 4–3) and EOJ is the end of job message. Completion of any user-entered command causes the UDOS prompt character > to be displayed.

### **4.6.3 System Control Commands.**

The user may control the execution of 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 UDOS. The space bar key is used to control UDOS 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 by an ASSIGN command.

*ESC*  
*Space Bar*

ESC  
or  
ESC ESC

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

- a) If input was being performed to UDOS, the Editor, or an application program, refer to paragraph 4.5.2 for a discussion of the actions taken.
- b) If an UDOS 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 UDOS 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 UDOS programs or the user program after it has been temporarily suspended by ESCAPE, either press RETURN or enter a valid UDOS command.

When the ESCAPE key is depressed, UDOS will respond with a double prompt 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 is discussed in paragraph 4.5.3.

## *Suspend*

### *Cont*

<p><u>S</u>SPEND NAME or <u>S</u>SPEND * or <u>S</u>SPEND /</p>
---

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 paragraph 4.6.7. 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

<p><u>C</u>ONT NAME or <u>C</u>ONT * or <u>C</u>ONT /</p>
---

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, UDOS 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* *Close*

<p><u>A</u>BORT NAME or <u>A</u>BORT * or <u>A</u>BORT /</p>
--

This command causes an active UDOS 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.

### \*ABT\* Error Responses:

- 24 — Job not active
- 31 — Parameter required

<p><u>C</u>LOSE CH (...CH)</p>
--------------------------------

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 UDOS 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*

**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

## *SEArch ON*

## *SEArch OFF*

### 4.6.4 System Option Commands.

The user may set the value of various system options that remain in effect during all subsequent operations, until removed or changed. To set system operations, the following commands are used:

SEARCH  
SYSTEM  
DEVICE  
CLOCK

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

SEARCH ON (N) or SEARCH OFF
-----------------------------------

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 2.

If automatic file searching is not being performed, i.e. SEARCH is off, then when the user specifies a filename with some command that he enters, UDOS 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, UDOS will search, in circular manner, 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***

### ***DEVICE***

#### ***CLOCK ON CLOCK OFF***

**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.

\*UDOS\* Error Responses:

9 — Invalid drive number

**DEVICE DEVICE U**  
or  
**DEVICE DEVICE D**

This command informs UDOS of the availability of a peripheral device. The argument DEVICE must be one of the system device names (see Table 4-1).

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

**CLOCK ON**  
or  
**CLOCK OFF**

This command enables or disables the 100 msec real time clock interrupt. (The real time clock is synchronized with the system clock and is available out of master memory for use by slave programs and other purposes.) Default value of CLOCK is ON.

\*CLK\* Error Responses:

30 — Invalid parameter  
31 — Parameter required

#### 4.6.5 System Utilities Commands.

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

FORMAT  
VERIFY  
RENAME  
DUP  
LDIR  
DELETE  
COPY  
PRINT

FORMAT initializes the diskette for use by the UNIVERSAL ONE 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***

### **FORMAT D (IDENT)**

All virgin diskettes must be formatted and verified before they can be used by UDOS.

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

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 UDOS. If a bad sector is detected on tracks 0 through 4 (the directory and UDOS area) the formatting process is aborted.

If the diskette will not be used for storage of system software, the area reserved for UDOS may be freed for other uses (after formatting is complete) by entering DELETE UDOS command. This, however, will prevent ever using this diskette for system programs.

During formatting, the ASCII character string IDENT is written to the diskette and serves as the diskette identification. This identification is always 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.

#### **\*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, the entire track on which the bad sector is located is 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 UDOS 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***

<p><u>RENAME</u> OLDFILE/D NEWFILE or <u>RENAME</u> D IDENT</p>
---

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 it 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



<b>DUP D1 D2 (IDENT)</b>
--------------------------

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*

## *DElete*

**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 UDOS 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

<u>COPY INPUT (...INPUT) OUTPUT</u>
-------------------------------------

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 completely 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***

## ***PRINTL***

<p><u>PRINT</u> FILENAME (DEVICE)(L1 L2) or <u>PRINTL</u> FILENAME (DEVICE)(L1 L2)</p>
--

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 as they are displayed or printed.

### **\*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

## 4.6.6 Object Program Utility Commands

The user will generally manipulate object program files to and from slave memory with these commands:

MODULE  
RHEX  
WHEX  
CSMS  
WSMS

MODULE writes a binary format 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 many semiconductor companies for the generation of PROMs and is described in Appendix F.

**MODULE FILENAME A1, A2, A3 (IDENT)**

This command writes a binary format load module to FILENAME. A1 and A2 are the memory bounds in the slave memory and A3 is the starting address of the program; 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 A1 and A2, as well as A3.

**\*MOD\*** Error Responses:

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

## *Rhex*

**RHEX (/BIAS) (DEVICE)**

This command reads an absolute hexadecimal object file into slave memory. BIAS is used to alter the absolute 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 polarity value of BIAS is assumed to be +.

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

### NOTE

The hexadecimal format varies between slaves, as determined by the microprocessor manufacturer.

#### \*RHX\* Error Responses:

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

<code>WHEX A1 A2 ... (,,A1 A2) (A3) (DEVICE)</code>
---

This command outputs an absolute hexadecimal format file from slave memory. The pairs A1,A2 are hexadecimal address constants that indicate the bounds of the slave memory segment to be written to the file. A3 is an optional program 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, the starting address vector A3 must be specified.

This command writes, in hexadecimal ASCII 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.

#### NOTE

The hexadecimal format varies between slaves, as determined by the microprocessor manufacturer.

#### \*WHX\* Error Responses:

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

## *CSms*

## *WSms*

### 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 read. 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



#### 4.6.7 Command Files.

UDOS provides the user with the capability of executing a sequence of UDOS commands by issuing a single command. This capability is implemented through the use of command files, which is a sequence of UDOS commands, identified by a single name. When the name of the command file (as an example, we shall use the name COM1) is used as a UDOS command:

```
> COM1 (r)
```

UDOS first determines that COM1 is not one of the basic UDOS commands and then searches the system directory for the file COM1. When UDOS locates COM1, it treats the first line in COM1 as an UDOS 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 UDOS command, UDOS will locate LISTALL and execute the first line as an UDOS 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.

UDOS also allows 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. For example, if LISTALL were:

```
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
```

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 paragraph 4.6.8).

Command files cannot be nested, but they can be chained. That is, if the last UDOS 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 UDOS 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.

#### **4.6.8 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 UDOS switch KILL to be set on or off.

If the KILL switch is on, a command file will be aborted if the current UDOS 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 UDOS 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 UDOS switch TYPE to be set on or off.

If the TYPE switch is on, UDOS command lines executed by the command file processor will be output to the system console. If the TYPE switch is off, UDOS command lines or 'EOJ' message will not be displayed on the console. Error messages from UDOS 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 UDOS.

The primary use of the \* command is in command files where it can be used to display comments around UDOS commands.

#### 4.6.9 UDOS Error Messages.

All UDOS error messages are of the following form:

\* id \* ERROR #

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

```
◆WHX◆ PAR 03  
◆WHX◆ ERROR 30
```

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

Table 4-2. UDOS Error Messages

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

Table 4-3. UDOS System Program Identifiers

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

# *Chapter* **5** *the Debugger*

## **5.1 INTRODUCTION.**

This chapter describes the Debugger, or the Debug Program. General topics include an overall description of the Debug Program, entry and exit from it, a sample debug session, and a description of each of the Debugger commands. Further information on versions of the Debugger, related to particular slave CPUs, is contained in the manual supplements provided with each of the slave CPU cards and emulation cables.

## **5.2 THE DEBUG PROGRAM.**

The Debugger is a subsystem of the UDOS, that is enhanced through UNIVERSAL ONE hardware features which allow it to control program execution on the slave CPU. When Debugger is executing, the user has a subset of the UDOS commands plus a set of Debugger commands at his disposal.

Functionally, the Debugger is a combination of software, hardware on the debug logic card, and the emulation cable. It can perform the following functions:

1. display memory and register contents, as well as Debug status, and allow these values to be modified;
2. control program execution and allow the user to request control at specified locations using breakpoints;
3. trace program execution and display relevant machine states;
4. allow debugging in the user's prototype system.

To accomplish these functions, the Debugger monitors the user's progress and state and saves necessary information. For example, 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.) As another example, 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 emulation cable allows the user to connect the slave CPU directly to his development hardware, where in-circuit-emulation may be performed.

There is a different version of the Debugger for each of the different slave CPUs available with the UNIVERSAL ONE system. The basic operation of Debugger and the functions of all Debug commands are the same for all versions, however some command parameter formats and some formats of displays generated in response to commands, vary from version to version. These differences are described in the manual supplement provided with each slave CPU.

