PRICE $19.95

WD/90
Pascal MICROENGINE™
Reference Manual

THE MICROENGINE COMPANY

Pascal MICROENGINE™ Computer

User's Manual

Preliminary Edition

March, 1979

Software is provided on a licensed basis only and is the property of the University of California with permission granted for use on an individual system basis only. Copies may be made for archival purposes only.

Information furnished by The MICROENGINE Company is believed to be accurate and reliable. However, no responsibility is assumed by The MICROENGINE Company for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of The MICROENGINE Company. The MICROENGINE Company reserves the right to change product specifications at any time without notice.

Copyright (c) 1979 by The MICROENGINE Company

First Printed March 1979

All rights reserved.

No part of this document may be reproduced by any means, nor translated, nor transmitted into a machine language without the written permission of The MICROENGINE Company.

MICROENGINE is a Registered Trademark of Western Digital Corporation.

The MICROENGINE Company is a wholly-owned subsidiary of Western Digital Corporation.

PREFACE

This manual describes the use and operation of the MICROENGINE Company's microcomputer products for developing and executing programs in the Pascal language. These products are:

- the WD/900 Pascal MICROENGINEtm Single-Board Computer, an 8 by 16-inch board containing a WD/9000 16-bit microprocessor that directly executes UCSD's Pascal P-code, 64K bytes of memory, 2 RS-232 asynchronous serial ports, 2 8-bit parallel ports, and a Floppy Disk Controller; and

- the WD/90 Pascal MICROENGINE Computer, a desktop microcomputer comprised of a Pascal MICROENGINE Single-Board Computer and power supplies enclosed in a stylized housing.

The manual is intended to serve both as an introductory guide and as a reference source. Chapter 1 presents an overview of the computer. It briefly describes hardware components of the computer; the software provided with the computer; the mechanical, environmental, and electrical requirements of the computer; and the computer operation. Chapters 2, 3, and 4 provide "how-to" information: how to install a new computer, how to operate the computer, and how to perform troubleshooting procedures. Chapter 5 presents detailed information on the computer's operation.

The software provided with the computer is the Pascal Operating System developed at the University of California at San Diego (UCSD). This operating system is described in the Pascal III.0 Operating System Reference Manual.

*****

This manual was prepared and edited using the UCSD Pascal Screen Oriented Editor and was printed using a PRINTRONIX Model P300 lineprinter. The lineprinter was provided by PRINTRONIX Corporation, Irvine, California.

CONTENTS

Page

1. OVERVIEW 1
   1.1 Components of the Computer 1
   1.2 Pascal Operating System 2
   1.3 Requirements of the Computer 3
   1.4 Computer Operation 3

2. INSTALLATION PROCEDURES 5
   2.1 Setting Option Switches 5
      2.1.1 Opening the Computer Housing 5
      2.1.2 Setting the Floppy Disk Controller Option Switches 7
      2.1.3 Setting the System Terminal Specifications Switches 8
      2.1.4 Setting the Serial Port Baud Rate Switches 10
      2.1.5 Reclosing the Computer Housing 10
   2.2 Installing System Components 11
      2.2.1 Installing the Computer 11
      2.2.2 Connecting the Peripherals 11
      2.2.3 Plugging in the Power Cord 12

3. OPERATING PROCEDURES 13
   3.1 Starting and Stopping the System 13
      3.1.1 Turning On the Power 13
      3.1.2 Loading the System Software 14
      3.1.3 Stopping the System 14
   3.2 Changing Hardware Option Switch Settings 15

4. TROUBLESHOOTING PROCEDURES 17
   4.1 The Troubleshooting Process: A General Discussion 17
   4.2 Symptom 1: Operating System Fails to Announce Itself 18
      4.2.1 Repeating the Software Loading Procedure 18
      4.2.2 Checking the Floppy Disk Controller Option Switches 18
      4.2.3 Measuring and Adjusting the Power Supplies 19
   4.3 Symptom 2: Transfers to the System Terminal Fail 20
   4.4 Symptom 3: Transfers to the Parallel Ports Device or Floppy Disk Fail 21
   4.5 Symptom 4: System Fails to Pass Diagnostic Program 21
   4.6 The Troubleshooting Process: a Summary Flow Chart

5. THEORY OF OPERATIONS 25
   5.1 Processor/Memory and Processor/Peripherals Communication 25
      5.1.1 Processor/Memory Communication 28
      5.1.2 Processor-Initiated Communication with Peripheral Devices 28
      5.1.3 Device-Initiated Communication with the Processor Interrupts 29
   5.2 Pascal MICROENGINE Processor 30
   5.3 Memory 31

   5.4 Serial Ports 31

5.4.1 Default Operation of the Serial Ports..... 33
5.4.2 Device Programming Mechanism: the Control Registers..... 37
5.5 Parallel Ports..... 43
5.5.1 Default Operation of the Parallel Ports..... 43
5.5.2 Device Programming Mechanism: the Control Register..... 46
5.6 Floppy Disk Controller and DMA Controller..... 49
5.6.1 Start-Up Procedures Involving the DMAC and FDC..... 51
5.6.2 Processor-Initiated Communication with the DMAC and FDC. 51
5.6.3 DMA Controller Organization and Operation..... 53
5.6.4 Floppy Disk Controller Organization and Operation..... 58

ILLUSTRATIONS

Figure Number	Title	
1-1	Pascal MICROENGINE Computer	1
2-1	Removing the Rear Panel	5
2-2	Removing the PC Board	5
2-3	Location of the Option Switch Packages	6
2-4	Air Vents	11
2-5	Rear Panel	11
3-1	ON/OFF Switch	13
4-1	RESET Button Location	19
4-2	Locations of the Power Supplies	19
4-3	Measuring and Adjusting a Power Supply	20
4-4	Fault-Finder Flow Chart	22
5-1	Pascal MICROENGINE Computer Components	26
5-2	Default Parallel Port Control Register Values	48
5-3	Format of First Data Word Transmitted to the FDC	53

TABLES

Table Number	Title	
2-1	Floppy Disk Controller Option Switch Settings	7
2-2	Standard System Terminal Switch Settings	8
2-3	System Terminal Specifications Switch Settings	9
2-4	Serial Port Baud Rate Switch Settings	10
5-1	Communication Bus Structure Control Lines	27
5-2	Standard General Device Addresses	29
5-3	Interrupt Conditions and Vector Addresses	30
5-4	Serial Port Cable Connector Pin Assignments	32
5-5	Serial Port Addresses	34
5-6	Serial Port Status Register Contents	35

5-7 Serial Port Control Register 1 Contents..... 37
5-8 Serial Port Control Register 2 Contents..... 41
5-9 Serial Port SYN, DLE, and Control Register Addresses..... 42
5-10 Parallel Port Cable Connector Pin Assignments..... 44
5-11 Parallel Ports Control Lines (Port C)..... 45
5-12 Parallel Ports Addresses..... 46
5-13 Parallel Ports Control Register Contents..... 47
5-14 Floppy Disk Cable Connector Pin Assignments..... 50
5-15 DMAC/FDC Addresses..... 52
5-16 DMAC Control Register Contents..... 54
5-17 DMAC Status Register Contents..... 55
5-18 Stepping Motor Rates for FDC Type I Commands..... 59
5-19 FDC Command Register Values for Type I Commands..... 60
5-20 FDC Status Register Contents for Type I Commands..... 61
5-21 FDC Command Register Values for Type II Commands..... 63
5-22 FDC Command Register Values for Type III Commands..... 64
5-23 FDC Status Register Contents for Type II and Type III Commands..... 64
5-24 FDC Command Register Values for the Force Interrupt Command (Type IV)..... 65

1. OVERVIEW

The WD/90 Pascal MICROENGINE Computer is a desktop microcomputer for developing and executing programs in the Pascal language. Figure 1-1 shows the outward appearance of the computer.

Figure 1-1. WD/90 Pascal MICROENGINE Computer

To create a complete computer system, the user need only add a system terminal, floppy disk drive, and any other desired I/O peripheral devices.

This introductory chapter provides an overview of the hardware components of the computer; the software provided with the computer; the mechanical, environmental, and electrical requirements of the computer; and the computer's operation. Subsequent chapters describe procedures for using the computer and also provide detailed information on the operation of the computer and its components.

1.1 COMPONENTS OF THE COMPUTER

The WD/90 Pascal MICROENGINE Computer is driven by Western Digital's WD/9000 Pascal MICROENGINE chip set, a stack-oriented 16-bit processor that directly executes UCSD's Pascal P-code. The processor and the other hardware components of the computer are on an 8 by 16-inch board enclosed in a low-profile (5-1/4 by 16-1/4 by 13-1/2 inch) stylized housing.

The WD/9000 processor is a hardware realization of UCSD's pseudo P-machine. The processor is comprised of five LSI/MOS circuits, each contained in a 40-pin package. The individual circuits are:

- The Data Chip, containing the microinstruction decoder, the arithmetic and logic unit (ALU), and the register file.

- The Control Chip, containing the macroinstruction decoder, portions of the control circuitry, the microinstruction counter, and input/output control logic.

- Three 22 by 512-bit MICROM chips, containing processor microinstructions.

The processor uses 4 power supplies (+5V, +12V, -12V, and -5V) and runs off a 3 MHz clock signal that is subdivided into 4 nonoverlapping phases (75 nanoseconds per phase). All I/O signals are compatible with transistor-transistor logic (TTL).

In addition to the five processor chips, the board contains:

- 64K bytes (32K words) of random-access memory (RAM).

- Two RS-232 asynchronous serial ports with switch-selectable baud rates from 110 to 19,200 bits per second.

- Two 8-bit parallel ports.

- A Floppy Disk Controller that is switch-selectable for single or double-density 5-1/4 or 8-inch floppy disks and which can control up to 4 drives of the same type. The Floppy Disk Controller operates under control of the Direct Memory Access Controller.

1.2 PASCAL OPERATING SYSTEM

The software provided with the Pascal MICROENGINE Computer is the Pascal Operating System developed at the University of California at San Diego (UCSD). This operating system includes:

- A Pascal Compiler.

- A BASIC Compiler.

- A File Manager.

- A Screen-Oriented Text Editor.

- A Debugging System.

The operating system is available in any of the following forms:

- o One 8-inch single-density floppy disk.
- o One 8-inch double-density floppy disk.
- o Two 5-1/4-inch single-density floppy disks.
- o Two 5-1/4-inch double-density floppy disks.

Chapter 3 of this manual describes the procedures for loading the operating system into the computer. Complete information of the use of the operating system is provided in the Pascal III.0 Operating System Reference Manual.

1.3 REQUIREMENTS OF THE COMPUTER

The Pascal MICROENGINE Computer has a straight forward set of easily satisfied requirements:

- o The computer requires a flat, hard surface to hold the board enclosure, which is 5-1/4 inches high, 16-1/4 inches wide, and 13-1/2 inches deep.
- o The computer can operate in temperatures in the range 0-50 degrees C. Humidity can range between 0% and 95% without affecting the computer's operation.
- o The computer normally runs on 110 volts AC. Optionally, the user can order a Pascal MICROENGINE Computer which runs on 220 volts AC. In either case, the line frequency range is 48-63 Hz.

1.4 COMPUTER OPERATION

Communication between components of the computer -- processor, memory, peripheral ports, and controllers -- is achieved via a processor-controlled bus structure. This structure is comprised of an address bus and latch, a bidirectional data bus, and several control lines.

The processor initiates communication with a peripheral device or a location in memory by loading the appropriate address onto the address bus. A peripheral device can initiate communicaton with the processor by generating an interrupt. Memory cannot initiate communication with the processor; all processor-memory communication is initiated by the processor. In addition, the Floppy Disk Controller operates under control of the Direct Memory Access Controller, allowing data transfers between a floppy disk and memory without involving the processor. Each of these communication mechanisms is discussed in Chapter 5.

2. INSTALLATION PROCEDURES

When a new Pascal MICROENGINE Computer is received, the first step is to unpack the components and verify that all of the items listed on the packing slip have been included and are intact. (In the event that any of the components are missing or have been damaged in shipping, contact your MICROENGINE Company representative immediately.)

After verifying that the delivery is in order, take the steps outlined in this chapter to install the computer.

2.1 SETTING OPTION SWITCHES

The Pascal MICROENGINE Computer provides switches for specifying the characteristics of the floppy disk containing the Pascal Operating System, for providing information about the system terminal, and for selecting the baud rate for each of the two serial ports.

2.1.1 Opening the Computer Housing

The first step is to open the computer housing. Begin by turning the computer on its side with the ON/OFF switch toward the top. (CAUTION: the computer housing has sharp edges. Be careful not to scratch your table top.) Remove the four screws located on the rear panel as illustrated in Figure 2-1.

Slide the rear panel and attached PC board about half way out as illustrated in Figure 2-2.

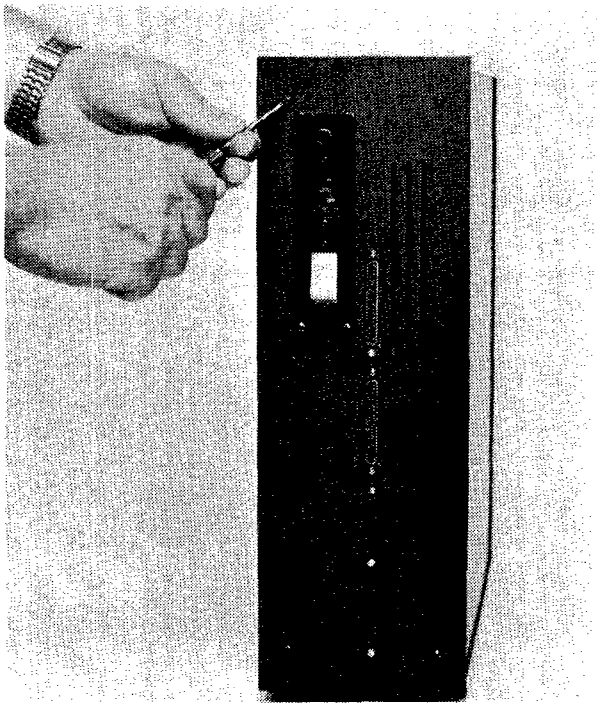


Figure 2-1. Removing the Rear Panel

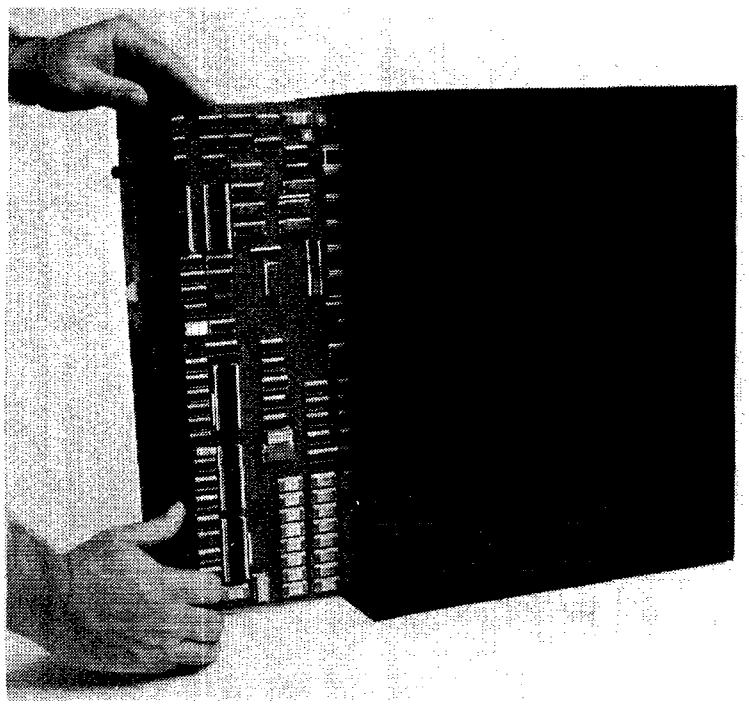


Figure 2-2. Removing the PC Board

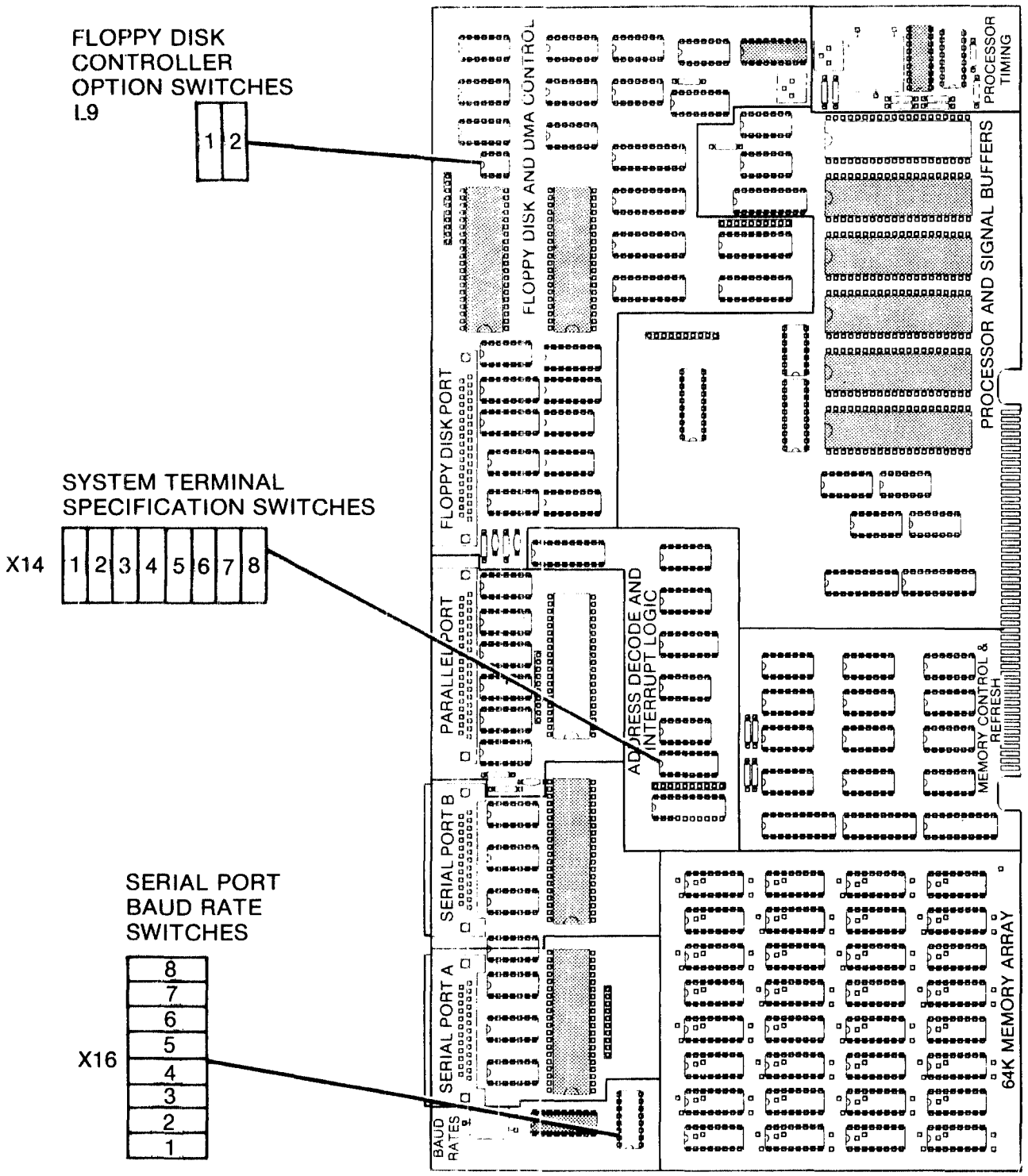


Figure 2-3. Location of the Option Switch Packages

2.1.2 Setting the Floppy Disk Controller Option Switches

Locate the Floppy Disk Controller option switch package (component designation L9) on the board as shown in Figure 2-3.

This package contains two switches. Switch S1 OFF corresponds to single-density mode, and switch 1 ON corresponds to double-density mode. Switch S2 OFF corresponds to 8-inch disks, and switch 2 ON corresponds to 5-1/4-inch disks.

Set these switches according to the characteristics of the floppy disk(s) containing the Pascal Operating System. Table 2-1 summarizes the possible combinations of disk characteristics and the corresponding option switch settings.

Table 2-1. Floppy Disk Controller Option Switch Settings

Operating System Resides On:	Set S1 to:	Set S2 to:
8-inch double-density disk	ON	OFF
8-inch single-density disk	OFF	OFF
5-1/4-inch double-density disks	ON	ON
5-1/4-inch single-density disks	OFF	ON

2.1.3 Setting the System Terminal Specifications Switches

After setting the Floppy Disk Controller option switches, the next step is to provide some information about the system terminal, i.e., the terminal to be used as the vehicle for communication between the computer and the user. A hardware option switch package (X14) is provided for supplying this information, located as shown in Figure 2-3.

Eight switches are provided.

Set these switches according to the characteristics of the system terminal. Table 2-2 outlines the settings for a standard system terminal: Table 2-3 describes the function of each switch. The controls on the system terminal must be set to inhibit parity generation (Operating System constraint). Switches S7 and S8 must be set to match the number of bits per character generated by the terminal.

Table 2-2. Standard System Terminal Switch Settings

Switch	Setting	
S1	1	("OFF" or "OPEN")
S2	0	("ON" or "CLOSED")
S3	0	("ON" or "CLOSED")
S4	1	("OFF" or "OPEN")
S5	-	Don't Care
S6	1	("OFF" or "OPEN")
S7	x	(Bits per character; must
S8	y	match setting on terminal.)

PARITY MUST BE TURNED OFF AT TERMINAL.

Table 2-3. System Terminal Specifications Switch Settings

Switch	Function
S1-3	Select the transmit and receive clock: S3 S2 S1 0 0 0 - Transmit and receive clock input (1X) 0 0 1 - Rate 1 (32X) 0 1 0 - Rate 2 (32X) 0 1 1 - Rate 3 (32X) 1 0 0 - Rate 4 (32X) 1 0 1 - Rate 4 - 2 (64X) 1 1 0 - Rate 4 - 4 (128X) 1 1 1 - Rate 4 - 8 (256X)
S4	In asynchronous mode (switch 6=0), controls the alternate Receiver clock rate: 0 - Receiver clock rate determined by switches 1-3. 1 - Receiver clock rate = Rate 1. In synchronous mode (switch 6=1), controls SYN character stripping: 1 - SYN character stripping enabled. 0 - SYN character stripping disabled.
S5	Controls whether parity is odd or even: 1 - odd parity. 0 - even parity.
S6	Controls the character mode: 1 - synchronous mode. 0 - asynchronous mode.
S7-8	Select the number of bits per character: S8 S7 0 0 - 8 bits. 0 1 - 7 bits. 1 0 - 6 bits. 1 1 - 5 bits.
where:	"0" corresponds to "ON" or "CLOSED" "1" corresponds to "OFF" or "OPEN"

2.1.4 Setting the Serial Port Baud Rate Switches

Locate the serial port baud rate selection switch package (X16) on the board as shown in Figure 2-3.

Four switches are provided for each serial port, allowing 16 possible rate selections for each port.

Set the switches for each serial port to the baud rates appropriate to the type of device to be connected to that port. Table 2-4 outlines the settings for commonly-used baud rates.

Table 2-4. Serial Port Baud Rate Switch Settings

	Switch Settings	
	Port A	Port B
Baud Rate	1 2 3 4	5 6 7 8
50	0 0 0 0	0 0 0 0
75	1 0 0 0	0 0 0 1
110	0 1 0 0	0 0 1 0
134.5	1 1 0 0	0 0 1 1
150	0 0 1 0	0 1 0 0
300	1 0 1 0	0 1 0 1
600	0 1 1 0	0 1 1 0
1200	1 1 1 0	0 1 1 1
1800	0 0 0 1	1 0 0 0
2000	1 0 0 1	1 0 0 1
2400	0 1 0 1	1 0 1 0
3600	1 1 0 1	1 0 1 1
4800	0 0 1 1	1 1 0 0
7200	1 0 1 1	1 1 0 1
9600	0 1 1 1	1 1 1 0
19200	1 1 1 1	1 1 1 1

where: "0" corresponds to "ON" or "CLOSED"
"1" corresponds to "OFF" or "OPEN"

2.1.5 Reclosing the Computer Housing

The computer housing can be reclosed by reversing the steps outlined in subsection 2.1.1. Slide the PC board and rear panel back in place and then attach the rear panel by replacing the four screws.

2.2 INSTALLING SYSTEM COMPONENTS

Once the option switches have been set, the various components of the system can be installed.

2.2.1 Installing the Computer

Place the housing containing the Pascal MICROENGINE Computer on a flat, hard surface. Be sure that neither the air intake vent nor the air outlet vent is blocked. As illustrated in Figure 2-4, The air intake vent is located on the bottom panel of the computer housing, and the air outlet vent is located on the rear panel.

Be sure that the computer is located near an electrical outlet, preferably within six feet, the length of the power cord.

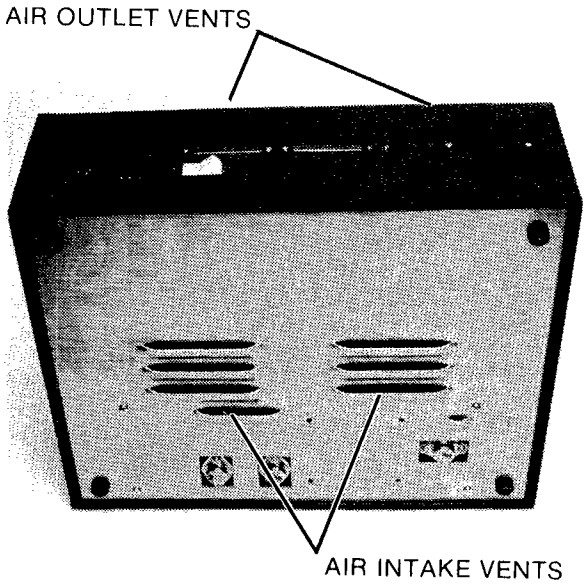


Figure 2-4. Air Vents

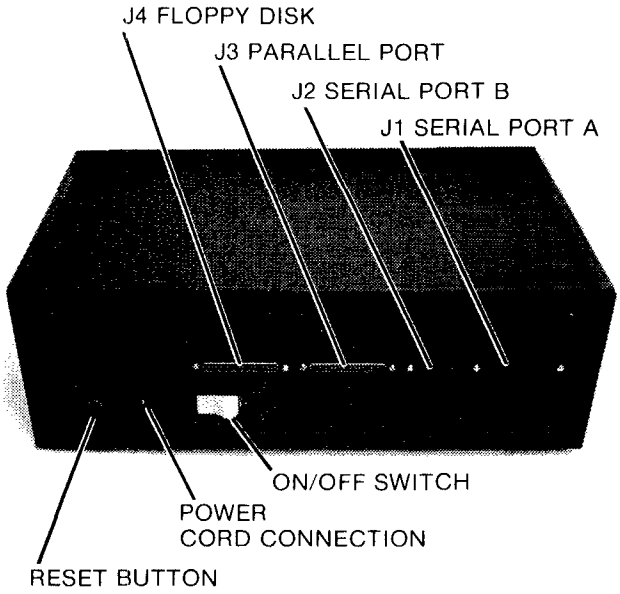


Figure 2-5. Rear Panel

2.2.2 Connecting the Peripherals

Before connecting any peripheral device to the Pascal MICROENGINE Computer, review the documentation provided by the manufacturer of that device. Perform the specified inspections and off-line start-up procedures.

After completing the peripheral device pre-installation inspection and verification procedures, locate the port connections on the rear panel of the computer housing, illustrated in Figure 2-5.

Connect the floppy disk drive cable to the connection labeled J4 FLOPPY DISK.

Connect the cable from the terminal to be used for system/user communication to the connection labeled J1 SERIAL PORT A.

If other devices are to be connected to the auxiliary serial port and/or the parallel port, connect those cables accordingly.

When a device is connected to the peripheral ports, the cable connector must be wired for either true data or false data. For true data, connect the TDO and TDI lines (pins 32 and 8, respectively) to ground (pin 7). For false data, connect TDO and TDI to the +5 volt supply (pin 26).

2.2.3 Plugging In the Power Cord

Next, locate the ON/OFF rocker switch on the rear panel of the computer housing, as illustrated in 2-5.

BE SURE THAT THIS SWITCH IS SET TO OFF. Then, locate the power cord connection near the ON/OFF switch on the rear panel. Plug the system power cord into this connection.

If the computer was ordered to operate on 110 volts, the power cord is delivered with a standard wall outlet plug attached. If the system was ordered to operate on 220 volts, the power cord is delivered without an outlet plug. Attach an appropriate plug to the cord.

After AC power is applied, the system is ready for operation. Operating procedures are outlined in Chapter 3.

3. OPERATING PROCEDURES

When the Pascal MICROENGINE Computer is properly installed (see Chapter 2), operation consists primarily of starting and stopping the system.

3.1 STARTING AND STOPPING THE SYSTEM

System start-up involves three operations:

1. Turn on the power.
2. Load the system diskette into the disk drive.
3. Depress the RESET button.

3.1.1 Turning On the Power

Turn on the power by setting the ON/OFF rocker switch to ON. The ON/OFF switch is located on the rear panel of the computer housing, as illustrated in Figure 3-1.

The ON/OFF switch lights when set to ON to indicate that the computer is receiving power. (If the switch fails to light, make sure that both ends of the power cord are plugged in properly. If the switch still fails to light, the user can assume that the wall outlet is not supplying power, since the switch light is wired directly to the power cord connection.)

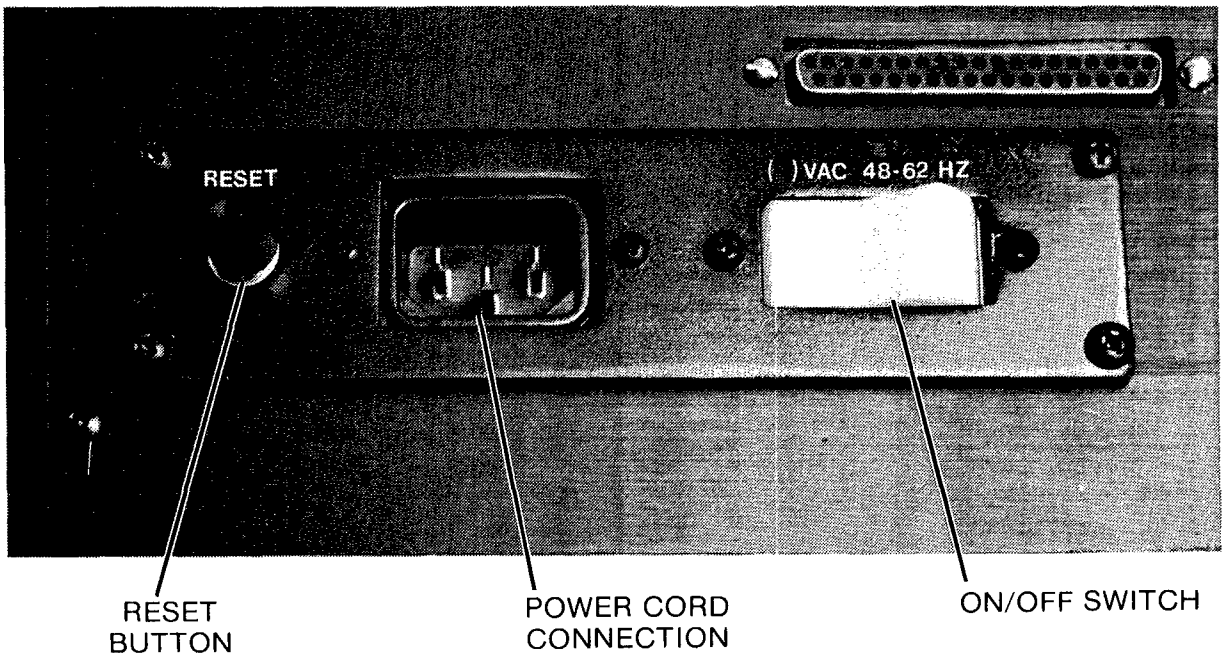


Figure 3-1. ON/OFF Switch

3.1.2 Loading the System Software

The software provided with the Pascal MICROENGINE Computer is the University of California at San Diego (UCSD) Pascal Operating System. This system includes:

- o Pascal Compiler
- o BASIC Compiler
- o File Manager
- o Screen-Oriented Text Editor
- o Debugging System

To load the operating system, simply load the operating system floppy disk into floppy disk drive (0) and depress the RESET button located next to the power cord connection on the rear panel (see Figure 3-1).

> NOTE: The MICROENGINE Company recommends that whenever the system is started, the steps just discussed be performed in the order described. That is, the ON/OFF switch should be set to ON before a floppy disk is loaded.

The operating system announces itself and then generates the line:

```
Command:  E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssem, D(ebug
```

to the screen of the system terminal, indicating that the system has been loaded successfully and is ready to accept any of the listed commands. The Pascal III.0 Operating System Reference Manual provides complete information on the user's options at this point. In the event that the command prompt line fails to appear repeat the above procedures carefully. If the prompt line still fails to appear consult Section 4.2.

3.1.3 Stopping the System

At the end of a session with the Pascal MICROENGINE Computer system, simply remove the software floppy disk(s) from the floppy disk drive(s) and set the ON/OFF switch to OFF.

> NOTE: The MICROENGINE Company recommends that these procedures be performed in this order whenever the system is stopped. That is, the floppy disk(s) should be removed before the ON/OFF switch is set to OFF.

3.2 CHANGING HARDWARE OPTION SWITCH SETTINGS

Chapter 2 outlines the procedures for setting the Pascal MICROENGINE Computer hardware option switches as part of the system installation process. (Switches are provided for specifying the floppy disk, system terminal and baud rates for the serial ports.) If the user wishes to change these settings, he follows the same procedures, outlined in Chapter 2.

4. TROUBLESHOOTING PROCEDURES

The Pascal MICROENGINE Computer requires no preventive maintenance. This chapter outlines procedures that are required only if a problem arises with the operation of the system.

After a general discussion of the troubleshooting process, this chapter describes specific procedures, presented in terms of the "symptom" displayed by the system. That is, this chapter details the steps to be taken if:

- o the Pascal Operating System fails to announce itself on the system terminal upon loading;
- o data transfers to/from the system terminal, the line printer or other device connected to the parallel ports, or a floppy disk fail; or
- o the system exhibits none of the above symptoms but operates erratically.

The last section of this chapter summarizes the total fault-finding process for a complete Pascal MICROENGINE Computer system, again in flow chart form.

NOTE: This chapter assumes that each of the system's peripheral devices was thoroughly inspected and tested before it was installed. If this is not the case, The MICROENGINE Company strongly recommends that the user perform these procedures before beginning the fault-finding process. Peripheral inspection and testing procedures are discussed in subsection 2.2.2.

4.1 THE TROUBLESHOOTING PROCESS: A GENERAL DISCUSSION

Fault-finding is largely a process of elimination. By performing the verification procedures outlined in this chapter, the user systematically rules out possible sources of the problem until the malfunctioning component is isolated. The procedures require a high-quality oscilloscope capable of resolving pulses of approximately 200 nanoseconds and a high input impedence (10 meg ohms or greater) digital voltage meter.

Once the defective component has been isolated, three possible courses of action are open:

- o Repair the component, following appropriate wiring diagrams.
- o Install a replacement component, obtained from a The MICROENGINE Company distributor or ordered directly from The MICROENGINE Company. A distributor list is available from any MICROENGINE Company representative.
- o Return the component for service. If a peripheral device is malfunctioning, return the device to it:

manufacturer for service according to that manufacturer's service agreement. In the case of a malfunctioning Pascal MICROENGINE Computer component, send that component to The MICROENGINE Company's factory for repair. If the computer is under warranty, the customer is responsible only for shipping the component to The MICROENGINE Company; The MICROENGINE Company provides required parts and labor and return shipping. If the computer is not under warranty, there is a fixed repair fee and the customer is responsible for all shipping. (For details, see the Pascal MICROENGINE Computer warranty statement and factory service agreement.

4.2 SYMPTOM 1: OPERATING SYSTEM FAILS TO ANNOUNCE ITSELF

Normally, when the Pascal Operating system is loaded, it announces itself on the system terminal and prompts the user to enter a command. If the announcement and prompt line do not appear when the operating system is loaded, follow the procedures outlined in this section.

4.2.1 Repeating the Software Loading Procedure

First, try pressing the RESET button. This button is located on the rear panel of the computer housing, as illustrated in Figure 4-1.

If the operating system still fails to announce itself, try reloading the floppy disk and pressing RESET. If this operation does not correct the problem, verify that the system terminal ON/OFF switch is set to ON. If it is not, set the switch to ON and then press the RESET button. If the operating system still fails to announce itself, try loading the backup operating system floppy disk and pressing RESET.

4.2.2 Checking the Floppy Disk Controller Option Switches

If none of the procedures outlined in subsection 4.1.1 correct the problem, check the settings of the Floppy Disk Controller hardware option switches. (As explained in subsection 2.1.2, these switches are set according to the size and density of the floppy disk or disks being loaded.)

If the switches are not set appropriately, change their settings as described in Section 2.1.1. Reclose the computer housing, reload the floppy disk, and press the RESET button.

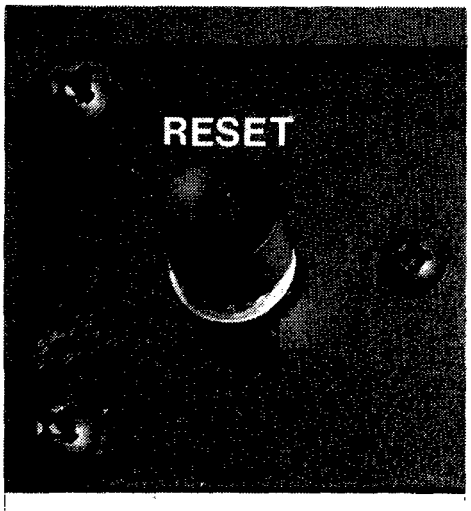


Figure 4-1. RESET Button Location

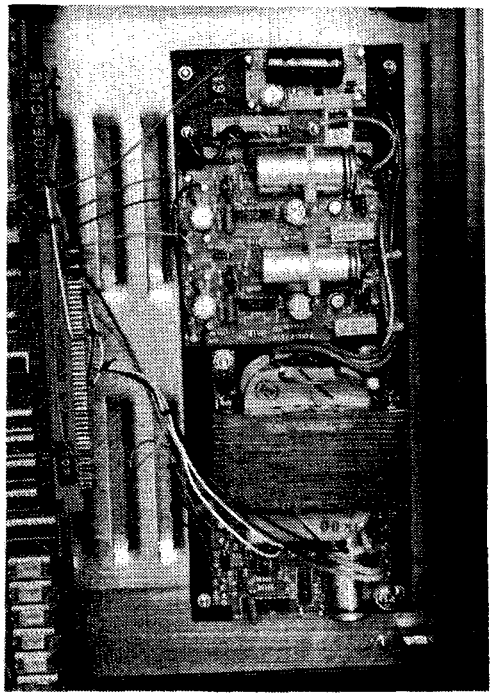


Figure 4-2. Location of the Power Supplies

4.2.3 Measuring and Adjusting the Power Supplies

If the operating system still fails to announce itself after the Floppy Disk Controller option switch settings have been checked, verify that the system power supplies are functioning properly and adjust them if they are not.

Begin by unplugging the system and opening the computer housing. Turn the computer on its side with the ON/OFF switch located at the top. (Caution: the computer housing has sharp edges, so be careful not to scratch your table top.) Remove the four screws on the computer panel and slide the rear panel and attached PC board all the way out of its mounting rails. Slide the top cover all the way out of its mounting rails and then replace the printed circuit board part-way back into its mounting rails. Locate the power supplies as illustrated in Figure 4-2.

Note that the board includes 4 adjustable power supplies: +5 volts, +12 volts, -12 volts, and -5 volts. Measure the voltage of each by clipping a digital voltmeter across each pair of terminals as illustrated in Figure 4-3.

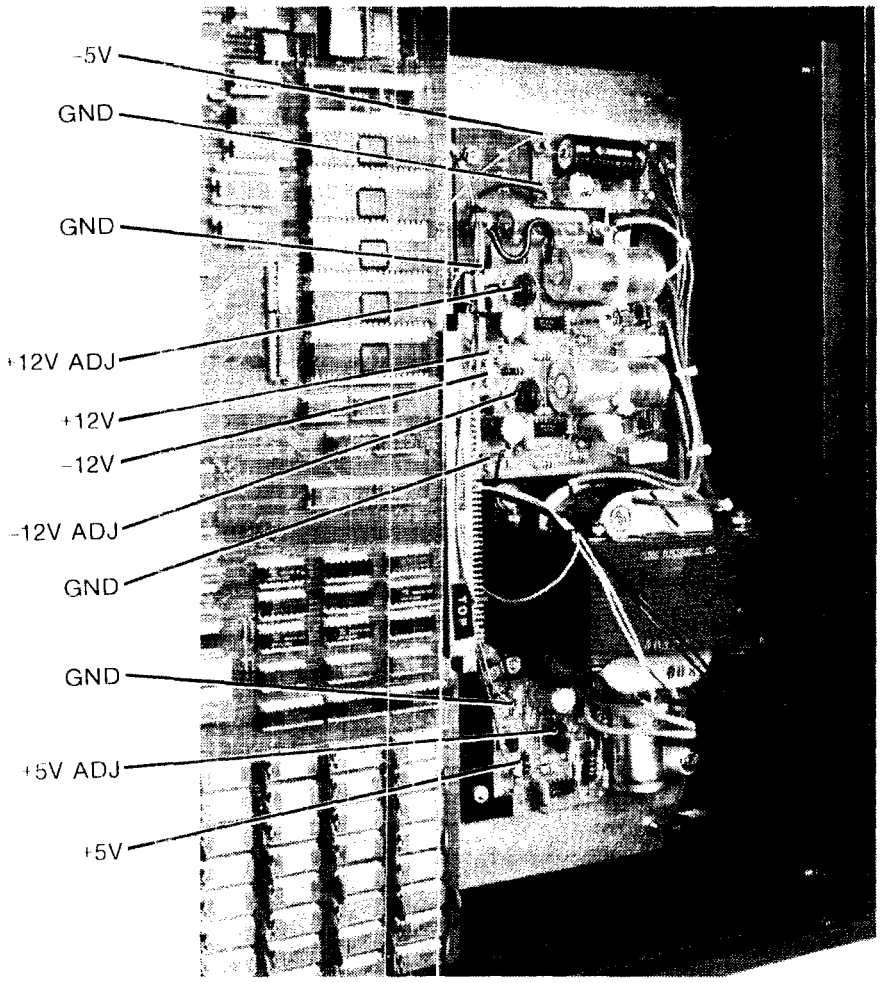


Figure 4-3. Measuring and Adjusting a Power Supply

If any of the supplies are not producing proper voltage, adjust them to within 10 millivolts of their nominal level. Insert a screwdriver into the slot provided on the power supply adjustment wheel for this purpose; turn the wheel to the right to increase the voltage or to the left to decrease the voltage. (See Figure 4-3.)

> NOTE: When replacing a power supply (or the entire board), measure the voltage produced by each new supply and, if necessary, adjust it by following the procedure just outlined.

After verifying that the hardware option switches are set correctly, the next step depends on whether the floppy disk drive is active or not. (Activity on the drive can be detected by clicking noises produced when a floppy disk is loaded into it.) The fault-finder flow chart presented in Figure 4-4 diagrams the steps to be taken in either situation; see Section 4.6.

4.3 SYMPTOM 2: TRANSFERS TO THE SYSTEM TERMINAL FAIL

If the Pascal Operating System announces itself on the system terminal and accepts commands from the terminal, but is unable to transmit data to the system terminal successfully, check the configuration of the Pascal Operating System. See the Pascal III.0 Operating System Reference Manual.

4.4 SYMPTOM 3: TRANSFERS TO THE PARALLEL PORTS DEVICE OR FLOPPY DISK FAIL

If the system is unable to perform successful transfers to the device connected to the parallel ports (e.g., a line printer) or to the floppy disk, perform the procedures diagrammed in Figure 4-4; see Section 4.6.

4.5 SYMPTOM 4: SYSTEM FAILS TO PASS DIAGNOSTIC PROGRAM

If the system displays none of the symptoms discussed in Sections 4.2 - 4.4 but behaves erratically, execute the system diagnostic program. Replace malfunctioning components according to the directions supplied by the program.

4.6 THE TROUBLESHOOTING PROCESS: A SUMMARY FLOW CHART

Figure 4-4 presents a flow chart diagramming the complete fault-finding process.

> NOTE: Use of this flow chart assumes a high level of competence with advanced digital circuits and a background in the use and operation of all peripherals used in the complete Pascal MICROENGINE System. Users not familiar with digital logic are advised to return the complete unit to The MICROENGINE Company for comprehensive automated analysis. This procedure saves the user downtime and learning delays encountered when trying to fault-find an advanced electronic unit such as the Pascal MICROENGINE.

To initiate use of the fault-finder flow chart, the following conditions must be met:

1) All line cords are plugged in, all power switches are "ON" with appropriate indicator also "ON", and all peripherals are on-line.

2) Operating system disk is in disk drive 0 and is in "RESET" condition.

3) User must have a high quality oscilloscope capable of resolving 200 nsec pulses and a high input impedence (10Meg ohm or greater) D.V.M. for accurate readings without circuit loading.

4) Pre-installation inspection and testing procedures have been performed for each peripheral device.

Figure 4-4. Fault-Finder Flow Chart

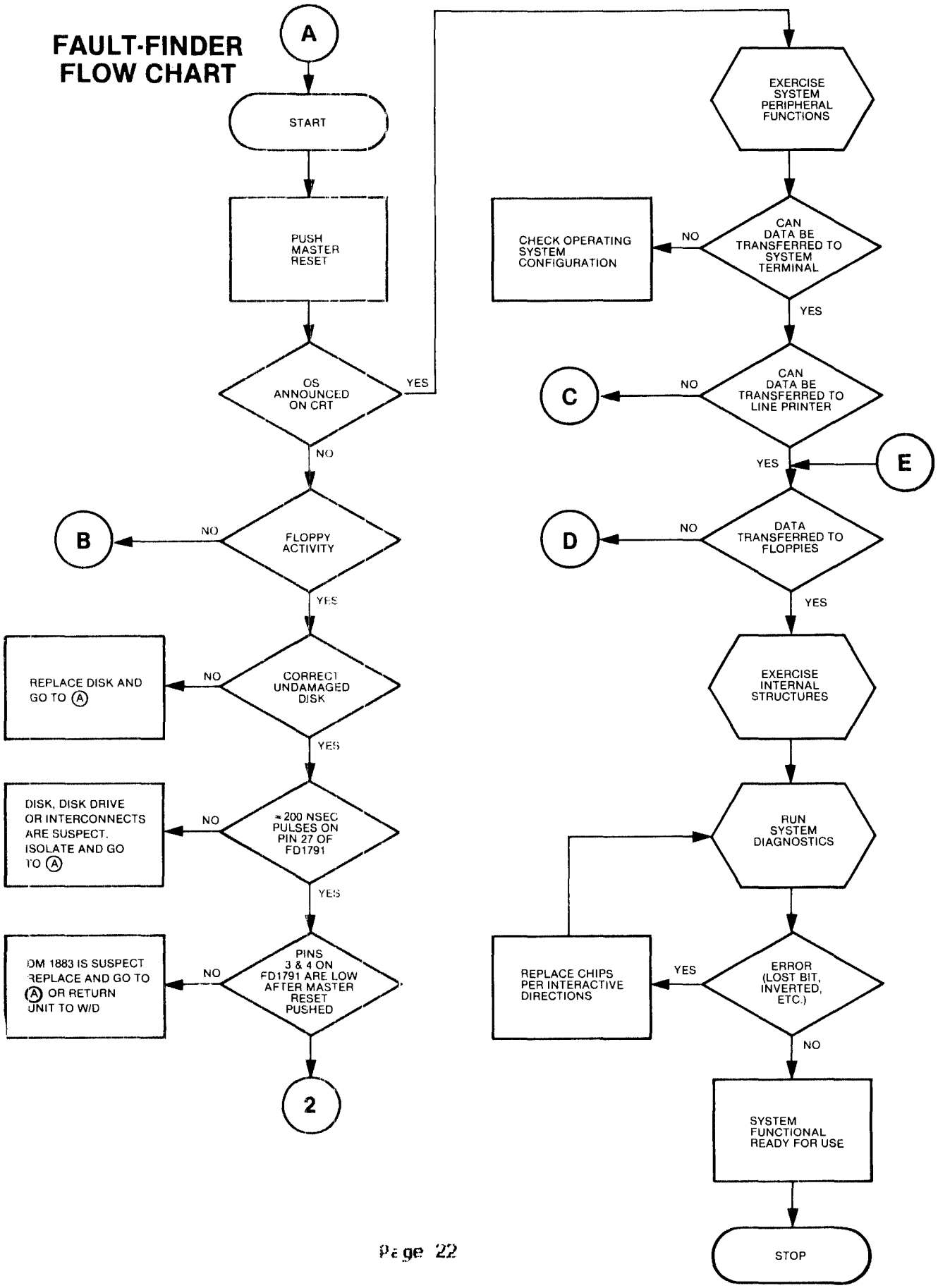


Figure 4-4. Fault-Finder Flow Chart (Continued)

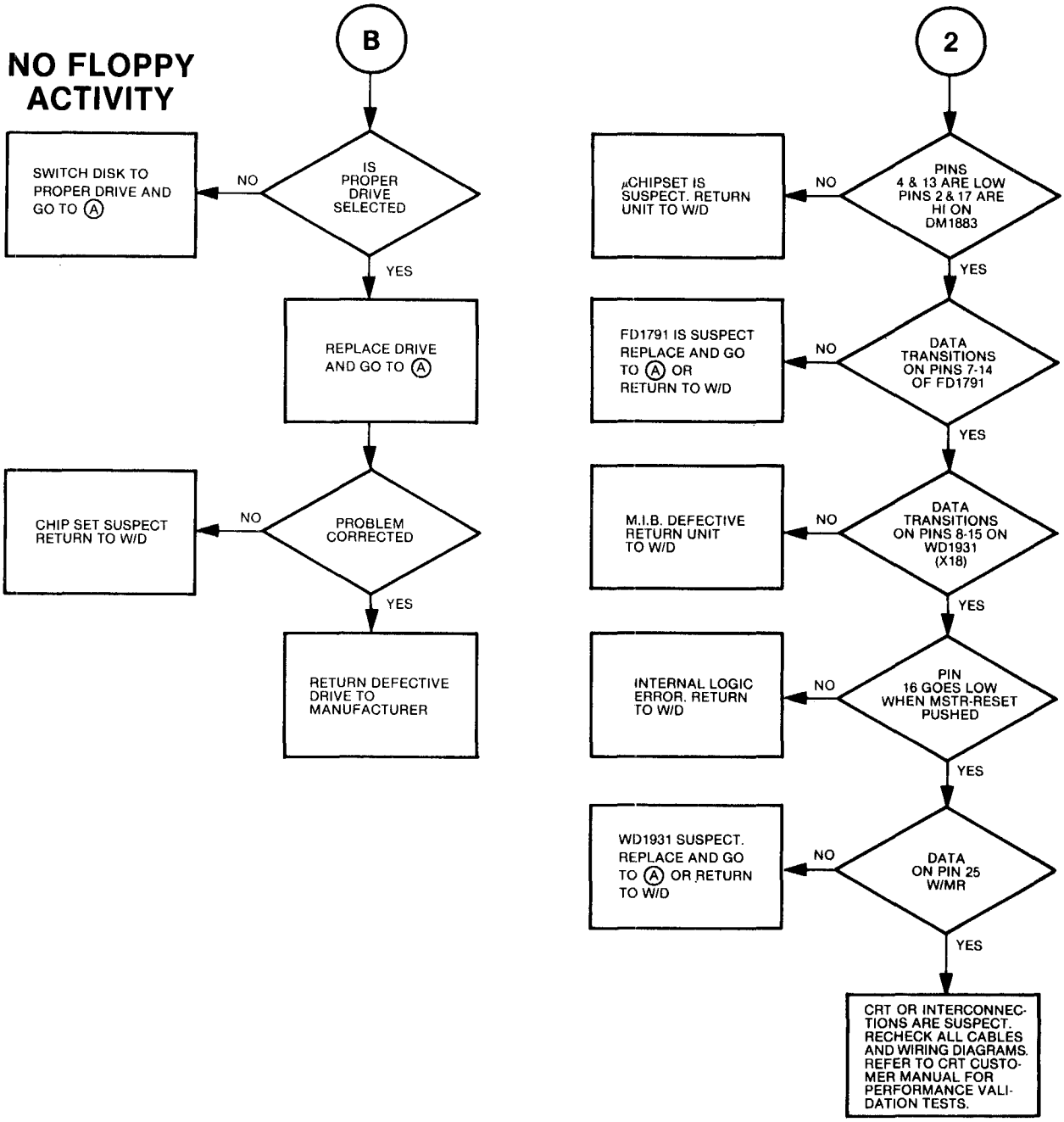
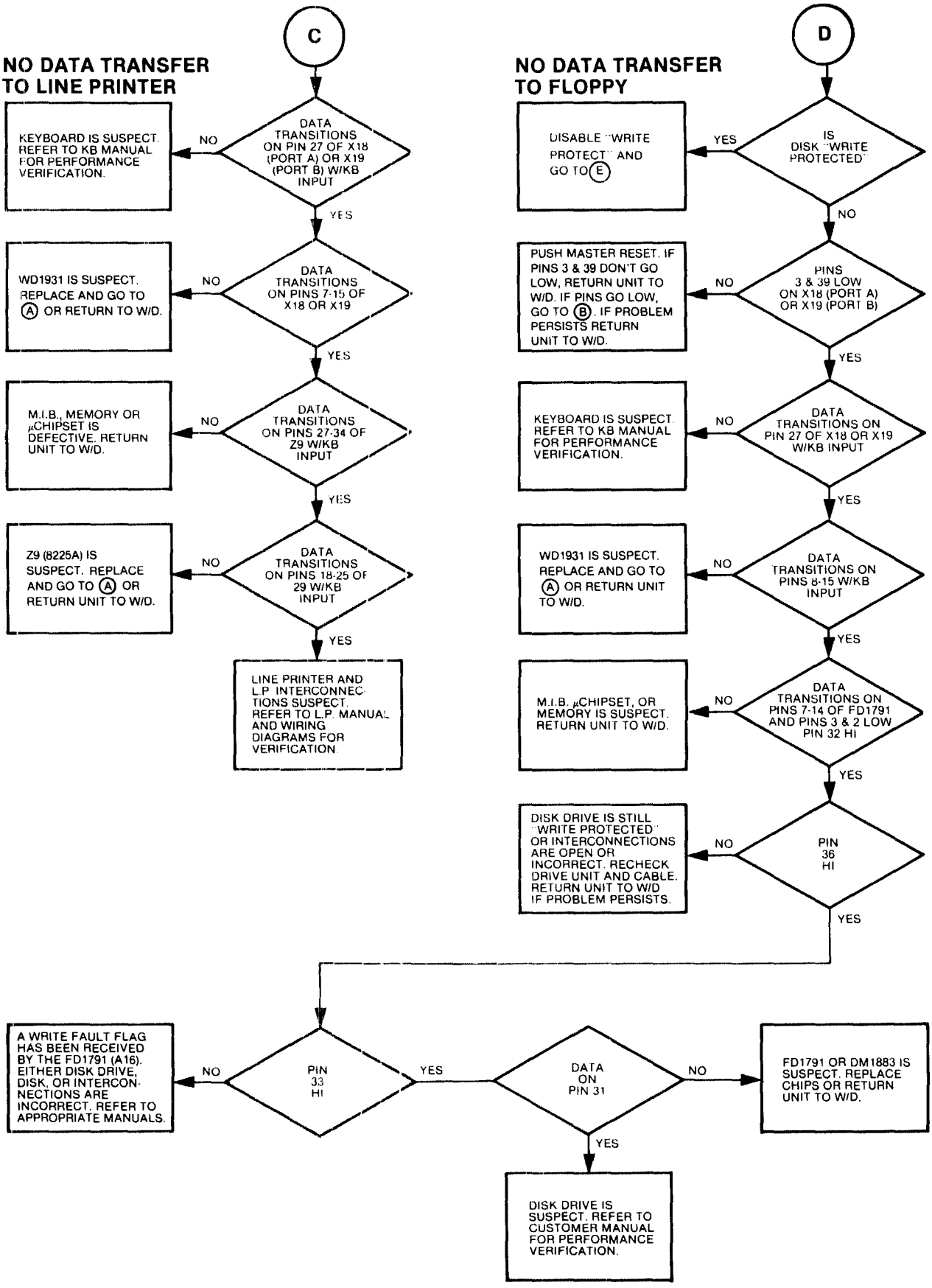


Figure 4-4. Fault-Finder Flow Chart (Continued)



5. THEORY OF OPERATIONS

This chapter describes the operation of the following Pascal MICROENGINE Computer components:

- o Pascal MICROENGINE processor
- o Memory
- o Serial ports
- o Parallel ports
- o Floppy Disk Controller and DMA Controller

Before discussing the operation of these components individually, this chapter presents an overview of the mechanisms for communication between the processor and memory and between the processor and the ports and controllers.

5.1 PROCESSOR/MEMORY AND PROCESSOR/PERIPHERALS COMMUNICATION

The Pascal MICROENGINE processor communicates with the peripheral ports and controllers and with memory via a 16-bit data bus, a 16-bit address bus, and several control lines. These busses and control lines emanate from a processor controlled bus structure. This structure is comprised of a 16-bit address bus, 16-bit address latch, a 16-bit bi-directional data bus, and latches and bus drivers as required for the control lines. The key control lines in this structure are listed in Table 5-1.

The processor initiates communication with a peripheral device or a location in memory by loading the appropriate address onto the address bus. A peripheral device can initiate communication with the processor by requesting an interrupt. (Memory cannot initiate communication with the processor; all processor-memory communication is initiated by the processor.) The following subsections describe each of these communication mechanisms.

In addition, the Floppy Disk Controller operates under control of the Direct Memory Access Controller, allowing data transfers between a floppy disk and memory without involving the processor. This type of communication is described in section 5.6.

Figure 5-1 illustrates the location of these components on the PC board.

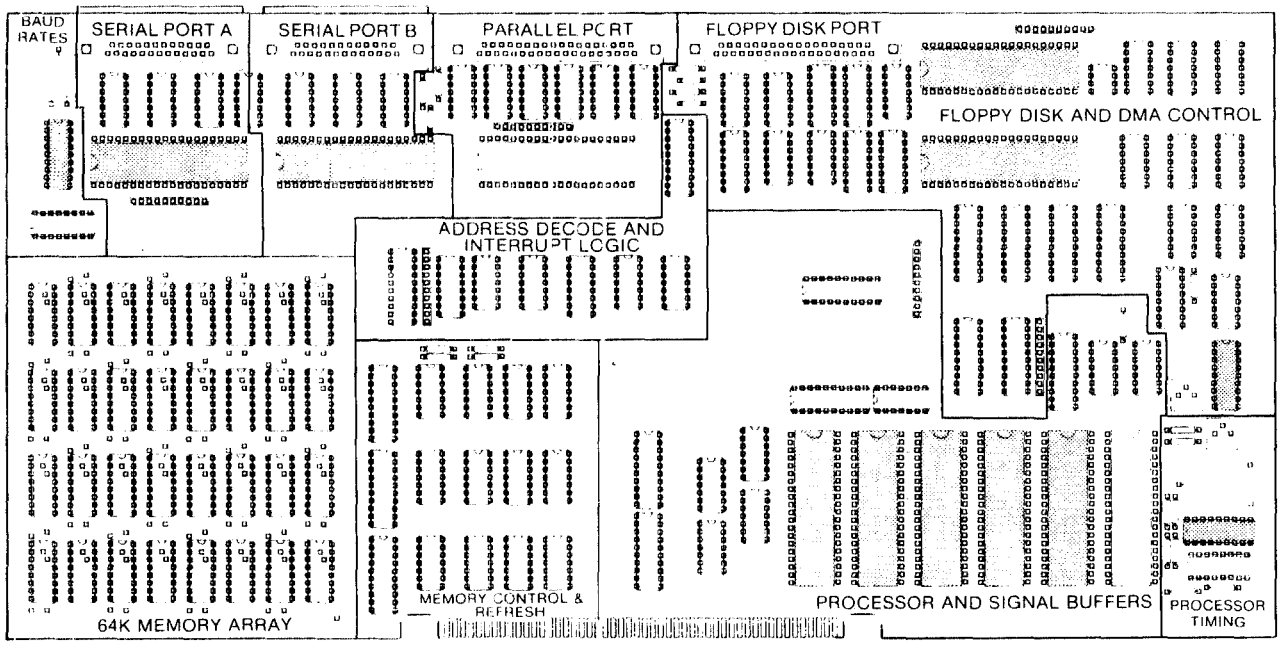


Figure 5-1. Pascal MICROENGINE Computer Components

Table 5-1. Communication Bus Structure Control Lines

Line	Function
SYNC	used by the processor to initiate a data transfer operation
REPLY	used by the ports and controllers to respond to the processor's data transfer signals.
DIN (Data-In)	used by the processor to cause the Read Data to be enabled onto the data bus.
DOUT (Data-Out)	used by the processor to cause the Write Data to be enabled onto the data bus.
WB (Write/Byte)	used by the processor to signify a byte (rather than word) output operation.
I0, I1, I2, and I3 (Interrupts)	used by the ports and controllers to request a processor interrupt. (Only I0 is used for all interrupts in the Pascal MICROENGINE Computer)
IACK (Interrupt Acknowledge)	used by the processor to signify that it is responding to an interrupt.
BUSREQ	used by the DMA Controller to request access to a bus for a word transfer operation.
BUSGRNT	used by the processor to signal the DMA Controller that the requested bus is being relinquished to the controller for a word transfer operation.
BUSY	used by the DMA Controller to signal that the bus requested by the processor is not currently available.
COMPUTE	used to control execution of microinstructions by the processor. This line is tied true.

5.1.1 Processor/Memory Communication

To initiate communication with a location in memory, the processor loads a 16-bit value between 0000 and EFFF onto the address bus. Any value in this range is a valid memory address and alerts memory for communication; the particular value corresponds to a 16-bit word at one of the 32,000 locations in memory.

The processor also signals either the Data-In (DIN) control line or the Data-Out (DOUT) control line, as appropriate to the type of I/O to be performed. The memory unit acknowledges the communication and generates REPLY to the processor and then responds to the DIN or DOUT line. On a Read operation, the memory unit places the contents of the addressed word onto the data bus; on a Write operation, data from the data bus is loaded into the addressed word in memory.

5.1.2 Processor-Initiated Communication with Peripheral Devices

To initiate communication with a peripheral device, the processor loads a 16-bit address onto the address bus, as when addressing memory. Unlike a memory address, which has a simple one-for-one correspondence with a word in memory, a device address has two components: bits 4-15 identify the port or controller to which the desired device is attached, and bits 0-3 identify the desired element within that port or controller. For example, the value:

FC20 (=1111 1100 0010 0000)

addresses the control register of serial port B. FC2x is the general address of serial port B, where the value of "x" (0 in the example above) identifies the particular element (the control register in the example) of the serial port.

The processor addresses all peripheral devices in this manner, with one exception. The general address FC1x references either serial port A or the system terminal specifications hardware option switches. (These switches are discussed in section 2.1.) If bit 3 is set, these switches are activated and bits 0-2 are ignored; if bit 3 is not set, serial port A is addressed and bits 0 and 1 identify the desired element of that port.

Table 5-2 lists the standard general device addresses. The values for selecting particular device elements are detailed in subsequent sections of this chapter.

Table 5-2. Standard General Device Addresses

Address	Unit
FC7x	Parallel ports
FC6x	Reserved
FC5x	Autoload
FC4x	Reserved for Interrupt priority latch
FC3x	Floppy Disk Controller and DMA Controller
FC2x	Serial port B
FC1x	Serial port A and system terminal specifications switches
FC0x	System value of nil

As in a memory access, the processor also signals the DIN or DOUT line when addressing a port or controller. The port or controller acknowledges the communication and generates REPLY to the processor and then responds to the DIN or DOUT line. On a Read operation, the device places data on the data bus; on a Write operation, data is loaded from the data bus into the device.

5.1.3 Device-Initiated Communication with the Processor: Interrupts

When an I/O device requires service, it signals the I0 control line. When the processor detects the signal, it enables the I0 line and disables the I1, I2, and I3 lines by loading the general device address FC4x onto the address bus, with:

0 0 0 1

as the value of "x." The device then places a vector address on the data bus. A vector address is a code used to direct the appropriate interrupt service routine. The processor signals the IACK control line to signify that it is responding to the interrupt.

If multiple interrupts occur simultaneously, the processor responds to them one at a time, in order of priority. Table 5-3 lists the interrupt conditions, their priorities, and the associated vector addresses.

Table 5-3. Interrupt Conditions and Vector Addresses

Interrupt	Priority	Vector Address
DMA	Highest	0020
Serial port A receive		0024
Serial port B transmit		0028
Serial port B receive		002C
Serial port A transmit		0030
Serial port A or B exception		0034
Parallel input port		0038
Parallel output port 2	Lowest	003C

5.2 PASCAL MICROENGINE PROCESSOR

The Pascal MICROENGINE Computer is driven by the Pascal MICROENGINE processor. This processor is comprised of five LSI/MOS circuits, each contained in a 40-pin package. The chips are:

- The Data Chip, containing the microinstruction decoder, the Arithmetic and Logic Unit (ALU), and the register file.
- The Control Chip, containing the macroinstruction decoder, portions of the control circuitry, the microinstruction counter, and input/output control logic.
- Three 22 by 512-bit MICROM chips, holding the microinstructions.

The processor uses four power supplies (+5 volts, +12 volts, -12 volts, and -5 volts) and runs off a 3-MHz clock signal that is subdivided into 4 nonoverlapping phases (75 nanoseconds per phase). All I/O signals are tri-state.

Figure 5-1 demonstrates the location of the processor chips on the board.

The processor chips are interconnected by an 18-bit microinstruction bus which provides bidirectional communication between chips for addresses and instructions. The 16-bit data access bus provides access to the inter-component communication bus structure. As described in section 5.1, this structure allows the processor to communicate with memory and the peripheral ports and controllers.

For more information on the Pascal MICROENGINE processor, see the "WD/9000 Pascal MICROENGINE Microprocessor Chip Set" data sheet.

5.3 MEMORY

The Pascal MICROENGINE Computer includes 32 16K RAM chips, providing 64K bytes (32K words) of MOS dynamic random access memory (RAM). Each RAM chip is organized as 16,384 words by 1 bit, and is packaged in a standard 16-pin dual in-line package. Figure 5-1 illustrates the location of the memory package on the board.

Section 5.1 describes the mechanism by which the processor communicates with memory. In the Pascal MICROENGINE Computer, the Floppy Disk Controller operates under control of the DMA Controller. As described in section 5.6, the DMA Controller allows for direct data transfers between the Floppy Disk Controller and memory, without involving the processor.

5.4 SERIAL PORTS

Two RS-232 asynchronous serial ports are provided with the Pascal MICROENGINE Computer. One of these ports -- the A port -- is used for connecting the system terminal to the system. The second port -- the B port -- is available for connecting any other RS-232 device to the system. (Section 2.2 describes the procedures for connecting terminals to the serial ports.) The location of the serial ports is presented in Figure 5-1.

Communication between the serial ports and the processor is achieved via the inter-component communication bus structure, described in section 5.1. A terminal is connected to a serial port via a serial port cable connector. This section is concerned with the operation of the serial ports within this structure. Table 5-4 lists the pin assignments of a serial port cable connector.

Table 5-4 Serial Port Cable Connector Pin Assignments

Pin Number	Signal	Description
J1-1 or J2-1	AA	FRAME GROUND
J1-2 or J2-2	BA	TD- WD 1931
J1-3 or J2-3	BB	RD- WD 1931
J1-4 or J2-4	CA	RTS WD 1931
J1-5 or J2-5	CB	CTS WD 1931
J1-6 or J2-6	CC	DSR WD 1931
J1-7 or J2-7	AB	SIGNAL GROUND
J1-8 or J2-8	CF	CARD WD 1931
J1-9 or J2-9		
J1-10 or J2-10		
J1-11 or J2-11		
J1-12 or J2-12		
J1-13 or J2-13		
J1-14 or J2-14		
J1-15 or J2-15	DB	IXRC WD 1931
J1-16 or J2-16		
J1-17 or J2-17	DD	IXRC WD 1931
J1-18 or J2-18		
J1-19 or J2-19		
J1-20 or J2-20	CD	DTR WD 1931
J1-21 or J2-21		
J1-22 or J2-22	CE	RING WD 1931
J1-23 or J2-23		
J1-24 or J2-24		
J1-25 or J2-25		

The serial ports have been implemented with two UC1931A/B devices. The UC1931 offers a number of programmable capabilities: the device can operate in either synchronous or asynchronous mode, for example. When the Pascal Operating System is loaded, it automatically programs the serial port devices according to a pre-defined set of specifications. Some of these specifications are provided within the operating system; others are read from the hardware option switches. (Recall from section 2.1 that the system provides hardware option switches for specifying system terminal characteristics and for defining the baud rate for each serial port.)

Subsection 5.4.1 describes how the serial ports operate under this set of specifications. For the user who wishes to program the serial port devices to operate in some other manner, subsection 5.4.2 describes the mechanisms for programming the devices and briefly discusses their full programmable capabilities.

5.4.1 Operation of the Serial Ports

Each serial port device includes several registers. Of primary interest are:

- Receiver Register. This 8-bit shift register receives data from the attached device. The incoming data is assembled and then transferred to the Receiver Holding Register with logic zeros filling any unused high-order bit positions.

- Receiver Holding Register. This 8-bit parallel buffer register presents the assembled characters to the data bus when requested through a Read operation. The term "Receiver" is used in this manual to refer to the Receiver Register and the Receiver Holding Register.

- Transmitter Holding Register. This 8-bit parallel buffer register holds parallel data transferred from the data bus by a Write operation. This data is transferred to the Transmitter Register when transmission is enabled.

- Transmitter Register. This 8-bit shift register is loaded from the Transmitter Holding Register. The data is serialized and presented to the attached device when transmission is enabled. In this manual, the term "Transmitter" refers to the Transmitter Holding Register and the Transmitter Register.

- Control Registers. These two 8-bit registers hold device programmed control words. The contents of these registers are described in subsection 5.4.2.

- Status Register. This 8-bit register holds information on the status of the port's operation.

The timing of the transfer of data to and from the data bus and the attached device is controlled by a BR1941L-6 Baud Rate Clock. The rate supplied by the Baud Rate Clock to each of the serial ports is set by the hardware option switches provided for this purpose. (These switches are described in subsection 2.1.3.)

Recall from section 5.1 that the processor initiates communication with serial port A by loading the general address FC1x onto the address bus; to address serial port B, the processor loads FC2x. In both cases, the value of "x" determines which serial port register is selected. Recall further that the general address for serial port A is used also to activate the system terminal specifications switch.

Table 5-5 lists values used to address serial port registers and the system terminal specifications switch.

In this table, (and subsequent tables in this chapter), a hyphen (-) denotes a

"don't care" value; i.e., any value may appear in the corresponding position.

Table 5-5. Serial Port Addresses

Operation	General Address	Value of "x"	Element Selected
Input from Serial Port A	FC1x	0 - 1 0	Status Register
		0 - 1 1	Receiver Holding Register
Output to Serial Port A	FC1x	0 - 1 1	Transmitter Holding Register
Activate System Terminal Specifications Switch	FC1x	1 - - -	-
Input from Serial Port B	FC2x	- - 1 0	Status Register
		- - 1 1	Receiver Holding Register

Character framing is provided by a Start bit (logic zero) at the beginning of a character and a Stop bit (logic one) at the end of a character. When information is transferred from the Transmitter Holding Register, transmission is initiated. A Start bit is inserted, followed by the serial output of the character (least significant bit first) with parity (if enabled) after the most significant bit. A stop condition (1, 1.5, or 2-bits) is then inserted. If the Transmitter Holding Register is full, the next character transmission starts after the Stop bit(s) of the character currently in the Transmitter Register have been transmitted. Otherwise, the Mark (logic one) condition is continually transmitted until the Transmitter Holding Register is loaded.

Reception of a character into the Receiver Register is initiated when the first Start bit after a preceding Stop bit is recognized. During the assembly of the character from serial to parallel, the Start and Stop bits (and the parity bit, if enabled) are stripped off. The assembly is completed when the Stop bit following the last character bit is received. If the Stop bit is a logic one, the character is determined to have correct framing and the port is prepared to receive the next character. If the Stop bit is a logic zero, a framing error has ocurred. The device assumes that the bit is the Start bit of the next character. Character assembly continues if the input is still a logic zero when sampled at the theoretical center of the assumed Start bit. As long as the received input is spacing, all zero characters are assembled. Error flags and data received interrupts are generated so that line breaks can be determined. After a character of all zeros is assembled along with a zero in the Stop bit location, the first received logic one is accepted as a Stop

bit; when this bit is received, the device is prepared to receive the next character.

As Table 5-5 has demonstrated, the contents of the serial port status registers can be read onto the data bus. Table 5-6 outlines the significance of each bit in the status register.

Table 5-6. Serial Port Status Register Contents

Bit	Name	Meaning
0	TRANSMITTER HOLDING REGISTER EMPTY	This bit is set to a 1 bit when the Transmitter Holding Register does not contain a character and the Transmitter is enabled. It is set to a 1 bit when the contents of the Transmitter Holding Register are transferred to the Transmitter Register. It is cleared to a 0 bit when the Transmitter Holding Register is loaded from the data bus or the Transmitter is disabled.
1	DATA RECEIVED	This bit is set to a 1 bit when the Receiver Holding Register is loaded from the Receiver Register and the Receiver is enabled. It is cleared to a 0 bit when the Receiver Holding Register is read onto the data bus or when the Receiver is disabled.
2	OVERRUN ERROR	This bit is set to a 1 bit when the previous character in the Receiver Holding Register has NOT been read when a new character is ready to be transferred to the Receiver Holding Register. Otherwise, the bit is cleared when a character is transferred to the Receiver Holding Register. It is cleared when the Receiver is disabled.

3	PARITY ERROR	This bit is set to a 1 bit when the Receiver and parity are enabled and the last received character has a parity error. The bit is set to a 0 bit if the character has correct parity.
4	FRAMING ERROR	This bit is set to a 1 bit if the bit after the last data bit of a character is a zero and the Receiver is enabled. This bit is set to a 0 bit if the bit after the last data bit of a character is a one.
5	CARRIER DETECTER	This bit is the complement of the CARRIER DETECTOR input (pin 39).
6	DATA SET READY	This bit is the complement of the DATA SET READY input (pin 33).
7	DATA SET CHANGE	This bit is set to a 1 bit when there is a change in the state of the DATA SET READY or CARRIER DETECTOR inputs with the DATA TERMINAL READY- output (pin 17) on, or the Ring Indicator is turned on with the DATA TERMINAL READY- output off. This bit is cleared to a 0 bit when the Status Register is read onto the data bus.

5.4.2 Device Programming Mechanism: the Control Registers

Each of the serial ports can be individually programmed to operate in either synchronous mode or asynchronous mode. Asynchronous mode is the default when the Pascal Operating System is loaded; the operation of a serial port in asynchronous mode has been described in subsection 5.4.1. In synchronous mode, two registers are of interest in addition to those discussed in subsection 5.4.1:

- o SYN Register. This 8-bit register holds the synchronization code used to establish character synchronization.

- o DLE Register. This 8-bit register holds the DLE character used when the optional transparent mode is in effect.

In synchronous mode, character framing is achieved by the SYN character, transmitted at the beginning of a block of characters. When the Receiver is enabled, it searches for two continuous characters matching the bit pattern contained in the SYN Register.

The serial port devices are programmed via two 8-bit control registers: Control Register 1 and Control Register 2. Tables 5-7 and 5-8 outline the contents of these registers.