If you are familiar with debugging programs, paragraph 5.3, INVOKING THE DEBUGGER, and paragraph 5.5, DEBUG COMMANDS, are recommended. If you are not familiar with debugging programs, the above paragraphs plus paragraph 5.4, SAMPLE DEBUG SESSION, are recommended. For the sample debug session, it is recommended that the reader be familiar with the basic architecture and operation of the 2650 microprocessor.

### **5.3 INVOKING THE DEBUGGER.**

There are three important facts that require explanation before discussing use of the Debugger:

1. The special UDOS keys, ESC and space bar retain their meanings while the Debugger is executing. Their use is discussed in paragraphs 4.5.2 and 4.5.3. Note in particular the impact of the ESC key on the EXAM command.
2. If it is necessary to switch from the master CPU to the slave, or 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 paragraph 5.5.
3. Executable programs are created and stored in the UNIVERSAL ONE system one of 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 UDOS command used to read hex format files.
  - b) Binary format: one byte of data is stored for each byte of object code. The UDOS command MODULE creates binary format files. LOAD is the UDOS command used to read binary format files.

When the user desires to invoke the Debugger, he first must issue the SLAVE command (paragraph 5.5), then load the program to be debugged into slave memory, by using either the LOAD or RHEX commands. Next, the DEBUG command can be issued to enter the Debug Program and the UDOS command CLOCK OFF must be issued to disable the real time clock interrupt.

When DEBUG is entered, the debug package appropriate to the slave microprocessor being used is loaded into master memory (overlay area 1). In addition, a small trace package, if necessary, is loaded into the slave memory. This package, 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. This package for most microprocessors will be contained in PROM on the slave CPU, instead of in slave memory.



After the Debug package has been loaded, the UDOS prompt character > is issued to the console. Whenever this prompt is displayed, the Debugger is ready to accept commands. Select the desired DEBUG mode (TRACE, BREAKPOINT, etc.), and you are ready to initiate execution of the application (user) program.

### NOTE

The commands available to the Debug user are listed in Appendix B. Note that several of the primary functions of the Debugger, such as examining the altering memory (the EXAM command), and execution control (the GO and EXQ commands) are UDOS commands that are also used by the Debugger. Other UDOS commands however, are not available to Debugger.

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

1. Depress the ESCAPE key twice. If the TRACE mode is active, a single depression is sufficient.
2. When the UDOS prompt character appears, enter the desired commands.
- 3: When it is necessary to continue the user program, entering the GO command will continue the user program from the point at which it was interrupted.

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

1. The user requests console control by depressing the ESCAPE 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 UDOS command ABORT. This may be accomplished by ABORT DEBUG or ABORT\*. In either case, both the Debugger and the user program are terminated.

### **5.4 SAMPLE DEBUG SESSION (USING A 2650 SLAVE).**

Let's monitor the program in Figure 5-1,a with the Debugger so that we may examine some of the debug features. The sample program has been assembled into hex object code, written to a disk file named DEMO. The starting location for the program is 3000.

The System is in the target slave mode 0 (see SLAVE command, paragraph 5.5) by default, so an initial SLAVE command is not required.

```

DEMO   ORG     H'3000'
R0     EQU     0
R1     EQU     1
*
*
TOPP   ADDZ    R1      ADD REGISTER 1 REGISTER 0
LOOP   ADDI,R0 1      INCREMENT R0
       CMI,R0 0      COMPARE R0 WITH 0
       BCFR,0 LOOP   IF COMPARE FAILED,BRANCH TO LOOP
       BCTR,0 TOPP   IF COMPARE SUCCEEDED,BRANCH TO TOP
       END      DEMO

```

a. Sample 2650 Microprocessor Program

```

1  > RHEX DEMOBJ
2  ♦RHX♦ EQU
3
4  > DEBUG
5
6  > SET R0 0 1
7
8  > DSTAT
9  P=0000                                R=00 01 00 00 00 00 00 00 00

```

b. Loading Object Code, Activating Debugger, and Initializing Slave Registers

```

1  > TR A S
2
3  > GO 3000
4  LDC INST  MNEMON  XP 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 40
6
7  > G
8  3001 8401  ADI ,00                01                02 01 00 00 00 00 00 40
9
10 > G
11 3003 E400  CMI ,00                00                02 01 00 00 00 00 00 40
12
13 > G
14 3005 987A  BFR ,00  - 3001          =3001 02 01 00 00 00 00 00 40

```

c. Single Step Trace All Mode

Figure 5-1. Displays During Sample Debugging Session

To load the hex code from the file DEMO, enter the UDOS command RHEX DEMO (line 1 of the Figure 5-1,b). This command loads the object code in the disk file DEMO into slave memory. (If the file DEMO contained a binary load module produced by the use of the MODULE command, the command LOAD DEMO would be used instead of RHEX.) To load the DEBUG package, enter the UDOS command DEBUG (line 4 of Figure 5-1,b). Both the object code from the sample program in Figure 5-1,a and the DEBUG trace package now reside in slave memory. The DEBUG package is located in master memory overlay area 1.

The sample program uses the slave CPU registers 0 and 1. If we wish to give these registers specified 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 0 1. (line 6 of Figure 5-1,b). 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 5-1,b) causes the information on line 9 of Figure 5-1,b 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. For the 2650 slave this display is organized as follows: the area in Figure 5-1,b indicated by ① displays the 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 upper and lower program status word values.

Suppose we wish to trace the execution of this program. Turn the TRACE function on, as shown on line 1 of Figure 5-1,c. 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 (see TRACE command description, paragraph 5.5). 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 5-1,c). 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.)

The Debugger now assumes control, proceeds with one step of program execution, and produces the TRACE display (lines 4 and 5 of Figure 5-1,c). The headings in line 4 have the following meanings (all values are in hex):

LOC	is the location of the last instruction executed.
INST	is the hex value of the last instruction executed.

MNEMON	is the 2650 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 (register 0)
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

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 the full 2650 mnemonic ADDZ, and the next nine entries indicate register contents.

We can single step through the next instruction by entering the UDOS command G (the GO command, line 7 of Figure 5-1,c). As can be seen in lines 7 - 8, as well as lines 10 - 11 and 13 - 14 of Figure 5-1,c, the Debugger performs a single step and then displays the TRACE information.

Suppose we do not wish to single step, but still wish to trace all the instructions executed. This can be accomplished by altering the TRACE mode: TR A (line 1 of Figure 5-2,a) 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 5-2,a), the Debugger takes control of the slave CPU after every slave CPU instruction is executed, and 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 TRACE (the effect can be noted on line 14 of Figure 5-2,a) and displays the double prompt >> (line 15 of Figure 5-2,a) to indicate readiness to accept commands.

Suppose we desire to not view any TRACE displays until the instruction at 3007 is executed. This can be accomplished by the actions shown in Figure 5–2,b. First we set a breakpoint by the command in line 1 of Figure 5–2,b. 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 5–2,b turns the TRACE mode off.

Execution is resumed using the GO command (line 5 of Figure 5–2,b). The Debugger monitors the slave program execution, and when the instruction at 3007 is executed, the display on lines 6 and 7 of Figure 5–2,b 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 wish to monitor the execution of all the branch instructions. This can be accomplished using the commands in Figure 5–2,c. First, let's set Register 1 to FA (line 1 of Figure 5–2,c). Then, using the DSTAT command, we can view the current DEBUG status (lines 3 and 4 of Figure 5–2,c). Note that the presence of the breakpoint at 3007 is indicated in this display. The WR following BP=3007, refers to the fact that either a read or a write to location 3007 will cause a break. Line 6 of Figure 5–2,c, TRA J, is the TRACE (TRA) command which requests that only branch (J) instructions be displayed.

When the slave program is continued with the GO command (line 8 of Figure 5–2,c), the Debugger displays the TRACE information for all branch instructions executed whether the branch was performed or not (lines 9 – 15 of Figure 5–2,c). The Debugger informs us that a break has taken place in line 16 of Figure 5–2,c.

If we desire to clear a breakpoint, the command in line 4 of Figure 5–2,c 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 5–2,c, the effect of the CLBP operation is clear.

When we are finished with a debug session, the Debugger must be exited using the UDOS command, ABORT (see line 8, Figure 5–2,c).