Table 5-7. Serial Port Control Register 1 Contents

Bit	Name	Function
0	DATA TERMINAL READY	This bit controls the DATA TERMINAL READY- output (pin 17), which controls the CD circuit of the attached terminal. When this bit is set to a 1 bit, the Carrier On and Data Set Ready interrupts are enabled. When this bit is set to 0, only the Ring On interrupt is enabled.

1	REQUEST TO SEND	This bit controls the REQUEST TO SEND- output (pin 32), which controls the CA circuit of the attached terminal. When this bit is set to a 1 bit and the CLEAR TO SEND- (pin 29) input is low, transmission of data to the attached terminal is enabled and Transmitter Holding Register Empty interrupts are generated. When this bit is set to a 0 bit, transmission is disabled after any current character has been transferred.
2	RECEIVER ENABLE	When set to a 1 bit, this bit enables the Receiver, allowing received characters to be placed in the Receiver Holding Register. Status bits 1, 2, and 3 are updated and a Data Received interrupt is generated. Character reception starts with a Start bit when in asynchronous mode or with 2 matches to the contents of the SYN Register when in synchronous mode.* When the RECEIVER ENABLE bit is set to a 0 bit, status bits 1, 2, 3, and 4 are cleared.
3	PARITY ENABLE	When this bit is set to a 1 bit, parity checking on received characters is enabled. In asynchronous mode,* generation of parity for transmitted characters is also enabled. When this bit is set to a 0 bit, parity checking and generation are both disabled in either mode.
4	ECHO MODE, DLE STRIP, or MISCELLANEOUS	The function of this bit depends on whether the device is operating in synchronous or asynchronous mode.* In asynchronous mode, bit 4 controls echo mode. If the Receiver is enabled and this bit is set to a 1 bit, the received data is echoed with a 1-bit delay. The clocked regenerated data, rather than the output of the Transmitter Register or a

		steady marking, is presented to the TRANSMITTED DATA output (pin 25). The Transmitter need not be enabled. In synchronous mode, the function of this bit is further dependent on whether the Receiver is enabled or not. If the Receiver is enabled and bit 4 is set to a 1 bit, received characters which match the contents of the DLE Register are stripped out and parity checking is disabled. When the Receiver is not enabled, bit 4 controls the MISCELLANEOUS- output (pin 5). When this bit is set to a 1 bit, this output is made low; when the bit is set to a 0 bit, the output is made high. When operating with a 32X clock (see Table 5-8, Bits 2-0), a 1 bit with the Receiver not enabled causes the Receiver bit timing to synchronize on mark-space transitions.
5	STOP BIT SELECTION, MISCELLANEOUS, TRANSMIT PARITY ENABLE, or FORCE DLE	The function of this bit depends on whether synchronous or asynchronous mode is in effect.* In asynchronous mode with the Transmitter enabled, bit 5 set to a 1 bit causes a single Stop bit to be transmitted; bit 5 set to a 0 bit causes 2 Stop bits to be transmitted for character lengths of 6, 7, or 8 bits, and 1.5 Stop bits for a character length of 5 bits. When the Transmitter is not enabled, bit 5 controls the MISCELLANEOUS- output (bit 5). When this bit is set to a 1 bit, this output is made low; when the bit is set to a 0 bit, the output is made high. In synchronous mode, the function of bit 5 is further dependent on bit 6. If both bits are set to a 1 bit, the contents of the DLE Register are transmitted prior to the next character loaded in the Transmitter Holding Register. When bit 5 is set to a 1 bit and

		bit 6 is set to a 0 bit, transmit parity is enabled. When bit 5 is set to a 0 bit, no parity is generated.
6	BREAK or TRANSMIT TRANSPARENT	The function of this bit depends on whether synchronous or asynchronous mode is in effect.* In asynchronous mode, when bit 6 is set to a 1 bit and the Transmitter is enabled, the TRANSMITTED DATA output (pin 25) is held in a spacing condition starting with the end of any current character. Normal Transmitter timing continues so that the break can be timed out by loading charaters into the Transmitter Holding Register. That is, interrupts are generated and the Transmitter operates normally except for the output which remains low while bit 6 is set to a 1 bit. In synchronous mode, bit 6 set to a 1 bit causes the Transmitter to be conditioned for the transparent transmission: idle fill is DLE-SYN and a DLE can be forced ahead of any character in the Transmitter Holding Register by use of bit 5.
7	LOOP/NORMAL	When this bit is set to a 0 bit, the device is configured to provide an internal data and control loop and the Ring On interrupt is disabled. When this bit is set to a 1 bit, the device is normal full duplex configuration and the Ring On interrupt is enabled.

* Synchronous/asynchronous mode is controlled by bit 5 of Control Register 2 (Table 5-8).

Table 5-8. Serial Port Control Register 2 Contents

Bit(s)	Name	Function
2-0	CLOCK SELECT	These bits select the transmit and receive clock as follows: 0 0 0 - Transmit and receive clock input (1X) 0 0 1 - Rate 1 (32X) 0 1 0 - Rate 2 (32X) 0 1 1 - Rate 3 (32X) 1 0 0 - Rate 4 (32X) 1 0 1 - Rate 4 - 2 (32X) (64X) 1 1 0 - Rate 4 - 4 (32X) (128X) 1 1 1 - Rate 4 - 8 (32X) (256X)
3	ALTERNATE RECEIVER CLOCK or STRIP SYN	The function of this bit depends on whether bit 5 selects asynchronous or synchronous mode. In asynchronous mode, if bit 3 is set to a 0 bit, Rate 1 is selected as the Receiver clock rate; if bit 3 is set to a 1 bit, the Receiver clock rate is determined by bits 2-0. (If 1X clocking is selected in bits 2-0, bit 3 must be 1.) In synchronous mode, if bit 3 is set to a 1 bit and the Receiver is enabled, received characters which match the contents of the SYN Register are stripped out and the SYN status bit is set with the next character. No SYN stripping occurs if bit 3 is set to a 0 bit.
4	PARITY ODD/EVEN	Odd parity is selected when this bit is set to a 1 bit; when it is set to a 0 bit, even parity is selected.
5	CHARACTER MODE	When this bit is set to a 1 bit, asynchronous mode is selected; when it is set to a 0 bit, synchronous mode is selected.

7-6	CHARACTER LENGTH	These bits select the number of bits per character as follows: 0 0 - 8 bits 0 1 - 7 bits 1 0 - 6 bits 1 1 - 5 bits

For example, when the Pascal Operating System is loaded, it reads the system terminal specifications hardware option switches and then loads the serial port A Control Register 2 with the corresponding values. Switches 1-8 correspond to Control Register 2 values 0-7 respectively.

The processor can load a value into either control register of either serial port, and it can read the current contents of either control register of either port. The processor can also load values into the SYN and DLE registers for use in synchronous mode; these registers cannot be read. Table 5-9 lists the values used by the processor to address the serial ports SYN, DLE, and control registers.

Table 5-9. Serial Port SYN, DLE, and Control Register Addresses

Operation	General Address	Value of "x"	Element Selected
Input from Serial Port A	FC1x	0 - 0 0	Control Register 1
		0 - 0 1	Control Register 2
Output to Serial Port A	FC1x	0 - 0 0	Control Register 1
		0 - 0 1	Control Register 2
		0 - 1 0	SYN and DLE Registers
Input from Serial Port B	FC2x	- - 0 0	Control Register 1
		- - 0 1	Control Register 2
Output to Serial Port B	FC2x	- - 0 0	Control Register 1
		- - 0 1	Control Register 2
		- - 1 0	SYN and DLE Registers

For more information of the serial port devices, see the "UC1931A/B" data sheet, available from Western Digital Corporation.

5.5 PARALLEL PORTS

The Pascal MICROENGINE Computer provides a parallel I/O capability in the form of an 8255A Programmable Peripheral Interface device, a standard 40-pin dual in-line package with 24 programmable pins. See Figure 5-1.

Communication between the parallel ports and the processor is achieved via the intra-component communication bus structure, described in section 5.1. A terminal is connected to the parallel ports via a parallel port cable connector. Table 5-10 illustrates the pin assignments of a parallel port cable connector.

This section is concerned with the operation of the parallel ports within this configuration. Subsection 5.5.1 describes the normal operation of the Pascal MICROENGINE Computer parallel ports device; that is, it explains how the device operates when it is programmed automatically by the Pascal Operating System. For the user who wishes to program the device to operate in some other manner, subsection 5.5.2 describes the mechanism for programming the device and briefly discusses the modes in which it can operate.

5.5.1 Default Operation of the Parallel Ports

When the Pascal Operating System is loaded, it automatically programs the parallel ports device to provide an 8-bit input port (Port A) for reading from the attached peripheral device, an 8-bit output port (Port B) for writing to the peripheral device, and an 8-bit control port (Port C). I/O data is transferred between the peripheral device and Ports A and B in conjunction with strobes, or "handshaking" signals. These signals are generated or accepted on Port C lines, as listed in Table 5-11. Input and output data are both latched.

Table 5-10 Parallel Port Cable Connector Pin Assignments

Pin Number	Signal	Description
J3-1	GND	
J3-2	PA1	DATA 1 PORT A
J3-3	PA0	DATA 0 PORT A
J3-4	PA4	DATA 4 PORT A
J3-5	PA6	DATA 6 PORTA
J3-6	SACKB-	ACKNOWLEDGE B
J3-7	STBA-	STROBE A
J3-8	PABIF-	OPEN- PA BUS IN FALSE
J3-9	SPC7	DATA 7 PORT C
J3-10	SOBFB-	OUTPUT BUFFER FULL B
J3-11	I/OPC6	
J3-12	IBFA	INPUT BUFFER FULL A
J3-13		
J3-14		
J3-15	PB6	DATA 6 PORT B
J3-16	PB4	DATA 4 PORT B
J3-17	PB3	DATA 3 PORT B
J3-18	PB0	DATA 0 PORT B
J3-19	RESET	
J3-20	GND	
J3-21	PA2	DATA 2 PORT A
J3-22	PA3	DATA 3 PORT A
J3-23	PA5	DATA 5 PORT A
J3-24	PA7	DATA 7 PORT A
J3-25	ACKB-	ACKNOWLEDGE B
J3-26		
J3-27	SPC6-	DATA 6 PORT C
J3-28		
J3-29	I/OPC7	
J3-30	OBFB-	OUTPUT BUFFER FULL B
J3-31	SIBFB-	INPUT BUFFER FULL B
J3-32	PBBOF	OPEN PB BUS OUT FALSE
J3-33	PB7	DATA 7 PORT B
J3-34	PB5	DATA 5 PORT B
J3-35	PB2	DATA 2 PORT B
J3-36	PB1	DATA 1 PORT B
J3-37	GND	

Table 5-11. Parallel Ports Device Control Lines (Port C)

Bit	Name	Function
0	INTRB (Interrupt Request, Port B)	A high on this output interrupts the processor when Port B is requesting service.
1	OBFB- (Output Buffer Full)	A low on this output indicates that data has been loaded into the Port B latch for output; it is, in essence, an acknowledgement.
2	ACKB- (Acknowledge Input) or INTEB- (Interrupt Enable, Port B)	On output, a high on this pin enables interrupts from Port B. On input, a low on this pin indicates that data output from Port B has been accepted by the peripheral device. When the Pascal Operating System is loaded, it makes this pin high to enable Port B interrupts.
3	INTRA (Interrupt Request, Port A)	A high on this output interrupts the processor when Port A is requesting service.
4	STBA- (Strobe Input)	A low on this input loads data into the Port A latch for input.
5	IBFA (Input Buffer Full)	A high on this output indicates that data has been loaded into the Port A latch for input; it is, in essence, an acknowledgement.
6	INTEA	A high on this output enables interrupts from Port A. When the Pascal Operating System is loaded, it makes this pin low to disable Port A interrupts.
7	--	This pin is not used.

Recall from section 5.1 that the processor initiates communication with the parallel ports by loading the general address FC7x onto the address bus, where the value of "x" determines which device element is selected. Table 5-12 lists the values used to address Ports A, B, and C.

Table 5-12. Parallel Ports Addresses

Operation	General Address	Value of "x"	Element Selected
Input from or output to parallel ports	FC7x	- - 0 0	Port A
		- - 0 1	Port B
		- - 1 0	Port C

5.5.2 Device Programming Mechanism: the Control Register

Parallel Ports A and B can be individually programmed to operate in any of three modes:

- Mode 0, the basic input/output mode. In this mode, data is simply read from or written to the port; no "handshaking" signals are required.

- Mode 1, the strobed input/output mode. This is the mode selected in the default device programming, described in subsection 5.5.1.

- Mode 2, the strobed bidirectional bus input/output mode. In this mode, data is transferred to and from a peripheral device on a single 8-bit bus. Handshaking signals are provided to maintain proper bus flow discipline in a manner similar to Mode 1.

Port C is programmed in two sections: the upper half (bits 0-3) and the lower half (bits 4-7). The mode of the upper half of Port C is defined with the mode of Port A; the mode of the lower half is defined with the mode of Port B.

The parallel ports device is programmed via the 8-bit Control Register. Table 5-13 outlines the contents of this register.

Table 5-13. Parallel Ports Control Register Contents

Bit	Name	Function
0	LOWER PORT C DIRECTION	When this bit is set to a 1 bit, the lines on the lower half of Port C are used as input lines. When this bit is set to a 0 bit, these lines are used as output lines.
1	PORT B DIRECTION	This bit, when set to a 1 bit, defines Port B as an input port. When this bit is set to a 0 bit, Port B is used as an output port.
2	PORT B AND LOWER PORT C MODE	When this bit is set to a 1 bit, Mode 1 is selected for Port B and the lower half of Port C. When this bit is set to a 0 bit, Mode 0 is selected.
3	UPPER PORT C DIRECTION	When this bit is set to a 1 bit, the lines on the upper half of Port C are used as input lines; when this bit is set to a 0 bit, these lines are used as output lines.
4	PORT A DIRECTION	When this bit is set to a 1 bit, Port A is used as an input port. When this bit is set to a 0 bit, Port A is used as an output port.
5,6	PORT A AND UPPER PORT C MODE	These bits select the mode for Port A and the upper half of Port C as follows: 0 0 Mode 0 0 1 Mode 1 1 - Mode 2

7	MODE SET FLAG	This bit controls the mode set flag. When it is set to a 1 bit, this flag is active. When it is set to a 0 bit, this flag is inactive. (This flag is normally inactive only while modes are being changed.)

For example, the value automatically loaded into the Control Register by the Pascal Operating System is shown in Figure 5-2.

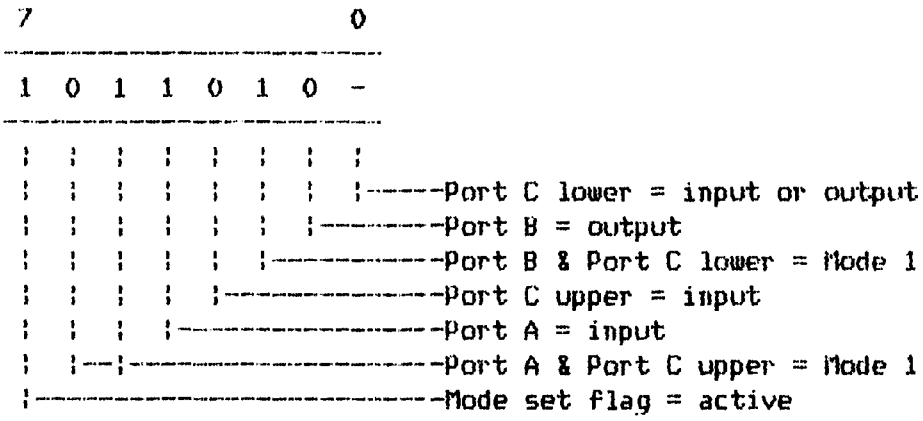


Figure 5-2. Default Parallel Port Control Register Values

To load a value into the Control Register, the processor uses the parallel port address, FC7x, with

- - 1 1

as the value of "x."

5.6 FLOPPY DISK CONTROLLER AND DMA CONTROLLER

The FD1791/2 Floppy Disk Controller (FDC) provided with the Pascal MICROENGINE Computer is switch-selectable for single- or double-density 8-inch or 5-1/4-inch disks and can handle up to four disk drives of the same type. The Floppy Disk Controller operates under control of the DM1883A/B Direct Memory Access Controller (DMAC), allowing data transfers between the floppy disk and memory without involving the processor.

Both the FDC and the DMAC are standard 40-pin dual in-line packages, located on the board as shown in Figure 5-1.

Communication between these controllers and the processor is achieved via the inter-component communication bus structure, described in section 5.1. A floppy disk drive is connected to the floppy disk controller via a Floppy Disk Controller cable connector. Table 5-14 lists the pin assignments of the FDC cable connector.

This section is concerned with the operation of the Floppy Disk Controller and the DMA Controller within this configuration. Subsection 5.6.1 describes automatic system start-up procedures that involve the DMAC and the FDC. In subsection 5.6.2, the mechanism by which the processor initiates communication with these controllers is discussed. Subsections 5.6.3 and 5.6.4 provide information on the operation of the DMAC and FDC, respectively.

Table 5-14. Floppy Disk Cable Connector Pin Assignments

Pin Number	Signal	Description
J4-1	GND	
J4-2	DSI	UNIT SELECT 1
J4-3	DS3	UNIT SELECT 2
J4-4	SDSEL	SIDE SELECT
J4-5	GND	
J4-6	STEP-	STEP
J4-7	GND	
J4-8	WD	WRITE DATA
J4-9		
J4-10	WPRT-	WRITE PROTECT
J4-11	GND	
J4-12	TRK0-	TRACK 00
J4-13	GND	
J4-14	RDTA	RAW READ DATA
J4-15	GND	
J4-16	READY	DRIVE READY
J4-17	GND	
J4-18		
J4-19	WF	WRITE FAULT
J4-20	GND	
J4-21	DS2	UNIT SELECT 2
J4-22	DS4	UNIT SELECT 4
J4-23	HLD-	HEAD LOAD
J4-24	GND	
J4-25	DIRC	
J4-26	GND	
J4-27	WG-	WRITE GATE
J4-28	GND	
J4-29	TG43	TRACK 43
J4-30	GND	
J4-31	IP	INDEX PULSE
J4-32	GND	
J4-33	RCLK	READ CLOCK
J4-34	GND	
J4-35	+5	
J4-36	+5	
J4-37	HLT	HEAD LOAD TIMING

5.6.1 Start-Up Procedures Involving the DMAC and FDC

When the Pascal MICROENGINE Computer is started, the settings of the Floppy Disk Controller hardware option switches take effect. (Recall from subsection 2.1.2 that the user sets these switches according to the characteristics of the floppy disk or disks to be loaded.)

- If the switch settings specify double-density operation, the DDEN- input (pin 37) of the FDC is made low. If single-density operation is indicated, DDEN- is made high.

- If the switch settings call for an 8-inch floppy disk, a 2 MHz clock signal is supplied to the CLK input (pin 24) of the FDC. If 5-1/4-inch floppy disks are indicated, a 1 MHz clock signal is supplied to the CLK input.

Also at system start-up, the AUTLD input (pin 37) of the DMAC is made high. When a floppy disk is loaded into a drive, this input causes an automatic boot loading of the first 64K words from the floppy disk into memory, starting at memory location 0. A high on the AUTLD input also places the DMAC in run mode and enables two of the device's three interrupt conditions. (See subsection 5.6.3.)

5.6.2 Processor-Initiated Communication with the DMAC and FDC

Recall from section 5.1 that the processor initiates communication with the DMAC or the FDC by loading the general device address FC3x onto the address bus. This address is transmitted to the DMAC, which tests bit 3 of the "x" field of the address. If this bit is high, one of the DMAC registers is selected, according to the value of bits 0-2, and the DMAC is prepared to respond to a Read or Write operation. If bit 3 is low, one of the FDC registers is selected. In this case, the DMAC is not prepared to respond to a Read or Write operation. Instead, the DMAC signals the FDC and the address is transmitted to the FDC. As when the DMAC is selected, the particular register of the FDC is selected according to the value of bits 0-2 of the device address "x" field.

Table 5-15 lists the values used by the processor to address the DMAC and the FDC. Information on the individual DMAC registers and FDC registers is provided in subsections 5.6.3 and 5.6.4.

Table 5-15. DMAC/FDC Addresses

Operation	General Address	Value of "x"	Element Selected
Autoload from floppy disk	FC5x	- - - -	DMAC AUTLD input
Input from FDC	FC3x	0 - 0 0	Status Register
		0 - 0 1	Track Register
		0 - 1 0	Sector Register
		0 - 1 1	Data Register
Output to FDC	FC3x	0 - 0 0	Command Register
		0 - 0 1	Track Register
		0 - 1 0	Sector Register
		0 - 1 1	Data Register
Input from or output to DMAC	FC3x	1 0 0 0	Control Register
		1 0 0 1	Status Register
		1 0 1 0	Transfer Count Low Register
		1 0 1 1	Transfer Count High Register
		1 1 0 0	Memory Address Low Register
		1 1 0 1	Memory Address High Register
		1 1 1 0	Memory Address Ext Register
		1 1 1 1	Interrupt Code Register

When the processor addresses the Floppy Disk Controller, it identifies the particular drive to be accessed and the side of the floppy disk to be accessed. This information is provided in the upper byte of each command word loaded onto the data bus after communication is established. Figure 5-3 diagrams the contents of this word.

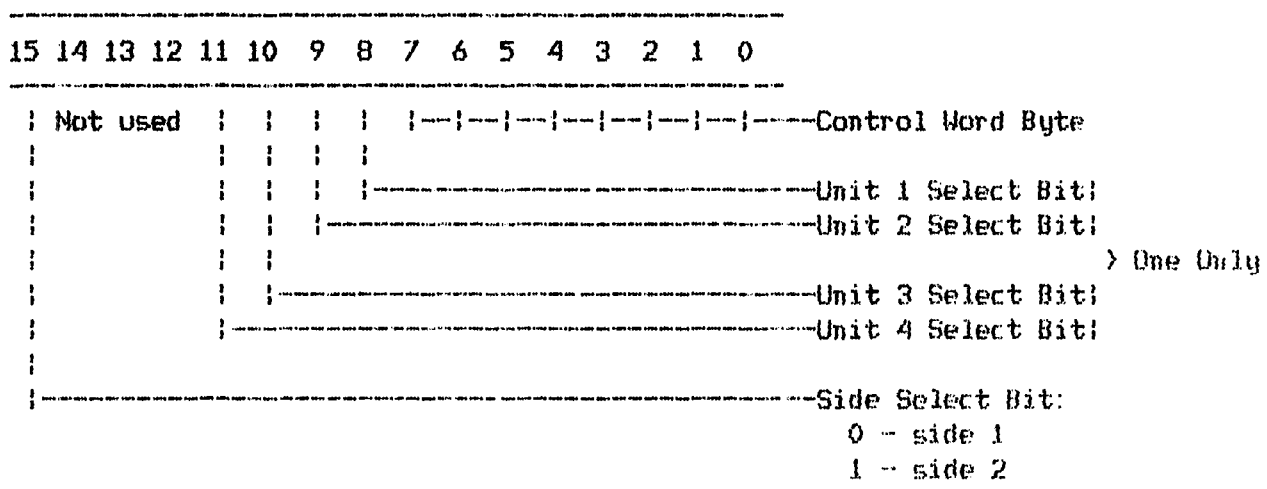


Figure 5-3. Format of First Data Word Transmitted to the FDC

5.6.3 DMA Controller Organization and Operation

The DMAC includes several registers. Of primary interest are:

- Transfer Count Register. This 16-bit counter register holds the two's complement of the block size -- i.e., the transfer count (in words) -- for DMA transfer operations. The low-order 8 bits are in TC Low and the high-order 8 bits are in TC High. This count is incremented after each DMA transfer.

- Memory Address Register. This 18-bit register occupies 3 DMA registers: bits 0-7 are held in MA Low, bits 8-15 are held in MA High, and bits 16 and 17 are held in MA Ext. The carry from bit 15 to 16 is enabled if and only if bit 6 of the Control Register is set to a 1 bit. (See Table 5-16.) The Memory Address Register is incremented by two after every DMA transfer and bit 0 is forced to 0.

- ID Code Register. This 8-bit programmable register contains a code for establishing a vector address during a DMAC interrupt. This register is not used.

- Control Register. This 8-bit register holds device programming specifications. The contents of this register are outlined in Table 5-16.

- Status Register. This 8-bit register contains 3 interrupt condition indicator bits and 5 bits which reflect the status of the Control Register. The contents of the Status Register are outlined in Table 5-17.

Table 5-16. DMAC Control Register Contents

Bit	Name	Function
0	RUN	When this bit is set to a 1 bit, the DMAC is placed in run mode; when it is set to a 0 bit, DMAC operation is terminated.
1	DEVICE INTERRUPT	This bit determines whether the device interrupt is enabled or not. When this bit is set to a 1 bit, a high on the DINTR input (pin 39) sets the INTR output (pin 40) low.
2	TIME-OUT INTERRUPT	This bit controls the time-out interrupt. When this bit is set to a 1 bit, the time-out one-shot sets the INTR output (pin 40) low. When this bit is set to a 0 bit, this interrupt is disabled. The time-out interrupt is set during a DMA transfer if REPLY- (pin 3) does not go low within 5 microseconds of MSYNC- (pin 16) going low.
3	TRANSFER COUNT ZERO INTERRUPT	This bit controls whether the transfer count equals zero interrupt is enabled or not. When this bit is set to a 1 bit, a zero in the Transfer Count Register sets the INTR- output (pin 40) low. When this bit is set to a 0 bit, this interrupt is disabled.
4	INPUT/OUTPUT MODE	This bit controls the I/O direction of the device. When set to a 1 bit, this bit sets Read mode; i.e., data is transferred from the Floppy Disk Controller to memory. When set to a 0 bit, this bit sets Write mode; i.e., data is transferred from memory to the Floppy Disk Controller.

5	HOLD BUS	When this bit is set to a 1 bit, the DMAC retains control of the data bus throughout the transfer of the entire block. When this bit is set to a 0 bit, the DMAC releases the data bus after each word transfer.
6	ADDRESS EXTENSION CARRY ENABLE	When set to a 1 bit, this bit allows a carry from DMA address bit 15 to propogate into bit 16. When this bit is set to a 0 bit, address extension carry is disabled.
7		This bit is not used.

Table 5-17. DMAC Status Register Contents

Bit	Name	Function
0	BYTE OR WORD	This bit indicates the status of the BOW input (pin 36). This bit is always set to a 0 bit, indicating that word mode is in effect. In word mode, the DMA memory address is incremented by two (bit 0 is forced to a 0 bit) after each DMA transfer. Bit 0 is a read-only bit.

1	DEVICE INTERRUPT	This bit is set to a 1 bit when a device interrupt condition occurs. Resetting this bit to a 0 bit resets the INTR- output (pin 40). Bit 1 is a read/write bit.
2	TIME-OUT INTERRUPT	This bit is set to a 1 bit when a time-out interrupt condition occurs. Resetting this bit to a 0 bit resets the INTR- output (pin 40). Bit 2 is a read/write bit.
3	TRANSFER COUNT ZERO INTERRUPT	This bit is set to a 1 bit when a transfer count equals zero interrupt condition occurs. When set, this bit sets the EOB output (pin 38). This bit is set to 0 when the Transfer Count Register is loaded with a non-zero value. Bit 3 is a read-only bit.
4	INPUT/OUTPUT MODE	This bit reflects the status of bit 4 in the Control Register. When this bit is set to a 1 bit, the DMAC operates in Read mode; when it is set to a 0, the DMAC operates in Write mode. Bit 4 is a read-only bit.
5	HOLD BUS	This bit reflects the status of bit 5 in the Control Register. When this bit is set to a 1 bit, the DMAC retains control of the data bus throughout an entire block transfer. When this bit is set to a 0 bit, the DMAC releases the bus after each word transfer. Bit 5 is a read-only bit.
6	ADDRESS EXTENSION CARRY ENABLE	This bit reflects the status of bit 6 in the Control Register. When this bit is set to a 1 bit, a carry is allowed from DMA address bit 15 to propagate into bit 16. When this bit is set to a 0 bit, address extension carry is disabled. Bit 6 is a read-only bit.
7	BUSY	This bit reflects the status of bit 0 of the Control Register. When this bit is set to a 1 bit, the DMAC is in run mode. When this bit is set to a 0 bit, DMAC operation is terminated. In the Status Register, BUSY is a read-only bit.

When the DMAC is in run mode, it waits for a Data Request input from the FDC. When a request is received, the DMAC requests control of the data bus from the processor. Once this request is granted, the DMAC controls data transfers between the FDC and memory. The direction of the transfer is determined by the status of the R/W output (pin 17), which is tied directly to bit 4 of the Control Register.

Three interrupt conditions can occur during DMAC operation:

- Device Interrupt. This condition occurs when the DINTR input (pin 39) is made high. A device interrupt occurs when the device requires service, when a failure occurs, when a task is completed, or in other situations.

- Transfer Count Equals Zero. This condition occurs when the Transfer Count Register is incremented to 0.

- Time-Out Interrupt. During a DMA transfer, the leading edge of the MSYNC- output (pin 16) triggers an internal time delay of about 5 microseconds. If the DMAC does not receive an active low REPLY- input (pin 3) within that time delay, a time-out interrupt condition occurs.

If any of these conditions occurs, the corresponding bit in the Status Register is set and the Status Register RUN bit is reset. If the appropriate enable bit in the Control Register is set, the INTR- output (pin 40) is made active.

For more information on the DM1883 device, see the "DM1883A/B Direct Memory Access Controller" data sheet, available from Western Digital Corporation.

5.6.4 Floppy Disk Controller Organization and Operation

The Floppy Disk Controller includes several registers. Of primary interest are:

- Data Shift Register. This 8-bit register assembles serial data from the RAW READ- input (pin 27) during Read operations and transfers serial data to the WRITE DATA output (pin 31) during Write operations.

- Data Register. This 8-bit register is used as a holding register during Disk Read and Write operations. In Disk Read operations, the assembled data byte is transferred in parallel to the Data Register from the Data Shift Register. In Disk Write operations, information is transferred in parallel from the Data Register to the Data Shift Register. In a Seek operation, the Data Register holds the address of the desired Track position. (The Disk Read, Disk Write, and Seek operations are described later in this subsection.)

- Track Register. This 8-bit register holds the track number of the current Read/Write head position. This register is incremented by 1 each time the head is stepped in (toward track 76) and decremented by 1 each time the head is stepped out (toward track 0).

- Sector Register. This 8-bit register holds the address of the desired sector position.

- Command Register. This 8-bit register holds the command currently in execution.

- Status Register. This 8-bit register holds device status information.

- CRC Logic Register. This logic is used to check or generate the 16-bit Cyclic Redundancy Check (CRC). The polynomial is:

$$F(x) = x^{16} + x^{12} + x^{5} + 1$$

  The CRC includes all information starting with the address mark up to the CRC characters.

When data is read from a floppy disk, it passes through the Data Separator, a counter separator comprised of two integrated circuits. The Data Separator is located on the board as shown in Figure 5-1.

The Data Separator logic serves to derive the flux transition spacings from the raw data; this rate is then supplied to the RCLK- input (pin 26). The raw

data is supplied to to the RAW READ- input (pin 27). Essentially, any transition on RAW READ- loads the Data Separator counter. When this counter reaches 0, the RCLK flip/flop is toggled.

The Floppy Disk Controller accepts 11 commands, grouped into 4 types. The following paragraphs describe the command types and the individual commands within each type.

Type I commands include the Restore, Seek, Step, Step In, and Step Out commands. Each of these commands includes a rate field (bits 0 and 1) which, in conjunction with the frequency of the CLK input, defines the stepping motor rate, as outlined in Table 5-18.

Table 5-18. Stepping Motor Rates for FDC Type I Commands

CLK Frequency	Rate Field Value	Rate Selected
2 MHz	0 0	3 ms
	0 1	6 ms
	1 0	12 ms
	1 1	15 ms
1 MHz	0 0	6 ms
	0 1	12 ms
	1 0	20 ms
	1 1	30 ms

Bit 2 of all Type I commands is a verification flag. If this bit is set to a 1 bit, a verification operation is performed on the destination track. If this bit is set to a 0 bit, no verification is performed.

The Type I commands also contain a head load flag in bit 3. If this bit is set to a 1 bit, the Read/Write head is loaded - - i.e., the HLD output (pin 28) is made active -- at the beginning of the command. If bit 3 is set to a 0 bit, HLD is deactivated at the beginning of the command. Once the head is loaded, it remains engaged until the FDC receives a command which specifically disengages the head. Or, if the FDC is busy for 15 revolutions of the disk, the head is disengaged automatically.

The Step, Step In, and Step Out commands include an update flag in bit 4. When this bit is set to a 1 bit, the Track Register is updated by 1 for each step. When this bit is set to a 0 bit, the Track Register is not updated.

The Type I commands are:

- Restore. If the Read-Write head is positioned over track 0 when this command is received, the Track Register is loaded with zeros and an interrupt is generated. Otherwise, stepping pulses at the

specified rate (see Table 5-17) are issued until the head is located over track 0.

- o Seek. This command assumes that the Track Register contains the track number of the Read-Write head's current position and that the Data Register contains the desired track number. Stepping pulses in the appropriate direction are issued until the contents of the Track Register are equal to the contents of the Data Register.

- o Step. This command causes the FDC to issue one stepping pulse to the disk drive in the same direction as the previous Step, Step In, or Step Out command. (At least one Step In or Step Out command must be executed before a Step command is received.)

- o Step In. This command causes the FDC to issue one stepping pulse in the direction toward track 76.

- o Step Out. This command causes the FDC to issue one stepping pulse in the direction toward track 0.

Table 5-19 summarizes the Command Register values for Type I commands.

Table 5-19. FDC Command Register Values for Type I Commands

Command	Command Register Value
Restore	0 0 0 0 h v r1 r2
Seek	0 0 0 1 h v r1 r2
Step	0 0 1 u h v r1 r2
Step In	0 1 0 u h v r1 r2
Step Out	0 1 1 u h v r1 r2

where:
- h - head load flag:
  - 1 - load head at beginning of command
  - 0 - unload head at beginning of command
- u - update flag:
  - 1 - update Track Register
  - 0 - do not update Track Register
- v - verification flag:
  - 1 - verify on last track
  - 0 - do not verify
- r1 and r2 - stepping motor rate (see Table 5-18)

Table 5-20 lists the Status Register contents when a Type I command is executed.

Table 5-20. FDC Status Register Contents for Type I Commands

Bit	Name	Meaning
0	BUSY	When set to a 1 bit, this bit indicates that command execution is in progress. When no command execution is in progress, this bit is set to a 0 bit.
1	INDEX	This bit is set to a 1 bit when an index mark is detected on the drive; otherwise, this bit is set to a 0 bit. This bit is an inverted copy of the IP input (pin 35).
2	TRACK 00	This bit is set to a 1 bit when the Read-Write head is positioned to Track 00; otherwise, this bit is set to a 0 bit. This bit is an inverted copy of the TR00 input (bit 34).
3	CRC ERROR	This bit is set to a 1 bit when CRC characters are encountered in an ID field; otherwise, this bit is set to a 0 bit.
4	SEEK ERROR	This bit is set to a 1 bit when the desired track was not verified; otherwise, this bit is set to a 0 bit.
5	HEAD LOADED	This bit is set to a 1 bit when the Read-Write head is loaded and engaged; otherwise, this bit is set to a 0 bit. This bit is the logical AND of the HLD output (pin 28) and the HLT input (pin 23).
6	PROTECTED	This bit is set to a 1 bit when Write Protect is activated; otherwise, it is set to a 0 bit. This bit is an inverted copy of the WRPT- input (pin 36).

7	NOT READY	This bit is set to a 1 bit if the drive is not ready. If the drive is ready, this bit is set to a 0 bit. This bit is an inverted copy of the READY input (pin 32), logically ORed with the MR- input (pin 19).

The Type II commands are Read Sector and Write Sector. Prior to loading a Type II command into the Command Register, the processor loads the desired sector number into the Sector Register. When a Type II command is received and an ID field is located on the disk, the track number of the ID field is compared with the Track Register. If they do not match, the next ID field is read. If there is a match, the sector number of the ID field is compared with the Sector Register. If they do not match, the next ID field is read. If there is a match, the CRC of the ID field is tested. If the CRC is incorrect, the next ID field is read. If the CRC is correct, the data field is then located and its contents are either read or written. The FDC must find an ID field with the correct track number, sector number, and CRC within four revolutions of the disk.

The Type II commands each provide flags for requesting a 15-microsecond delay before the Read-Write head is loaded against the medium, and for specifying that multiple sectors are to be read or written. The Write Sector command also provides a flag for controlling the writing of the data address mark.

The Type II commands are:

- Read Sector. This command causes the data field associated with the selected ID field (correct track number, sector number, and CRC) to be presented to the processor.

- Write Sector. This command causes the FDC to write data bytes from the Data Shift Register to the data field of the selected sector (correct track number, sector number, and CRC).

The Command Register values for Type II commands are listed in Table 5-21.

Table 5-21. FDC Command Register Values for Type II Commands

Command	Command Register Value
Read Sector	1 0 0 m - e 0 0
Write Sector	1 0 1 m - e - a
	where: m - multiple record flag: 1 - multiple records 0 - single sector e - head loading delay flag: 1 - 15-ms delay 0 - no delay a - data address mark flag: 1 - F8 (Deleted Data Mark) 0 - FB (Data Mark)

The contents of the Status Register have the same meanings on a Type II command as on a Type III command. The Status Register contents for Type II and III commands are described in Table 5-23, presented after the discussion of the Type III commands.

The Type III commands are:

- Read Address. This command causes the 6 bytes of the next encountered ID field to be assembled and transferred to the Data Register. The ID field bytes are: track address, side number, sector address, sector length, CRC 1, and CRC 2.

- Read Track. This command causes reading to begin with the leading edge of the next encountered index mark and to continue until the next index pulse. As each byte is assembled, it is transferred to to the Data Register.

- Write Track. This command causes bytes from the Data Register to be written to the disk, starting with the leading edge of the next encountered index pulse and continuing until the next index pulse.

Table 5-22 lists the Command Register values for Type III commands.

When a Type II or Type III command is executed, the contents of the Status Register are as listed in Table 5-23.

Table 5-22. FDC Command Register Values for Type III Commands

Command	Command Register Value
Read Address	1 1 0 0 0 1 0 0
Read Track	1 1 1 0 0 1 0 -
Write Track	1 1 1 1 0 1 0 0

Table 5-23. FDC Status Register Contents for Type II and Type III Commands

Bit	Name	Meaning
0	BUSY	When set to a 1 bit, this bit indicates that a command is under execution. If no command is under execution, this bit is set to a 0 bit.
1	DATA REQUEST	This bit is a copy of the DRQ output (pin 38). When set to a 1 bit, it indicates that the Data Register is full on a Read operation or empty on a Write operation. Otherwise, this bit is set to a 0 bit.
2	LOST DATA	This bit is set to a 1 bit if the processor does not respond to a DRQ signal (pin 38) in one byte time. Otherwise, this bit is set to a 0 bit.
3	CRC ERROR	This bit is set to a 1 bit if an error is found in one or more ID fields or in the data field. Otherwise, this bit is set to a 0 bit.
5	RECORD TYPE/ WRITE FAULT	On a Read Sector command, this bit indicates the record-type code from the data field address mark. On any Write command, this bit is set to a 1 bit if a write fault occurs. This bit is not used by the Read Track command.

6	WRITE PROTECT	On any Write command, this bit is set to a 1 bit if Write Protect is active; otherwise, it is set to a 0 bit. This bit is not used by the Read Sector or Read Track command.
7	NOT READY	This bit is set to a 1 bit if the drive is not ready. If the drive is ready, this bit is set to a 0 bit. This bit is an inverted copy of the READY input (pin 32), logically ORed with the MR input (pin 19).

Type IV consists of a single command: the Force Interrupt command. If another command is under execution when the Force Interrupt command is loaded into the Command Register, that command is terminated. Depending on the value of bits 0-3 of the Command Register, an interrupt may be generated. Table 5-24 lists the Command Register values for the Force Interrupt Command.

Table 5-24. FDC Command Register Values for the Force Interrupt Command (Type IV)

Command	Command Register Value
Force Interrupt, no interrupt generated	1 1 0 1 0 0 0 0
Force Interrupt, Not-Ready to Ready Transition interrupt generated	1 1 0 1 0 0 0 1
Force Interrupt, Ready to Not-Ready Transition interrupt generated	1 1 0 1 0 0 1 0
Force Interrupt, Index Pulse interrupt generated	1 1 0 1 0 1 0 0
Force Interrupt, Immediate Interrupt generated	1 1 0 1 1 0 0 0

If a Force Interrupt command is received while another command is under execution, the Status Register BUSY bit (bit 0) is reset and the rest of the Status Register bits are unchanged. If a Force Interrupt command is received while no other command is being executed, the BUSY bit is reset and the rest of the Status Register bits are updated or cleared. In this case, the Status Register bits have the same meanings as on a Type I command, as outlined in Table 5-20.

For more information on the FD1791 Floppy Disk Controller, see the "FD1791/2 Floppy Disk Controller" data sheet, available from Western Digital Corporation.

THE MICROENGINE COMPANY

Pascal MICROENGINE™ Computer

Pascal Operations Manual

Preliminary Edition

March, 1979

Software is provided on a licensed basis only and is the property of the University of California with permission granted for use on an individual system basis only. Copies may be made for archival purposes only.

Information furnished by The MICROENGINE Company is believed to be accurate and reliable. However, no responsibility is assumed by The MICROENGINE Company for its use; nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of The MICROENGINE Company. The MICROENGINE Company reserves the right to change product specifications at any time without notice.

Copyright (c) 1979 by The MICROENGINE Company

First Printed March 1979

All rights reserved.

No part of this document may be reproduced by any means, nor translated, nor transmitted into a machine language without the written permission of The MICROENGINE Company.

MICROENGINE is a Registered Trademark of Western Digital Corporation.

The MICROENGINE Company is a wholly-owned subsidiary of Western Digital Corporation.

PREFACE

This manual is organized primarily for reference purposes. The first section gives a brief overview of the Pascal Operating System. Section 2 gives details, including commands, of each of the separate parts of the system. Section 3 deals with Pascal programming considerations necessary to function within the system. Section 4 describes the utilities available. Appendices are for quick reference capability.

*****

This manual was prepared and edited using the UCSD Pascal Screen Oriented Editor and was printed using a PRINTRONIX Model P300 lineprinter. The lineprinter was provided by PRINTRONIX Corporation, Irvine, California.

WESTERN DIGITAL

PASCAL OPERATIONS MANUAL

OUTLINE

Page

Section 1 INTRODUCTION....1

Section 2 PASCAL OPERATING SYSTEM....3
- 2.1 The Operating System Hierarchy....3
- 2.2 Outer Level Commands....4
- 2.4 Screen-Oriented Editor....6
  - 2.4.1 Moving Commands....6
  - 2.4.2 Text-Changing Commands....8
  - 2.4.3 Formatting Commands....15
  - 2.4.4 Miscellaneous Commands....16
  - 2.4.5 L2 Editor....19
- 2.5 Yet Another Line-Oriented Editor (YALOE)....22
  - 2.5.1 Special Key Commands....23
  - 2.5.2 Input/Output Commands....24
  - 2.5.3 Moving Commands....26
  - 2.5.4 Text-Changing Commands....28
  - 2.5.5 Miscellaneous Commands....30
- 2.6 File Handler....32
  - 2.6.1 "F" Prompt Line Commands....33
  - 2.6.2 "?" Prompt Line Commands....39
- 2.7 Pascal Compiler....43
  - 2.7.1 Compiler Option Syntax....45
  - 2.7.2 Compiler Options....45
- 2.8 Basic Compiler....49
  - 2.8.1 Features of UCSD Basic....49
  - 2.8.2 UCSD Basic Enhancements....53
- 2.9 Linker....53
  - 2.9.1 Using the Linker....54
  - 2.9.2 Linker Conventions and Implementation....55
- 2.10 Debugger....57
  - 2.10.1 Debugger Modes....57
  - 2.10.2 Commands....59

Section 3 PASCAL PROGRAMMING CONSIDERATIONS....63
- 3.1 Intrinsics....63
  - 3.1.1 Character Array Manipulation Intrinsics....64
  - 3.1.2 I/O Intrinsics....66
  - 3.1.3 String Intrinsics....69
  - 3.1.4 Miscellaneous Intrinsic Routines....71
- 3.2 Files....72
  - 3.2.1 Text Files....72
  - 3.2.2 Code Files....72
  - 3.2.3 Data Files....73
  - 3.2.4 Foto Files....73

3.2.5 Bad Files....73
3.2.6 Work Files....73
3.2.7 Volumes....73
3.2.8 File Names....75
3.3 Segments....75
3.4 Linkages....76
3.4.1 Pascal to Pascal Linkages (Units)....76
3.5 Long Integers....79
3.6 UCSD Pascal Enhancements....80
3.6.1 Case Statements....80
3.6.2 Comments....81
3.6.3 Dynamic Memory Allocation....81
3.6.4 EOF(F)....81
3.6.5 EOLN(f)....82
3.6.6 Files....82
3.6.7 GOTO and EXIT Statements....84
3.6.8 Packed Variables....84
3.6.9 Parametric Procedures and Functions....87
3.6.10 Program Headings....87
3.6.11 READ and READLN....87
3.6.12 RESET(f)....88
3.6.13 REWRITE(f)....88
3.6.14 Segment Procedures....88
3.6.15 Sets....89
3.6.16 Strings....89
3.6.17 WRITE and WRITELN....90
3.6.18 Implementation Size Limits....90
3.6.19 Extended Comparisons....91

Section 4 UTILITY PROGRAMS....93
4.1 Setup....93
4.1.1 Miscellaneous Information....93
4.1.2 General Terminal Information....93
4.1.3 Control Key Information....94
4.1.4 Video Screen Control Characters....96
4.2 Bootstrap Copier....97
4.3 Duplicate Directory Utilities....97
4.4 Librarian....98
4.5 Library Map....99
4.6 P-Code Disassembler....99
4.6.1 P-Code Disassembler....99
4.6.2 Data Segment Reference Statistics....100
4.6.3 Opcode, Procedure Call and Jump Statistics....102
4.7 Patch/Dump....102
4.7.1 Console Mode....102
4.7.2 Patchwork Mode....103
4.7.3 Wholewriter Mode....104
4.8 Calculator....104
4.9 GOTOXY Procedure Binder....106

APPENDICES

Appendix A Command Summaries....107
A.1 Outer Level....107
A.2 Screen Oriented Editor....107
A.3 YALOE....108
A.4 File Handler....109
A.5 Pascal Compiler....110
A.6 Debugger....111

Appendix B Tables....113
B.1 Runtime Errors....113
B.2 Syntax Errors Version 3....114
B.3 I/O Results....117
B.4 Unit Numbers....117
B.5 P-Machine Op Codes Version 3....118
B.6 Assembler Syntax Errors....127
B.7 ASCII Codes....129

Appendix C UCSD Pascal Syntax Diagrams....131

Appendix D Glossary....135

ILLUSTRATIONS

2-1 Operating System Overview....3
2-2 Example of Auto-Indent....9
2-3 Example of Delete Command....10
2-4 Example of Exchange....12
2-5 Example of Replace....15
2-6 Example of Margin....16
2-7 Examples of Moving Commands....28
2-8 Examples of Text-Changing Commands....29
2-9 Example of Macro Command....32
2-10 File Specification....33
2-11 Example of Compiler Display....45
2-12 Example of Compiled Program Source Listing....57
2-13 Example of Entering Examine Mode....58
2-14 Using Debugger Commands to Debug a Program....60

3-1 Examples of Character Array Manipulation Intrinsics....65
3-2 Examples of String Intrinsics....70
3-3 I/O Devices....75
3-4 Syntax for a Unit Definition....78
3-5 Example Uses of Long Integers....80
3-6 Example of Fallthrough in a Case Statement....81
3-7 Using Mark and Release to Change Heap Size....82
3-8 Example of Using Untyped Files....84
3-9 Example of using the Exit Statement....85
3-10 Examples of Packed Arrays and Records....86
3-11 Examples of Set Comparisons....89

4-1 Disassembly Example....101
4-1 Calculator Examples....105

SECTION 1

INTRODUCTION

The Pascal Operating System described in this document is designed to run on the MICROENGINEtm computer. For optimum use of the system, a CRT is recommended for use as the operator's console. The Pascal language used in the system is the modification of Standard Pascal developed at the University of California, San Diego (UCSD Pascal).

The Operating System consists of a number of modules, each designed for a particular program control purpose. These modules are accessed by commands that are displayed in a prompt line on the operator's console. As each module is accessed, it generates the display of a prompt line giving the commands that are to be used in it. The modules are:

- Editor (either for a CRT or line-oriented console)
- File Handler
- Pascal Compiler
- Basic Compiler
- Linker
- Debugger

Basic to the design of the system is the "work file", which is an area used for program development. Only one work file area is provided by the system. If another is needed, the contents of the work file may be saved under a separate file name for later reference (see Section 2.6, File Handler, the S(ave command). However, any number of files can be retrieved and combined into one work file.

This document was written with the assumption that the user is already familiar with the Pascal language and using computer systems. The following documents will provide supplemental information on both Pascal and this system:

- Pascal User Manual and Report (Standard Pascal)
  Kathleen Jensen and Niklaus Wirth
  Springer-Verlag, New York, (c)1975
- Microcomputer Problem Solving Using Pascal
  Kenneth L. Bowles
  Springer-Verlag, New York, (c)1977
- Pascal MICROENGINE Computer User's Manual

SECTION 2

PASCAL OPERATING SYSTEM

2.1 THE OPERATING SYSTEM HEIRARCHY

The Pascal Operating System has what can be described as a tree-like structure. The root of the tree corresponds with the outer level. Commands from the root reach out to activate the leaves, or lower levels of the system. Figure 2-1 diagrams this tree-like heirarchy.

OPERATING SYSTEM OVERVIEW

```
Outer Commands: E(dit, F(ile, R(un, C(omp, L(ink, X(ecute, D(ebug
¦
¦
--Edit: A(djust, C(py, D(lete, F(ind, I(nsrt, J(mp, R(place, Q(uit, X(chng, Z(ap
¦
--Filer: G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans, D(ate, Q(uit
¦
--Run: Invokes the Compiler, Linker and executes the program.
¦
--Compile: Translates Pascal source programs into P-code. Options are placed
¦          in program text.
¦
--Link: Links the system library to translated program.
¦
--Execute: Executes compiled and linked programs.
¦
--Debug: Interactive program which aids in debugging executing programs.
```

The UCSD Pascal Operating System is a single user program development system which aids in developing and executing user application programs.

Figure 2-1. Operating System Overview

The lower levels of the system are accessed by outer level commands. A command structure within each level enables the user to direct its execution. The commands are listed in a prompt line. The prompt line for the outer level commands appears after Autoload, after execution of a lower level, and following the entry of some lower level commands, such as Q(uit in the File Handler. Within each of the lower levels, infrequently used commands may not appear on the prompt line if there is insufficient room on the line.

The lower levels of the Pascal Operating System are:

- Screen-Oriented Editor - An editor specifically designed for use with video display terminals to insert or delete text in the work file or any text file.

- Line-Oriented Editor (YALOE) - An editor specifically designed for use with teleprinters to insert or delete text in the work file or any text file.

- File Handler - Used primarily for maintenance of files stored on the disk.

- Pascal Compiler - A one-pass compiler used to compile programs.

- Basic Compiler - Used to compile programs.

- Linker - Used to combine precompiled files, written in Pascal .

- Debugger - Used to facilitate debugging.

Normally, an installation will have just one of the editors and one of the compilers, depending on system configuration and need.

Other functions are available as utilities. These include the desk calculator, the patch/dump utility, the terminal configuration setup program and a bootstrap mover.

2.2 OUTER LEVEL COMMANDS

The prompt line for the outer level commands appears automatically after Autoload and after any of the lower levels have completed execution. The prompt line offers a command for each of the other levels of the operating system. The format of the prompt line is:

Command: E(dit, R(un, F(iler, C(omp, L(ink, X(ecute, D(ebug

The operator invokes the execution of the individual commands by entering the capitalized character on the operator's console. The function of each command is as follows:

E(dit

By entering an "E", either the Screen-Oriented Editor or the Line-Oriented Editor (YALOE) is brought into memory from disk, depending on whether a CRT or a teleprinter is used as the operator's console in the system. The work file text is read into the Editor buffer automatically if one is present. Otherwise, the Editor prompts for a file.

R(un

Entering an "R" causes the code file associated with the current work file to be executed. If a code file does not currently exist, the system Compiler is called automatically. If the compilation requires linkage to separately compiled code, the Linker also is called automatically and assumes the use of the file *SYSTEM.LIBRARY. The program is executed after a successful compilation and linkage.

F(ile

Entering an "F" calls the File Handler into memory from disk.

C(omp

Entering a "C" initiates either the Pascal Compiler or the Basic Compiler, whichever is a part of the system.

L(ink

The "L" command starts the Linker to allow users to link routines from libraries other than *SYSTEM.LIBRARY.

X(ecute

After an "X" command is entered, the user is prompted for the file name of a previously compiled code file. If the file requested exists, it is executed. Otherwise, the message "can't find file" is returned. (Note: The ".CODE" suffix on such a file is implicit and does not need to be entered.) If the code file is composed of several separately compiled files, and one part has not been linked, the message "file (fileid) not linked in" is returned. Programs can be executed either by use of X(ecute for a file that has already been compiled or by use of F(ile, G(et the file, Q(uit the Filer, and then R(un the program.

D(ebug

Entering a "D" causes the current work file to be executed. If the program in the work file has not been compiled, the Compiler will be called first, as with the R(un command. During execution, if a runtime error occurs or if a user- defined breakpoint or halt is encountered, the Debugger is called.

The X(ecute command can be used to execute utility functions. These functions include a desk calculator, a librarian and library map, and a GOTOXY procedure binder, among others. These are described in Section 4.

2.4 SCREEN-ORIENTED EDITOR

The Screen-Oriented Editor is designed to provide a window into the file with the video display terminal. The window shows that portion of the file in which editing is taking place. When entering any file, the Editor displays the start of the file in the upper left corner of the screen. That is where the cursor is originally positioned, also. The cursor is a marker that indicates where the action is taking place. When the user enters an Editor command to go to a part of the file that is not displayed, the window is updated to show that portion of the file. Thus, the user does not need to operate on any portions of the text that are not seen on the screen; however, he has the option to do so.

When the Editor has been called by entering the outer level "E" command, the Editor displays a prompt line at the top of the screen. The prompt line reminds the user of the current mode and the options available for that mode. Only the most commonly used options appear on the prompt line. The format is:

```
>Edit:  A(djust C(py D(lete F(ind I(nsrt J(mp R(place Q(uit X(chng Z(ap
```

Note that the cursor is never really "at" a character position; it is between the character where it appears (for ease of display) and the one immediately preceding. This is most clearly noticed in the I(nsrt command which inserts in front of the character at which the cursor is located.

In the Editor, some of the options are referred to as commands and some as modes. When an option executes a task and returns control to the Editor, that option is called a command. When an option issues a prompt for another level of options, it is called a mode. On entering or returning to the edit level, the Editor always displays the "Edit:" prompt line.

Repeat factors are allowed by any of the commands to repeat the action of the command as many times as indicated by the immediately preceding number. For example, entering

```
2 (down-arrow)
```

will cause the (down-arrow) command to be repeated twice, moving the cursor down two lines. The assumed repeat factor is 1 if no number is typed before the command. A slash (/) typed before the command indicates an infinite number of repeats.

Some commands are directional. If their direction is forward, they operate forward through the file; if backwards, they operate in the reverse. When direction affects the commands, it is specifically noted.

2.4.1 Moving Commands

The moving commands move the cursor from one location to another to position it for the next editing function. Many of these commands are initiated by keys on the CRT keyboard. They include:

Command/Key	Function
⟨down-arrow⟩	Moves cursor down
⟨up-arrow⟩	Moves cursor up
⟨right-arrow⟩	Moves cursor right
⟨left-arrow⟩	Moves cursor left
"⟨" or "," or "-"	Changes the direction to backward
")" or "." or "+"	Changes the direction to forward
⟨space⟩	Moves direction
⟨back-space⟩	Moves left
⟨return⟩	Moves to the beginning of the next line

Direction is always indicated by an arrow (⟩ or ⟨) in front of the prompt line. The direction is forward when the Editor is entered, but can be changed by typing the appropriate command whenever the "Edit:" prompt line is present. On many standard keyboards, the period (.) can be used for forward because it is the lower case for ")"; and the comma (,) can be used for backward, being the lower case for "⟨".

Repeat factors can be used with any of the keyboard commands given above.

The Editor maintains the column position of the cursor when executing the ⟨up-arrow⟩ and ⟨down-arrow⟩ commands. When the cursor is outside the text, the Editor treats the cursor as though it were immediately after the last character or before the first in the line.

The moving commands that do not have special function keys on the CRT keyboard are jump, page and equals. They are described below.

⟩JUMP

JUMP mode is entered by typing "J" for J⟨mp. When the JUMP mode is entered, the prompt line appears:

JUMP: B⟨eginning E⟨nd M⟨arker ⟨esc⟩

Entering "B" or "E" moves the cursor to the beginning or end of the file, displays the "Edit:" prompt line and the first or last page of the file. Entering "M" produces the prompt line:

Jump to what marker?

The user then enters the name of the marker, followed by a ⟨return⟩. The Editor moves the cursor to the place in the file where the marker having that name was previously set. If a marker of that name is not in the file, the error message displayed will be:

ERROR: Marker not there. Please press ⟨space-bar⟩ to continue.

The instructions for setting a marker in a file are given in Section 2.4.4, Miscellaneous Commands, under Set.

PAGE

The PAGE command is executed in response to typing "P". PAGE moves the cursor one whole page up or down, depending on the direction of the arrow at the beginning of the prompt line. The cursor moves to the start of the top line. To move several pages at one time, the repeat factor may be used with this command.

EQUALS

The EQUALS command is executed in response to "=". It makes the cursor jump to the beginning of the last section of text that was inserted, found or replaced from anywhere in the file. EQUALS works from anywhere in the file and is not direction-sensitive. When an INSERT, FIND or REPLACE command is executed, the beginning of that function is saved. However, if a copy or deletion has been made between the beginning of the file and that absolute position, the beginning location is altered and is no longer correct for an EQUALS command.

2.4.2 TEXT-CHANGING COMMANDS

The commands described below change the text of the file.

INSERT

INSERT mode is entered by typing "I" for Insrt. When the INSERT mode is entered, the prompt line appears:

```
Insert: Text{(bs) a char, (del) a line} [(etx)
        accepts, (esc) escapes]
```

The text to be inserted is typed in. It may be followed immediately by (etx) or (esc). Before insertion, a character can be deleted by backspacing ((bs) a char) or an entire line can be deleted ((del) a line).

A new line can be started at a level of indentation set by options in the SET mode by typing:

```
(return)
```

Direction does not affect the INSERT mode, but is indicated by the arrow in the first position of the prompt line.

An insertion that is made and corrected is available for use in the COPY mode. However, if (esc) is used,

no string is available for COPY.

AUTO-INDENT and FILLING

AUTO-INDENT and FILLING are used to control the left and right margins, respectively. Both are in effect if True and not in effect if False. (See SET for setting them to True or False.)

If AUTO-INDENT is True, a <return> causes the next line to start at the same level of indentation as the immediately preceding line. If False, indentation has to be adjusted by spacing for any line that is not to start at the normal left-most position in the line. An example is shown in Figure 2-2. When the left margin is controlled by AUTO-INDENT (True), the level of indentation is changed by using the <space> and <backspace> keys immediately following a <return>.

If FILLING is True, all insertions are forced within the right margin by automatically inserting a <return> between words whenever the right margin would have been exceeded. The Editor considers anything between two spaces or between a space and a hyphen to be a word. An example is shown in Figure 2-2. (See SET to set margins.)

FILLING also causes adjustment of the right margin on the portion of the paragraph following an insertion. However, any line beginning with the Command character (see SET) is not touched, and that line is considered to terminate the paragraph.

```
Line 1    Original indentation
Line 2    <RET> causes indentation to the level of the line above
  Line 3  Indention was changed by <space> <space>
  Line 4  <RET> causes indentation to the level of the line above
 Line 5   Indentation was changed by <Backspace>
```

Figure 2-2. Example of Auto-Indent

DELETE

DELETE mode is entered by typing "D" for D(lete. On entering DELETE mode, the following prompt line appears:

```
>Delete: < > <Moving commands> {<etx> to delete,
         ,<esc> to abort}
```

The cursor must be positioned at the first character to be deleted. When entering DELETE, the Editor remembers where the cursor is. That position is called the anchor. As the cursor is moved from the anchor position using the moving commands, text in its path will disappear. To accept the deletion, type <etx>; to escape, type <esc>. When <etx> is typed, the Editor saves everything that was deleted for COPY to use. When <esc> is typed, nothing is deleted but the copy buffer contains what would have been deleted. An example is shown in Figure 2-3.

This is the text before deleting:

This sentence of the text is to remain the same. This sentence is to be modified by the delete command.

- -

Position the cursor over the "T" in the second occurrence of "to". Enter a "D" followed by typing 5 <space> keys and a <etx> key. This will result in the following text:

- -

This sentence of the text is to remain the same. This sentence is modified by the delete command.

Figure 2-3. Example of Delete Command

The repeat factor may be used to delete several lines at once by prefacing a <return> or any other of the moving commands with the desired repeat number while in DELETE mode.

ZAP

The ZAP command is executed by typing "Z" for Z(ap. This command deletes all text between the start of what was previously found, replaced or inserted and the current position of the cursor. ZAP is designed to be used immediately after FIND, REPLACE or INSERT. If more than 80 characters are being zapped, the Editor will ask for verification.

If a FIND or REPLACE is made with a repeat factor and then ZAP is called, only the last find or replace will be zapped.

Whatever is deleted by ZAP is available for use with COPY.

COPY

COPY mode is entered by typing "C" for C(py. When entering the COPY mode, the following prompt line appears:

>COPY: B(uffer F(ile (esc)

Enter "B" to copy text in the copy buffer into the file at the location of the cursor when the "C" was typed. On completion of the COPY, the cursor returns to a position immediately preceding the text that was copied. The use of COPY does not change the contents of the copy buffer.

Enter "F" to copy text from another file. The following prompt line will appear:

>COPY: FROM WHAT FILE[MARKER,MARKER]?

Any file may be specified. Text is assumed. Markers can be set so that a given portion of a file can be copied. On completion of the COPY (from file), the cursor returns to the beginning of the text just copied from the file. Use of COPY does not alter the contents of the file being copied.

The copy buffer is affected by the following commands:

DELETE - On accepting a deletion, the buffer is loaded with the deletion; on escape, the buffer is loaded with what would have been deleted.

INSERT - On accepting an insertion, the buffer is loaded with the insertion; on escape, the buffer is empty.

ZAP - When the ZAP command is used, the buffer is loaded with the deletion.

When the deletion is greater than the buffer space available, after typing (etx), the Editor will give the warning:

There is no room to copy the deletion. Do you wish to delete anyway? (y/n)

EXCHANGE

EXCHANGE mode is entered by typing "X". On entering

the EXCHANGE mode, the following prompt line appears:

```
EXCHANGE: TEXT ((bs) a char; [(esc) escapes; (etx)
          accepts]
```

EXCHANGE replaces one character in a file for each character of text entered. An example is shown in Figure 2-4. Backspacing one character will cause the original character in that position to reappear. Typing (esc) leaves the EXCHANGE mode without making any of the changes indicated since entering the mode, while typing (etx) accepts the changes as part of the file.

NOTE: EXCHANGE does not allow typing past the end of the line or typing in a carriage return.

This is the text before exchanging:

```
PROGRAM EXCHANGE;
  BEGIN
    WRITELN('THIS TEXT IS FINE');
    WRITELN('THIS TEXT IS NOTFINE');
END.
```

- -

Position the cursor to the first "N" in "NOTFINE". Enter an "X" followed by typing "CHANGED" and a (etx) key. This will result in the following text:

- -

```
PROGRAM EXCHANGE;
  BEGIN
    WRITELN('THIS TEXT IS FINE');
    WRITELN('THIS TEXT IS CHANGED');
  END.
```

Figure 2-4. Example of Exchange

FIND and REPLACE

The following rules apply to both FIND and REPLACE:

1. The repeat factor is valid and must be typed before typing "F" or "R". The current repeat factor appears in brackets on the prompt line.

2. The Editor provides two string storage variables:

```
targ  the target string (used by both)
sub   the substitute (REPLACE only)
```

Both delimiters of either string will be the same. The Editor considers any character that is not alphabetic or numeric to be a delimiter. (space) is a particularly common delimiter.

3. Text is scanned in the direction of the arrow on the prompt line. The target pattern can be found only if it appears in that section of the text.

4. In LITERAL mode, the Editor will look for any occurrences of the target string; in TOKEN mode, it looks for an isolated occurrence. Isolation means a string is surrounded by any combination of delimiters. To use the LITERAL mode, type "L" after the prompt line and before the target line; to use TOKEN, type "T". The default value found in the Environment may be overridden by typing "L" or "T". TOKEN ignores spaces within strings so that both "(',')" and (', ') are considered to be the same string.

5. To use the same string as used previously, type "S". For example, typing "RS/(any-string)/" causes REPLACE to use the previous target string; while typing "R/(any-string)/S" causes the previous substitute string to be used.

FIND mode is entered by typing "F". On entering FIND mode, one of the prompt lines below will appear:

```
)Find[n]:  L(it (target) =)
)Find[n]:  T(ok (target) =)
```

FIND mode finds the repeat factor [n] occurrence of the (target) string starting with the current position and moving in the direction shown by the arrow.

REPLACE mode is entered by typing "R". On entering the REPLACE mode, one of the prompt lines below will appear:

```
)Replace[n]:  L(it V(fy (tart) (sub) =)
)Replace[n]:  T(ok V(fy (targ) (sub) =)
```

REPLACE mode finds the repeat factor [n] occurrence of the target string and replaces it with the substitute string.

The verify option (V(fy) allows examination of the

target string (to the limit set by the repeat factor) to decide if it is to be replaced. Whenever REPLACE has found the target pattern in the file and verification has been requested, the following prompt line appears:

```
>Replace: (esc) aborts, 'r' replaces, ' ' doesn't
```

Typing an "R" will cause replacement; typing a space will cause a continuation of a search for the next occurrence if the limit of the repeat factor has not yet been reached. The repeat factor counts the number of times an occurrence is found, not the number of times "R" is entered. If a slash (/) is used as the repeat factor, every occurrence of the target string will be replaced.

If the Editor cannot find the target string, the error message appears:

```
ERROR: Pattern not in the file. Please press
       (space bar) to continue.
```

An example of REPLACE is shown in Figure 2-5.

This is the text before replacing:

```
PROGRAM REPLACE;
  BEGIN
    WRITELN('SOME WORDS');
    WRITELN('MORE WORDS');
    WRITELN('EVEN MORE WORDS');
  END.
```

- -

Position the cursor to the beginning of the text. Enter a 3R and the following would be displayed:

- -

```
>REPLACE[3]: L(it V(fy (targ) (sub?) => T/WORDS//STUFF/
```

- -

This will result in the following text:

```
PROGRAM REPLACE;
  BEGIN
    WRITELN('SOME STUFF');
    WRITELN('MORE STUFF');
    WRITELN('EVEN MORE STUFF');
```

```
END.
```

Figure 2-5. Example of Replace

2.4.3 Formatting Commands

The formatting commands ADJUST and MARGIN are used to control indentation and page format on a line-by-line basis (ADJUST) or by paragraph (MARGIN).

ADJUST

ADJUST mode is entered by typing "A". On entering the ADJUST mode, the following prompt line appears:

```
>Adjust: L(just R(just C(enter <left,right,up,down-
         arrows> {<etx> to leave}
```

ADJUST mode adjusts indentation on a line-by-line basis. On any line, the right-arrow and left-arrow commands move the whole line one space to the right or left, respectively, each time the arrow is typed. Type <etx> when indentation is adjusted.

To adjust a sequence of lines, adjust one, then use the up-arrow and down-arrow commands to adjust the line above or below, respectively, by the same amount. Repeat factors can be used before any of the arrows.

"L" and "R" are used to left- and right-justify lines to margins set in the Environment. "C" will center the line between the set margins. Typing an up- or down-arrow will justify or center the line above or below to the same specification.

MARGIN

The MARGIN command is executed by typing "M". This command does not appear on the prompt line. MARGIN is Environment-dependent and cannot be executed except when FILLING is set to True and AUTO-INDENT is set to False.

Three parameters are used by margin: right-, left- and paragraph- margin. MARGIN deals with only one paragraph at a time. It realigns the text to compress it as much as possible without violating the three margins. To set the margin values, see SET mode. An example is shown in Figure 2-6.

This paragraph has been MARGINed with the parameters set:

Left margin 10
Right margin 70
Paragraph margin 18

The Margin Command is executed by typing "M" when the cursor is in the paragraph to be margined. The Margin Command deals with only one paragraph at a time and realigns the text to the specification set in the environment.

This paragraph has been MARGINed with the parameter set:

Left margin 20
Right margin 60
Paragraph margin 0

The Margin Command is executed by typing "M" when the cursor is in the paragraph to be margined. The Margin Command deals with only one paragraph at a time and realigns the text to the specification set in the environment.

Figure 2-6. Example of Margin

For purposes of formatting, a paragraph is defined as the lines of text occurring between two blank lines. To MARGIN a paragraph, move the cursor anywhere in the paragraph and type "M". With an exceptionally long paragraph, the routine may take several seconds in execution before redisplaying the paragraph.

Any given line of text can be protected from being MARGINed if the the Command character appears as the first non-blank character on the line. The MARGIN treats that line as though it were entirely blank. (See SET for setting the Command characters.)

WARNING: Do not use MARGIN within a line that starts with the Command character.

2.4.4 Miscellaneous Commands

SET

SET mode is entered by typing "S". This command does not appear on the prompt line. On entering the SET mode, the following prompt line appears:

>Set: M(arker E(nvironment (esc)

Markers are a convenience in a long file. They are set by moving the cursor to the position in the text to be marked, then typing "M". The following prompt line will appear:

```
Name of marker?
```

The name may be up to 8 characters followed by a (return). Marker names are case-sensitive; upper and lower cases of the same letter are considered to be different letters. If the marker already existed, it will be reset.

Only ten markers are permitted in a file at any one time. On typing "SM", if an overflow occurs, the following prompt will appear:

```
Marker ovflw.

Which one to replace.

0)  name1

1)  name2

    ..

    ..

9)  name10
```

Choose a number 0 through 9, type that number, and now that space will be available for setting the desired marker.

Once markers are set, they can be jumped to by using the M(arker option in the JUMP mode.

If a copy or deletion is made between the beginning of the file and the position of the marker, the absolute position of the marker will be changed.

Through SET mode, the Editor enables the user to set the Environment to meet the needs of the editing to be done. When "E" for E(nvironment is typed, the following prompt appears:

```
>E(nvironment: (options) (etx) or (sp) to leave

        A(uto indent            True
        F(illing                False
        L(eft margin            0
        R(ight margin           79
```

```
P(ara margin            5
C(ommand ch             ^
T(oken def              True

nnnn bytes used, nnnn available

Patterns:
        target = 'xyz',  subst = 'abc'
```

The option values given in the prompt are defaults for the Sorok 120. By typing the appropriate letter, any or all of the options may be changed. Each option is described below.

A(uto indent - Set to True (turned on) by typing "AT" and to False by "AF". Affects only the INSERT mode.

F(illing - Set to True (turned on) by "FT" and to False by "FF". Affects the INSERT mode and allows the MARGIN command to function.

Margins - Set by typing the appropriate letter ("L", "R" or "P") followed by a positive integer of less than 4 digits and a <space>. When Filling is True, the margins set here are the ones that affect INSERT and MARGIN and the center and justify commands in ADJUST.

C(ommand ch - Set by typing "C" followed by any character. Affects the MARGIN command and the Filling option of the INSERT mode.

T(oken def - Set to True by "TT" and to False by "TF". If True, Token is the default; if False, Literal is the default. Affects FIND and REPLACE.

VERIFY

The VERIFY command is executed in response to typing "V". The status of the Editor is verified by displaying the updated screen. The Editor attempts to adjust the window so the the cursor is at the center of the screen.

QUIT

QUIT mode is entered by typing "Q". On entering the QUIT mode, the following prompt appears:

```
>Quit:
   U(pdate the workfile ard leave
   E(xit without updating
   R(eturn to the editor without updating
```

W(rite to a file name and return

One of the four options must be selected by typing the appropriate letter. The options are described below:

U(pdate - The Editor writes the file just modified into the workfile and stores it as SYSTEM.WRK.TEXT. It is available for either the Compile or Run options or for the Save option in the Filer. The Filer regards it as the text file.

E(xit - The Editor leaves without making any changes in SYSTEM.WRK.TEXT. Any modifications made since entering the Editor are NOT recorded in the permanent work file.

R(eturn - The Editor returns without updating. The cursor is returned to the exact place in the file it occupied when "@" was typed. Usually, this command is used after typing "@" unintentionally.

W(rite - With this option, a further prompt appears:

```
>@uit:
Name of output file (<cr> to return) -->
```

The modified file may be written to any file name. If it is an existing file, the modified file will replace it. The command can be aborted by typing <return> instead of a file name; return will be to the Editor. After the file has been written to disk, the prompt will be:

```
>@uit:
Writing . . . . . .
Your file is nnnn bytes long.
Do you want to E(xit from or R(eturn to the Editor?
```

Typing "E" exits from the Editor and returns to the Outer Command level. Typing "R" returns the cursor to the same position in the file as when the "@" was typed.

2.4.5 L2 Editor

The L2 Editor handles files larger than can fit into the main memory buffer at one time (i.e., what can fit into the space available on the disk). Also, it automatically makes a backup copy of the file being edited. There are very few differences between the L2 Editor and the Screen-Oriented Editor previously defined in this section. The differences are described below.

If, on entering the Editor by typing "E", there is not room enough on the

disk, the following error message will be displayed:

ERROR: Not enough room for backup!

The disk must be K(runched by the Filer to combine unused blocks at the end, or a file must be removed, or another disk must be used. Then, when "E" is typed, the following prompt lines appear:

```
Copying to filename.back.
>Edit    (same as for standard Editor)
Reading . . . . . .
```

In Section 2.3, the description of entering a workfile and getting a program applies also to the L2 Editor.

When all changes and additions have been made, the Editor is exited as usual, by typing "Q", except that the W(rite option is not available. The other three options have added features:

U(pdate - supplies additional information to indicate the file name and length. Below is an example of the extra information given when a new file is created:

```
Writing.#
The workfile, #:F1.TEXT, is 73 blocks long.
The backup file is X:F1.BACK.
```

The newly edited file is .TEXT; and the original file with no modifications is .BACK.

E(xit - no .BACK file is made and the existing one is removed.

R(eturn - no changes.

A few of the Editor commands are handled differently. They are:

JUMP - The prompt line is the same, however the "B" and "E" refer to the beginning and end of the buffer, not the file.

FIND - The Editor displays "Finding . . ." and gives the pattern. If the pattern is not in the buffer, it displays:

```
End of buffer encountered.  Get more from disk?
(Y/N)
```

On typing "Y", the Editor will move another section of the file into the buffer and continue searching. FIND is still directional.

SET - Markers are set by typing "SM", as in the standard Editor. However, up to 20 markers are

permitted, rather than 10. The environment is set by typing "SE". The prompt displayed gives additional information, as follows:

```
>Environment:  options <etx> or <sp> to lave
  A<uto Indent
  F<illing
  L<eft margin
  R<ight margin
  P<ara margin
  C<ommand ch
  T<oken def

  nnnn bytes used.   nnnn available.
```

There are n pages in the left stack, and n pages in the right stack. You have n pages of room, and at most n pages worth in the buffer.