## 5.5 DEBUG COMMANDS.

This section lists commands that are used with the Debugger. Besides the DEBUG command itself, there are eight commands that are used both with the Debugger and under UDOS. These commands are:

GO	EXAM
LOAD	PATCH
XEQ	STATUS
DUMP	SLAVE

```

1 > TR A
2
3 > GO 3000
4 3000 81 ADZ ,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 ADI ,00
15 >>

```

a. 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 00 21
7 3007 BREAK
8
9 >

```

b. Using Breakpoints

```

1 > SET R1 FA
2
3 > DSTAT
4 P=3007 BP=3007 WP R=00 FA 00 00 00 00 00 00 21
5
6 > TRA J
7
8 > GO 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

```

c. Clearing Breakpoints and Terminating a Debug Session

```

1 > DSTAT
2 P=3007 BP=3007 WP 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
8 > ABORT DEBUG

```

d. Clearing Breakpoints and Terminating a Debug Session

Figure 5-2. Typical Displays During Various Debugging Modes

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 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. STATUS displays the status of the slave CPU and the job being executed by it. SLAVE sets the emulation mode.

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.

Note that as with all UNIVERSAL ONE system commands, the RETURN key (r), is used to start execution of any command.

**DEBUG (ADDRESS) (DEVICE)**

This command causes the Debug Program 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. 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.

**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 at which it was stopped.

\*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 previously 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.

## *Xeq*

### *Dump*

#### \*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, previously created using the MODULE command, to be loaded into slave memory and executed. This command is the equivalent of LOAD FILENAME followed by the GO 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

#### DUMP A1 (A2) (DEVICE)

This command causes the contents of slave memory to be displayed on DEVICE, beginning with address A1. In the display two hexadecimal characters represent 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 always automatically 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 are 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 a single byte of the slave memory at location ADDRESS to be displayed on the console. The user then has several options: a) display the next sequential byte at ADDRESS; b) display address of the current location and its contents; c) replace the current memory byte with data entered on the control console 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 address of the current byte and the byte.
HEX DATA PAIR	Replace the current memory location with a 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  $10_{16}$ .

The ESCAPE Key has a different interpretation when the EXAM command is being used. Refer to paragraph 4.5.2 for details.

For example, if locations 3000-3003 contain AF, CB, DF, F8 respectively, the EXAM command can be used as follows (user interaction underlined):

```
>EXAM 3000
3000=AF CB DF F8(r)
>
```

When the space bar is depressed, the next sequential byte is displayed. When return is entered, the command is terminated. To increment each location, this sequence can be used:

```
>EXAM 3000
3000=AF-01 CB-02 DF-03 F8-04 . . .
```

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*

### *S*Status

#### 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, entered by the user on the control console.

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 STATUS information, displayed and possible values of the displayed parameters:

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 (MODE) (DEV ADDR)**

This command selects and activates the slave CPU and sets its mode of operation. MODE designates the mode (also called emulation mode) in which the slave CPU will operate. The default value of MODE is 0. DEV ADDR gives the address of the slave CPU board. The default value of DEV ADDR is determined by diskette being used.

The possible values for MODE are:

- 0 – System Mode. Uses slave memory and system I/O.
- 1 – Partial emulation mode. Uses slave memory, user prototype I/O and user clock.
- 2 – Full emulation mode. Uses user prototype memory, I/O, and clock.

Note that in mode 2 the TRACE JUMP option is not available, (see TRACE command description).

The possible values for DEV ADDR are:

- 0 – 2650
- 1 – 8080
- 2 – 6800

\*SLV\* Error Responses:

- 31 – Parameter required
- 32 – Too many parameters
- 54 – Invalid mode
- 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 write or read 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***    ***RESet***  
***SEt***     ***DStat***

**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:

BREAKPOINT 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.

**SET Rm A1 (. . .An)**

This command causes the specified slave CPU registers to be preloaded with the hexadecimal constants A1 thru An. 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.

The format of this command will vary according to the requirements of the microprocessor.

\*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 will vary somewhat depending on the slave microprocessor.

*TRace OFF*  
*TRace All (Step)*  
*TRace Jmp (Step)*

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 display 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 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

# Chapter 6 the Editor

## 6.1 INTRODUCTION.

This chapter describes the Editor program. The Editor is discussed by examining the UDOS command EDIT, presenting a sample edit, detailing all the Editor commands, and listing all the messages which the Editor may display to the operator.

## 6.2 EDITOR OVERVIEW.

The major function of the UNIVERSAL ONE Editor is to create new source programs or to change existing source programs. The 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 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, paragraph 6.3 the detailed description of the commands, paragraph 6.5, and the Editor messages, paragraph 6.6, will be most helpful.

## ***EDIT***

If you are not familiar with editors, paragraph 6.4, which describes a typical edit session, will be helpful in illustrating the use of the Editor commands.

### **6.3 UDOS COMMAND EDIT.**

You may start the Editor by utilizing the UDOS command EDIT. 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. The Editor accomplishes this 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 the Editor would create the temporary \*ATA1 as 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 UDOS. The EDIT request will be rejected if an attempt is made to do so.

## NOTE

While the Editor is executing, the special UDOS keys, ESCAPE and space bar, retain their special functions. Consult paragraphs 4.5.2 and 4.5.3 for an explanation of their use.

**6.4 EDIT EXAMPLE.**

Let us go through an example of editing. Suppose you have conceived and coded a program for a 2650 slave, as shown in Figure 6-1,a, and wish to create a new file DADDSB, which will contain the source program data. Start the Editor program by typing:

```
> EDIT DADDSB/0 (r) (form 2, see paragraph 6.3)
```

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

```
◆◆ EDIT VER U.1 ◆◆
◆◆ NEW FILE ◆◆
◆
```

The \* is the Text Editor prompt character, which indicates that the Editor is ready to accept commands. Figures 6-2 through 6-7 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 R2,R3
◆
DADD STRP,R1 DAR1
      ADDP R3 DAR1
      PPSL WC
      ADDZ R2
      STRZ R2
      CPSL WC
      RETC,UN
DAR1 RES 1
```

Figure 6-1. A Sample Source Program

The first command entered, line 1 of Figure 6-2, 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.



**6.4.1 Input and Edit Modes.** 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 6-2). The Editor acknowledges this command by displaying INPUT: to remind you of its mode (line 3 of Figure 6-2). You may then enter the source program (lines 4-14 of Figure 6-2). As can be seen, errors have occurred (lines 9 and 13 of Figure 6-2). 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 6-2).

**6.4.2 Setting Tabs.** The effect of entering the TAB character can be seen by examining, for example, lines 9 and 23 in the display of the buffer (see Figure 6-2). 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 TAB S commands. The default TAB character is CTRL-I and the default TAB stops are 8, 16, 24, 32, 40, 48, 56, and 64.

**6.4.3 Displaying Buffer and Making Corrections.** 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 6-2). On line 17 of Figure 6-2, 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 6-2) 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 6-2, 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 6-3). In this case, the \$ is the delimiting character, which means that the \$s tell the Editor where the data starts and where the data ends. The Editor finds the first line in the buffer that contains WD, moves the line pointer to the beginning of the line, and displays the line (line 2 of Figure 6-3). To alter the WD to WC, enter S (the SUBSTITUTE command), a space, then \$WD\$WC\$ (line 3 of Figure 6-3). 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. (Any character that will not appear in the string itself can be used as the delimiter, in place of \$ .)

The Editor performs the substitution and displays the line as altered (line 4 of Figure 6-3). To change RETDMYNN to RETC,UN, find the line by entering F (FIND), space, then \$RET\$ to locate this string (line 5 of Figure 6-3). The Editor prints the line on which it locates RET (line 6 of Figure 6-3). 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 6-3). 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 6-3).

To insure that the changes were performed correctly, go to the top of the buffer and display its contents (see Figure 6-4).

```

1  ♦TAB .
2  ♦I
3  INPUT:
4  ♦ DOUBLE PRECISION ADD. A IN R0,R1. B IN R2,R3
5  ♦ON RETURN, A+B IS 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 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 6-2. Entering Text and Displaying the Buffer

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 6-4), 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 UDOS 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 and UDOS will display its prompt character (line 17 of Figure 6-4).

```

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 6-3. Use of FIND, SUBSTITUTE and REPLACE Commands

```

1  ♦B
2  ♦TY 55
3  ♦ DOUBLE PRECISION ADD  A IN R0,R1          B IN R2,R3
4  ♦ON RETURN, A+B IS 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
17  >

```

Figure 6-4. Displaying the Buffer and Filing

**6.4.4 Editing In New Data.** Suppose you wished to expand DADDSB/0 to include not only a double precision add, but a double precision subtract, as in Figure 6–5.

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

Start the Editor by entering:

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

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

When a new filename is the sole argument to an EDIT command, the file is treated as the PRIMARY OUTPUT file and there is no PRIMARY INPUT file. This is as it should be, since if you are in the process of creating a new file which will contain unique information, there is no need for a PRIMARY INPUT file. In this case, DADDSB/0 is an existing file which contains the double precision addition routine, so this EDIT command requests that DADDSB/0 be edited to itself, as explained in paragraph 6–3.

```

♦ DOUBLE PRECISION ADD  A IN R0,R1      B IN R2,R3
♦ ON RETURN, A+B IS R2,R3
♦
DADD  STRR,R1 DAR1
      ADDR  R3      DAR1
      PPSL  WC
      ADDZ  R2
      STRZ  R2
      CPSL  WC
      RETC,UN
DAR1  RES  1
♦♦ EOF ♦♦
♦TAB .
♦I
INPUT:
♦ DOUBLE PRECISION SUBTRACT. A IN R2,R3. B IN R0..,R1
♦ ON RETURN, A-B IS IN R2,R3
♦
DSUB.STRR,R0.DSRD
.STRR,R1.DSR1
.SUBR,R3.DSR1
.PPSL.WC
.SUBR,R2.DSRD
.CPSL.WC
.RETC,UN
DSRD.RES.1
DSR1.RES.1
.END.DADD

```

Figure 6–5. Sample Double Precision Add and Subtract Programs

When the Editor displays its prompt character, `*`, you can proceed. Since the new text is to be appended to the existing text in DADDSB, you must read the existing file into the buffer. This is accomplished by entering `G` (the GET command), space, and then `20`, an arbitrarily large number that will result in DADDSB/0, which we know to be approximately 10 lines long, being read into the buffer. (See line 1 of Figure 6-6). 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 6-6).

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 6-6). Display the buffer by entering `TY 55` (line 4 of Figure 6-6). This command displays the buffer and prints `** EOF **` to indicate the bottom of the buffer (lines 5 – 16 of Figure 6-6). (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 6-6). The TAB character is specified as a `.` in line 19. Enter the input mode by entering `I` (line 20 of Figure 6-6). Enter the source data (lines 22 – 35 of Figure 6-6). The effect of the TAB character can be seen in lines 52 to 61 of Figure 6-6, 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 6-6. To do these tasks, you must first position the line pointer to point to the line that begins with `*` DOUBLE PRECISION SUBTRACT. This can be accomplished by moving the line pointer down the buffer. Enter `D` (the Move Line Pointer Down the Buffer Command), space, `10` (line 1 of the Figure 6-7). 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 6-7. 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, (r)`, since the default value for the number of lines to move is 1 (line 3 of Figure 6-7).

To enter the `*` line, enter `I` (the `I` is an INSERT line command unless it is immediately followed by a RETURN in which case the user enters the INPUT mode), space, `*`, `(r)` (line 5 of Figure 6-7). To enter the second `*` line between the two temporary variables, `DSR0` and `DSR1`, and the subtract routine, go to the bottom of the buffer. This is accomplished by entering `END` (line 6 of Figure 6-7). The Editor indicates the line pointer's position at the bottom of the buffer by displaying `** EOF **` (line 7 of Figure 6-7). 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 (r)`, (line 8 of Figure 6-7). This command moves the line pointer up three lines and displays the line (line 9 of Figure 6-7). To enter the `*` line, enter `I` (INSERT), space, `*`, `(r)` (line 10 of Figure 6-7).

```

♦GET 20
♦♦ EOF ♦♦
♦B
♦TY 55
♦ DOUBLE PRECISION ADD A IN R0,R1      B IN R2,R3
♦ON RETURN, A+B IS R2,R3
♦
DADD  STRR,R1 DAR1
      ADDR  R3      DAR1
      PPSL  WC
      ADDZ  R2
      STRZ  R2
      CPSL  WC
      RETC,UN
DAR1  RES  1
♦♦ EOF ♦♦
♦END
♦♦ EOF ♦♦
♦TAB .
♦I
INPUT:
♦ DOUBLE PRECISION SUBTRACT. A IN R2,R3. B IN R0,R1
♦ ON RETURN, A-B INNS IN R2,R3
♦
DSUB STRR,R0 DSR0
     STRR,R1 DSR1
     SUBR,R3 DSR1
     PPSL,WC
     SUBR,R2 DSR0
     CPSI,WC
     RETC,UN
DSR0.RES.1
DSR1.RES.1
.END.DADD

♦B
♦T 25
♦ DOUBLE PRECISION ADD A IN R0,R1      F IN R2,R3
♦ON RETURN, A+B IS 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 DSR0
      STRR,R1 DSR1
      SUBR,R3 DSR1
      PPSL  WC
      SUBR,R2 DSR0
      CPSI  WC
      RETC,UN
DSR0  RES  1
DSR1  RES  1
      END  DADD
♦♦ EOF ♦♦
♦

```

Figure 6-6. Adding Data to an Existing File

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

## 6.5 EDITOR COMMAND DESCRIPTIONS.

This section provides detailed descriptions of all the 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 explained.

### 6.5.1 Editor Command Line.

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

COMMAND PARAMETERLIST
-----------------------

where:

- COMMAND identifies the particular action desired
- 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 paragraph 6.5.2.

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 $BADLINE$ : K 1 (r)
```

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

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

### 6.5.2 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.

```

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  DSR0 RES 1
10 ♦I ♦
11 ♦B
12 ♦T 55
13 ♦ DOUBLE PRECISION ADD A IN R0,R1 B IN R2,R3
14 ♦ON RETURN, A+B IS 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 DSR0
29 STRR,R1 DSR1
30 SUBR,R3 DSR1
31 PPSL WC
32 SUBR,R2 DSR0
33 CPSL WC
34 RETC,UN
35 ♦
36 DSR0 RES 1
37 DSR1 RES 1
38 END DADD
39 ♦♦ EOF ♦♦
40 ♦FILE
41 ♦♦ EOF ♦♦
42 ♦SLJ♦ EDJ

```

&gt;

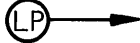
Figure 6-7. Inserting Lines Into the Buffer



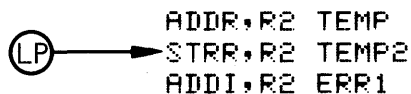
Conventions used in the command descriptions are:

- a) The symbol N, used with several commands, 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, except when it is used with the COPY command and, when an alternate file is specified with the GET and PUT commands. In these cases, N must be specified. In addition, N may be directly appended (without a space) 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 in paragraph 6.4.3.
- 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 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, used as a pair, execute a command line repetitively, see paragraph 6.5.9). To enter these characters into the buffer, they must be entered while in the INPUT mode or when a / prefix is used. (See paragraphs 6.4.1 and 6.5.3 for INPUT, paragraph 6.5.9 for / ).

### 6.5.3 Insertion Commands.

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,R0 DARR2  
LP → ADDR,R1 DARR3
```

and the command

```
♦INSERT STRR,R0 DARR5
```

was performed, the buffer would be altered to

```
      ADDR,R0 DARR2  
      STRR,R0 DARR5  
LP → ADDR,R1 DARR3
```

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,R0 DARR2  
LP → ADDR,R1 DARR3
```

and the sequence

## Kill

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

was performed, the buffer would be altered to

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

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

### 6.5.4 Deletion Commands.

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, only the current line is deleted.

For example, if the buffer appears as follows:

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

and the following command is performed

```
♦K4
```

the buffer will be changed to

```
(LP) → LINE 5
RETC,UN
```

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

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

## *Substitute*

- 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 `q+n` 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.

### 6.5.5 Alteration Commands.

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.

`STRING 2` may contain TAB characters (see paragraph 6.5.9). 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:

```
(LP) → AA.....AABBCC.....CC (63A's and 62 C's)
```

## Replace

and the command

```
S $BB$BBBBB$  
** TRUNCATED **
```

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

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

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+/  
WINES BY +RIDGE+
```

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:

```
(LP) → ADDR,R2 DAR1
```

the command

```
♦R STRR,R2 DAR1
```

will result in the current line being altered to:

ⓁP → STRR,R2 DAR1

The position of the line pointer is not altered. 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.

### 6.5.6 Search Commands.

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:

ⓁP → LINE 1  
LINE 2  
LINE 3  
LINE 4  
LINE 5

and the command:

♦F \$4\$

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

LINE 1  
LINE 2  
LINE 3  
ⓁP → LINE 4  
LINE 5

## Get

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 paragraph 6.5.8) the search starts at the current line plus one.

### 6.5.7 I/O Commands.

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 three 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) An existing file in the disk can be written over, under control of the Editor, and thus be destroyed.
- 3) System devices, such as CONO, CONI, LPT1, etc., can also be specified as the input or output files in all I/O commands.

#### 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:

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

LP →

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.

PUT N (FILENAME)
PUTK 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



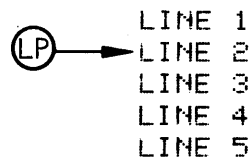
## List Copy

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. Thus, with the PUTK and PUT commands there are two possibilities:

- a) The line pointer points to a line which needs to be deleted from the buffer. In the case, PUTK is used and the line pointer is repositioned to the line immediately following the deleted text.

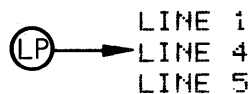
For example, if the buffer appears as follows:



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 PUT is used and 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.

*Begin*      *End*  
*Down*      *Up*  
*N*      *Again*

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

### 6.5.8 Buffer Line Pointer Commands.

The user may alter the position of the buffer 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 one line below the last line of 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 q+n 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.

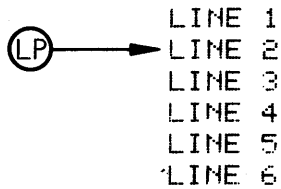
### 6.5.9 Utilities.