Markers: <P1 P2 >P3
(no arrow indicates the marker is in the current buffer)

Created mm dd yy: Last updated mm dd yy (Revision n).

The L2 Editor contains two new commands, BANISH and NEXT. They are described below.

BANISH

BANISH is entered by typing "B". The following prompt line will appear:

```
>Banish:  To the L<eft or R<ight <esc>
```

BANISH moves characters from the buffer into the stack to provide more room in the buffer to avoid overflow when doing a large insertion or copy. The left and right stacks are ahead of and behind the cursor, respectively. The screen is the boundary for the operation.

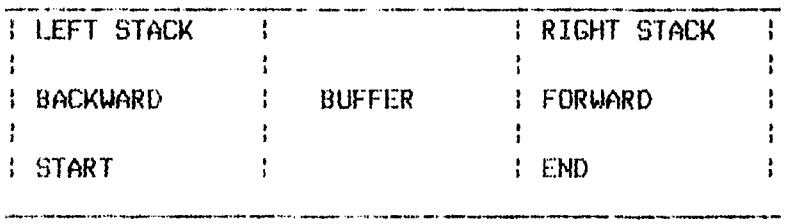
NEXT

NEXT is entered by typing "N" to move beyond the bounds of the buffer. The following prompt line appears:

```
Next: F<orwards, B<ackwards in the file: S<tart,
      E<nd of the file.  <esc>
```

When using "F" or "B", an implicit banish occurs using the cursor as the point of reference. With "F",

everything above the screen is banished to the left stack. More characters are added to the bottom of the screen to extend the buffer in the forward direction. With "B", the characters below the cursor are banished to the right stack and the lower part of the screen becomes blank. More characters are added above the screen. Thus, the symbolic file can be diagrammed as shown below.



2.5 YET ANOTHER LINE-ORIENTED EDITOR (YALOE)

The YALOE text editor is designed for use in systems that have a teleprinter or teletypewriter as the system operator's console rather than a video display terminal.

The Editor assumes the existence of a workfile, but is not dependent on it. The workfile can be created after entering YALOE. If a workfile already exists, the Editor will print:

```
Workfile STUFF read in.
```

If the Editor is called and the workfile is empty, the Editor will give the message:

```
No workfile read in.
```

The Editor operates in either Command Mode or Text Mode. The Editor is in Command Mode when entered. In Command Mode, all keyboard input is assumed to be commands. Each command may be terminated by (esc). The commands may be strung together. No commands in a string (or singly) will be executed until the final command in the string is followed by (esc) (esc). Spaces, carriage returns and tabs within a command string are ignored unless they appear in a text string. When the execution of a command string is complete, the Editor prompts for the next command with an asterisk (*). In contrast to other levels of the Pascal Operating System, a prompt line of available commands is not given.

If an error is encountered during command execution, the command will be terminated at that point without completing execution.

The Text Mode is entered whenever a command is typed that must be followed by text. Then all succeeding characters are considered to be text until the next (esc). The commands that require text are F(ind, G(et, I(nsert, M(acro define, R(ead file, W(rite to file, and eX(change.

NOTE: When typed, (esc) echoes a dollar sign ($). The (esc) terminates each text string and causes the Editor to re-enter the Command Mode. A double (esc) terminates the command string and causes the Editor to start execution.

The workfile is stored in the text buffer. This area is allocated dynamically by the ? command (section 2.5.5). The Editor can work only on files that fit completely within the text buffer.

The cursor is the position in the file where the next command will be executed. Most edit commands function in relation to the cursor.

Some of the YALOE commands described here require a command argument to precede the command letter. Usually, the argument specifies the number of times the command should be performed or the particular portion of text to be affected by the command. With some commands, the specifications are implicit and no argument is needed. The command arguments used by YALOE are:

n — Any integer, signed or unsigned. Unsigned integers are assumed to be positive. In a command that accepts an argument, the absence of one implies 1 (only one execution) or minus 1 if only the minus sign is present.

m — A number in the range of 0 through 9.

o — The beginning of the current line.

/ — The same as 32700. A "-/" is -32700. It is used for a large repeat factor.

= — Represents -n where n equals the length of the last text argument used. Applies only to the J, D, and C commands.

2.5.1 Special Key Commands

Various keys on the keyboard have special functions, some of which are unique to YALOE. These commands are described below. Those control keys that do not appear below are ignored and discarded by YALOE.

(esc)

The escape key is echoed as a dollar sign ($) on the operator's console. A single (esc) terminates a text string. A double (esc) executes a command string.

RUBOUT
(linedel)

On hard-copy terminals, line delete is echoed as "(ZAP" and a carriage return. On others, it clears

the current line on the screen. In both cases, the contents of that line are discarded by the Editor.

CTRL H
(chardel)

On hard-copy terminals, character delete is echoed as a percent sign (%) followed by the character deleted. Deletions are done right to left, with each character deleted erased by the %, up to the beginning of the command string. CTRL H may be used in both Command and Text Modes.

CTRL X

CTRL X causes the Editor to ignore the entire command string, and respond with a carriage return and an asterisk (*) to prompt for another command. The command string being ignored may be on several lines. All lines back to the previous * prompt are ignored. (A character delete is confined to one line.)

CTRL O

CTRL O causes the Editor to switch to the optional character set (bit 7 turned on). This applies only to the TERAK 8510A.

NOTE: If strange characters start appearing on the terminal CRTL O may have been hit accidentally. This is corrected by typing CRTL O again.

CTRL F
Flush

CTRL F causes the Editor to discard all output to the terminal until the next CTRL F is typed.

CTRL S
stop

CTRL S causes the Editor to hold all output to the terminal until the next CTRL S is typed.

2.5.2 Input/Output Commands

The commands that control I/O are described below.

LIST

The LIST command is specified by typing "L" for L(ist. It causes the Editor to print a specified number of lines on the terminal without moving the cursor. Variations of this command are illustrated in

the examples below.

*-3L$$ Prints all characters starting at the third preceding line and ending at the cursor.

*5L$$ Prints all characters beginning at the cursor and terminating at the fifth carriage return (line).

*0L$$ Prints from the beginning of the current line up to the cursor.

VERIFY

The VERIFY command is specified by typing "V" for V(erify. It causes the Editor to print the current text line on the terminal. The position of the cursor within the line has no effect on the command and the cursor is not moved. No arguments are used. VERIFY is equivalent in effect to a "*0L$$" list command.

WRITE

The WRITE command is specified by typing "W" for W(rite followed by the file title, in the following format:

*W(file title)$

The file title is any legal file title described in Section 3.2, except that the file type is not given. The Editor will automatically append ".TEXT" as a suffix unless the title ends with ".", "]" or ".TEXT". If the title ends in ".", the period will be stripped from it.

The WRITE command will write the entire text buffer to a file having the given file title. It will not move the cursor or alter the contents of the text buffer. If the volume specified by the file title has insufficient room for the text buffer, the following error message will be given:

OUTPUT ERROR. HELP!

The text buffer can be written to another volume.

READ

The READ command is specified by typing "R" for R(ead followed by the file title, in the following format:

*R(file title)$

The Editor will attempt to locate the file title as given. If no file is found having that title, a ".TEXT" is appended and a new search is made.

The READ command inserts the specified file into the text buffer starting at the location of the cursor. If the file read in does not fit, the entire text buffer will be undefined in content. This is an unrecoverable error.

QUIT

The QUIT command is specified by typing "Q" for Q(uit. It has several forms, as follows:

QU	Quit and update by writing out a new SYSTEM.WRK.TEXT
QE	Quit and escape Editor; do not alter the work file.
QR	Do not quit; return to the Editor

If "Q" is typed alone, a prompt will be sent to the terminal giving the above choices. An option must be entered (U, E or R).

The "QU" command is a special case of the WRITE command. If QU does not work, "W" can be used to write out SYSTEM.WRK.TEXT followed by "QE" to exit from the Editor. "QR" is used to return to the Editor after a "Q" has been typed accidentally.

ERASE

The ERASE command is specified by typing "E" for E(rase. It functions only with video display terminals and causes the Editor to erase the screen.

O

The O command is specified by typing "O". It functions only with video display terminals and causes the Editor to display the text around the cursor each time the cursor is moved. The argument for the O command specifies the number of lines to be displayed. This option is in a disabled state when the Editor is entered. If needed, it must be enabled by using the O command. A second O disables the option. The location of the cursor is denoted by a split in the line of text.

2.5.3 Moving Commands

The moving commands relocate the cursor to a new position. These commands are important because most other editing commands are dependent on cursor positioning. The moving commands are described below.

The direction of cursor movement is specified in the commands by the sign of the argument. A positive (+n) argument gives the number of characters or lines to move in a forward direction; and a negative argument (-n), in a backward direction.

Carriage return characters are treated the same as any other character in text except that the (cr) denotes the end of a line of text.

Examples of the moving commands are given in Figure 2-7. In the examples, the cursor position is indicated by an up arrow (^) although the cursor does not actually appear on the teleprinter or teletypewriter.

JUMP

> The JUMP command is specified by typing "J" for J(ump. JUMP moves the cursor a specified number of characters in the text buffer. Movement may be either forward or backward and is not restricted to just the current line.

ADVANCE

> The ADVANCE command is specified by typing "A" for A(dvance. ADVANCE moves the cursor a specified number of lines. The cursor is then positioned at the beginning of the line to which it moved. An argument of zero moves the cursor to the beginning of the current line. Movement may be either forward or backward.

Here are the original lines and the cursor position.

THE TIME HAS COME(cr)

THE WALRUS SAID^(cr)

TO TALK OF MANY THINGS(cr)

Example 1. *8J$$ moves the cursor forward 8 characters to the next line between the K and the space.

TO TALK^ OF MANY THINGS(cr)

Example 2. *-2A$$ moves the cursor to the beginning of the second preceding line.

^THE TIME HAS COME(cr)

Example 3. *BGTWINE$=J$$ moves the cursor to the beginning of the text buffer, then starts searching for the string "TWINE". When the string is found, the cursor will be positioned immediately before it.

Figure 2-7. EXAMPLES OF THE MOVING COMMANDS

BEGINNING

The BEGINNING command is specified by typing a "B" for B(eginning. BEGINNING moves the cursor to the beginning of the text buffer.

GET and FIND

The search commands GET and FIND are synonymous. GET is specified by typing "G" and FIND by typing "F". With either command, the current text buffer is searched starting from the location of the cursor for the nth occurrence of a specified text string. Upon completion of a successful search, the cursor is positioned immediately following the nth occurrence if n is positive and immediately before if n is negative. If the search is unsuccessful, the Editor generates an error message, and the cursor is positioned at the end of the buffer if n is positive and at the beginning if n is negative.

2.5.4 Text Changing Commands

The text-changing commands add to, remove or change the text. They are described below. Examples are given in Figure 2-8.

INSERT

The INSERT command is specified by typing "I" for I(nsert. INSERT causes the Editor to enter Text Mode to add characters immediately following the cursor until an ⟨esc⟩ is typed. After insertion is completed, the cursor is positioned immediately following the last character inserted.

Occasionally with a large insertion, the temporary buffer will become full. Before this happens, the following message is printed on the operator's console.

Please finish.

Typing ⟨esc⟩ ⟨esc⟩ will terminate the insertion at that point so that the temporary buffer can be emptied into the text buffer. Insertion can then be continued by again typing "I" to re-enter the Text Mode. Forgetting to type the "I" will cause the characters that are next entered as insertions to be interpreted as commands.

⁎-4D$$	Deletes the four characters immediately preceding the cursor, even if they are on the previous line.
⁎B$GTWINE $=D$$	Moves the cursor to the beginning of the text buffer, searches for the string "TWINE", and deletes it.
⁎/K$$	Deletes all lines in the text buffer from the line in which the cursor is positioned to the end of the buffer.
⁎0CAAA$$	Replaces the characters from the beginning of the line to the cursor with "AAA" (same as ⁎0XAAA$$).
⁎BGA$=CB$$	Searches for the first occurrence of "A" and replaces it with "B".
⁎-3XNEW$$	Exchanges all characters beginning with the first character on the third line back and ending at the cursor with the string "NEW".

Figure 2-8. EXAMPLES OF TEXT-CHANGING COMMANDS

DELETE

The DELETE command is specified by typing "D" for D(elete. DELETE removes a specified number of characters from the text buffer, starting with the position of the cursor. On completion of the deletion, the cursor is positioned immediately following the deleted text.

KILL

The KILL command is specified by typing "K" for K(ill. KILL deletes a specified number of lines from the text buffer starting at the position of the cursor. On completion, the cursor is positioned at the beginning of the line following the deleted text.

CHANGE

The CHANGE command is specified by typing "C" for C(hange. CHANGE replaces n characters, starting at the position of the cursor, with the given text string. On completion, the cursor is positioned immediately following the changed text.

EXCHANGE

The EXCHANGE command is specified by typing "X" for

eX(change. EXCHANGE exchanges n lines, starting with the line on which the cursor is located, with the indicated text string. The cursor remains at the end of the changed text on completion of the command.

2.5.5 Miscellaneous Commands

SAVE

The SAVE command is specified by typing "S" for S(ave. SAVE copies the specified number of lines into the save buffer, starting at the cursor. On completion, the cursor position is unchanged and the contents of the text buffer are unaltered. Each time save is executed, the previous contents of the save buffer, if any, are destroyed. If executing a SAVE will cause the text buffer to over- flow, the Editor will generate a message and not perform SAVE.

UNSAVE

The UNSAVE command is specified by typing "U" for U(nsave. UNSAVE inserts the entire contents of the save buffer into the text buffer at the cursor. On completion, the cursor is still positioned before the inserted text. If the text buffer does not have enough room for the contents of the save buffer, the Editor will generate a message to this effect and and not execute the unsave.

The save buffer may be removed by typing "OU".

MACRO

A macro is a single command that performs a string of standard, but related commands. Any group of frequently used commands can be grouped into a macro to eliminate the need for having to write the whole setof instructions whenever they are needed. The user may create macros by using the M(acro command. The MACRO command is specified by typing "M" for M(acro in the following format:

```
mM%command string%
```

where m is an integer in the range of 0 through 9. MACRO is used to define a maximum of 10 macros. The default number is 1. The command string is stored into the macro buffer m. The command string delimiter (% in the above case) is always the first character following the "M". The delimiter may be any character that does not appear in the macro command string itself. The second occurrence of the delimiter terminates the macro.

All characters except the delimiter are legal macro command string characters, including a single (esc). All YALOE commands are legal. An example is given in Figure 2-9.

If an error occurs when defining a macro, the following error message is generated:

```
Error in macro definition.
```

The macro will have to be redefined.

N (Execute Macro)

The N command is specified "N" in the following format:

```
nNm$
```

to execute a specified macro command string. The n is simply any command argument (E.G., a repeat factor) and m is the macro number to be executed. If m is omitted, 1 is assumed. Because m is technically a command text string, the N command must be terminated by (esc) (echoed as $).

Attempts to execute undefined macros result in the generation of the following error message:

```
Unhappy macnum.
```

Errors encountered during macro execution generate:

```
Error in macro.
```

? (List)

The ? command is specified by typing "?" to print a list of all commands and the sizes of the text buffer, save buffer, and the memory still available for expansion.

```
x4M1ZFPREFACE$=CEND PREFACE$V$%$$
```

This example defines macro number 4. When macro 4 is executed, the Editor will look for the string "PREFACE", change it to "END PREFACE", and then display the change to verify it.

Figure 2-9. EXAMPLE OF MACRO COMMAND

2.6 FILE HANDLER

The File Handler handles, identifies, structures and restructures the files described in Section 3.2. The file basic to the Pascal operating system is the work file which is a temporary copy of the file being modified or created. The work file name on the diskette is #SYSTEM.WRK.TEXT when the text of a file is being changed. When a code version is first created, the name is #SYSTEM.WRK.CODE.

Many File Handler commands require a file specification. The diagram in Figure 2-10 illustrates the syntax of file specification. Whenever a specification is requested, as many files as desired may be specified, separating the specifications for each file with commas and terminating the list with a carriage return. Commands operating on single file names will keep reading the names from the list and operating on them until there are no more names. Commands operating or two file names at once (e.g., CHANGE and TRANSFER) will take file names in pairs until one or none remain. In this case, if only one name remains, the File Handler will prompt for the second name. If an error is detected in the list, the entire list will be flushed.

The rules for legal file and volume names are given in Section 3.1, Files.

The File Handler performs the requested action on all files meeting the specifications. The wild card characters "=" and "?" are used to specify subsets of the directory. For example, a file specification containing the subset-specifying string "PUB=TEXT" notifies the File Handler to perform the requested action on all files whose names begin with the string "PUB" and end with the string "TEXT".

If a "?" is used instead, the File Handler requests verification before performing the requested action on each file meeting the specified criteria.

Either or both strings may be empty. For example, a subset specification "=<string>" or "<string>=" or even "=" is valid. In the last case, where both strings are empty, the File Handler acts on every file in the volume directory. The same applies to "?" used alone, except here the File Handler verifies first.

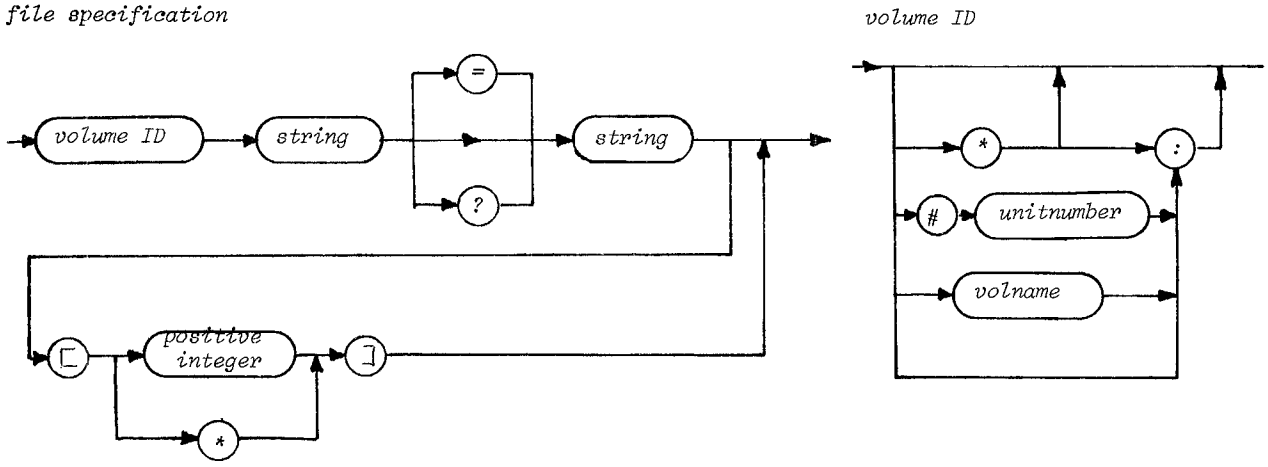


Figure 2-10 . FILE SPECIFICATION

The File Handler is entered by typing "F" when at the Outer Level of commands. The following prompt line will appear:

```
Filer:  G(et, S(ave, W(hat, N(ew, L(dir, R(em, C(hng, T(rans, D(ate, Q(uit
```

Additional File Handler commands are displayed in response to typing "?". The prompt line is:

```
Filer:  B(ad-blks, E(xt-dir, K(rnch, M(ake, P(refix, V(ols, X(amine, Z(ero
```

The operator invokes the execution of individual commands by typing the character on the operator's console. Many of the commands give additional prompt lines to get all of the information necessary for execution. Answering a Yes/No question on a prompt line with any character other than "Y" constitutes a No answer. Typing an <esc> will return control to the Outer Level of commands.

2.6.1 "F" Prompt Line Commands

GET

The GET command is specified by typing "G" for G(et. GET loads the designated file into the workfile. When "G" is typed, the File Handler responds with the following prompt line:

```
Get what file?
```

The entire file specification is not necessary. If the volume ID is not given, the default disk is assumed. Wildcards are not allowed, and the size specification is ignored. The suffixes ".TEXT" and ".CODE" are not required if both are present and both are loaded. If only one is present, the File Handler will load that one regardless of whether code or text has been specified.

When the File Handler has completed loading, it responds with one of the following messages:

```
Text and Code file loaded.
Code file loaded.
Text file loaded.
```

SAVE

The SAVE command is specified by typing "S" for S(ave. SAVE saves the work file under the file name specified by the user in response to the prompt:

```
Save as what file?
```

The entire file specification is not necessary. If the volume ID is not given, the default disk is assumed. Wild cards are not allowed, and the size option is ignored. The File Handler automatically appends ".TEXT" or ".CODE", as appropriate, so these suffixes must NOT be entered.

Any illegal characters in the file name will be ignored except a colon (:). If the file specification includes the volume ID, the File Handler assumes that the work file is to be saved on another file. In that case, a colon is assumed to separate the file name from the volume name.

For example, if the operator enters "BLACK:BART" in response to the prompt "Save as what file?", the File Handler will generate the prompt line:

```
Would you like BART.TEXT written to BLACK: ?
```

A "Y" answer to this prompt will cause the File Handler to attempt a transfer of the work file to the specified volume and file (see TRANSFER).

WHAT

The WHAT command is specified by typing "W" for W(hat. WHAT identifies the name and status (saved or not saved) of the work file.

NEW

The NEW command is specified by typing "N" for N(ew. NEW clears the work file. No file specifications are permitted. If a work file is already present, the File Handler responds with:

```
Throw away current workfile?
```

A response of "Y" will clear the work file, while "N" returns the user to the outer level of the File Handler. If a backup work file exists, the following prompt line is generated:

```
Remove (workfile name).BACK?
```

LIST

The LIST command is specified by typing "L" for L(dir. LIST lists a disk directory, or some subset thereof, to the volume and file specified. The default is CONSOLE:. The File Handler responds to "L" with the following prompt line:

```
Dir listing of what vol?
```

The user may list any subset of the directory using the wildcard option, and may also write the directory or a subset to a volume or file name other than CONSOLE. File specification must be in terms of source and destination.

Source file specification consists of a mandatory volume ID and optional subset-specifying strings, which may be empty. If the latter are used, one of the wildcard characters is required. A string (i.e., the full file name including ".TEXT") may not be used as part of the source file specification unless a wildcard character is used. Source file information is separated from destination file information by a comma.

Destination file specification includes a volume ID and, if the volume is block-structured, a file name. File size will be ignored.

This command is most frequently used to list an entire directory. The bottom line of the listing gives the number of files listed out of the number on the volume, and the number of blocks used, and the number of blocks unused.

REMOVE

The REMOVE command is specified by typing "R" for R(em. REMOVE removes file entries from the directory. When "R" is typed, the File Handler responds with the following prompt line:

```
Remove what file?
```

One specification is required for each file to be

removed. Wildcards are legal. Size information is ignored.

NOTE: SYSTEM.WRK.TEXT and SYSTEM.WRK.CODE should be removed only by the NEW command.

The use of just the initial letter of several file names will remove all files beginning with that letter. Typing the file specification "=" will remove every file in the directory unless a subset(s) is also specified. However, before finalizing any wildcard removes, the File Handler prompts with:

```
Update directory?
```

A "Y" causes all specified files to be removed. "N" returns the user to the outer level of the File Handler without removing anything.

CHANGE

The CHANGE command is specified by typing "C" for C(hange. CHANGE changes the file or volume name. When "C" is typed, the File Handler responds with the following prompt:

```
Change what file?
```

This command requires two file specifications: the file to be changed, and the new name. The first is separated from the second by either a (ret) or a comma. Any volume ID information in the second file specification is ignored because the old and new files are on the same volume. Size is also ignored.

Wildcard specifications are legal. If a wildcard character is used in the first file specification, then it must be used in the second. The subset-specifying strings in the first are replaced by the analogous strings (replacement strings) in the second.

The File Handler will not make the name change if the new name exceeds 15 characters.

On completion of the change, the File Handler reports what changes were made. For example, in response to:

```
HIGH:ST=TEXT,LOW=END
```

The File Handler will respond with:

```
HIGH:START.TEXT changed to LOWRT.END
HIGH:STOPS.TEXT changes to LOWPS.END
```

The TRANSFER command is specified by typing "T" for T(ransfer. TRANSFER copies the specified file to the given destination, leaving the source file intact. When "T" is typed, the File Handler responds with the prompt line:

Transfer what file?

File specifications for both the source and destination files are required, separated by a (ret) or a comma. Wildcards are permitted, and size information is recognized for the destination file. The source specification is given in response to the above prompt line. If the destination is not also given, the File Handler then responds with the prompt line:

To where?

On a one-drive machine, the source disk must NOT be removed until prompted to insert the destination disk, as follows:

Put in (destination volume)
Type (space) to continue

After the transfer has taken place, the File Handler notifies the user as follows:

(source vol:file) transferred to (destination vol:file)

On a one-drive machine the user will have to alternate inserting the source and destination disks until the transfer is complete.

A file can be transferred to another volume without changing name and without specifying the destination file name by other than a dollar sign ($) to signify that the name is the same. The destination volume still must be specified.

WARNING: The file name for the destination cannot be omitted completely. Otherwise, the directory for the complete volume may be wiped out. If the file name is omitted and no "$" is used, the File Handler will query:

Possibly destroy directory of (destination vol) ?

A "Y" answer will wipe out the directory of the whole destination volume.

Files may be transferred to volumes that are not block structured (e.g., CONSOLE or PRINTER) by specifying the appropriate volume ID. A file name then is ignored. The user should make certain that the destination volume is on-line.

Transfer can also be made from a non-block-structured device if it is an input device. In this case, any file name in the source file specification is unnecessary and will be ignored, if present.

If the source file specification includes a wildcard character, and the destination is to a block-structured device, then the destination file specification must also contain a wildcard character. The subset-specifying strings for the source will be replaced by analogous strings (replacement strings) in the destination. Any of the subset-specifying or replacement strings may be empty. The file specification of "=" specifies every file on the volume.

WARNING: The output buffer may overflow if the transfer of the whole disk is handled in this way. A better way of handling volume-to-volume transfers is by specifying only the source and destination volume IDs. Transferring from one block-structured volume to another causes the destination volume to be "wiped out" so that it becomes an exact copy of the source. A prompt is generated to verify if that is what is wanted:

```
Possibly destroy directory of (destination vol) ?
```

With a "Y" answer, the directory will be destroyed. A "N" answer will return control to the outer level of the File Handler. Often the "Y" answer is desirable to create a copy of the source disk for backup purposes. In this case, the name of the destination volume probably should be changed to indicate that it is a backup of the source volume (see CHANGE for name change). The source volume will not be destroyed.

Using "=" as the destination file name specification has the effect of replacing the subset-specifying strings in the source specification with nothing. The "?" may be used in place of the "=", but then the user will be asked for a verification before the transfer is performed.

WARNING: Wildcard characters in specifications should not be used on same-disk transfers. The results are unpredictable.

A file can be transferred from a volume to the same volume by specifying the same volume name for both source and destination. This is useful to relocate a file on the disk. Specifying the number of blocks desired will cause the File Handler to copy the file into the first available area of at least that size. If no size specification is given, the file is written into the largest unused area.

On a same-disk transfer, if the same file name is specified for both the source and destination, the File Handler rewrites the file to the size-specified area and removes the older copy.

QUIT

The QUIT command is specified by typing "Q" for Q(uit. QUIT returns the user to the Outer Level of commands. No file specification is allowed.

2.6.2 "?" prompt Line Commands

BAD BLOCKS

The BAD BLOCKS command is specified by typing "B" for B(ad blocks. BAD BLOCKS scans the disk and detects bad blocks. When "B" is typed, the File Handler responds with the following prompt line:

```
Bad blocks scan of what volume?
```

The user must specify the volume ID. The File Handler checks each block on the given volume for errors and lists the number of each bad block. Bad blocks can often be fixed or marked (see EXAMINE).

EXTENDED LIST

The EXTENDED LIST command is specified by typing "E" for E(xt-dir. EXTENDED LIST lists the directory in more detail than the LIST command. All prompts and wildcard options are the same as for the LIST command. All files and unused areas are listed along with the corresponding block length, last modification date, starting block address, number of bytes in the last block in the file, and the file kind. The summation line at the end of the list is the same as that for LIST.

KRUNCH

The KRUNCH command is specified by typing "K" for K(runch. KRUNCH moves the files on the specified

volume so that unused blocks are grouped at the end. When "K" is typed, the File Handler responds with the following prompt line:

Crunch what vol?

When the user responds with the volume ID, the File Handler generates:

Are you sure you want to crunch <volume ID> ?

A "Y" answer initiates the crunch. A "N" returns the user to the outer level of the File Handler.

The specified volume must be on-line. A bad block scan of the volume before crunching is strongly recommended in order to avoid writing files over bad areas of the disk. If bad blocks are found, they must be either fixed or marked before crunching (see EXAMINE).

As each file is moved, its name is reported on the console. If SYSTEM.PASCAL is moved, the system must be reinitialized by bootstrapping. Until this task has been completed, do not touch the disk, reset-switch or disk-drive door.

MAKE

The MAKE command is specified by typing "M" for M(ake. MAKE creates a directory entry with the specified file name. When "M" is typed, the File Handler responds with the following prompt line:

Make what file?

The file specification must be entered. In this case, the size is extremely useful because, if it is omitted, the File Handler creates the file by using the largest unused area of disk. The file size is given by following the volume ID and file name with the desired number of blocks enclosed in brackets. Two special ways of specifying size are:

[0] The same as omitting size. The file is created in the largest unused area.
[*] The file is created in the second largest or half largest unused area, whichever is larger.

PREFIX

The PREFIX command is specified by typing "P" for P(refix. PREFIX changes the current default to the

volume specified. The user must enter a volume ID. The rest of a file specification is unnecessary. The specified volume does not have to be on-line.

When "P" is typed, the File Handler responds with the prompt line:

```
Prefix titles by what vol?
```

The current default volume can be determined by responding to the prompt with ":".

VOLUMES

The VOLUMES command is specified by typing "V" for V(ols. VOLUMES lists all of the volumes currently on-line, along with their associated unit (device) numbers. No prompt line is displayed and no file specification is allowed. An example list of volumes is:

```
Volumes on-line
  1   CONSOLE:
  2   SYSTERM:
  3 * ONDISK
  4   PRINTER:
Prefix is - ONDISK
```

The asterisk (*) denotes the system (or boot-disk) volume. This is the default volume unless the prefix has been changed (see PREFIX). Block-structured devices are indicated by "*" or "#".

EXAMINE

The EXAMINE command is specified by typing "X" for X(amine. EXAMINE attempts to physically recover suspected bad blocks that have been detected by the B(AD BLKS command. When "X" is typed, the File Handler responds with the following prompt line:

```
Examine blocks on what volume?
```

When the user responds by typing in a volume ID, the prompt is generated:

```
Block number range?
```

The user enters the block number(s) detected by the BAD BLOCKS scan. If any files are endangered, the following prompt appears:

```
File(s) endangered:
  filename
```

Try to fix them?

A "Y" answer will cause the File Handler to examine the blocks and return either of the messages:

Block <block number> may be ok
Block <block number> is bad

In the first case, the bad block has probably already been fixed. In the second case, the File handler will offer the user the option of marking the blocks.

The user responds by giving the block number(s) of those to mark bad. These blocks will not be shifted by KRUNCH and will be rendered effectively harmless.

An "N" answer to the "fix them?" prompt returns the user to the outer level of the File Handler.

The volume to be examined must be on-line.

WARNING: A block which "may be ok" is probably physically ok but may contain garbage. Fixing a block means that the block is read, written out to the block, and read again. If the two reads are the same, the message "may be ok" applies. If the reads are different, the block is declared bad and may be marked as such.

ZERO

The ZERO command is specified by typing "Z" for Zero. ZERO reformats the specified volume, rendering the previous directory unobtainable. When "Z" is typed, the File Handler responds with the following prompt line:

Zero dir of what vol?

The user then enters the volume ID. This is followed by the prompt line:

Destroy <volume name> ?

A "Y" answer prompts:

Duplicate dir?

If the answer is "Y", a duplicate directory will be maintained. In case the disk directory is destroyed, a utility COPYDUPDIR can use the duplicate directory to restore the disk. The File Handler will then give the current number of blocks on the disk and ask if that is the desired number:

(current number of blocks on disk) blocks?

A "N" answer generates a prompt requesting the desired number:

# of Blocks?

Table 2-1 gives the correct number of blocks for several types of disks. A "Y" answer to "current number of blocks" generates:

New vol name?

The user can enter any valid volume name. The File Handler then queries to validate the name:

(new volume name) correct?

A "Y" answer causes the File Handler to generate the message:

(new volume name) zeroed

A "N" answer to the "new volume name" question and to the questions "destroy volume name" and "duplicate directory" returns the user to the outer level of the File Handler.

Table 2-1. BLOCK QUANTITIES ON DISK

NO OF BLOCKS	DISK TYPE
156	Single-density, soft-sectored, 5 1/4" floppy
312	Double-density, soft-sectored, 5 1/4" floppy
624	Double-density, dual sided, soft-sectored, 5 1/4" floppy
494	Single-density, soft-sectored, 8" floppy
988	Double-density, soft-sectored, 8" floppy
1976	Double-density, dual sided, sot-sectored 8" floppy

2.7 PASCAL COMPILER

The Pascal Compiler is a one-pass recursive descent compiler. It is invoked

by using the C(ompile or R(un commands in the Outer Level of the Pascal Operating System commands.

The Compiler normally compiles the work file, if one exists. Otherwise, it prompts the user for a source file name by the following prompt line:

```
Compile what text?
```

During the course of compilation, the Compiler will display messages on the operator's console detailing the progress of the compilation. Note that this display can be inhibited in one of two ways. The Q+ (quiet compile) option, which is described later in this section, will suppress the display. Also, if the HAS SLOW TERMINAL boolean in the system communication area is False, the display is suppressed (see Section 4.1).

An example of the output to the operator's console is shown in Figure 2-11. The identifiers appearing on the screen are the same as those in the program. The identifier for a procedure is displayed as soon as its compilation has started. The numbers appearing within [ ] give the number of 16-bit words available for symbol table storage at that point. The numbers within < > are the current line numbers. Each dot on the screen represents one source line compiled.

```
SEND.TEXT

Compiling...
```

```
PASCAL Compiler [3.0] (Unit Compiler)
<---0>.....
LAINIT [1710 words]
<--43>.....
GETFILE [1692 words]
<--52>.....
WRITEIT [1674 words]
<--71>.....
NEWLINE [1634 words]
<--84>.....
<-134>.....
<-184>.....
COPYIT [1616 words]
<-192>.....
SEND [1627 words]
<-205>.....

211 lines
Smallest available space = 1616 words
```

Figure 2-11. EXAMPLE OF COMPILER DISPLAY

If no syntax errors are detected, the compilation is considered to be successful. In this case, the Compiler writes a code file called *SYSTEM.WRK.CODE to disk. This is the code file that is executed if the user has specified the R(un command.

When the Compiler detects a syntax error, the symbol in the source where the error is detected is indicated by the marker "<<<<" and the text surrounding the error and an error number are displayed. If both the @+ (quiet compile) and L+ (list source on disk) options are selected, the compilation will continue with the syntax error being listed on the file and the console remaining undisturbed.

If those options are not selected, the Compiler gives the user the choice of three options--typing <space>, <esc> or "E".

A <space> instructs the Compiler to continue with the compilation; an <esc> causes termination; and "E" is an E(ditor command that puts the system under control of the Editor with the cursor located at the error. Syntax errors are listed in Appendix B.2. All error numbers will be accompanied by a text message of explanation on entry to the Editor if the file *SYSTEM.SYNTAX is available.

2.7.1 Compiler Option Syntax

The Compiler may be instructed to generate code according to certain options. These options are written as comments in the program text and are preceded by a dollar sign ($). The general format is:

```
(*$<option sequence> <any comment>*)
Where (* and *) bracket a comment.
```

In the option sequence, options are separated by commas. Each option is designated by a letter followed by either a plus, if the option is to be activated, or by a minus, if the option is negated. When "+" or "-" is not specified, a + is assumed. When default options are desired, they do not need to be included in the option sequence. Three of the options may be followed by file names rather than "+" or "-". They are I (when including another source file), L (when listing to a non-default destination), and U (when naming the system library file).

D

The D option causes the Compiler to issue breakpoint instructions in the code file during compilation so that the Debugger can be used more effectively. The default value is D-. The effects of "+" and "-" are:

D- Omit breakpoint instructions during compilation.

D+ Insert breakpoint instructions.

G

The G option affects the boolean variable GOTOOK in the Compiler. This boolean is used by the Compiler to determine whether it should allow the use of the Pascal GOTO statement within the program. The default value is G-. The effects of "+" and "-" are:

G- Generate a syntax error on encountering a GOTO statement.

G+ Allow the use of the GOTO statement.

NOTE: In the Pascal program, in some cases where the GOTO statement may be used, other statement such as FOR, WHILE or REPEAT may be more appropriate.

I

The I option has two forms. When it is followed immediately by a "+" or "-", it affects the boolean variable IOCHECK in the Compiler. When it is followed by a file name, it causes the Compiler to include a different source file into the compilaton at that point.

When followed by "+" or "-", the default value is I+. The effects of the signs are:

I+ Generate code after each I/O statement to see if the I/O wasaccomplished successfully. If not, the program will be terminated with a runtime error.

I- Do not generate any I/O checking. In case of an unsuccessful I/O operation, the program is NOT terminated with a runtime error.

The I- option is frequently used with system level programs that already check the IORESULT function after each I/O operation. The system program can then detect and report I/O errors without terminating abnormally. However, this may be at the expense of increased I/O errors and possibly severe program bugs.

The syntax for instructing the Compiler to include another source file is:

```
(*$I(file name) *)
```

The comment must be closed at the end of the file name, and no other options can follow. If a file name starts with a "+" or "-", a blank must be inserted

between the "I" and the file name.

If the initial attempt to open the included file fails, the Compiler appends ".TEXT" to the name and tries again. If the second attempt fails, or if an I/O error occurs while reading the included file, the Compiler responds with a fatal syntax error.

An included file may be inserted at any point into the original program, provided the rules governing the normal ordering of Pascal declarations will not be violated. The Compiler will even accept included files that contain the declarations CONST, TYPE, VAR, PROCEDURE and FUNCTION even though the original program has already completed its declarations. To do so, the "I" comment must appear between the last VAR declaration and the first PROCEDURE or FUNCTION declaration of the original program.