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, UDOS, TAB, TABS, TYPE, ?, / and the iterate command function, m < command > .

**AGAIN**

This command performs the previous command, as long as it is one of the repeatable commands. For example, if the buffer appears as follows,

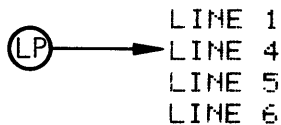
## FILE



and the command

♦K2

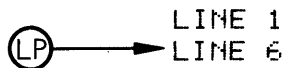
were performed, the buffer would be altered to,



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 and control will return to UDOS.

TYPE N

This command displays N lines of the buffer 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 and control returns to UDOS.

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.

When a character is defined as the tab character, it will not be displayed, but the following character will start at the next tab position.

An example of using C as the tab character is:

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

# TABS

*m*

TABS C1 C2 C3 . . .

This command sets the tab positions to the given columns C1 C2 C3. . . 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 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:

```
ⓁP → 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:

```

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

```

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

**UDOS**

This command suspends the Editor and returns control to UDOS. The Editor may be continued using the UDOS 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 line, the < , > , and: characters do not perform their usual functions. For example, the command

♦B:F \$LEFTANGLE,< \$

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

♦/B: F \$LEFTANGLE,<\$

would find the string 'LEFTANGLE', < . If FIND or SUBSTITUTE command is the first command in a command line, then the / is not required at the beginning of the line.

**BRIEF**

This command changes the state of a 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

## Macro

current line pointer after completing any of the commands END, UP, DOWN, FIND, SUBSTITUTE, and REPLACE. If the BRIEF switch is on, EDITOR will not type the current line. For example, if the buffer appears as follows:

```
ⓁP → 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  
♦D1
```

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.

### 6.5.10 MACROS

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

```
MACRO m=COMMANDLINE
```

This command is used to define a MACRO. The m is an integer which identifies the macro, and must be greater than 0 and less than 128. COMMANDLINE can be any normal command line but can not contain a macro execution or definition command; this will result in error when the macro is executed.

If a MACRO m already exists, and MACRO m=COMMAND LINE 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.

## 6.6 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 \*\***

Output diskette is full.

**\*\* NUMBER? \*\***

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.

**\*\* UDOS STAT= XX \*\***

XX is the UDOS 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 Chapter 9.

**\*\* NO PI \*\***

For this editing session there is no PRIMARY INPUT file; the user may not do "GET's" without specifying an Alternate Input file.

**\*\* NO PO \*\***

For this editing session there is no PRIMARY OUTPUT file; the user may not do "PUT's" without specifying an Alternate Output file.

**\*\* READ FILE? \*\***

An attempt was made to read from a non-existent file or an illegal input device.

**\*\* (INPUT) \*\***

The Editor response is in reference to an input attempt.

**\*\* (OUTPUT) \*\***

The Editor response is in reference to an output attempt.

**\*\* 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 INFILENAME OUTFILENAME" where INFILENAME and OUTFILENAME were not the same file and INFILENAME 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.

**\*\* ABORTED \*\***

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 paragraph 6.5.5.)

# *Chapter* **7** *the Assembler*

## **7.1 INTRODUCTION.**

This chapter describes operation of the Assembler. Topics covered include an overall description of the Assembler, procedure for using the Assembler, and an example of a typical assembly listing. Further information on a particular Assembler version for a specific slave CPU is contained in the manual supplement provided with that slave CPU card.

## **7.2 ASSEMBLER OVERVIEW.**

The Assembler is the system program used to translate source code into object code that is executable by the slave CPU. Each UNIVERSAL ONE system software package contains a particular version of the Assembler, suitable for that slave CPU which the software package supports. The Assemblers provided in different system software packages are those supplied by the microprocessor manufacturers. For example, for the 8080A slave CPU package it will be the MDS 800 Macro Assembler, for the 2650 package it will be the SIGNETICS TWIN 2650 Assembler, etc.

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. Different hex object code formats are described in the manual supplements for the different microprocessors.
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. For details on assembly language syntax, instruction codes, and other related material, consult the manual supplement for a specific microprocessor.
3. It will display errors on the console, if not overridden by a command parameter.

## **7.3 USING THE ASSEMBLER.**

The Assembler may be invoked when the UNIVERSAL ONE system is under the control of UDOS, by using the ASM command. The user must ensure that two conditions exist before the Assembler is used:

## **ASM**

1. The source program is present on a floppy disk file.
2. UDOS is ready to accept commands. UDOS presents its prompt character > when it is ready for commands.

To execute the Assembler, the user enters the following UDOS command:

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

where:

**SOURCEFILENAME** is the name of the disk file 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.

**OPTIONS** one or more optional parameters used by Assemblers of different slaves. For example, a particular slave may recognize **WIDE** as a designation that the output line is to be 120 print positions wide; or **NOERR** to indicate that errors should not be displayed on the console.

In response to the command, the Assembler will proceed to assemble the object code and will display the listing on the system control console or another specified output device. When the Assembler has completed its task, UDOS will display its prompt character > , to indicate it is ready for other 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.

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. For example,  
ASM (r) is not a valid command

### **UNACCEPTABLE INPUT DEVICE:**

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

### **INPUT FILE ASSIGN ERROR – SRB STAT=XX**

The SRB Status Codes are listed in Chapter 9.

## **7.4 LOADING AN ASSEMBLED PROGRAM.**

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

1. Ensure that the object file is present on a disk file and that UDOS is ready to accept commands.

2. Enter the UDOS command:

```
RHEX OBJECTFILE
```

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

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

Hex object code programs created on paper tape outside the UNIVERSAL ONE system (for example, by the 8080A cross-assembler) can be read into slave memory by the RHEX command or to a disk file by using the UDOS COPY command. 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.

### 7.5 SAMPLE ASSEMBLY LISTING.

If the double precision add/subtract subroutine, discussed in the Chapter 6 edit example, is to be assembled, the following sequence has to be performed:

1. Six EQU assembler directives have to be entered into the source file. These are needed to define the contents of various CPU registers (R0, R1, R3, UN, WC), related only to the 2650 slave CPU (for details see the 2650 slave CPU manual supplement).
2. The command

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

will write the object code produced on file DADOBJ/0 (see Figure 7-1), and produce the listing in Figure 7-2 on the line printer.

```

♦ DOUBLE PRECISION ADD  A IN R0,R1      B IN R2,R3
♦ ON RETURN, A+B IS R2,R3
♦
R0      EQU      0
R1      EQU      1
R2      EQU      2
R3      EQU      3
UN      EQU      3
WC      EQU      8
♦
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 DSR0
        STRR,R1 DSR1
        SUBR,R3 DSR1
        PPSL   WC
        SUBR,R2 DSR0
        CPSL   WC
        RETC,UN
♦
DSR0    RES      1
DSR1    RES      1
        END      DADD

```

Figure 7-1. Sample 2650 Slave Program Listing Ready for Assembly

TWIN ASSEMBLER VER 2.0

PAGE 0001

```

LINE ADDR  OBJECT  E SOURCE
0001          ♦ DOUBLE PRECISION ADD  A IN R0,R1      B IN R2,R3
0002          ♦ON RETURN, A+B IS 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  DAR1
0012 0002 8B07          ADDR,R3  DAR1
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          DAR1     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 C80B      DSUB     STRR,R0  DSR0
0024 000E C90A          STRR,R1  DSR1
0025 0010 AB08          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

♦SLJ♦ EDJ

*Note: refer to 2650 slave manual supplement  
for an explanation of the format in this listing.*

Figure 7-2. Sample 2650 Slave Assembly Listing

# *Chapter* **8** *PROM Programmer*

## **8.1 INTRODUCTION.**

This chapter describes the optional PROM programming facilities of the UNIVERSAL ONE system. Topics covered include a general description of the PROM programming software and hardware, instructions for programming a PROM, and a description of the associated commands.

## **8.2 PROM PROGRAMMING HARDWARE AND SOFTWARE.**

The optional PROM programming facility is used to manually program PROM chips in the UNIVERSAL ONE system or to output programming data to another device, such as a paper tape punch. The PROM programming facility consists of one or two programmer circuit cards and the PROM Programmer software. The current hardware and software can support both the 82S115 bipolar fusible link PROM and the 1702A MOS erasable PROM. Other hardware and software, in development at the time of publication, will support the 2708 and 2704 MOS erasable PROMs.

The programmer circuit board is installed in the development computer card cage and during programming controls the data flow (i.e., power application) to the PROM chip inserted in one of the development computer front panel sockets. A different programmer card is required for each different type PROM to be programmed.

The PROM Programmer software is included in UDOS overlay area 1, on the system diskette, and can be utilized for programming either the 82S115 or 1702A PROM. Three software commands are used to read, write, or compare data in a PROM. Other software will need to be written on the system diskette to support additional PROM types.