The Compiler cannot keep track of nested include comments. If an included file contains another include comment, a fatal syntax error will be generated.

The include comment is useful for compiling very large programs in smaller, more easily managed segments.

L

The L option tells the Compiler whether to generate a source program listing to a given file. The default value is L-. The effect of the sign is:

L-	No compiler listing will be made.
L+	The compiler listing will be sent to a disk file named "#SYSTEM.LST.TEXT".

The user may override this default destination by specifying a file name following "L". To specify a file name within a control comment, refer to the description of the "I" option (include another source file).

NOTE: The file containing the program listing may be edited the same as any other file if the file name contains the suffix ".TEXT". Otherwise, the file will be treated as data rather than text.

Next to each source line in the program listing, the Compiler lists the line number, segment procedure number, procedure number and the number of bytes or words (bytes for code, words for data) required by that procedure's declarations or code to that point. The Compiler also indicates if the line is within the

code to be executed or is part of the declarations for the procedure by flagging the line with "D" for declaration and an integer (0 through 9) for the lexical level of statement nesting within the code part. If the D+ option is selected, the listing will include an "*" to specify breakpoints.

Q

The Q option is for the "quiet compiler". It is used to suppress the output of procedure names and line numbers detailing the progress of the compilation to the operator's console. The default value is set to the current value of the SLOWTERM attribute of the system communication record SYSCOM^ (actually SYSCOM^.MISCINFO.SLOWTERM). The effect of the signs is:

Q+	Suppress output to the CONSOLE device.
Q-	Send procedure name and line number output to the CONSOLE device.

R

The R option affects the value of the boolean variable RANGECHECK in the Compiler. If RANGECHECK is True, the Compiler will output additional code to check on array subscripts and the assignments to variables of subrange types. The default value is R+. The effect of the signs is:

R+	Turns on range checking.
R-	Turns off range checking.

NOTE: Programs compiled with the R- option selected will run slightly faster. However, if an invalid index occurs or if an invalid assignment is made, the program will NOT be terminated with a runtime error.

S

The S option determines whether the Compiler will operate in swapping mode. In swapping mode, only one of the two main parts (declarations or statements) is in main memory at one time. This makes an additional 2500 words available for symbol table storage. However, compilation is slower. On full size, single-density floppy disks, the compile time is doubled. This option must be set before the Compiler encounters any Pascal syntax. The default value is S-. The effect of the signs is:

S+	Puts the Compiler in swapping mode.
S-	Puts the Compiler in nonswapping mode.

U

This option sets the boolean variable SYSCOMP in the compiler which is used to determine whether the compilation is a user program compilation or a system program compilation. The default value is U+. The effect of the signs is:

U+ The compilation is to take place on the user program lexical level.

U- Compilation is to take place at the system lexical level. This also sets the following options: R-, G+, and I-.

NOTE: Selecting U- will generate programs that may not behave as expected. It is not recommended for non-systems work without knowing the method of operation.

The U option also is used to name a library file. The named file becomes the file in which subsequent USEd UNITs are sought. The default file for the library is "#SYSTEM.LIBRARY".

An example of a USES clause with the U option is given below.

```
USES UNITA,UNITD, Found in #SYSTEM.LIBRARY
  $U NEW.CODE
     UNITB
  $U OLD.CODE
     UNITC,UNITE;
```

2.8 BASIC COMPILER

The Basic Compiler has been written in the Pascal language. It is invoked the same as the Pascal Compiler as a result of the C(ompile or R(un command given when at the Outer Level of commands for the Pascal Operating System. The file name for the Basic Compiler must be changed from its original name of BASIC.COMPILER to #SYSTEM.COMPILER for it to be invoked by the system. Thus, only one compiler will be in the system configuration, Basic or Pascal.

The Basic program is created by using one of the system editors. The main features of UCSD Basic are described below in sufficient detail for those who are already familiar with Basic to understand what is required by this Compiler.

2.8.1 Features of UCSD Basic

The Basic Compiler has real and string variables. When a real variable is applied to indexing or other integer purposes, the rounded value of the number is used. In the functions described below, "x" and "y" can be real variables or expressions which are equivalent to real variables. In like manner, "s1" and "s2" can be string variables or expressions which are equivalent to a

string.

VARIABLE NAMES

Real variables: letter(digit)

String variables: letter(digit)$. (The digit is optional.)

INTRINSIC ARITHMETIC FUNCTIONS

ATN(x)	Returns the angle in radians whose tangent is x.
EXP(x)	Returns the base of the natural logarithms raised to the power of x.
INT(x)	Returns the value of x rounded to the nearest integer.
LOG(x)	Returns the log (base 10) of x.
LN(x)	Returns the natural log of x.
MOD(x,y)	Returns x to modulo y
SIN(x)	Returns the sine of angle x, where x is in radians.
COS(x)	Returns the cosine of angle x, where x is in radians.

INTRINSIC STRING FUNCTIONS

CAT$(s1,s2,. . .)	Returns a string which is equal to the concatenation of all the strings in the parameter list.
COP$(s1,x,y)	Returns a copy of a portion of the string s1, y consecutive characters, starting with the character position x.
DEL$(s1,x,y)	Returns the contents of the string s1 with y consecutive characters deleted, starting with character position x.
INS$(s1,s2,x)	Returns the contents of s2 with s1 inserted immediately before character position x.
LEN$(s1)	Returns the length of the string s1.
POS$(s1,s2)	Returns an integer that is equal to the position of the first character in the first occurrence of the string s1 in the string s2.

OTHER FUNCTIONS

ORD(s)	Returns the ASCII value of the first character of the string(s).

STR$(x) Returns the string containing the character associated with the ASCII value x.

GET$ Reads a single character from the keyboard without prompt or echoing, and returns it as a string; no arguments are required.

OLD(c,s)
NEW(c,s) The numeric constant c (no fraction part) becomes associated with the disk file whose name is in s. OLD expects the file to already exist; NEW creates a new one with the name s, removing any previous file of that name. These functions must occur before related print or input statements. The numbers may be reassigned, and must be in the range 0 to 16. For best results, use only at the beginning of a program. ".TEXT" must be appended to the file name for the file to be edited by either of the system editors. These functions return IORESULT.

PROGRAMMING STATEMENTS

Arithmetic statements and operations

-,+	subtract,add
/,*	divide,multiply
^,**	exponentiation

Relational operators

=	equals
<>,><	not equals
>	greater than
<	less than
>=,=>	greater than or equal
<=,=<	less than or equal

INPUT list
INPUT #c list Inputs from the main system device, usually the keyboard. If the optional #c is present, input is from the disk file number c. The input list may contain any combination of real and string variables. When a program expects input, the prompt "?" is printed. Input of real numbers may be terminated with any non-numeric character. Input of strings must be terminated with a <ret>.

determined by the variable name. The array indices are 0..n1,0..n2,. . . Both real and string multi-dimensional arrays can be used. If no dimensions are declared, they are assumed to be 0..10,0..10,0..1, 0..1,. . . The number of dimensions automatically declared depends on the number that are used in the program, but must be constant over all uses of the array.

GOSUB linenumber Executes a subroutine call. The calling address is placed on the subroutine stack. Subroutine calls may be recursive.

RETURN Returns to the line after the last GOSUB that is still pending. It pops the top address off the stack and uses it as the return address. A RETURN when no GOSUBs are pending is an error.

GOTO linenumber Program execution jumps to the given line number.
REM text Remark line.

2.8.2 UCSD Basic Enhancements

The following are unique to UCSD Basic:

- For loops, var=exp1 is done before exp2 or exp3 are evaluated.
- Continuation of statements is allowed. Any line not beginning with a line number is assumed to be a continuation.
- All parameter functions are called by value. Parameters cannot be used to return values from a function. Function calls are allowed to be recursive.
- Arrays of more than two dimensions are allowed. String functions and procedures are those found in the UCSD Pascal language.
- Tab stops are not allowed in printing. All list elements are printed without spaces between them. The carriage return can be suppressed by ";" as the last symbol of the line.
- Subroutines may be recursive.
- In-line comments may be inserted. The portion of any line following "@" is ignored by the Compiler.
- The code of PASCAL FUNCTIONs may be added to the Basic Compiler as new standard Basic functions. This is accomplished by a straight-forward addition to the Basic Compiler.

2.9 LINKER

The Linker allows the user to combine pre-compiled Pascal files so that they

may be executed as one file. Normally, the pre-compiled files are resident in the file *SYSTEM.LIBRARY and are combined with the current work file.

In writing programs that utilize pre-compiled routines or subprograms, the user must declare them in the calling program to be EXTERNAL, or as SEGMENT PROCEDURES much as PROCEDURES or FUNCTIONS may be declared to be FORWARD (see Section 3.3 for further details on segmenting a program). The Compiler will then inform the system that linking is required before execution. The Linker also can be used to link in UNITs, groups of routines that will be used together to perform a common task. Any files that reference UNITs or EXTERNAL routines and have not yet been linked may be compiled and saved, but must be linked before execution.

2.9.1 Using the Linker

The Linker may be entered by typing either "L" for L(inker for "R" for R(un when in the Outer Level of commands. The Linker must be invoked explicitly in the following cases:

- If the file into which the routines are to be linked is not the work file.
- The external routines to be linked reside in library files other than *SYSTEM.LIBRARY.

When "L" is typed, the Linker responds with the following prompt:

```
Host file?
```

The host file is the file into which the routines or units are to be linked. If the work file is to be used, an asterisk and a return is typed rather than a file name. Any file name entered will automatically be appended with ".CODE" by the Linker. The Linker than asks for the name(s) of the library file(s) in which the units or external routines are to be found:

```
Lib file? (codefile identifier)
```

Up to eight library file names may be entered. Typing an asterisk (*) and a return will cause the Linker to reference the *SYSTEM.LIBRARY. The Linker notifies the user about each library file that is successfully opened; for example:

```
Lib file? * (ret)

Opening *SYSTEM.LIBRARY
```

When all library file names have been entered, the user must type a return to proceed. The Linker then prompts with:

```
Map file? (file identifier) (ret)
```

The Linker writes the map file to the file requested. The map file contains information relevant to the linking process. Responding with a return will suspend this option. Unless a period is the last letter of the file name, the

Linker will automatically append ".TEXT" to the name.

After the Linker has read all of the segments required to enable the linking process, it prompts the user for the destination file name for the linked code output:

```
Destination File?
```

The destination file often will be the same as the host file. Linking will begin after a (return) is typed following the output file name. Simply a (return) only causes the output file to be placed on the work file, #SYSTEM.WRK.CODE.

During the linking process, the Linker will report on the operator's console all segments being linked and all external routines being copied into the output file. The linking process will be aborted if any required segments or routines are missing or undefined. The user will be informed by one of the following messages, as appropriate:

```
Unit <identifier> undefined

Proc <identifier> undefined

Func <identifier> undefined

Global <identifier> undefined

Public <identifier> undefined
```

When typing "R" for R(un, if the program in the work file contains EXTERNAL declarations or uses UNITs, the Linker is automatically invoked after the Compiler. The Linker will search the #SYSTEM.LIBRARY for the routines or units specified and will attempt to link them.

If the routine or unit is not in the #SYSTEM.LIBRARY, the Linker will respond with one of the messages given above, as appropriate.

2.9.2 Linker Conventions and Implementation

A codefile may contain up to sixteen segments. Block 0 of the program code file contains information regarding name, kind, relative address and length of each code segment. This information is called the segtable, and is formatted in a record as follows:

```
RECORD
  DISKINFO: ARRAY[0..15] OF
              RECORD
                CODELENG, CODEADDR: INTEGER
              END
  SEGNAME: ARRAY[0..15] OF PACKED ARRAY[0..7] OF CHAR;
  SEGKIND: ARRAY[0..15] OF (LINKED,HOSTSEG,SEGPROC,UNITSEG,
                            SEPRTSEG);
  TEXTADDR: ARRAY[0..15] OF INTEGER;
END
```

CODELENG gives the length of the segment in words, and CODEADDR gives the block address. A description of SEGKIND follows:

LINKED	The code segment is fully executable. Either all external references have been resolved or none were present.
HOSTSEG	The outer block of a Pascal program if the program has external references.
SEGPROC	A Pascal segment procedure.
UNITSEG	A compiled segment.
SEPRTSEG	A separately compiled procedure of function (e.g., assembly language code files or Pascal UNITs that are not SEGMENT UNITs.

If a segment contains unresolved external references, the Compiler generates linker information. This information is in a series of variable-length records, one for each UNIT, routine or variable that is referenced in but is external to the source. The first eight words of each record contain:

```
LIENTRY=RECORD
        NAME: ALPHA;
        CASE LITYPE: LITYPES OF
          UNITREF,
          GLOBREF,
          PUBLREF,
          PRIVREF,
          SEPPREF,
          SEPFREF,
          CONSTREF:
            (FORMAT: OPFORMAT;          format of lientry; name can be BIG,
                                        BYTE, or WORD
             NFEFS: INTEGER;            # of references to lientry name in
                                        compiled code segment
             NWORDS: LCRANGE);          size of privates in words
          GLOBDEF:
            (HOMEPROC: PROCRANGE;       which procedure it is in
             ICOFFSET: ICRANGE);        byte offset in p-code
          PUBLDEF:
            (BASEOFFSET: LCRANGE);      compiler-assigned word offset
          CONSTDEF:
            (CONSTVAL: INTEGER);        user's defined value
          EXTPROC, EXTFUNC,
          SEPROC, SEPFUNC:
            (SRCPROC:  PROCRANGE;       procedure # in source segment
             NPARAMS: INTEGER);         # of parameters expected
          EOFMARK:
            (NEXTBASELC: LCRANGE);      private var allocation info
          END(lientry)
```

If the LITYPE is one of the first case variant, a list of code pointers into the code segment follows this portion of the record. Each pointer is the absolute byte address within the code segment of a reference to a variable, UNIT or routine named in lientry. These are 8-word records; but only the first NREFs are valid.

2.10 DEBUGGER

The Interactive Debugger is included in the Pascal Operating System to facilitate debugging the Pascal program. For optimum use of the Debugger, two Compiler options should be turned on: D+ and L+. The D+ option generates breakpoint instructions within a program as it is being compiled. The breakpoints are necessary for the use of the Crawl, Walk or Breakpoint commands. The L+ option writes a compiled source listing of the program on disk. The Debugger uses this listing (file name *SYSTEM.LST.TEXT) while in the CRAWL or WALK mode or when a breakpoint is executed.

Both options have their drawbacks. D+ causes a sightly larger code file to be created; L+ requires space on disk. However, these options can be turned on and off as needed so that they can be activated for troublesome pieces of code only, if desired.

A sample program to be debugged is shown in Figure 2-12. In the compiled source listing shown, the first column contains the line numbers, the second has the segment numbers, and the third, the procedure numbers. In the procedure number column, as asterisk (*) after the number indicates that the line has at least one conditional halt (breakpoint) associated with it; otherwise, a colon (:) appears. The letter following the asterisk or colon indicates whether the offset represents a code (C) or data (D) offset. If the offset is "C", the offset for that procedure is given as the first instruction generated for the line. If the offset is "D", the number given as the first instruction represents the word offset in the data area where storage for that line of the procedure begins.

```
 1   1    1:D     1 (*$D+,LDEBUG.TEXT*)
 2   1    1:D     1 PROGRAM DEBUG;
 3   1    1:D     3 VAR A : INTEGER;
 4   1    1:D     4
 5   1    2:D     1 PROCEDURE DIVIDE;
 6   1    2:D     1 VAR B : REAL;
 7   1    2:0     0 BEGIN (*DIVIDE*)
 8   1    2*1     0  B := 5/A;
 9   1    2*0    11 END (*DIVIDE*);
10   1    2:0    26
11   1    1:0     0 BEGIN (*DEBUG*)
12   1    1*1     0  A := 0;
13   1    1*1     7  DIVIDE;
14   1    1*0    11 END (*DEBUG*).
```

The debugger is entered by typing "D" for D(ebug, rather than "R" for R(un, while in the Outer Level of commands for the Pascal Operating System. If the program work file has not been compiled, the Compiler will be called first, automatically. However, if a runtime error occurs during compilation, or if a breakpoint or halt is encountered, Debugger is called.

After "D" is typed, the Debugger displays a message giving the release number and date of the release:

```
PASCAL INTERACTIVE DEBUGGER - January 1978
```

The Debugger is in EXAMINE mode when entered. This mode is used to peruse portions of memory, set or clear breakpoints, resume execution of the program, or enter WALK or CRAWL mode to execute the program one statement at a time. The execution options are prompted by:

```
>EXAMINE: 1.. ( (links, M(ove, (, ), L(ink, D(ata, S(tack, H(eap,
E(rase, U(pdate, (crtl-U(p), (crtl-D(own), C(rawl, W(alk, R(esume,
(esc)
```

R(esume runs the program normally until a BREAK or breakpoints are encountered or a non-fatal runtime error occurs. C(rawl puts the program into the CRAWL mode to execute one statement at a time, waiting for input from the user between steps. W(alk puts the program into WALK mode to execute the program one statement at a time at an adjustable rate. The other commands are described later in this section. Whenever the EXAMINE mode is entered, the prompt line appears. If entered as a result of an execution error, additional information is given, as shown in Figure 2-13.

```
(*$D+,LDEBUG.TEXT*)
PROGRAM DEBUG;
VAR A : INTEGER;

PROCEDURE DIVIDE;
VAR B : REAL;
BEGIN (*DIVIDE*)
 B := 5/A;
END (*DIVIDE*);

BEGIN (*DEBUG*)
 A := 0;
 DIVIDE;
END (*DEBUG*).
```

Figure 2-13. EXAMPLE OF ENTERING EXAMINE MODE

The bottom line gives the reason why the EXAMINE mode was entered. It may be some type of execution error, a user break, termination of WALK or CRAWL mode, or execution of a breakpointed statement. In the case shown above, it was a floating point error (a divide by zero, to be specific). The procedure in which the error occurred is given by Proc and Seg.

In the CRAWL mode, information about a statement is given prior to its execution. If *SYSTEM.LST.TEXT exists, the compiled listing line containing

the statement is displayed. Otherwise, the information displayed includes the line number, the number of the segment and procedure, and the code offset of the first instruction. The user then has two options.

If <space> is typed, the Debugger will execute the line and continue. If "Q" is typed, the Debugger will leave the CRAWL mode and reenter the EXAMINE mode.

As in the CRAWL mode, when in the WALK mode information is displayed prior to execution of the statement. On typing "W" to enter the WALK mode, the following prompt line appears:

DELAY:

The user then enters an integer that the Debugger uses as the number of seconds to delay between executing each statement in the program. The BREAK key is used to reenter the EXAMINE mode.

2.10.2 Commands

An example of how the commands are used to debug a program is given in Figure 2-14. The commands are described below.

LINKS

Entering a number between 0 and 9 gives the number of links to move up or down the dynamic or static chain. The direction is determined by the first character of the EXAMINE prompt line. A forward arrow indicates that movement will be in the direction of the older calls (if dynamic) or ancestors (if static). The reverse arrow indicates that movement will be towards more recent calls. The type of links to be traversed, STATIC or DYNAMIC, is specified to the right of DEFAULTLINK (as shown in the example in Figure 2-12).

NOTE: Movement towards descendents is not allowed.

MOVE

Typing "M" specifies the M(ove command that is used to find a specified procedure and make it the current procedure. This command has two parameters:

PROC

Procedure number of the desired procedure. Default is the number of the bombed procedure (the one at the bottom of the call chain). Typing <return> will give the default and bypass the normal search.

SEG

Segment number of the desired procedure. Typing <return> will give the default segment (which must be preceded by a <return> for the default procedure).

Figure 2-14. USING DEBUGGER COMMANDS TO DEBUG A PROGRAM

After the procedure and segment numbers have been entered, the Debugger will search up the dynamic links starting at the caller of the current procedure. Note that this implies that one can never move to the current procedure because the Debugger will not find it. If the specified procedure is found, it becomes the current procedure and the information in the prompt line will be updated. Otherwise, the current procedure remains unchanged.

<

Typing "<" or "," changes the direction of link traversal (movement) to be down the call chain (i.e., go towards the callees).

>

Typing ">" or "." changes the direction of link traversal to be up the call chain (i.e., go towards the callers).

LINK

Typing "L" specifies the L(ink command to toggle the DEFAULTLINK from DYNAMIC to STATIC, and vice versa.

DATA

Typing "D" specifies the D(ata command to examine the data and parameter segment of a procedure. The Debugger prompts for four parameters:

The only parameters that need to be entered are those other than the defaults. Typing (cr) at any point tells the Debugger to use the default values for the remaining parameters. Typing (space) delimits a parameter and lets the Debugger prompt for the next one. The parameters are:

OFFSET

Default value is the last offset displayed plus 1. Beginning value is 1. The offset may be changed by entering an integer.

LENGTH

The beginning default value is the minimum of the text buffer size (15 for 24-line screens) and DATA plus PARAM. After that, it is the last length specified in the D(ata or S(tack command. LENGTH determines the number of words to be displayed.

PROC

The default value is the number of the current procedure. Any procedure may be specified that is higher in the call chain.

SEG

The default value is the segment to which the current procedure belongs.

When the Debugger finds the specified procedure, it will display the data, wrapping around to the top of the screen and erasing information in the memory display data bugger, if necessary.

When an offset to be displayed is larger than PARAM plus DATA for a procedure, the message is generated at the bottom of the screen:

Warning - offset too large

The invalid data will not be displayed.

STACK

Typing "S" specifies the S(tack command that is used for examining the stack area belonging to a specific procedure. Parameters are specified in the same way as for the DATA command; but the first offset is 0, not 1.

HEAP

Typing "H" specifies the H(eap command that displays a portion of memory specified by an octal address and a length.

ERASE

Typing "E" specifies the E(rase command that clears the memory display buffer on the screen.

UPDATE

Typing "U" specifies the U(pdate command that refreshes the memory display buffer. S(tack, D(ata and H(eap commands save the procedure numbers and offsets displayed in the memory buffer. When "U" is typed, the buffer is erased. The saved numbers are used to locate any information belonging there. UPDATE is not able to refresh any of the information that belongs to procedures that are below the current procedure in the call chain. It will generate the message:

Proc not found.

CRTL-U

Typing "U" specifies the (CRTL-U(p) command that moves the asterisk (*) up one line.

CTRL-D

Typing "D" specifies the (CRTL-D(own) command that moves the asterisk (*) down one line.

CRAWL

Typing "C" specifies the C(rawl command that resumes execution of the program in CRAWL mode at the point in the program where the Debugger was invoked.

WALK

Typing "W" specifies the W(alk command that resumes execution of the program in the WALK mode starting where the Debugger was invoked.

RESUME

Typing "R" specifies the R(esume command that resumes normal execution of the program where the Debugger was invoked.

ESCAPE

Typing (esc) specifies escape, return to the Outer Level of commands.

CARRIAGE RETURN

Typing (cr) specifies a carriage return that clears the line with the asterisk (*) and moves down one line.

BREAKPOINT

Typing "S" for S(et or "C" for C(lear sets or clears a breakpoint. They both require line numbers.

SET

Enter a line number of a line that has an asterisk in the compiled listing. Whenever a statement in that line is about to be executed, the Debugger will be called.

CLEAR

Enter (cr) to clear all breakpoints or enter the line number of an active breakpoint.

SECTION 3

PASCAL PROGRAMMING CONSIDERATIONS

Many aspects of the Pascal Operating System need to be considered when programming in Pascal because they have an influence on how a program should be written. These aspects are described in this section.

3.1 INTRINSICS

Users of intrinsics should be fluent in Pascal and experienced in the use of the Operating System. All necessary range and validity checks are the responsibility of the user. Some intriniscs do no range checking. Those which are particularly dangerous are noted in their descriptions.

The required parameters are listed along with the function/procedure identifier. Optional parameters are in square brackets []. The default values are in metabrackets {} on the line below. Within each subsection, functions and procedures are given in alphabetic order.

The following terms are used in the explanation of the Intrinsics:

ARRAY : a PACKED ARRAY OF CHARacters

BLOCK : one disk block, (512 bytes)

BLOCKS : an INTEGER number of blocks

BLOCKNUMBER : an absolute disk block address

BOOLEAN : any BOOLEAN value

CHARACTER : any expression which evaluates to a character

DESTINATION : a PACKED ARRAY OF CHARacters to write into or a STRING, context dependent

EXPRESSION : part or all of an exprssion, to be specified

FILEID : a file identifier, must be

```
VAR fileid: FILE OF (type);
        or  TEXT;
        or  INTERACTIVE;
        or FILE;
```

INDEX : an index into a STRING or PACKED ARRAY OF CHARacters, context dependent or as specified.

NUMBER : a literal or identifier whose type is either INTEGER or REAL.

RELBLOCK : a relative disk block address, relative to the start of the file in context, the first block being block zero.

SIMPLVARIABLE	: any declared PASCAL variable which is of one of the following TYPEs: BOOLEAN CHAR REAL STRING or packed array [..] OF CHAR
SIZE	: an INTEGER number of bytes or characters; any integer value
SOURCE	: a STRING or PACKED ARRAY OF CHARacters to be used as a read-only array, context dependent or as specified.**
SCREEN	: an array 9600 bytes long; or as needed
STRING	: any STRING, call-by-value unless otherwise otherwise spcified, ie. may be a quoted string, or string variable or function which evaluates to a STRING
TITLE	: a STRING consisting of a file name
UNITNUMBER	: physical device numbr used to determine device handler used by the interpreter
VOLID	: a volume identifier, STRING [7]

** in string intrinsics, SOURCE is going to have to be a string, in intrinsics that deal with packed arrays of characters, it may be either. A word of caution about using STRINGs in intrinsics that expect character arrays, the zeroeth element of the string is the length byte, which may cause the programmer some unexpected problems were he not aware of that fact.

3.1.1 Character Array Manipulation Intrinsics

The Character Array Manipulation Intrinsics are byte oriented. No range checking of any sort is performed on the parameters passed to them; so handle with care. The user must know what he is doing because the system does not protect itself from these operations. Examples are shown in Figure 3-1. The intrinsic SIZEOF (Section 3.1.4) is meant for use with these intrinsics to "remember" the number of bytes of a parameter.

Example of SCAN:

```
PROGRAM SCANTEST;
VAR EX : PACKED ARRAY[0..37] OF CHAR;
    I  : INTEGER;

BEGIN (*SCANTEST*)
 EX := ' EXAMPLE OF CHARACTER ARRAY INTRINSICS';
 I := SCAN(-25,=':',EX[25]);
 WRITELN (I);
```

```
 I := SCAN(100,<>' ',EX[0]);
 WRITELN (I);
END (*SCANTEST*).
```

Examples of MOVELEFT
MOVERIGHT

```
PROGRAM MOVETEST;
VAR BUF1 : PACKED ARRAY [0..19] OF CHAR;
    BUF2 : PACKED ARRAY [0..20] OF CHAR;

BEGIN (*MOVETEST*)
 BUF1 := 'MOVE CHARACTERS LEFT';
 BUF2 := 'THESE CHARACTERS.....';
 MOVELEFT(BUF1,BUF2,5);
 WRITELN (BUF2);
END (*MOVETEST*).
```

Figure 3-1. EXAMPLES OF CHARACTER ARRAY MANIPULATION INTRINSICS

```
FUNCTION SCAN (LENGTH, PARTIAL EXPRESSION, ARRAY) : INTEGER;
```

This function returns the number of characters from the starting position to where it terminated. Termination comes when matching the specified LENGTH or satisfying the EXPRESSION. The ARRAY should be packed and may be subscripted to denote the starting point. If the EXPRESSION was satisfied on the character at which ARRAY is pointed, the value returned will be zero. If the LENGTH passed was negative, the number returned will be negative and the function will have scanned backward. The PARTIAL EXPRESSION must be in the following format:

"<>" or "=" followed by character expression

```
PROCEDURE FILLCHAR (DESTINATION, LENGTH, CHARACTER);
```

This procedure takes a (subscripted) packed array of characters and fills it with the number (LENGTH) of CHARACTERs specified. This can be done using a MOVELEFT procedure (described below); but FILLCHAR is twice as fast because no memory reference is needed for the source. FILLCHAR will optimize word moves only if the DESTINATION is below the I/O page. Word moves are not done to the I/O page because some hardware relies on byte addressing in this address space.

```
PROCEDURE MOVELEFT (SOURCE, DESTINATION, LENGTH);
PROCEDURE MOVERIGHT (SOURCE, DESTINATION, LENGTH);
```

These procedures do mass moves of bytes for the LENGTH specified. MOVELEFT starts from the left end of the SOURCE and moves bytes to the

left end of the DESTINATION, traveling right. MOVERIGHT starts from the right end, traveling left. Both are needed when working on a single array in which the order of the characters moved is critical.

MOVERIGHT never attempts to optimize word moves. MOVELEFT will optimize only if the DESTINATION is at an address below the I/O page. Word moves are not done to the I/O page because some hardware relies on byte addressing in this address space.

3.1.2 I/O Intrinsics

```
PROCEDURE CLOSE (FILEID OPTION);
```

OPTIONS include ", LOCK", ", NORMAL", ", PURGE" and ", CRUNCH". Note the commas.

A normal CLOSE is done when the OPTION is null. CLOSE simply sets the file state to closed. If the file was opened using REWRITE and is a disk file, it is deleted from the directory.

The LOCK option will cause the disk file associated with the FILEID to be made permanent in the directory if the file is on a directory structured device and the file was opened with a REWRITE; otherwise a normal close is done.

The PURGE option will delete the title associated with the FILEID from the directory. The unit will go off-line if the device is not block-structured.

The intent of the CRUNCH option is to lock a file with a minimum number of blocks of useful information. (This option is currently undefined.)

Regardless of option, all CLOSEs will mark the file closed and will make the implicit variable FILEID^ undefined. CLOSEing an already closed file causes no action.

```
FUNCTION EOF (FILEID) : BOOLEAN;
FUNCTION EOLN (FILEID) : BOOLEAN;
```

EOF and EOLN return False after the file specified is reset. They both return True on a closed file. If FILEID is not present, the fileid INPUT is assumed (e.g., IF EOF THEN. . .). When EOF (FILEID) is True, FILEID^ is undefined.

When GET (FILEID) sets FILEID^ to the EOLN or EOF character, EOLN (FILEID) will return True, and FILEID^ (in a FILE OF CHAR) will be set to blank.

While doing puts or writes at the end of a file, if the file cannot be expanded to accommodate the PUT or WRITE, EOF (FILEID) will return True.

```
PROCEDURE GET (FILEID);
PROCEDURE PUT (FILEID);
```

GET (FILEID) will leave the contents of the current logical record pointed at by the file pointers in the implicitly declared window variable FILEID^ and increment the file pointer.

PUT (FILEID) puts the contents of FILEID^ into the file at the location of the current file pointers and then updates those pointers.

Both procedures are used on typed files, files for which a type is specified in the variable declaration (i.e., "FILEID : FILE OF type "). Untyped files are simply declared as " FILEID: FILE;". "F: FILE OF CHAR" is equivalent to "F: TEXT". In a typed file, each logical record is a memory image fitting the description of a variable of the associated (type).

```
FUNCTION IORESULT : INTEGER;
```

After any I/O operation, IORESULT contains an INTEGER value that corresponds to the values given in Appendix B.3.

```
PROCEDURE PAGE (FILEID);
```

PAGE (FILEID) sends a top-of-form (ASCII FF) to the file.

```
PROCEDURE READ(LN) (FILEID, SOURCE);
PROCEDURE WRITE(LN) (FILEID, SOURCE);
```

These procedures may be used only on TEXT (FILE OF CHAR) or INTERACTIVE files for I/O. The three types of INTERACTIVE files are INPUT, OUTPUT and KEYBOARD. INPUT results in echoing of characters typed to the console. OUTPUT allows the user to halt or flush the output. KEYBOARD does no echo; it allows the programmer complete response to user typing.

If "FILEID," is omitted, INPUT or OUTPUT (as appropriate) is assumed. A READ (STRING) will read up, but not including, the end-of-line character (carriage return) and leave EOLN (FILEID) True. This means that any subsequent reads of string variables will return the null string until a READLN or READ (character) is executed.

```
PROCEDURE RESET (FILEID, [TITLE]);
PROCEDURE REWRITE (FILEID, TITLE);
```

These procedures open files for reading and writing and mark the file as open. The FILEID may be any Pascal-structured file. TITLE is a string containing any legal file title. REWRITE creates a new file on disk for output files; RESET marks an already existing file open for I/O. If the device specified is a non-directory-structured device (e.g., PRINTER), the file is opened for input, output or

both, in either case.

If the file is already open when the RESET or REWRITE is attempted, an error is returned in IORESULT. The state of the file remains unchanged.

RESET (FILEID) without an optional string parameter rewinds the file by setting the file pointers back to the beginning (0 record) of the file. The boolean functions EOF and EOLN will not be set by the implied GET in RESET.

With files of the INTERACTIVE type, these functions act differently. On files of other types, RESET will do an initial GET to the file, setting the window variable to the first record in the file. On INTERACTIVE files, RESET will not do the GET.

```
PROCEDURE SEEK (FILEID, INTEGER);
```

SEEK changes the file pointers so that the next GET or PUT uses the INTERGERth record of FILEID. Records in files are numbered starting with 0. A GET or PUT must be executed between SEEK calls because two consecutive SEEKs may cause unpredictable junk to be held in the window and associated buffers.

```
FUNCTION UNITBUSY (UNITNUMBER) : BOOLEAN
```

This function returns a boolean value. If the value is True, the device specified is waiting for an I/O transfer to complete. For example:

```
UNITREAD (1(*CONSOLE*), CH[0], 1(*1character), 1(*ASYNCH*));
WHILE UNITBUSY (1) (*while read hasn't taken place*) DO
WRITELN ('Please type a character.');
```

Execution of the example will result in the continuous ouptut of the line 'Please type a character' until a character is typed.

```
PROCEDURE UNITCLEAR (UNITNUMBER);
```

This procedure cancels all I/Os to the specified unit and resets the hardware to its power-up state.

```
PROCEDURE UNITREAD (UNITNUMBER, ARRAY, LENGTH, [BLOCKNUMBER], [INTEGER]);
PROCEDURE UNITWRITE (UNITNUMBER, ARRAY, LENGTH, [BLOCKNUMBER], [INTEGER]);
                                                 {SEQUENTIAL}      {0}
```

These procedures are dangerous because no range checking is done.

These are the low-level procedures that do I/Os to various devices. The UNITNUMBER is the integer name of the device. ARRAY is any declared packed array. It may be subscripted to indicate a starting position to do the transfers from/to. LENGTH is an integer giving the

number of bytes to transfer. BLOCKNUMBER is required only when using a block-structured device, and is the absolute block number where the transfer will start from/to. If omitted, BLOCKNUMBER is assumed to be 0. The INTEGER value is optional and assumed to be 0. If 1, it indicates that the transfer is asynchronous. If BLOCK- NUMBER is omitted, but INTEGER is included, a comma is used to hold the placement of parameters.

PROCEDURE UNITWAIT (UNITNUMBER);

This procedure waits for the specified device to complete the I/O in progress.

3.1.3 String Intrinsics

To maintain the integrity of the LENGTH of a string, only string functions or full-string assignments should be used to alter strings. Moves and single-character assignments do not affect the length of a string, which means that the programmer must do range checking. The individual elements of STRING are of CHAR type and may be indexed 1. . LENGTH(STRING). Accessing the string outside this range will have unpredictable results if range-checking is off, or may cause a runtime error if range-checking is on.

Examples of String Intrinsics are given in Figure 3-2.

FUNCTION CONCAT (SOURCEs) : STRING

This function returns a string that is the concatenation of all the strings passed to it. There may be any number of source strings, separated by commas.

FUNCTION COPY (SOURCE, INDEX, SIZE) : STRING

This function returns a string containing SIZE characters copied from SOURCE starting at the INDEXed position.

FUNCTION LENGTH (STRING) : INTEGER

This function returns the integer value of the length of STRING.

```
PROGRAM STRINTST;

VAR name,text,pattern,first,second,third : STRING;
    start,get,toomany,more : STRING;
    long : INTEGER[8];
    I : INTEGER;

BEGIN (*STRINTST*)

 I := LENGTH('ABC');
 WRITELN (I);
 name := 'JOHN SMITH';

 I := LENGTH(name);
 WRITELN(I);
 text := 'THIS IS AN EXAMPLE OF STRING INTRINSIC';
 pattern := 'EXA';

 I := POS(pattern,text);
 WRITELN(I);

 first := 'ABCDE';
 second :='FGHIJ';
 third := CONCAT(first,second);
 WRITELN (third);

 start := 'HERE IS A STRING OF CHARACTERS';
 get := COPY(start,POS('C',start),10);
 WRITELN(get);

 toomany :='THIS STRING HAS TOO MANY CHARACTERS';
 DELETE(toomany,17,9);
 WRITELN(toomany);

 more :=' TOO MANY';
 INSERT(more,toomany,16);
 WRITELN(toomany);

 long := 1000000;
 STR(long,more);

 WRITELN('$',more);
END(*STRINTST*).
```

Figure 3-2. EXAMPLES OF STRING INTRINSICS

```
FUNCTION POS (STRING, SOURCE) : INTEGER
```

This function returns the position of the first occurrence of the pattern (STRING) to be scanned in SOURCE. The INTEGER value of the first position in the matched pattern will be returned. If the pattern was not found, zero will be returned.

```
PROCEDURE DELETE (DESTINATION, INDEX, SIZE) : STRING
```

This procedure deletes SIZE characters from DESTINATION starting at the INDEXed position.

```
PROCEDURE INSERT (SOURCE, DESTINATION, INDEX)
```

This procedure inserts SOURCE into DESTINATION starting with the INDEXed position in DESTINATION.

3.1.4 Miscellaneous Intrinsic Routines

```
PROCEDURE GOTOXY (XCOORD, YCOORD);
```

This procedure sends the cursor to the specified coordinates. The upper left corner of the screen is assumed to be 0,0. This procedure defaults to a Datamedia-terminal. For systems other than Datamedia or Terak 8510a, a new GOTOXY must be bound in (see Section 4.10).

```
PROCEDURE HALT;
```

This procedure generates a HALT opcode that causes a non-fatal runtime error to occur. When HALT is executed, the Debugger is invoked. If the Debugger is not in core when a HALT occurs, a fatal runtime error will occur (#14).

```
FUNCTION LOG (NUMBER) : REAL;
```

This function returns the log base ten of NUMBER.

```
PROCEDURE MARK (VAR HEAPPTR: ^INTEGER);
PROCEDURE RELEASE (VAR HEAPPTR: ^INTEGER);
```

These procedures allocate and return heap memory allocations to the system. HEAPPTR is of type ^INTEGER. MARK sets HEAPPTR to the current top- of-heap. RELEASE sets the top-of-heap pointer to HEAPPTR.

```
FUNCTION PWROFTEN (EXPONENT: INTEGER) : REAL;
```

This function returns the value of ten to the EXPONENT power. EXPONENT must be an integer in the range of 0 through 37.

```
FUNCTION SIZEOF (VARIABLE OR TYPE IDENTIFIER): INTEGER;
```

This function returns the number of bytes that a parameter occupies in

the stack. It is used with the FILLCHAR and MOVExxxx intrinsics.

```
PROCEDURE TIME (VAR HIWORD, LOWORD: INTEGER);
```

This procedure returns the current value of the system clock. The value is given in 60ths of second, assuming a 16-bit integer size and a 32-bit clock word. HIWORD contains the most significant portion. Both HIWORD and LOWORD must be VARiables of type INTEGER.

WARNING: The sign of LOWORD may be negative because the time is represented as a 32-bit unsigned number. This function currently is undefined.

3.2 FILES

A file may be defined as a body of information that is stored on an I/O device. A file is referenced by the Pascal program and the Pascal Operating System by the file name. The suffix of the file name is dependent on file type. The following types of files are used by the Pascal Operating System:

Reserved Suffix	Contents of File
.TEXT	Human-readable text
.CODE	Machine-executable code
.DATA	Data file
.FOTO	One Terak screen image
.BAD	A physically damaged area of disk

3.2.1 Text Files

The text file is composed of 1024-byte pages, where a page is defined as:

```
(⟦DLE⟧⟦indent⟧⟦text⟧⟦CR⟧⟦DLE⟧⟦indent⟧⟦text⟧⟦CR⟧. . .⟦nulls⟧)
```

Data Link Escapes are followed by an indent code, which is a byte that contains the value 32+ (number to indent). The nulls at the end of the page follow a carriage return in all cases. They pad to the end of the page to give the Compiler integral numbers of lines on a page. The DLE and indent code are optional.

The first page of a text file is the header page that is reserved for information for the Text Editor. When a user program opens a text file and REWRITEs or RESETs it with a title ending in ".TEXT", the I/O subsystem will create, then skip over, the header page. This page facilitates users in editing their I/O data. The File Handler will transfer the header page only on a disk-to-disk transfer, and will omit it on a transfer to a serial device (e.g., to a PRINTER or CONSOLE).

3.2.2 Code Files

The first block of information in a code file describes the code kept in the file. Heading the block is an array of 16 word pairs, a pair for each segment

on the disk. The first word of the pair gives the block number within the segment where code begins. The second word gives the number of bytes of code located there.

Following this array is a series of 16 eight-character arrays that describe the segments by name. These eight characters identify the segment at compile time.

Then follows a 16-word array of state descriptors. The values in this array tell what kind of segment is at the described location. The values are:

```
LINKED
HOSTSEG
SEGPROC
UNITSEG
SEPRTSEG
```

The remaining 144 words of the block are reserved for future use.

3.2.3 Data Files

The content and format of the data files are up to the user.

3.2.4 Foto Files

Foto files contain screen images. Each one is declared in Pascal as follows:

```
TYPE SCREEN = PACKED ARRAY[0..239,0..319] OF BOOLEAN;
VAR FOTOFILE: PACKED FILE OF SCREEN
```

3.2.5 Bad Files

Bad files are those files that protect a user from using a physically bad block of disk. They are marked bad by the File Handler after a bad block scan has been done and the bad blocks have been examined (see Section 2.6).

3.2.6 Work File

In addition to the permanent files described above, the Pascal Operating System supports a work file that is a temporary copy of the file being modified. The work file is used by the File Handler, Editor and Compiler. When the text part of a work file is changed, the system stores it on disk as "#SYSTEM.WRK.TEXT". When the code version is first created, it is named "#SYSTEM.WRK.CODE".

3.2.7 Volumes

A volume is any I/O device. A block-structured device is one that can have a directory (e.g., disk). A non-block-structured device does not have an internal structure. It simply produces or consumes a stream of characters (e.g., printer and console). Table 3-3 gives the volume names reserved for non-block-structured devices, the unit number associated with each device, and the unit numbers associated with the system and alternate disks.

FIGURE 3-3 I/O DEVICES

UNIT NUMBER	VOLUME ID	DESCRIPTION
1	CONSOLE:	Screen and keyboard with echo
2	SYSTERM:	Screen and keyboard without echo
4	<volume name>:	System disk
5	<volume name>:	Alternate disk
6	PRINTER:	Line Printer
8	REMOTE:	Additional peripherals
9-12	<volume name>:	Additional disk drives

Volume names for block-structured devices can be assigned by the user. The name must not exceed seven characters in length and may not contain "=", "$", "?" or ",". The character "#" is the reserved volume ID of the system disk, the disk upon which the system was booted. The character ":", when used alone, is the volume ID of the default disk. The system and default disks are equivalent unless the default prefix has been changed (see Section 2.6, PREFIX). "#<unit number>" is equivalent to the name of the volume in the disk drive at the current time.

3.2.8 File Names

A legal file name may not exceed 15 characters and may not include the characters "=", "$", "?" or ",". Lower-case letters will be translated to upper case. Blanks and non-printing characters will be removed. Legal characters are the alphanumberics and the special characters "-", "/" "\", "_" and ".". Special characters normally are used to indicate heirarchic relationships between files and to distinguish related files of different types. The wild card characters "=" and "?" are used to specify subsets of the directory (see Section 2.6, File Specification).

3.3 SEGMENTS

Segmenting a program so that procedures have to be in memory only when they are in use has many advantages:

- Large pieces of one-time code (e.g., initialization procedures) can be put into a segment.
- The work can be divided among several programmers, each coding, compiling and debugging his own segment, to be linked later by the Linker program.
- A program can be configured to suit storage requirements.

A maximum of six SEGMENT procedures are available to the Pascal programmer. The disk that holds the code file for the program must be on-line and in the same drive as when the program was started whenever a SEGMENT is called.

Otherwise the system will attempt to retrieve and execute whatever information currently resides on that particular location on the disk.

SEGMENT procedures must be the first procedure declarations containing code-generating statements. Declarations of SEGMENT procedures and functions in UCSD Pascal are identical to those in standard Pascal, except that they are preceded by the reserved word "SEGMENT".

As an example, when the user wishes to put initialization procedures into a segment because they are one-time-only procedures, the declaration might be:

```
SEGMENT PROCEDURE INITIALIZE;
BEGIN
  (*  Pascal code *)
END;
```

The Linker program that can link separately compiled SEGMENTs together is described in Section 2.9.

3.4 LINKAGES

Frequently used routines and data structures can be separately compiled or assembled and can be stored in libraries until needed (see Section 4.4, Librarian). These externally compiled structures then can be integrated into the files needing their capabilities. A file that references such a structure need not compile it directly into its code file; the Linker copies the existing code of the structure into the host code file (see Section 2.9, Linker).

the INTERFACE part of the UNIT as though that part belonged to the host program itself. Because the constants, types, variables, functions and procedures declared in the INTERFACE part are global, name conflicts may arise with identifiers in the host program. The programmer may not use identifiers that are in use by the UNIT. Procedures and functions may not Use UNITs locally.

The syntax for a UNIT definition is shown in Figure 3-4. The declarations of routine headings in the INTERFACE part are similar to forward declarations; therefore, when the corresponding routines are defined in the IMPLEMENTATION part, formal parameter specifications are not repeated.

A UNIT may use another UNIT, as shown in the example in Figure 3-5. In this case, the USES declaration must appear at the beginning of the INTERFACE part.

> NOTE: Variables of type FILE must be declared in the INTERFACE part of a UNITS. A FILE declared in the IMPLEMENTATION part will cause a syntax error at compile time.

A user may define a UNIT in-line, after the heading of the host program. In this case, the user compiles both the UNIT and the host program together. Subsequent changes in either require a recompilation of both.

```
⟨Compilation unit⟩      : := ⟨Program heading⟩; ⟨Unit definition⟩;
                             ⟨Uses part⟩ ⟨Block⟩ :
                             ⟨Unit definition⟩; ⟨Unit definition⟩.

⟨Unit definition⟩       : := ⟨Unit heading⟩;
                             ⟨Interface part⟩
                             ⟨Implementation part⟩
                             End

⟨Unit heading⟩          : := Unit ⟨Unit identifier⟩ :
                             Separate unit ⟨Unit identifier⟩

⟨Unit identifier⟩       : := ⟨Identifier⟩

⟨Interface part⟩        : := Interface
                             ⟨Uses part⟩
                             ⟨Constant definition part⟩
                             ⟨Type definition part⟩
                             ⟨Variable declaration part⟩
                             ⟨Procedure heading⟩ : ⟨Function heading⟩

⟨Implementation part⟩   : := Implementation
                             ⟨Label declaration part⟩
                             ⟨Constant definition part⟩
                             ⟨Type definition part⟩
                             ⟨Variable declaration part⟩
                             ⟨Procedure and Function declaration part⟩

⟨Uses part⟩             : := Uses ⟨Unit identifier⟩
                              , ⟨Unit identifier⟩; : ⟨Empty⟩
```

Figure 3-4. SYNTAX FOR A UNIT DEFINITION

A UNIT or group of UNITs can be compiled separately and stored in a library. After compiling a host program that uses a UNIT that is stored in a library, the user must link that UNIT into the host program by executing the Linker. If a user calls R(un and an unlinked code file has been requested, the Linker will be called automatically. If X(ecute is called in such a case, the system will issue a reminder to link the code (see Section 2.2, Outer Level Commands).

If the host program has changes, the user must recompile and link in the UNIT. If the IMPLEMENTATION part is changed, the UNIT must be recompiled, and then all compilation units that use the UNIT must be relinked. Changes in the INTERFACE part require a recompilation of not only the UNIT, but of all compilation units that use it. Then, all compilation units must be relinked.

The Compiler generates Linker information in the contiguous blocks that follow a program that uses UNITs. This information includes locations of references to externally defined identifiers (see Section 2.9, Linker).

3.5 LONG INTEGERS

is suitable for business, scientific or other applications where there is a need for extended number length with complete accuracy. The four basic standard arithmetic operations (addition, subtraction, multiplication and division) are supported, as well as routines facilitating conversion to strings and standard INTEGERs. Strong type checking is enforced to reduce potential errors. I/O, in-line declaration of constants, and inclusion in structured types are fully supported and are analogous to the usage of standard INTEGERs.

LONG INTEGERS are declared by using the standard identifier "INTEGER" followed by a length attribute enclosed in square brackets. The length is given as an unsigned number, not larger than 36, that denotes the minimum number of decimal digits to be represented. In the example below, the variable Z is capable of storing up to a 12-decimal digit signed number:

```
VAR Z: INTEGER[12];
```

LONG INTEGERs may be used, generally, anywhere a REAL would be syntatically correct. However, care must be taken to ensure that sufficient words have been allocated by the declared length attribute for storage of the result of assignment or arithmetic expression statements. INTEGER expressions are implicitly converted as required upon assignment to, or arithmetic operations with, a LONG INTEGER.

However, the reverse is not true. The LONG INTEGER probably should not be used as a subrange. Examples of uses of the LONG INTEGER are shown in Figure 3-5.

Arithmetic operations that may be used in conjunction with the LONG INTEGER are:

```
+, -, *, DIV, unary plus/minus
```

On assignment, the length of the LONG INTEGER is adjusted during execution to the declared length attribute of the variable. Therefore, an interrupt (overflow) may result when the intermediate result exceeds the number of words required to store at least 37 decimal digits, or when the final result is assigned to a variable with an insufficient length attribute. All of the standard relational operators may be used with mixed INTEGER and LONG INTEGER.

The function TRUNC will accept a LONG INTEGER as well as a REAL as an argument. The function becomes TRUNC(L), where "L" is a LONG INTEGER. Interrupt (overflow) will result if "L" is greater than MAXINT.

The procedure STR(L,S) will convert the INTEGER or LONG INTEGER "L" into a string, complete with minus sign if needed, and will place it in the STRING "S".

An attempt to declare a LONG INTEGER in a parameter list other than for the routines TRUNC and STR will result in a compile-time error. The error may be

circumvented by creating a type that is called LONG INTEGER, as follows:

```
PROGRAM LINTEGER;
 VAR L : INTEGER[20];
     I : INTEGER;
 BEGIN (*LINTEGER*)
  L := 9876543210;
  L := -L;
  L := L-L;
  L := 256;
  I := TRUNC(L);
 END (*LINTEGER*).

TYPE LONG = INTEGER 15
PROCEDURE OVERSIZE(ACCOUNT: LONG);
```

Figure 3-5. EXAMPLE USES OF LONG INTEGERS

The LONG INTEGER is stored as a multi-word, twos-complement binary number. System routines do the I/O conversions as required. Maximum storage efficiency is achieved by dynamic expansion and contraction of word allocation as required. During LONG INTEGER operations, the length is placed on the stack above the number itself. Note that the declared length attribute may be equal to or less than this length.

3.6 UCSD PASCAL ENHANCEMENTS

Presented here is a summary of the areas in which UCSD Pascal differs from Standard Pascal as well as special enhancements offered by UCSD Pascal. The Standard Pascal referred to here is defined in PASCAL USER MANUAL AND REPORT (2nd edition) by Kathleen Jensen and Niklaus Wirth (Springer-Verlag, 1975). Many of the differences are in the areas of files and I/O. Some of the key differences from a programming standpoint are in EOF, EOLN, READ, WRITE, RESET and REWRITE.

3.6.1 Case Statements

In Standard Pascal, if no label is equal to the value of the case statement selector, the result of the case statement is undefined (Jensen and Wirth).

In UCSD Pascal, if no label matches the value of the case selector, the next statement executed is the statement following the case statement. An example is shown in Figure 3-8. Note that a semicolon is NOT permitted before the "END" of a case variant field declaration within a RECORD declaration. See Appendix B.2 for revised syntax diagrams for (field list).

```
PROGRAM FALLTHRU;
 VAR I : INTEGER;
```

```
BEGIN (*FALLTHRU*)
 I := 25;
 CASE I OF
  0 : WRITELN('I = 0');
  1 : WRITELN('I = 1');
 END(*CASE*);
WRITELN('NEITHER');
END (*FALLTHRU*).
```

Figure 3-6. EXAMPLE OF FALLTHROUGH IN A CASE STATEMENT

3.6.2 Comments

A comment is any text that appears between the symbols "(*" and "*)" or the symbols "{" and "}". Comments are ignored by the Compiler unless the first character of a comment is "$", in which case, the comment is interpreted to be a Compiler control comment (see Section 2.7, Pascal Compiler). Note that matching symbols must be used; they may not be mixed. This feature allows a user to nest comments. For example:

```
{ XCP := XCP + 1;  (* NESTED COMMENT *) }
```

The matching symbols have been used as pairs within pairs of different symbols. Using the same pair for nesting will result in a syntax error.

3.6.3 Dynamic Memory Allocation

In Standard Pascal, DISPOSE asks that storage occupied by one particular variable be released by the system for other uses.

In UCSD Pascal, DISPOSE is not implemented. However, it can be approximated by a combined use of the intrinsics MARK and RELEASE.

Storage is allocated for variables by the standard procedure NEW in a stack-like structure called a "heap". The program in Figure 3-9 illustrates how MARK and RELEASE can be used to change the size of the heap. As NEW is used to create a new variable, the size of the heap is augmented by the size of the variable. When the variable is no longer needed, RELEASE resets the top-of-heap address that was set originally by MARK.

A series of calls to NEW between calls to MARK and RELEASE will result in the immediate release of storage occupied by several variables at RELEASE time. Note that, due to the stack nature of the heap, it is not possible to release memory used by a single item in the middle of the heap. This is why MARK and RELASE only approximate the function of DISPOSE.

Careless use of MARK and RELEASE can lead to "dangling pointers" that point to areas of memory that are no longer a part of the defined heap space.

3.6.4 EOF(F)

When text file F is being used as an input file from the CONSOLE device, to

set EOF to True, the user must type the EOF character. The system default EOF character is control-C. (To change the default character, see Section 4.1, Setup.)

If F is closed, EOF(F) will return True for any FILE F. If F is a file of type TEXT and EOF(F) is True, then EOLN(F) is also True. After a RESET(F), EOF(F) is False. If EOF(F) becomes True (end-of-file is reached) during a GET(F) or READ(F), the data obtained is invalid.

```
PROGRAM HEAPCHNG;
 TYPE STUDENT =
        RECORD
          NAME : PACKED ARRAY [0..10] OF CHAR;
          ID   : INTEGER
        END;
 VAR S : ^STUDENT; (* '^' MEANS POINTER*)
     HEAP : ^INTEGER;

 BEGIN (*HEAPCHNG*)
   MARK(HEAP);
   NEW(S);
   S^.NAME := 'SMITH, JOHN';
   S^.ID := 2656;
   RELEASE(HEAP);
 END (*HEAPCHNG*).
```

Figure 3-7. USING MARK AND RELEASE TO CHANGE HEAP SIZE

When a user program starts execution, the system automatically performs a RESET on the predeclared files INPUT, OUTPUT and KEYBOARD.

The default file for EOF and EOLN is INPUT.

3.6.5 EOLN(F)

EOLN(F) is defined only if F is a text file. F is defined as a text file when the window variable F^ is of (type)CHAR. EOLN becomes True after reading the end-of-line character (cr). The carriage return must be typed immediately following the last character on the line. If a space is typed first, then followed by a (cr), EOLN will remain False, and another READ will take place.

3.6.6 Files

Several aspects of file handling are described below. The enhancements presented bring UCSD Pascal closer to the standard definition of the language. Note that UCSD Pascal includes untyped files that are not available to the Standard Pascal user.

WARNING: It is not currently possible to READ or WRITE to files of types

other than TEXT or FILE OF CHAR.

Interactive Files

The standard predeclared files INPUT and OUTPUT will always be defined as type INTERACTIVE. They behave exactly as do files of type TEXT. All files other than INTERACTIVE operate exactly as described in Jensen and Wirth, including the functioning of EOF(F), EOLN(F) and RESET(F). For more details concerning files of type INTERACTIVE, see Section 2.6 (READ, READLN and RESET).

Untyped Files

Untyped files are unique to UCSD Pascal. An untyped file can be thought of as a file without a window variable F^ to which all I/O must be accomplished (using BLOCKREAD and BLOCKWRITE). Any number of blocks can be transferred using either BLOCKREAD or BLOCKWRITE. These functions will return the actual number of blocks read/written. When untyped files are used, it is advisable to specify the Compile option "I", thus requiring that the function IORESULT and the number of blocks transferred will be checked after each BLOCKREAD or BLOCK-WRITE to detect any I/O errors. An example of a program that uses untyped files is shown in Figure 3-8.

Random Access of Files

Individual records in a file can be accessed randomly by the intrinsic SEEK. The two parameters for SEEK are the file identifier and an integer giving the record number to which the window should be moved. The first record of a structured file has the number 0. SEEK always sets EOF and EOLN to False. The subsequent GET or PUT will set these conditions as appropriate. Attempts to PUT records beyond the physical end of file will set EOF to True.

```
(*$I-*)
PROGRAM FILEXAMP;
 VAR S,D : FILE;
     BUF : PACKED ARRAY[0..511] OF CHAR;
     BLKN, BLKSTRAN : INTEGER;
     IOERR : BOOLEAN;
 BEGIN (*FILEXAMP*);
  IOERR := FALSE;
  RESET(S,'FROM.DATA');
  REWRITE(D,'TO');
  BLKN := 0;
  BLKSTRAN := BLOCKREAD(S,BUF,1,BLKN);
  WHILE (NOT EOF(S)) AND (IORESULT = 0)
          AND (NOT IOERR) AND (BLKSTRAN=1) DO
   BEGIN
    BLKSTRAN := BLOCKWRITE(D,BUF,1,BLKN);
    IOERR := ((BLKSTRAN < 1) OR (IORESULT <> 0));
    BLKN := BLKN + 1;
```

```
    BLKSTRAN := BLOCKREAD(S,BUF,1,BLKN);
  END (*WHILE*);
 CLOSE(D,LOCK);
END (*FILEXAMP*).
```

Figure 3-8. EXAMPLE OF USING UNTYPED FILES

3.6.7 GOTO and EXIT Statements

The GOTO statement prohibits a GOTO to a label that is not within the same block as the statement. This is a limitation that is not imposed on the GOTO statement in Standard Pascal. Because of this limitation, the examples on pages 31-32 of Jensen and Wirth do not apply.

EXIT is a UCSD extension statement. Its only parameter is the identifier of the procedure to be exited. The EXIT statement was created because of the occasional need for a means to abort a complicated, and possible deeply nested, series of procedure calls upon encountering an error. The recursive descent UCSD Pascal Compiler contains an example of the EXIT statement being used in this way. However, the use of this statement is discouraged.

> NOTE: The use of an EXIT statement to exit a function can result in the function returning undefined values if no assignment is made to the function identifier prior to the execution of the EXIT statement.

If the identifier in the EXIT statement is that of a recursive procedure, the most recent invocation of that procedure will be exited. Upon EXIT, an implicit CLOSE(F) is done on local files that were opened during execution of the procedure being exited. An example of using EXIT is shown in Figure 3-9.

3.6.8 Packed Variables

Described below are packed arrays and records, using packed variables as parameters, and (not) using PACK and UNPACK.

```
            PROGRAM EXITTEST;
VAR S : STRING;
    I : INTEGER;

                    PROCEDURE CALL; FORWARD;

                    PROCEDURE PRINT;
                     BEGIN (*PRINT*)
                      WRITELN('-->');
                      READLN(S);
                      WRITELN(S).
```

```
 IF S[1] = '*' THEN EXIT(CALL);
 WRITELN('LEAVE PRINT');
END (*PRINT*);

PROCEDURE CALL;
 BEGIN (*CALL*)
  PRINT;
  WRITELN('LEAVE CALL');
 END (*CALL*);

PROCEDURE COUNT;
 BEGIN (*COUNT*)
  IF I (= 10 THEN CALL;
  WRITELN('LEAVE COUNT');
 END (*COUNT*);

BEGIN (*EXITTEST*)
  I := 0;
  WHILE NOT EOF DO
   BEGIN
    I := I+1;
    COUNT;
    WRITELN;
   END (*WHILE*);
END (*EXITTEST*).
```

Figure 3-9. EXAMPLE OF USING THE EXIT STATEMENT

Packed Arrays

The UCSD Pascal Compiler will pack arrays if the ARRAY declaration is preceded by the word PACKED. For example:

```
ARRAY[0..9] OF CHAR;
PACKED ARRAY[0..9] OF CHAR;
```

The array in the first declaration will occupy ten 16-bit words of memory, with each element occupying one word. The array in the second declaration will be packed into a total of five words, since each 16-bit word contains two 8-bit characters. Thus each element is eight bits long.

Examples of packed arrays that are not of type CHAR are given in Figure 3-10.

Due to the recursive nature of the Compiler, the following two declarations are not equivalent:

```
PACKED ARRAY[0..9] OF ARRAY[0..3] OF CHAR;
PACKED ARRAY[0..9,0..3] OF CHAR;
```

In the first declaration, the second ocurrence of ARRAY causes the

packing option in the Compiler to be turned off, giving an unpacked array of 40 words. The array in the second declaration occupies a total of 20 words because ARRAY appears only once. If the second occurrence of ARRAY in the first declaration had also been preceded by PACKED, the two declarations would have been equivalent.

```
PROGRAM PACKTST;
 VAR A: PACKED ARRAY [0..9] OF 0..2000;
     B: PACKED ARRAY [0..15] OF BOOLEAN;
     C: PACKED RECORD
          D: BOOLEAN;
          CASE E: BOOLEAN OF
          TRUE: (F: INTEGER);
          FALSE: (G  PACKED ARRAY [0..7] OF CHAR)
          END;
 BEGIN
 END.
```

Figure 3-10. EXAMPLES OF PACKED ARRAYS AND RECORDS

An array will be packed only if the final type of array is scaler, subrange, or a set that can be represented in eight bits or less or if the final type is BOOLEAN or CHAR. No packing is done if the array cannot be expressed in a field of eight bits.

No packing occurs across word boundaries. If the type of element to be packed requires a number of bits that does not divide evenly by 16, unused bits will be at the high end of each of the words that comprise the array.

NOTE: It is illegal to assign a string constant to an unpacked ARRAY OF CHAR, although it may be assigned to a PACKED ARRAY OF CHAR. Also, comparisons between an ARRAY OF CHAR and a string constant are illegal. This is because of size differences.

A PACKED ARRAY OF CHAR may be output with a single WRITE statement, and may be initialized by using the intrinsics FILLCHAR and SIZEOF.

Packed Records

As with arrays, the Compiler will pack records if the RECORD declaration is preceded by PACKED. In the example below, the entire record is packed into one 16-bit word.

```
 VAR A:  PACKED RECORD
           Q,R,S: 0..31;
           B: BOOLEAN
         END;
```

The variables Q, R and S each take up five bits. The boolean variable is allocated to the sixteenth bit.

PACKED RECORDS may contain fields that also are packed, either arrays or records. But PACKED must occur before every occurrence of RECORD to retain packed qualities throughout all fields of the record.

A case variant may be used as the last field of a PACKED RECORD. The amount of space allocated to it will be the size of the largest variant among the cases.

Using Packed Variables as Parameters

Packed variables may be passed as call-by-value parameters to a procedure or function. However, they may not be passed as call-by-reference parameters.

PACK and UNPACK

UCSD Pascal does NOT support the standard procedures PACK and UNPACK. (Jensen and Wirth, 106).

3.6.9 Parametric Procedures and Functions

UCSD Pascal does NOT support the use of procedures and functions as formal parameters in the parameter list of a procedure or function.

3.6.10 Program Headings

A list of file parameters may follow the file identifier. However, they are IGNORED by the Compiler and have no effect on the program being compiled. Any file declarations other than the three predeclared files (INPUT, OUTPUT and KEYBOARD) of type INTERACTIVE must be declared along with the other VAR declarations for the program.

3.6.11 READ and READLN

In Standard Pascal, the procedure READ requires that the window variable F^ be loaded with the first character of the file when the file is opened. If effect, the statement READ(F,CH) would be equivalent to:

```
CH: =F^;
GET(F);
```

To be responsive to the demands of an interactive programming environment, UCSD Pascal defines the additional file type INTERACTIVE. Declaring a file to be of type INTERACTIVE is equivalent to declaring it to be type TEXT, except that the definition of READ(F,CH) is reversed:

```
GET(F);
CH: =F^;
```

The standard definition of the procedure READ requires tha the process of opening a file include loading the window variable F^ with the first character of the file. In an interactive environment it is inconvenient to require a user to type a character of the input file when it is open to avoid the program "hanging" when it is first opened. To overcome this, UCSD Pascal has reversed the order. This difference affects the way in which EOLN must be used when reading from a text file of the type interactive. EOLN only becomes true after reading the end of line character, a (return). The character returned as a result of the READ is a blank.

Three predeclared text files (INPUT, OUTPUT and KEYBOARD) of type INTERACTIVE are opened automatically for a user program. The file INPUT defaults to the console device. The statement READ(INPUT,CH), where CH is a character variable, will echo the character typed from the console back to the console. WRITE statements to the file OUTPUT will cause the output to appear on the console, by default. The file KEYBOARD is the non-echoing equivalent to INPUT. For example, the following two statements are equivalent to READ(INPUT,CH

```
READ (KEYBOARD, CH);

WRITE (OUTPUT, CH);
```

3.6.12 RESET(F)

In Standard Pascal, the procedure RESET resets the file window to the beginning of file F. The next GET(F) or PUT(F) affects record 0 of the file. Also, the window variable F^ is loaded with the first record of the file.

In UCSD Pascal, the standard conventions hold true unless the file is of type INTERACTIVE. In that case, the window variable is NOT loaded. Thus, the UCSD equivalent of the Standard RESET(F) is the two-statement sequence:

```
RESET(F);
GET(F);
```

UCSD Pascal also provides an alternative form of opening a pre-existing file. In it, RESET has two parameters; the file identifier followed by either a string constant or variable, whichever corresponds to the directory file name of the file being reopened.

3.6.13 REWRITE(F)

REWRITE opens and creates a new file. It has two parameters: the file identifier followed by either a string constant or variable, whichever corresponds to the directory file name of the file being created.

REWRITE performs the same as the UCSD intrinsic OPENNEW, and will replace it eventually (see Section 3.1, Intrirsics).

3.6.14 Segment Procedures

The SEGMENT PROCEDURE is a UCSD extension to Pascal. With it, the programmer

can segment a large program so that the entire program does not have to be in memory at once. For further information, see Section 3.3, Segments.

3.6.15 Sets

All of the Standard Pascal constructs for sets are supported by UCSD Pascal (see p. 50-51 of Jensen and Wirth). Sets of enumeration values are limited to positive integers only. Also, a set is limited to 255 words and 4080 elements. Comparisons and operations are allowed only between sets that are either of the same base type or subranges of the same underlying type. Examples are shown in Figure 3-11.

```
PROGRAM SETST;
 VAR SET1: SET OF 0..49;
     SET2: SET OF 0..99;

 BEGIN (*SETST*)
   SET1 := [0,5,10];
   SET2 := [10,20,30];
   IF SET1 = SET2 THEN
      WRITELN('THEY ARE EQUAL')
   ELSE
      WRITELN('THEY ARE NOT EQUAL');
 END(*SETST*).
```

Sets of different underlying types cannot be compared:

```
PROGRAM SETCOMP;
 TYPE INGREDIENTS = (FLOUR,SUGAR,EGGS,SALT);

 VAR I: SET OF INGREDIENTS;
     N: SET OF 0..49;

 BEGIN (*SETCOMP*)
   I := [FLOUR];
   N := [1,2,3,4,5];
   IF I = N THEN   (------ ERROR WILL OCCUR HERE
   WRITELN('EQUAL');
 END (*SETCOMP*).
```

Figure 3-11. EXAMPLES OF SET COMPARISONS

3.6.16 Strings

STRING variables are unique to UCSD Pascal. Essentially, they are PACKED ARRAYs of CHAR with a dynamic LENGTH attribute, the value of which is returned by the string intrinsic LENGTH. The default maximum length of a string variable is 80 characters. This value can be overridden in the declaration of a string by appending the desired length within [] after the type iden- tifier

STRING. For further information and examples, see Section 3.1.1, String Intrinsics.

A string variable has an absolute maximum length of 255 characters. Assignment to string variables can be performed using the assignment statement, UCSD string intrinsics, or with a READ statement. For example:

```
TITLE:=' THIS IS MY STRING ';
READLN(MYSTRING);
NAME:= COPY(MYSTRING,1,21);
```

The individual characters within a string are indexed from 1 to the length of the string. A string variable may not be indexed beyond its current dynamic length.

String variables may be compared to other string variables, no matter what the current dynamic length of either. The lengths do not have to be equal.

One of the most common uses of string variables in UCSD Pascal is reading file names from the console device. When a string variable is used as a parameter to READ or READLN, all characters up to the end-of-line character (carriage return) in the source file will be assigned to the string variable. In reading string variables, the single statement READLN(S1,S2) is equivalent to the two-statement sequence:

```
READ(S1);
READLN(S2);
```

3.6.17 WRITE and WRITELN

The procedures WRITE and WRITELN follow the conventions of Standard Pascal except when applied to a variable of type BOOLEAN. UCSD Pascal does not support the output of the words TRUE or FALSE when writing out the value of a boolean variable.

For writing variables of type STRING, see Section 3.1.1, String Intrinsics. When a string variable is written without specifying a field width, the actual number of characters written is equal to the dynamic length of the string. If the field width specified is longer than the dynamic length, leading blanks are inserted. If the field width is smaller, excess characters will be truncated on the right.

3.6.18 Implementation Size Limits

The maximum size limitation of UCSD Pascal are:

1. Maximum number of bytes of object code in a procedure or function is 1200. Maximum number of words for local variables in a procedure or function is 16383.

2. Maximum number of characters in a string variable is 255.

3. Maximum number of elements in a set is 255 * 16 = 4080.

4. Maximum number of segment procedures and functions is 16, of which nine are reserved for the Pascal system and seven are available to the user.

5. Maximum number of procedures or functions within a segment is 127.

3.6.19 Extended Comparisons

UCSD Pascal permits = and <> comparisons of any array or record structure.

SECTION 4

UTILITIES

4.1 SETUP

Certain information about the user's system configuration is kept in a file called SYSTEM.MISCINFO. During system initialization, this file is read into memory. From there, it is accessed by many parts of the Pascal Operating System, particularly, where applicable, by the Screen-Oriented Editor.

Much of the information in this file must be set up by the user to conform to his hardware configuration and particular needs. Most concerns the nature of the terminal and keyboard, although there is some miscellaneous information.

SETUP is run by typing "X" for eX(ecute at the Outer Level of commands, then the file name SETUP, followed by a carriage return. The following prompt line appears:

```
SETUP: C(HANGE T(EACH) H(ELP) Q(UIT)
```

The program is self-teaching. Typing "H" for H(ELP) will produce an explanation of what the other commands do. If the SETUP program is not on the disk, the following message will appear:

```
no file setup.CODE
```

SETUP does not tell the system how to do random cursor addressing on the user's terminal. If this feature is part of the user's hardware configuration, information on using the feature can be found in Section 4.9, GOTOXY Procedure Binder.

4.1.1 Miscellaneous Information

HAS 8510A

> If TRUE, the system is running on a Terak 8510A hardware configuration; otherwise FALSE.

HAS CLOCK

> If TRUE, a real-time clock is available; otherwise FALSE. The real-time clock module is assumed to be a line frequency clock. When available, the clock is used by the system to optimize disk directory updates (see Section 3.1.6, Time Intrinsics).

4.1.2 General Terminal Information

HAS LOWER CASE

> If TRUE, the terminal has lower case; otherwise FALSE.

HAS RANDOM CURSOR ADDRESSING

If TRUE, the terminal has random cursor addressing; otherwise FALSE. This applies only to video terminals.

HAS SLOW TERMINAL

If TRUE, the terminal has a baud rate of 600 or less; otherwise FALSE. When TRUE, the system issues abbreviated prompt lines and messages.

NON-PRINTING CHARACTER

Any printing character may be entered here to indicate the character that should be displayed to indicate the presence of a non-printing character. The suggested character is ASCII "?".

SCREEN HEIGHT

Enter the number of lines displayed on the screen of a video terminal. Otherwise, enter 0 for hard-copy terminal or one in which paging is not appropriate.

VERTICAL MOVE DELAY

Enter the number of nulls to send after a vertical cursor move. The nulls will be sent after a carriage return, ERASE TO END OF LINE, ERASE TO END OF SCREEN and MOVE CURSOR UP. Many types of terminals require a delay after certain cursor movements to enable the terminal to complete the movement before the next character is sent.

4.1.3 Control Key Information

Some keyboards generate two codes when a single key is typed. That is indicated according to the following format:

PREFIXED[<fieldname>] TRUE

The prefix for all such keys must be the same. For example, many keys function as escape keys in addition to their named function. If a user's keyboard had a vector pad that generated the value pairs ESC "U" and ESC "D" for the Uparrow and Downarrow keys, respectively, the following values should be entered:

KEY FOR MOVING CURSOR UP	ASCII "U"
KEY FOR MOVING CURSOR DOWN	ASCII "D"
LEAD-IN KEY FOR KEYBOARD	ESC
PREFIXED[KKEY FOR MOVING CURSOR UP]	TRUE
PREFIXED[KEY FOR MOVING CURSOR DOWN]	TRUE

The following keys may apply to all terminals.

KEY FOR BREAK

Typing the BREAK key causes the program currently executing to be terminated immediately with a runtime error. This should be set to something that is difficult to hit accidentally.

KEY TO DELETE CHARACTER

This key removes one character from the current line. It may be typed until nothing is left on the line. The suggested setting is ASCII BS.

KEY TO DELETE LINE

This key will cause the current line of input to be erased. The suggested setting is ASCII DEL.

KEY TO END FILE

This is the console end-of-file character that sets the boolean function EOF to True. This applies only to INPUT or KEYBOARD files or the unit CONSOLE. The suggested setting is ASCII ETX.

KEY FOR FLUSH

This is the console output cancel character. When the FLUSH key is pressed, output to the file OUTPUT is undisplayed until FLUSH is pressed again or the system writes to the file KEYBOARD. Processing is uninterrupted even though the output is not displayed. The suggested setting is something that is difficult to hit accidentally.

KEY FOR STOP

This is the console output stop character. When preseed, output to the file OUTPUT ceases. Output resumes where it left off when the key is pressed again. This function is useful for reading data that is being displayed too fast for easy reading. The suggested setting is ASCII DC3.

The following keys are applicable only to video terminals that have selective erase.

EDITOR "ACCEPT" KEY

In the Screen-Oriented Editor, this key is used to accept commands, thus making permanent any action taken. The suggested setting is ASCII ETX.

EDITOR "ESCAPE" KEY

In the Screen-Oriented Editor, this key is used to escape from commands, reversing any action taken. The suggested setting is ASCII ESC.

KEY TO MOVE CURSOR UP
DOWN
LEFT
RIGHT

These keys are used by the Screen-Oriented Editor for cursor control.

If the keyboard has a vector pad, the keys must be set to the value it generates. Otherwise, four keys may be chosen in the pattern of a vector pad (e.g., "O", ".", "M" and ";") and be assigned the control codes that correspond to them. A prefix character may also be used.

4.1.4 Video Screen Control Characters

The video screen control characters are sent by the computer to the terminal to control the actions of the terminal. The terminal manual will give the appropriate values. If a terminal does not have one of these characters, the field should be set to 0, unless otherwise directed.