## **8.3 USING THE PROM PROGRAMMER.**

There are three sockets on the front panel of the development computer (see Figure 3-5). 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.)

To program a PROM, first load the fully debugged PROM object code from the disk file into slave memory and then proceed according to the following procedure:



## ***Rprom***

1. Always turn PROM power off whenever inserting a PROM in its socket (or removing it). Power to the socket is controlled by the PROM PWR switch on the front panel of the development computer. (The PPWR indicator above the switch is lighted when power is on.)
2. Insert PROM in its correct socket. Use of the wrong socket is likely to cause permanent damage to the PROM. Align pin 1 of the PROM with pin 1 of the socket. Pin 1 is adjacent to the lever.
3. When inserting or removing the PROM leave the socket lever up; push down on the lever to clamp the PROM in the socket.
4. Acquire the UDOS prompt character > on the control console and then enter the desired PROM Programmer command, or sequence of commands. The commands that can be used are RPPROM, WPPROM and CPPROM. RPPROM is used to read the contents of a PROM into slave memory. WPPROM is used to write binary code from slave memory to the PROM. CPPROM is used to compare the contents of slave memory with the contents of a PROM. See paragraph 8.4 below for a more complete description of each command.
5. If instead of programming, it is desired to output the code to peripheral equipment, use the WHEX (or WSMS) command (see paragraph 4.6.6).

### **8.4 PROM PROGRAMMER COMMANDS.**

The PROM Programmer utilizes three commands: RPPROM, WPPROM, and CPPROM. These commands are stored in UDOS overlay area 1 and can be used whenever the UDOS prompt character > appears on the control console.

RPPROM (A1) (N) (A2) (A3) (C)
-------------------------------

This command is used to read the contents from 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 type to be read. If N is equal to 1 the 1702A PROM is specified. If N is equal to 2 the 82S115 PROM 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 power 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 (socket) 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 of 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 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 9

## Supervisor Call Interface

### 9.1 INTRODUCTION.

This chapter describes Supervisor Call (SVC) software of the UNIVERSAL ONE system. A general description of the SVC concept and applications is at the beginning of the chapter, followed by a description of various component parts of the SVC software.

### 9.2 GENERAL DESCRIPTION OF SUPERVISOR CALLS.

In the structure of UNIVERSAL ONE system software no direct communication is allowed between the slave CPU and system peripherals. However, Supervisor Calls are a means by which a user program, running under the slave CPU, can gain access to system peripherals by generating interrupts to the master CPU. A program running under the slave CPU can issue one of six SVCs – SVC1, SVC2, etc. – to acquire either an I/O connection with system peripherals or obtain a UDOS program service.

The SVC is actually an extended slave CPU I/O instruction that is decoded on the debug logic card to generate a master CPU interrupt. Corresponding to each SVC there is a Service Request Block (SRB), which is an 8-byte block that identifies certain specifics and stores status related to the I/O or service requested by the SVC.

The SVC is initiated by a slave CPU I/O instruction to slave CPU device ports F2 thru F7, followed by a slave CPU NOP instruction.

<u>SVC No.</u>	<u>Slave CPU I/O Port</u>	<u>SVC Pointer</u>
SVC1	F7	40
SVC2	F6	42
SVC3	F5	44
SVC4	F4	46
SVC5	F3	48
SVC6	F2	4A

When the program running under the slave CPU issues an SVC, the debug logic card decodes the SVC, halts the slave and interrupts the master CPU. The SVC is vectored to a unique location in master memory where the UDOS program SVC Processor module (refer to Chapter 4), which services all SVC interrupts, is located. Under control of the SVC Processor, the master CPU looks at the SRB pointer in slave

memory, uses the pointer to access the SRB and then executes the SRB. During this time the slave CPU remains in the halted state.

All SVC I/O operations are performed by UDOS running under the master CPU. Data to be input from floppy disk files is first read into UDOS buffers and then deblocked to the user's buffer, as required. Data to be output to floppy disk files are accumulated in UDOS buffers and then output to the file as full sectors, under control of the UDOS File Manager. Input/output operations on devices other than floppy disk files are performed directly through the user's buffer.

### 9.3 SERVICE REQUEST BLOCK (SRB).

The user must place an SRB in the slave memory, corresponding to each of the six SVCs. All of the information needed to perform the function requested by the SVC caller is contained in the SRB.

An SRB consists of eight contiguous bytes, located in the first 16K page of common (slave) memory. The contents of an SRB are as follows:

<u>Byte</u>	<u>Symbol</u>	<u>Content</u>
1	SFC	SVC function code
2	SCH	Channel number
3	STAT	Status
4	SDAT	Single byte data
5	BCNT	I/O byte count
6	BMAX	I/O buffer length
7-8	BPTR	I/O buffer pointer

A description of each entry in the SRB is given below:

#### 9.3.1 SRB Bytes.

**SVC Function Code (SFC).** The SVC function code specifies the I/O or service function which is to be performed as a result of the SVC call. A list of 25 different functions supported by UDOS is given in Appendix D, and each function is described in more detail in paragraph 9.3.2.

**Channel Number (SCH).** A logical channel number must be specified for all SVC I/O function codes. The channel number must be in the range 0-7. When a logical channel is assigned to a physical device or file, the channel stays connected to the device or file until a "CLOSE" function is issued on the channel. The same channel number can be used in more than one SRB.

**Status (STAT).** For all I/O operations, an indication of the result of the I/O request is returned by the master CPU and written in the location of the SRB byte STAT. If a READ and PROCEED, or WRITE and PROCEED function is requested, STAT will first be set by master CPU to indicate that the I/O operation is in progress. When the I/O operation is complete, STAT will be set to indicate the result of the operation. A list of status codes is given in Appendix E.

**Single Byte Data (SDAT).** This location is used by the master CPU to return single byte data requested by an SVC function other than an I/O function. For I/O, SDAT is the physical status of the device being accessed.

**I/O Byte Count (BCNT).** This location is used by the master CPU to return the actual number of bytes to be input or output by a READ or WRITE operation. For line oriented ASCII read or write operations, this count is the actual number of characters including the end-of-line character EOL. For binary read or write operations the count is the actual number of bytes. BCNT is also used in conjunction with SDAT to return double byte data requested by a non-I/O SVC function (e.g., GET TIME).

**I/O Buffer Length (BMAX).** The maximum number of bytes of data to be input to or output from the user's buffer must be written in this byte by the user prior to initiating the SVC. Once set by the user, it is not disturbed by UDOS.

**I/O Buffer Pointer (BPTR).** For all SVC I/O functions and for some non-I/O SVC functions, the user must provide the starting address of a buffer in the first 16K page of common memory. Unless otherwise specified in the SVC function description, data transfers to or from the user program are performed through the buffer pointed to by BPTR.

### 9.3.2 SVC Function Descriptions.

**Assign Channel.** An application program running under the slave CPU has eight logical channels, or ports, through which it can perform I/O. Any logical channel can be assigned to any physical device attached to the system. A floppy disk file is treated in the same manner as an independent physical device.

The physical device or floppy disk file to which a logical channel is to be assigned is given as a string of ASCII characters terminated by an EOL character.

To assign a channel, the SRB byte SCH must contain a channel number in the range 0–7 and BPTR must contain the starting address of the device name string. If a channel is assigned to a floppy disk file which does not exist, the file will be automatically created and the STAT byte in the SRB will be set to a 1 to indicate that it is a new file.

**Read or Write ASCII.** An ASCII line is defined as a string of ASCII characters terminated by an end-of-line character EOL (a normal ASCII carriage return). An application program running under the slave CPU can input or output a line through a channel which has been previously assigned to a floppy disk file or other I/O device. The required settings for the SRB are minimal.

To read a line from a peripheral source into an application program buffer, BPTR must contain the starting address of a buffer into which the line is to be input. BMAX must contain the maximum number of characters the user wants placed in the buffer in the event an EOL character is not in the data stream. The user's buffer must be able to contain BMAX+1 bytes of data because, if an EOL character is not found in the

data stream and BMAX bytes have been placed in the buffer, an EOL character will be placed in the buffer at the BMAX+1 position. This procedure assures the user that an ASCII input line will always be terminated by an EOL.

To write a line to a peripheral device or file, BPTR must contain the starting address of the line to be output. BMAX must contain the maximum number of ASCII characters to be output in the event the line is not terminated by an EOL character.

After the I/O function is completed, BCNT will contain the actual number of ASCII characters, including the EOL, which were input or output to or from the user's buffer. The actual number of characters input or output may be less than the maximum specified by BMAX as a result of encountering the normal EOL character, and end-of-file on a read before finding a EOL, or an end-of-device on a write. All but the first return a non-zero status.

The maximum length of a line supported by UDOS is 255 characters, plus the EOL character, for a total of 256. Thus, BMAX must be greater than or equal to 1 and less than or equal to 255.

**Read or Write Binary.** An application program running under the slave CPU can input or output a block of binary bytes through a channel which has been previously assigned to a floppy disk file or other I/O device.

To read a block of binary bytes from a peripheral source, BPTR must contain the starting address of a buffer into which the data is to be input. BMAX must contain the number of bytes to be input to the buffer.

To write a block of binary bytes to a peripheral device or file, BPTR must contain the starting address of the buffer from which the data is to be output. BMAX must contain the number of bytes to be output from the buffer.

After the I/O function is completed, BCNT will contain the actual number of bytes which were input or output to or from the user's buffer.

Binary read and write operations are performed strictly under count control. A user may input or output up to 256 bytes of data. In the case of read or write, a BMAX value of 0 is taken to mean 256.

**Close File.** The CLOSE FILE function disconnects the given channel from the floppy disk file or other I/O device to which it was assigned.

If the channel was assigned to a floppy disk write file, the UDOS buffer used for the file will be output and a logical end-of-file will be recorded on the file before it is disconnected. Subsequently, when the file is read, the end-of-file condition will be sensed and indicated in the STAT byte of the SRB.

For physical devices other than the floppy disk, an appropriate clearing action will be taken.

**Rewind File.** The REWIND function applies only to floppy disk files. It has the effect of positioning a file at its beginning. If a device other than a floppy disk file has been assigned to the channel, the REWIND function will be treated as a NOP.

To rewind a file, SRB byte SCH must contain the channel number to which the floppy disk file has been assigned. When a floppy disk file is "rewound" it is treated the same as if it had just been assigned. If the first I/O operation for the rewound file is a read, data will be input from the file in the normal manner. If the first I/O operation for the rewound file is a write, the sectors previously allocated for the file will have no significance and the file will be treated as if it were a new file.

**Delete File.** The DELETE function causes the floppy disk file assigned to the given channel to be deleted from the directory of the diskette and, as a consequence, also causes the channel to be disconnected from the file.

If a device instead of a floppy disk file has been assigned to the channel, the DELETE function will be treated the same as CLOSE function.

**Rename File.** To rename the floppy disk file which has been assigned to the given channel, BPTR must contain the starting address of the name (given as an ASCII line) to which the file is to be renamed.

A file which is to be renamed must not be in the process of being read or written, i.e., the final must have just been assigned or rewound. If a device other than a floppy disk file has been assigned to the channel for which the RENAME function is entered, the function will be treated as a NOP.

**Get Parameter (From Procedure Parameter Buffer).** If an application program running under the slave CPU has been invoked as a procedure from the system console, often there are parameters in the procedure line. This SVC is used to get a particular parameter for the application program.

Parameters are stored in the UDOS procedure parameter buffer in master memory. The parameters are identified by number according to the order in which they appear in the command line and exist as strings of ASCII characters terminated by an EOL character. The desired parameter is requested as a number in SRB byte SDAT. The parameter is returned to the user as an ASCII line, starting at the location contained in BPTR.

When a procedure command line is entered, parameters are delimited by a space, comma, or EOL character. A comma or space delimiter will be replaced with an EOL character before the parameter is stored in the UDOS parameter buffer. A parameter may be omitted from the ordered sequence by two consecutive commas. If a parameter has been omitted, the first character in the user's buffer will be an EOL character.

If the given parameter number is greater than the number of parameters included in the command line, the first byte of the user's buffer will be a -1, followed by an EOL, and the SRB status byte will indicate status code 06.

**Get Parameter (From Slave Parameter Buffer).** If an application program running under the slave CPU has been invoked by a LOAD or EXECUTE command, often



there are parameters in the command line. This SVC gets a particular parameter for the application program.

If parameters are present, they will be stored in the UDOS slave parameter buffer in master memory. The parameters are identified by number according to the order in which they appear in the command line and exist as strings of ASCII characters terminated by an EOL character.

The desired parameter is requested as a number in SRB byte SDAT. The parameter is returned to the user as an ASCII line, starting at the location contained in BPTR.

When the above UDOS command is entered, parameters are delimited by a space, comma, or EOL character. A comma or space delimiter will be replaced with an EOL character before the parameter is stored in the UDOS parameter buffer. A parameter may be omitted from the ordered sequence by two consecutive commas. If a parameter has been omitted, the first character in the user's buffer will be an EOL character.

If the given parameter number is greater than the number of parameters included in the command line, the first byte of the user's buffer will be a -1, followed by an EOL, and the SRB status byte will indicate status code 06.

**Load Overlay.** Overlays for the user's programs can be stored on disk as load modules, complete with memory beginning, end, and start addresses. The resident part of the application program loads an overlay by presetting the SRB bytes and then issuing the SVC. The file name of the overlay is given as an ASCII string terminated by an EOL. The BPTR byte in the SRB must point to this string. The header information in the load module determines where the overlay is to be loaded in memory. The result of the load operation is returned in STAT. The overlay is not started, and control remains with the requesting program.

**Execute Overlay.** This SVC function is called and performed in the same way as the LOAD OVERLAY function with the exception that the overlay is executed after it is loaded at the starting address given in the header of the load module. This SVC provides the capability of chaining separate programs, as distinct from overlays.

**Suspended Execution.** This SVC function will cause the requesting program to be suspended at the place the SVC is issued. The action is similar to an I/O and wait. The program can be restarted again by an operator command issued from the system console.

**Exit.** This SVC function will cause the program running under the slave CPU to be terminated. The channels previously assigned by or for the program will not be closed.

**Abort.** This SVC function will cause the program running under the slave CPU to be terminated. The channels previously assigned by or for the program, if not already closed, will be closed.

**Get Time.** The accumulative milliseconds since system start-up time (module 2<sup>16</sup>) is returned in SDAT and BCNT bytes. Milliseconds will not accumulate if the CLOCK has been disabled by the UDOS command CLOCK OFF.

**Get Overlay Address.** The memory bounds of the last overlay loaded into common memory and the execution address of the overlay are stored in six consecutive bytes starting at the address given in BPTR. The first two bytes will contain the low load address, the next two bytes will contain the high load address, and the last two bytes will contain the execution address.

**Get Device Status.** The status of the device assigned to the given logical channel (SCH), as obtained from the physical device, is returned in SDAT. A default of zero will be returned if there is no physical status available.

**Get Device Type.** The identification number of the device assigned to the channel number in (SCH) is returned in SDAT and the device type is returned in BCNT. The devices are identified as follows:

<u>Device Name</u>	<u>Device I.D.</u>	<u>Device Type</u>
CONI (Console Input)	1	1
CONO (Console Output)	2	2
LPT1 (Line Printer)	3	2
HSPT (H.S. P/T READER)	4	1
TTYR (TTY P/T READER)	6	1
DISK FILE	-1	43

<u>DEVICE TYPES:</u>		
1	ASCII read	
41	Binary read	
2	ASCII write	
42	Binary write	
3	ASCII read/write	
43	Binary read/write	

The device types specified for the UDOS !/O device represent the way in which the UDOS commands treat the devices in normal usage. A user application program can read from any input device in either ASCII or binary and can write to any output device in either ASCII or binary.

The CONO, CON1, and the floppy disk are sharable devices which can be assigned to more than one channel. The LPT1, HSPT, and TTYR are non-sharable devices and can only be assigned to one channel at any given time.

A user application program can have a maximum of seven channels assigned to floppy disk files.

**Get Last Console Input Char.** The last character input from the control console is returned in SDAT. If sensed in a loop, while performing extensive calculations or I/O, it provides the user program with a way to respond to a request for attention or other action by the operator.

# Appendix **A** *UDOS Command Summary*

The short form required to invoke the command is underlined.

	PAGE
<u>ABORT</u> NAME .....	4-13
or	
<u>ABORT</u> *	
or	
<u>ABORT</u> /	
<u>ASSIGN</u> CH DEVICE (... CH DEVICE) .....	4-14
<u>ASM</u> SOURCEFILE (LISTFILE) (OBJECTFILE) ( <u>W</u> IDE) ( <u>N</u> OERR) ....	7-2
<u>BKPT</u> ADDRESS ( <u>W</u> RITE) ( <u>R</u> EAD) .....	5-13
<u>CLBP</u> (ADDRESS) .....	5-14
<u>CLOCK</u> <u>ON</u> .....	4-16
or	
<u>CLOCK</u> <u>OFF</u>	
<u>CLOSE</u> CH (... CH) .....	4-13
<u>CONT</u> NAME .....	4-12
or	
<u>CONT</u> *	
or	
<u>CONT</u> /	
<u>COPY</u> INPUT (...INPUT) OUTPUT .....	4-23
<u>CPROM</u> (A1) (N) (A2) (A3) (C) .....	8-3
<u>CSMS</u> (ADDRESS) (DEVICE) .....	4-28
<u>DEBUG</u> (DEVICE) (ADDRESS) .....	5-9
<u>DELETE</u> FILENAME/D (,,,FILENAME/D) .....	4-22
<u>DEVICE</u> DEVICE U .....	4-16
or	
<u>DEVICE</u> DEVICE D	
<u>DSTAT</u> .....	5-14

	PAGE
<u>D</u> UMP A1 (A2) (DEVICE) .....	5-10
<u>D</u> UP D1 D2 (IDENT) .....	4-21
<u>E</u> DI (INFILENAME) (OUTFILENAME) .....	6-2
<u>E</u> XAM ADDRESS .....	5-11
<u>F</u> ORMAT D (IDENT) .....	4-18
<u>G</u> O (ADDRESS) .....	5-9
<u>K</u> ILL ON .....	4-31
or	
<u>K</u> ILL OFF	
<u>L</u> DIR (D) (.) (/) (DEVICE) .....	4-22
<u>L</u> OAD FILENAME .....	5-9
<u>M</u> ODULE FILENAME A1, A2, A3 (IDENT) .....	4-25
<u>P</u> ATCH ADDRESS HEX-STRING .....	5-12
<u>P</u> RINT FILENAME (DEVICE) (L1 L2) .....	4-24
or	
<u>P</u> RINTL FILENAME (DEVICE) (L1 L2)	
<u>R</u> ENAME OLDFILE/D NEWFILE/D .....	4-20
or	
<u>R</u> ENAME D IDENT	
<u>R</u> ESET .....	5-14
<u>R</u> HEX (/BIAS) (DEVICE) (Format depends on type of slave CPU) .....	4-26
<u>R</u> PROM (A1) (N) (A2) (A3) (C) .....	8-2
<u>S</u> EARCH ON (N) .....	4-15
or	
<u>S</u> EARCH OFF	
<u>S</u> ET Rm (A1 (...Ai) (Format depends on type of slave CPU) .....	5-14
<u>S</u> LAVE (MODE) (DEV ADDR) .....	5-13
<u>S</u> TATUS .....	5-12
<u>S</u> SPEND (NAME) .....	4-12
or	
<u>S</u> SPEND *	
or	
<u>S</u> SPEND /	
<u>S</u> YSTEM D .....	4-16
<u>T</u> RACE OFF .....	5-15
or	
<u>T</u> RACE ALL (STEP) (A1 A2)	
or	
<u>T</u> RACE JMP (STEP) (A1 A2)	

	PAGE
<u>T</u> YPE ON .....	4-31
or	
<u>T</u> YPE OFF	
<u>V</u> ERIFY D .....	4-19
<u>W</u> HEX A1 A2 ... („A1 A2) (A3) (DEVICE) .....	4-27
<u>W</u> PROM (A1) (N) (A2) (A3) (C) .....	8-3
<u>W</u> SMS (ADDRESS) (DEVICE) .....	4-28
<u>X</u> EQ FILENAME .....	5-10
* COMMENT .....	4-32

# Appendix **B** *Debugger Command Summary*

The minimum terms required to invoke the commands are underlined.

The following commands are used both by the Debugger and by UDOS (DEBUG is used only during UDOS to invoke the Debugger).

	PAGE
<u>A</u> BORT .....	4-13
<u>A</u> SSIGN .....	4-14
<u>C</u> LOSE .....	4-13
<u>D</u> EBUG .....	5-9
<u>D</u> ELETE .....	4-22
<u>D</u> UMP .....	5-10
<u>E</u> XAM .....	5-11
<u>G</u> O .....	5-9
<u>L</u> OAD .....	5-9
<u>P</u> ATCH .....	5-12
<u>S</u> LAVE .....	5-13
<u>S</u> TATUS .....	5-12
<u>S</u> YSTEM .....	4-16
<u>X</u> EQ .....	5-10

The following commands are used only by the Debugger (only after the DEBUG command).

	PAGE
<u>B</u> KPT .....	5-13
<u>C</u> LBP .....	5-14
<u>D</u> STAT .....	5-14
<u>R</u> ESET .....	5-14
<u>S</u> ET .....	5-14
<u>T</u> RACE .....	5-15

# Appendix **C** Editor Command Summary

The short form required to invoke a command is underlined.

	PAGE
<u>A</u> GIN .....	6-21
<u>B</u> EGIN .....	6-21
<u>B</u> RIEF .....	6-25
<u>C</u> OPY IN INFILE (OUTFILE) .....	6-20
<u>D</u> OWN n .....	6-21
<u>E</u> ND .....	6-21
<u>F</u> ILE .....	6-22
<u>F</u> IND \$STRING\$ .....	6-17
<u>G</u> ET N (FILENAME) .....	6-18
<u>I</u> NPUT .....	6-13
<u>I</u> NSERT STRING .....	6-13
<u>K</u> ILL N .....	6-14
<u>L</u> IST N .....	6-20
<u>M</u> ACRO m=COMMANDLINE .....	6-26
<u>M</u> ACRO m .....	6-27
<u>N</u> .....	6-21
<u>P</u> UT N (FILENAME) .....	6-19
<u>P</u> UTK N (FILENAME) .....	6-19
<u>Q</u> UIT .....	6-23
<u>R</u> EPLACE STRING .....	6-16
<u>U</u> DOS .....	6-25
<u>S</u> UBSTITUTE \$STRING <sup>1</sup> \$STRING <sup>2</sup> \$ .....	6-15

	PAGE
<u>TAB</u> CHAR .....	6-23
<u>TABS</u> C1 C2 C3 ... .....	6-24
<u>I</u> TYPE N .....	6-23
<u>U</u> P n .....	6-21
m<commands> .....	6-24
? .....	6-25
L .....	6-25



# *Appendix* **D** *SVC Function Codes*

<u>HEX CODE</u>	<u>FUNCTION</u>
10	ASSIGN channel to device or file
01	READ ASCII and WAIT
81	READ ASCII and PROCEED
02	WRITE ASCII and WAIT
82	WRITE ASCII and PROCEED
C1	READ BINARY and PROCEED
42	WRITE BINARY and WAIT
C2	WRITE BINARY and PROCEED
03	CLOSE device or file on channel
04	REWIND file on channel
05	DELETE file on channel
06	RENAME file on channel
13	GET PARAMETER (From Procedure Parameter Buffer)
1C	GET PARAMETER (From Slave Parameter Buffer)
17	LOAD OVERLAY
18	EXECUTE OVERLAY
19	SUSPEND EXECUTION
1A	EXIT
1F	ABORT
11	GET TIME (milliseconds)
12	GET OVERLAY ADDRESSES
15	GET DEVICE STATUS
14	GET DEVICE TYPE
16	GET LAST CONSOLE INPUT CHAR.

# **Appendix** ***E*** **SRB Status Codes**

- 00 — FUNCTION COMPLETE / NO ERROR
- 01 — CHANNEL ASSIGNED TO NEW FILE
- 02 — ILLEGAL CHANNEL NUMBER
- 03 — CHANNEL NOT ASSIGNED
- 04 — CHANNEL BUSY
- 05 — ILLEGAL FUNCTION CODE
- 06 — NO EOL ON ASCII READ
- 07 — NO EOL ON ASCII WRITE
- 08 — ILLEGAL DRIVE NUMBER
- 09 — FILE IN USE
- 0A — DEVICE NOT OPERATIONAL
- 0B — DEVICE NOT AVAILABLE
- 0C — DEVICE NOT READY
- 0D — DEVICE IN USE
- 0E — DIRECTORY READ ERROR
- 0F — DIRECTORY WRITE ERROR
- 10 — DIRECTORY FULL
- 11 — DEVICE READ ERROR
- 12 — DEVICE WRITE ERROR
- 13 — CODE NOT ASSIGNED
- 14 — CODE NOT ASSIGNED
- 15 — FILE NAME IN USE
- 16 — ILLEGAL FILE NAME
- 17 — FILE IN R/W PROGRESS
- 18 — CHANNEL ALREADY ASSIGNED
- 19 — INCORRECT DISKETTE
- 7F — I/O IN PROGRESS
- FF — END OF FILE OR END OF DEVICE

# Appendix **F**

## **SMS Tape Format**

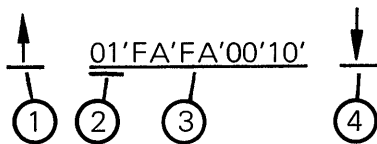
An SMS tape consists of a block of data, preceded by a TAPE ON character (CTRL-R or hex '12') and followed by a TAPE OFF character (CTRL-T or '14'). When the TAPE ON character is read, the address counter is set to zero. This means that the next data byte will be stored at location 0. When the TAPE OFF 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



- 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** **G** **System**  
**Readiness Test**

(to be supplied)

# *Appendix* **H** *System Utility Command Files*

Millenium 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 UDOS binary load file
2. All UDOS overlays, including the Assembler and the 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.