On some terminals, a two-character sequence is required for some functions (e.g., ESC plus a character). If the first character for all of the functions is the same, it can be set as the value of the field LEAD-IN TO SCREEN. Then the field PREFIX[(fieldname)] must be set to TRUE for each two-character function.

BACKSPACE

This character causes the cursor to move one space to the left.

ERASE LINE

This character causes the erasure of all characters on the line where the cursor is currently located. The cursor is relocated to the beginning of the line.

ERASE SCREEN

This character erases the entire screen. The cursor is repositioned in the upper left hand corner of the screen.

ERASE TO END OF LINE

This character causes the erasure of all characters from the current position of the cursor to the end of the line. The cursor location is unchanged.

ERASE TO END OF SCREEN

This character causes the erasure of all characters from the current position of the cursor to the end of the screen. The cursor location is unchanged.

MOVE CURSOR HOME

This character causes the cursor to be relocated to "home", which is the upper left hand corner of the screen.

NOTE: If the terminal does not have such a character, the field should be set to ASCII CR (carriage return).

MOVE CURSOR UP

LEFT

These characters cause the cursor to move non-destructively one space in the direction indicated.

4.2 BOOTSTRAP COPIER

The bootstrap copier writes the first two blocks of the specified file to the specified unit. This utility is on the file BOOTER.CODE. It is run by typing "X" for eX(ecute at the Outer Level of commands, followed by BOOTER. A prompt line appears to ask for the unit number of the volume on which to write the bootstrap.

Following the entry of the unit number (see Appendix B.4), the file name to write as the bootstrap will be asked for.

To copy the bootstrap from an existing disk, give the disk name. Then the bootstrap will be copied from the disk named to the unit numbered.

4.3 DUPLICATE DIRECTORY UTILITIES

These are two utilities to handle duplicate directories. COPYDUPDIR copies the duplicate directory, and MARKDUPDIR marks a disk that currently is not maintaining a duplicate directory. They are described below.

COPYDUPDIR

This program copies the duplicate directory into the primary directory location. It is entered by typing "X" for eX(ecute while in the Outer Level of commands, followed by COPYDUPDIR. The program then prompts for the drive (4 or 5) in which the copy is to take place.

If the disk is not currently maintaining a current directory, a message is generated.

If no duplicate directory is found after the drive number has been entered, a message is generated.

If a duplicate directory is found, then a prompt will ask if the directory currently in blocks 2-5 is to be destroyed.

A "Y" answer will cause the execution of the copy. Any other answer will abort the program.

If the disk is not currently maintaining a current directory, the RECOVER program should be run. This is described

MARKDUPDIR

This program will mark a disk that is currently not maintaining a duplicate directory. It is entered by typing "X" for eX(ecute while in the Outer Level of commands, followed by MARKDUPDIR. The program will prompt to get the unit (4 or 5) that is to be marked.

Blocks 6-9 must be free. The program will check for this and will give the following message if the blocks appear to be in use:

> A "Y" response, indicating that the user is sure that the blocks are free, will execute the mark. Any other character will abort the program.

Blocks 6-9 can be checked by using the E(xtended command in the File Handler. The extended listing will show where the first file starts. If the first file starts at block 6, or it if starts at block 10 but there is a four-block unused section at the top, then the disk has not been marked. However, if the first file starts at block 10 and these are no unused blocks at the beginning, the disk has been marked.

In the examples below, the disks have not been marked.

```
SYSTEM.PASCAL        31      10-Jan-79     6     Codefile

unused                4      10-Jan-79     6     Codefile
SYSTEM.PASCAL        31      10-Jan-79    10     Codefile
```

Below is the directory of a properly marked disk.

```
SYSTEM.PASCAL        31       10-Jan-79    10     Codefile
```

4.4 LIBRARIAN

The librarian allows the user to link separately compiled Pascal units and separately assembled subroutines into a library file. The librarian is entered by typing "X" for eX(ecute while in the Outer Level of commands, followed by LIBRARY.

Before adding a segment to the #SYSTEM.LIBRARY, the user must create a new file into which each segment that is wanted from the original #SYSTEM.LIBRARY is linked. Then it is possible to add segments by linking from another code file into the new file being created.

On entering the librarian program, the user is prompted for the name of the output code file:

```
Output Code File ->
```

The program then prompts:

```
Link Code File ->
```

to which the user should respond with #SYSTEM.LIBRARY. Then the program displays the names of all segments currently linked into the input library, as well as their length in bytes. A maximum of 16 segments are permitted in any Pascal program or library. After the program lists the segments, it prompts:

```
Segment # to link and (space), N(ew file, Q(uit, A(bort
```

The user responds with the number of the segment within the link code file

that is to be linked into the new library file, followed by <space>. Next, the user enters the number of the segment in the output file to be linked into (i.e., the new library), followed by a <space>. For each segment linked, the program reads the segment from the input file and writes it to the output file at the segment requested. It then displays the segments currently in the output library.

When all needed segments have been linked, a new input file is requested by typing "N" for N(ew. Once the needed segments from all input files have been linked, the user locks the output file by typing "Q" for Q(uit, followed by <cr>. The linking process is aborted by typing "A" for A(bort. In that case, control returns to the Outer Level of commands.

The old #SYSTEM.LIBRARY should either be removed or have its name changed if it resides on the same disk as the new. The name of the new library should be changed to #SYSTEM.LIBRARY in order to be used.

4.5 LIBRARY MAP

The library map program produces a map of a library or code file and lists the linker information mantained for each segment of the file. It is entered by typing "X" for eX(ecute while in the Outer Level of commands, followed by LIBMAP. The program will return with a prompt asking for a library file name.

An asterisk (#) will indicate #SYSTEM.LIBRARY. The ".CODE" suffix may be suppressed when requesting a library or file other than the #SYSTEM.LIBRARY by appending a period to the full file name. For example:

typing:	will reference the file:
#	#SYSTEM.LIBRARY
DIGITAL	:DIGITAL.CODE
DIG.LIBRARY.	:DIG.LIBRARY

The library map utility usually is used to list library definitions. However, when the program prompts for a reference list.

typing a "Y" in response will cause the program to include intra-library symbol references. A negative response is indicated by typing a <space> or <cr>. Then the program prompts for an output file name.

If the extra period at the end of the file name entered is not used, the program will automatically append ".TEXT".

Several libraries may be mapped at the same time. Typing a <cr> when prompted for a file name will quit the program and return control to the Outer Level of commands.

4.6 P-CODE DISASSEMBLER

The P-code disassembler inputs a UCSD code file and outputs symbolic pseudo-assembly code (P-code) along with statistics on opcode frequency, procedure calls and data segment references. The disassembler is helpful to the user in optimizing programs and provides a source of information on the subtleties of

the UCSD implementation of Pascal. All statistics gathered are static in that they are collected by making a pass through the code file rather than collecting them dynamically while the code file is running.

4.6.1 Disassembly

The disassembler is invoked by typing "X" for eX(ecute while at the Outer Level of commands, followed by DISASM.I5. The file OPCODES.I5 must be on the system disk. On entering the disassembler, the first prompt is for an input code file:

The suffix ".CODE" is assumed, and therefore is not required. The code file must be one that has been generated by the Pascal Compiler. If a program USES a UNIT, the disassembly program will include the UNIT only if the code file has been linked. Assembly language routines linked into a Pascal host will not be included in the disassembly.

The next prompt asks if the first physical byte (byte 0) of a machine word is the most significant byte of the word.

The next prompt is for an output file for the disassembled output. Because the output file has not been defined by type, either CONSOLE: or PRINTER: (if on-line) may be used.

Then the user must decide if he wishes to take control of the disassembly to disassemble only selected procedures rather than all of them in the file.

If the user responds with a "Y", a message warns that all statistics are gathered only on the procedures that are disassembled. The Segment Guide then displays the segments in the file by name so that a particular segment can be selected. Then the Procedure Guide is listed when an "L" is typed to give the procedures contained in the segment (see Section 4.6.2 below for more details).

The Segment Guide may be reentered by typing "Q" while in the Procedure Guide, enabling the user to disassemble several procedures in several segments on a selective basis. The Segment Guide is exited by typing "Q". Figure 4-1 gives an example of a Pascal program, its disassembly and its statistics.

```
 1    1     1:D      1 (*$L DISASSM.TEXT*)
 2    1     1:D      1 PROGRAM DISASSM;
 3    1     1:D      3
 4    1     1:D      3   VAR J,I : INTEGER;
 5    1     1:D      5       BUF : ARRAY[0..6] OF INTEGER;
 6    1     1:D     12
 7    1     1:0      0   BEGIN
 8    1     1:1      0     J :=4;
 9    1     1:1      5     I :=J+1;
10    1     1:1     10     BUF[J] :=200;
11    1     1:0     22   END.
```

Sample Pascal Program

```
1    1     1:D      1 (*$L DISASSM.TEXT*)
2    1     1:D      1 PROGRAM DISASSM;
3    1     1:D      3
4    1     1:D      3   VAR J,I : INTEGER;
5    1     1:D      5       BUF : ARRAY[0..6] OF INTEGER;
6    1     1:D     12
7    1     1:0      0   BEGIN
8    1     1:1      0     J :=4;
9    1     1:1      5     I :=J+1;
10   1     1:1     10     BUF[J] :=200;
11   1     1:0     22   END.
```

Sample Program Disassembled

Figure 4-1. DISASSEMBLY EXAMPLE

4.6.2 Data Segment Reference Statistics

The most common use of the references to data segments for a particular procedure is to optimize the procedure's code file. By rearranging the order of the declaration of variables, the offset for a given variable may be changed within the data segment. The first 16 words offset into the data segment are the fastest and have optimized one-byte instructions. Offsets from 17 to 127 result in instructions at least two bytes long; greater than 127, at least three bytes long. If the most frequently used variables have the smallest offsets, considerable code file space, and possible execution time, may be saved.

The Procedure Guide listing gives all of the procedures in a selected segment by number, lex level and data segment size. Refering to the previous section, the listing is made in response to typing an "L" after the segment has been selected.

4.6.3 Opcode, Procedure Call and Jump Statistics

The opcode, procedure call and jump statistics are collected as an aid to optimizing the architecture of P-code. They are of little use to the programmer. The last prompt of the program asks for the name of the file to which these statistics should be dumped.

4.7 PATCH/DUMP

The patch/dump program is entered by typing "X" for eX(ecute while in the Outer Level of commands, followed by PATCH. On entering the program, the following prompt line appears:

```
C(onsole, P(atchwrite, W(holewrite, Q(uit
```

Typing "C" puts the program into console mode for working with and altering the file. Typing "P" puts the program into patchwrite mode for dumping a file in hex, decimal, octal or ASCII format. Typing "W" puts the program into wholewrite mode for dumping, concatentating and/or moving blocks in files. "Q" exits from the patch/dump program and returns control to the Outer Level of commands. Console, patchwrite and wholewrite modes are described below.

4.7.1 Console Mode

In the console mode, the prompt line changes with each command. The full prompt line on entering the mode is:

```
Patch:  R(ead, S(ave, H(ex, M(ixed, G(et, Q(uit [nn]
```

The number in the square bracket at the end of the line is the current block being patched. The first command to use is G(et, which will return with the prompt:

```
Filename: (cr for unit i/o)
```

In response, the name of the file to be patched is entered. However, a carriage return is typed instead if the disk (or other device) has no

directory or has some problem with the directory. Typing <cr> will generate the prompt:

Unitnum to patch [4,5,9..12] (0 will Quit)

When either a file name or a unit number has been entered, the next command to execute is R(ead, which will read a block from the file/unit. On entering R(ead, the prompt is:

BLOCK:

The block number of the file/unit specified is entered. Note that no range checking is provided on this read. Now, using the H(ex command will display the block entirely in hexadecimal characters. The M(ixed command will display the block in ASCII characters, where possible, and hexadecimal values elsewhere. The prompt for M(ixed is:

Alter: pad vector 1,5,3,0 0..F hex characters, S(tuff, Q(uit

The vector keys on the terminal control cursor movement. Note that the cursor will not move off the data. Typing a hex character changes the character at the location of the cursor only if one or more of the data positions is changed.

The S(tuff command displays the prompt line:

Stuff for how many bytes:

Enter a number from 0 to 512, followed by a carriage return to cause patch to accept the number. The next prompt line is:

Fill with what hex pair:

Enter a byte value if hexadecimal. The data will reappear on the screen with the number of bytes specified filled with the value specified. Filling starts with the location of the cursor.

Typing a "Q" will transfer control from the alter mode back to the console mode. S(ave is the next command to be executed to write the changed data back where it was read from.

The console mode clears its memory after each session.

4.7.2 Patchwork Mode

In the patchwork mode, a full screen prompt appears:

```
This procedure writes out sequential blocks to any file as a patch dump.
Type the prefix character of the option to be changed.  Type 'P' to PRINT,
'Q' to QUIT.

     A( Input File
     B( Begin Block #
     C( Num. of Blocks
```

E(Output File
G(Hexadecimal
H(ASCII
I(Decimal
J(Octal
K(Decimal Bytes
L(Octal Bytes
M(Krunch
N(Double Space

Following each field is its current value. Typing the character in front of the field positions the cursor after the field and removes the current value. Typing "Y" or "T" sets a boolean value to True; any other character sets the field to False. The input and output file fields require a file name followed by a carriage return. The integer fields (begin block and num. of blocks) require a number followed by a carriage return or space. Any other character sets the field to some unspecified value.

Both Print and Quit cause control to return to the outer level of the patch/dump program. Print first dumps the file in the requested format. The format may be specified by the options krunch and double space. Krunch, when True, removes blank lines between logical output lines. Double space, when True, double spaces all output. The default is none.

Note that the patchwrite mode remembers its parameters across sessions while remaining in the patch/dump program. Patchwrite paginates its output. After each block is written, a form feed is generated, PAGE(OUTPUTFILE).

4.7.3 Wholewrite Mode

In the wholewrite mode, a full screen prompt appears:

This procedure writes any number of blocks from an existing file to a new file, unchanged. Simply specify the necessary parameters. Type 'P' to PUT, 'Q' to QUIT.

I(nput File
S(tart Block
N(umber of Blocks
O(utput File

Fields are changed the same way as for the patchwork mode. Wholewrite mode allows mixing/matching and mingling of files. Both Put and Quit return control to the outer level of the patch/dump program. Put first writes to the file.

Note that the wholewrite mode remembers its parameters across sessions while remaining in the patch/dump program.

4.8 THE CALCULATOR

The calculator program is entered by typing "X" for eX(ecute while in the Outer Level of commands. On entering the calculator, the following prompt appears:

->

This prompt expects a one-line expression in algebraic form as a response. Up to 25 different variables are available, each with different values assigned using the syntax of the given grammar. Only the first eight letters may be used to distinguish between variables. Variables having a value may be used as constants. The two built-in variables are PI(3.141593) and E(2.718282). No distinction is made between upper and lower case letters.

The Pascal MOD function (\) rounds the operands to integers. WARNING: because this uses the Pascal definition of MOD (Jensen and Wirth, p.108), the results obtained may not be as expected.

The operand of the factorial FAC function also is rounded to an integer which must be between 0 and 33, inclusive, or the expression will be rejected.

The uparrow is used for exponentiation. The answer is calculated by using e ^ Y LN (X). Therefore, the operand must be positive or the expression will be rejected.

The constant LASTX is assigned the value of the previous correct expression and may be used in the next expression. This does away with the necessity for reentering the same expression.

Angles for the TRIG functions must be in RADIANS. Degree-to-radian conversion is accomplished by RADANGLE = (PI/180) * DEGANGLE.

The calculator program will bomb on an execution error if an overflow or underflow occurs. If this happens, all user-assigned variables and their values will be lost. Type carriage return immediately following the prompt to leave the calculator. Calculator examples are given in Figure 4-2.

```
-> PI
     3.14159

-> E
     2.71828

-> A = (FAC(3)/2)
     3.00000

-> 3 + 6
     9.00000

-> A + 6
     9.00000

-> (RET)   To end the program
```

Figure 4-2. CALCULATOR EXAMPLES

4.9 GOTOXY PROCEDURE BINDER

The GOTOXY binder alters the SYSTEM.PASCAL on the default P(refix disk to create and bind into the system (once only) the GOTOXY procedure that enables the system to communicate correctly with a video terminal. Only system configurations containing a video terminal need GOTOXY. The coordinates for the upper lefthand corner of the video screen must be X=0, Y=0.

The GOTOXY binder is entered by typing "X" for eX(ecute while in the Outer Level of commands, followed by BINDER. When entered, it prompts for:

```
local GOTOXY
```

the procedure that must be created to suit the needs of the particular installation.

To create "local GOTOXY", examine the file GOTOXY.TEXT, that is on the release disk, with the Screen-Oriented Editor. This file contains a few procedures for doing GOTOXY cursor addressing on several types of video terminals. If the procedure needed is in the file, remove it from comments, comment out any other procedures, recompile it and run BINDER on it. BINDER is a self-instructing program. Directions for entering it are given above. Possible errors that may occur while reviewing the GOTOXY.TEXT and selecting the procedure needed are:

Possible Error	Fix
Nil memory reference at compile time	Remove the program heading and try again
Value range error when executing BINDER	(**U-*) should be the first thing in the GOTOXY file.

If the needed procedure is not in the file, it must be created. The created procedure cannot be named GOTOXY because this identifier is predeclared at the "$U-" level of compilation.

APPENDIX A

COMMAND SUMMARIES

A.1 Outer Level
A.2 Screen-Oriented Editor
A.3 Line-Oriented Editor
A.4 File Handler
A.5 Pascal Compiler
A.6 Debugger

Appendix A.1 Outer Level

A(ssem	Invokes the system Assembler.
C(omp	Invokes the system Compiler (Pascal or Basic).
D(ebug	Invokes the Debugger.
E(dit	Invokes the system Editor (Screen-Oriented Editor or Yet Another Line-Oriented Editor).
eX(ecute	Executes the code file.
F(iler	Invokes the File Handler.
L(ink	Invokes the Linker.
R(un	Executes the code file associated with the current work file. If none exists, the Compiler is automatically called, followed by the Linker, if necessary, before execution.

Appendix A.2 SCREEN-ORIENTED EDITOR

⟨down-arrow⟩	moves ⟨repeat-factor⟩ lines down
⟨up-arrow⟩	" " lines up
⟨right-arrow⟩	" " spaces right
⟨left-arrow⟩	" " spaces left
⟨space⟩	" " spaces in direction
⟨back-space⟩	" " spaces left
⟨tab⟩	moves ⟨repeat-factor⟩ tab positions in direction
⟨return⟩	moves to the beginning of line ⟨repeat-factor⟩ lines in direction.
"⟨" "," "_"	change direction to backward
"⟩" "." "+"	change direction to forward
"="	moves to the beginning of what was just found/replaced/inserted/exchanged

A(djust: Adjusts the indentation of the line that the cursor is on. Use the arrow keys to move. Moving up (down) adjusts line above (below) by same amount of adjustment as current line. Repeat factors are valid.

C(opy: Copies what was last inserted/deleted/zapped into the file at the position of the cursor.

D(elete: Treats the starting position of the cursor as the anchor. Use any moving commands to move the cursor. (etx) deletes everything between the cursor and the anchor.

F(ind: Operates in L(iteral or T(oken mode. Finds the (targ) string. Repeat factors are valid, direction is applied. "S" = use same string as before.

I(nsert: Inserts text. Can use (backspace) and (del) to reject part of your insertion.

J(ump: Jumps to the beginning, end or previously set marker.

M(argin: Adjusts anything between two blank lines to the margins which have been set. Command characters protect text from being margined. Invalidates the copy buffer.

P(age: Moves the cursor one page in direction. Repeat factors are valid, direction is applied.

Q(uit: Leaves the editor. You may U)pdate, E)xit, W)rite, or R)eturn.

R(eplace: Operates in L(iteral or T(oken mode. Replaces the (targ) string with the (subs) string. V(erify option asks you to verify before it replaces. "S" option uses the Same string as before. Repeat factors replace the target several times. Direction is valid.

S(et: Sets M(arkers by assigning a string name to them. Sets E(nvironment for A(uto-indent, F(illing, margins, T(oken, and C(ommand characters.

V(erify: Redisplays the screen with the cursor centered.

eX(change: Exchanges the current text for the text typed while in this mode. Each line must be done separately. (back-space) causes the original character to re-appear.

Z(ap: Treats the starting position of the last thing found/replaced/inserted as an anchor and deletes everything between the anchor and the current cursor position.

(repeat-factor) is any number typed before a command. Typing a / is the infinite number.

Appendix A.3 YALOE

n - an argument m - macro number

nA: Advance the cursor to the beginning of the n th line from the current

position.

B: Go to the Beginning of the file.

nC: Change by deleting n characters and inserting the following text. Terminate text with (esc).

nD: Delete n characters.

E: Erase the screen.

nF: Find the n th occurrence from the current cursor position of the following string. Terminate target string with (esc).

nG: Get - ditto -

H: - invalid -

I: Insert the following text. Terminate text with (esc).

nJ: Jump cursor n characters.

nK: Kill n lines of text. If current cursor position is not at the start of the line, the first part of the line remains.

nL: List n lines of text.

mM: Define macro number m.

nNm: Perform macro number m, n times.

nO: On, off toggle. If on, n lines of text will be displayed above and below the cursor each time the cursor is moved. If the cursor is in the middle of a line then the line will be split into two parts. The default is whatever fills the screen. Type 0 to turn off.

P: - invalid -

Q: Quit this session, followed by:

U:(pdate	Write out a new SYSTEM.WRK.TEXT
E:(scape	Escape from session
R:(eturn	Return to editor

R: Read this file into buffer (insert at cursor);
'R' must be followed by (file name) (esc);
WARNING: If the file will not fit into the buffer, the content of the buffer becomes undefined!

nS: Put the next n lines of text from the cursor position into the Save Buffer.

T: - invalid -

U: Insert (Unsave) the contents of the Save Buffer into the text at the cursor; does not destroy the Save Buffer.

V: Verify: display the current line

W: Write this file (from start of buffer);
'W' must be followed by (file name) (esc)

nX: Delete n lines of text, and insert the following text; terminate with (esc)

Y: - invalid -

Z: - invalid -

Appendix A.4 File Handler

B(ad-blks — Scans the disk and detects bad blocks, listing the number of each.

C(hange — Changes file or volume name.

D(ate — Lists current system date and enables user to change date; format is dd-mmm-yy.

E(xt-dir — Lists the directory in more detail than the L(dir command.

G(et — Loads the designated file into the work file.

K(runch — Moves the files on the specified volume so that unused blocks are combined at the end of the disk; disk files only.

L(dir — Lists a disk directory, or subset of one, to the volume and file specified; default is CONSOLE:).

M(ake — Creates a directory entry with the specified filename.

N(ew — Clears the work file.

P(refix — Changes the current default volume to the volume specified.

Q(uit — Returns control to the Outer Level of commands.

R(em — Removes file entries from the directory.

S(ave — Saves the work file under the specified file name.

T(rans — Copies (transfers) the specified file to the specified destination volume; directories are not changed automatically.

V(ols — Lists the volumes currently on-line along with their corresponding device numbers.

W(hat — Identifies the file name and state (saved or not) of the work file.

X(amine — Attempts to recover bad blocks physically; a bad-block scan should be done first.

Z(ero — Reformats the specified volume and makes the old directory irretrievable.

Appendix A.5 Pascal Compiler

D: — Causes the Compiler to insert breakpoint instructions into the code file during compilation so that the Debugger can be used more effectively. Default is '-', no breakpoint.

G: — Affects the boolean variable GOTOOK to allow the use of the Pascal GOTO statement in the program. Default is '-', no GOTO.

I: — When followed by a '+', causes the Compiler to generate code after any I/O statement to check for successful completion of I/O. This is the default.

When followed by a '-', inhibits I/O checking.

When followed by a file name, includes another source file into the compilation.

L: Causes the Compiler to generate a listing of the source program on a specified file. If a '+', the default file is *SYSTEM.LIST.TEXT. Default if '-', no listing.

Q: "Quiet compile" option used to suppress output to the CONSOLE listing procedure names and line numbers during compilation. Default is set to the current value of SYSCOM^.MISCINFO.SLOW-TERM.

R: Affects the value of the boolean variable RANGECHECK to perform checking on array subscripts and assignments to variables of subrange types. Default is '+', code for checking is inserted.

S: Causes the Compiler to operate in swapping mode so that only one of the two main parts of the Compiler (declarations processor or statement handler) is in main memory at one time; freeing about 2500 words for symbol table storage. Default is '-', no swapping.

U: Affects the boolean variable SYSCOMP to determine if the compilation is of a user program or a system program. Default is '+', user program.

When followed by a file name, U: names the library file.

Appendix A.6 Debugger

EXECUTE OPTIONS

C(rawl Executes a program one statement at a time, waiting for input from user between stops.

R(esume Runs program normally until a BREAK or breakpoint is encountered or a non-fatal runtime error occurs.

W(alk Runs a program one statement at a time at an adjustable rate.

EXAMINE MODE OPTIONS

'(' or ',' Changes the direction of link traversal to go toward the callees, down the call chain.

')' or '.' Changes the direction of link traversal to go toward the callers, up the call chain.

(esc) Returns control to the Outer Level of commands.

(CR) Clears the line with the '*' and moves down one line.

(CTRL-D)	Moves the '*' down one line.
(CTRL-U)	Moves the '*' up one line.
# links	Indicates how many links to move up or down the dynamic or static chain; enter a number 0-9.
B(reakpoint	Used to S(et or C(lear a breakpoint.
C(rawl	Resumes running the program in crawl mode at the point the Debugger was invoked.
D(ata	Used to examine the DATA and parameter segments of a procedure.
E(rase	Clars the memory display buffer on the screen.
H(eap	Displays the portion of memory specified by the octal address and length.
L(ink	Toggles the default link from dynamic to static, and vice-versa.
M(ove	Finds the specified procedure and makes it the current procedure.
R(esume	Resumes normal execution of the program at the point the Debugger was invoked.
S(tack	Examines the stack area belonging to the specified procedure.
U(pdate	Refreshes the memory display buffer.
W(alk	Resumes execution of the program in walk mode at the point where the Debugger was invoked.

APPENDIX B
TABLES

B.1 Runtime Errors
B.2 Syntax Errors
B.3 I/O Results
B.4 Unit Numbers
B.5 P-Machine Opcodes
B.6 Assembler Syntax Errors
B.7 ASCII

Appendix B.1 RUNTIME ERRORS

Version 3

0	System error	FATAL
1	Invalid index, value out of range (XINVNDX)	
2	No segment, bad code file (XNOPROC)	
3	Procedure not present at exit time (XNOEXIT)	
4	Stack overflow (XSTKOVR)	
5	Integer overflow (XINTOVR)	
6	Divide by zero (XDIVZER)	
7	Invalid memory reference (bus timed out) (XBADMEM)	
8	User Break (XUBREAK)	
9	System I/O error (XSYIOER)	FATAL
10	User I/O error (XUIOERR)	
11	Unimplemented instruction (XNOTIMP)	
12	Floating Point math error (XFPIERR)	
13	String too long (XS2LONG)	
14	Halt, Breakpoint (without debugger in core) (XHLTBPT)	
15	Bad Block	

All fatal errors either cause the system to rebootstrap, or if the error was totally lethal to the system, the user will have to reboot. All errors cause the system to re-initialize itself (call system procedure INITIALIZE).

Appendix B.2 SYNTAX ERRORS

Version 3

```
  1:  Error in simple type
  2:  Identifier expected
  3:  'PROGRAM' expected
  4:  ')' expected
  5:  ':' expected
  6:  Illegal symbol
  7:  Error in parameter list
  8:  'OF' expected
  9:  '(' expected
 10:  Error in type
 11:  '[' expected
 12:  ']' expected
 13:  'END' expected
 14:  ';' expected
 15:  Integer expected
 16:  '=' expected
 17:  'BEGIN' expected
 18:  Error in declaration part
 19:  Error in (field-list)
 20:  '.' expected
 21:  '*' expected
 22:  'Interface' expected
 23:  'Implementation' expected
 24:  'Unit' expected

 50:  Error in constant
 51:  ':=' expected
 52:  'THEN' expected
 53:  'UNTIL' expected
 54:  'DO' expected
 55:  'TO' or 'DOWNTO' expected in for statement
 56:  'IF' expected
 57:  'FILE' expected
 58:  Error in (factor) (bad expression)
 59:  Error in variable

101:  Identifier declared twice
102:  Low bound exceeds high bound
103:  Identifier is not of the appropriate class
104:  Undeclared identifier
105:  Sign not allowed
106:  Number expected
107:  Incompatible subrange types
108:  File not allowed here
109:  Type must not be real
110:  (tagfield) type must be scalar or subrange
111:  Incompatible with (tagfield) part
112:  Index type must not be real
113:  Index type must be a scalar or a subrange
114:  Base type must not be real
115:  Base type must be a scalar or a subrange
116:  Error in type of standard procedure parameter
117:  Unsatisfied forward reference
```

118: Forward reference type identifier in variable declaration
119: Re-specified params not OK for a forward declared procedure
120: Function result type must be scalar, subrange or pointer
121: File value parameter not allowed
122: A forware declared function's result type can't be re-specified
123: Missing result type in function declaration
124: F-format for reals only
125: Error in type of standard procedure parameter
126: Number of parameters does not agree with declaration
127: Illegal parameter substitution
128: Result type does not agree with declaration
129: Type conflict of operands
130: Expression is not of set type
131: Tests on equality allowed only
132: Strict inclusion not allowed
133: File comparison not allowed
134: Illegal type of operand(s)
135: Type of operand must be boolean
136: Set element type must be scalar or subrange
137: Set element types must be compatible
138: Type of variable is not array
139: Index type is not compatible with the declaration
140: Type of variable is not record
141: Type of variable must be file or pointer
142: Illegal parameter solution
143: Illegal type of loop control variable
144: Illegal type of expression
145: Type conflict
146: Assignment of files not allowed
147: Label type incompatible with selecting expression
148: Subrange bounds must be scalar
149: Index type must be integer
150: Assignment to standard function is not allowed

151: Assignment to formal function is not allowed
152: No such field in this record
153: Type error in read
154: Actual parameter must be a variable
155: Control variable cannot be formal or non-local
156: Multidefined case label
157: Too many cases in case statement
158: No such variant in this record
159: Real or string tagfields not allowed
160: Previous declaration was not forward
161: Again forward declared
162: Parameter size must be constant
163: Missing variant in declaration
164: Substitution of standard proc/func not allowed
165: Multidefined label
166: Multideclared label
167: Undeclared label
168: Undefined label
169: Error in base set
170: Value parameter expected

171: Standard file was re-declared
172: Undeclared external file
174: Pascal function or procedure expected
182: Nested units not allowed
183: External declaration not allowed at this nesting level
184: External declaration not allowed in interface section
185: Segment declaration not allowed in unit
186: Labels not allowed in interface section
187: Attempt to open library unsuccessful
188: Unit not declared in previous uses declaration
189: 'Uses' not allowed at this nesting level
190: Unit not in library
191: No private files
192: 'Uses' must be in interface section
193: Not enough room for this operation
194: Comment must appear at top of program
195: Unit not importable

201: Error in real number - digit expected
202: String constant must not exceed source line
203: Integer constant exceeds range
204: 8 or 9 in octal number
250: Too many scopes of nested identifiers
251: Too many nested procedures or functions
252: Too many forward references of procedure entries
253: Procedure too long
254: Too many long constants in this procedure
256: Too many external references
257: Too many externals
258: Too many local files
259: Expression too complicated

300: Division by zero
301: No case provided for this value
302: Index expression out of bounds
303: Value to be assigned is out of bounds
304: Element expression out of range
398: Implementation restriction
399: Implementation restriction

400: Illegal character in text
401: Unexpected end of input
402: Error in writing code file, not enough room
403: Error in reading include file
404: Error in writing list file, not enough room
405: Call not allowed in separate procedure
406: Include file not legal

Appendix B.3 I/O RESULTS

Version 3

0	No error
1	Bad Block, Parity error (CRC)
2	Bad Unit Numer
3	Bad Mode, Illegal operation
4	Undefined hardware error
5	Lost unit, Unit is no longer on-line
6	Lost file, File is no longer in directory
7	Bad Title, Illegal file name
8	No room, insufficient space
9	No unit, No such volume on line
10	No file, No such file on volume
11	Duplicate file
12	Not closed, attempt to open an open file
13	Not open, attempt to access a closed file
14	Bad format, error in reading real or integer
15	Ring buffer overflow

Appendix B.4 UNIT NUMBERS

Version 3

NUMBER	VOLUME NAME
0	<empty>
1	CONSOLE
2	SYSTERM
3	GRAPHIC
4	floppy0
5	floppy1
6	PRINTER
7	available - <unimplemented>
8	REMOTE <reserved for future use>
9	block1
10	block2
11	block3
12	block4

Devices 9 - 12 are block-structures devices, in most cases (RK-05).

.1

Mnemonic	Instruction Code	Parameters	Description
Constant One Word Loads			
SLDC	0..31		Short Load Word Constant (Value 0-31). Pushes the opcode, with high byte zero, onto stack.
LDCN	152		Load Constant Nil. Pushes nil onto stack.
LDCB	128	UB	Load Constant Byte. Pushes UB, with high byte zero, onto stack.
LDCI	129	W	Load Constant Word. Pushes W onto stack.
LCA	130	B	Load Constant Address. Pushes the word Address of the constant with offset B in constant word block.
Local One Word Loads and Store			
SLDL1..16	32..47		Short Load Local Word. SLDLX fetches the word with offset X in MP activation record and pushes it.
LDL	135	B	Load Local Word. Fetches the word with offset B in MP activation record and pushes it.
LLA	132	B	Load Local Address. Fetches address of the word with offset B in MP activation record and pushes it.
STL	164	B	Store Local. Stores Tos into word with offset B in MP activation record.
Global One Word Loads and Store			
SLDO1..16	48..63		Short Load Global Word. SLDOX fetches

			the word with offset X in base activation record and pushes it.
LDO	133	B	Load Global Word. Fetches the word with offset B in base activation record and pushes it.
LAO	134	B	Load Global Address. Pushes the word addres of the word with offset B in base activation record.
SRO	165	B	Store Global Word. Stores Tos into the word with offset B in base activation record.

Intermediate One-Word Loads and Store

LOD	137	DB,B	Load Intermediate Word. DB indicates the number of static links to traverse to find the activation record to use. B is the offset within the activation record.
LDA	136	DB,B	Load Intermediate Address.
STR	166	DB,B	Store Intermediate Word.

Indirect One-Word Loads and Store

STO	196		Store Indirect. Tos is stored into the word pointed to by Tos-1.

Extended One-Word Loads and Store

LDE	154	UB,B	Load Word Extended. UB is segment number. B is the offset within the segment.
LAE	155	UB,B	Load Address Extended.
STE	217	UB,B	Store Word Extended.

Multiple Word Loads and Stores (Sets and Reals)

LDC	131	B,UB	Load Multiple Word Constant. B is the offset within the constant word block, and UB is the number of words to load. Push the block onto the stack.
LDM	208	UB	Load Multiple Words. Tos is a pointer

			to the beginning of a block of UB words. Push the block onto the stack.
STM	142	UB	Store Multiple Words. Tos is a block of UB words, Tos-1 is a word pointer to a similar block. Transfer the block from the stack to the destination block.

Byte Arrays

LDB	167		Load Byte. Push the byte (after zeroing high byte) pointed to by byte pointer Tos.
STB	200		Store Byte. Store byte Tos into the location specified by Byte Pointer Tos-1.

Record and Array Indexing and Assignment

MOV	197	B	Move Words. Tos is a source pointer to a block of B words, Tos-1 is a destination pointer to a similar block. Transfer the block from the source to the destination.
SINDO..7	120..127		Short Index and Load Word. SINDX indexes the word pointer Tos by X words, and pushes the wod pointed to by the result.
IND	230	B	Static Index and Load Word. Indexe the word pointer Tos by B words, and pushes the word pointed to.
INC	231	B	Increment Field Pointer. The word pointer Tos is indexed by B words and the resultant pointer is pushed.
IXA	215	B	Index Array. Tos is an integer index, Tos-1 is the array base word pointer, and B is the size (in words) of an array element. A word pointer to the indexed element is pushed.
IXP	216	UB1,UB2	Index Packed Array. Tos is an integer index, Tos-1 is the array base word pointer. UB(1) is the number of element-per-word, and UB(2) is the field-width (in bits). Compute and push a packed field pointer.

LDP	201	Load A Packed Field. Push the field described by the packed field pointer Tos.
STP	202	Store Into A Packed Field. Tos is the data, Tos-1 is a packd field pointer. Store Tos into the field described by Tos-1.

Logicals

LAND	161	Logical And. And Tos into Tos-1.
LOR	160	Logical Or. Or Tos into Tos-1.
LNOT	220	Logical Not. Take one's complement of Tos.
LEUSW	180	Compare Unsigned Word <=. Compare unsigned word of Tos-1 to unsigned word of Tos and push true or false.
GEUSW	181	Compare Unsigned Word >=. Compare unsigned word of Tos-1 to unsigned word of Tos and push true or false.

Integers

ABI	224	Absolute Value of Integer. Take absolute value of integer Tos. Result is undefined if Tos is initially -32768.
NGI	225	Negate Integer. Take the two's co complement of Tos.
ADI	162	Add Integers. Add Tos and Tos-1.
SBI	163	Subtract Integers. Subtract Tos from Tos-1.
MPI	140	Multiply Integers. Multiply Tos and Tos-1. This instruction may cause overflow if result if larger than 16 bits.
DUP1	226	Copy Integer. Duplicate one word on Tos.
DVI	141	Divide Integers. Divide Tos-1 by Tos and push quotient.

MODI	143	Modulo Integers. Divide Tos-1 by Tos and push the remainder.
CHK	203	Check Against Subrange Bounds. Insure that Tos-1 < = Tos-2 < = Tos, leaving Tos-2 on the stack. If conditions are not satisfied a run-time error occurs.
EQUI	176	Compare Integer =. Compare Tos-1 to to Tos and push true or false.
NEQI	177	Compare Integer <>. Compare Tos-1 to to Tos and push true or false.
LEQI	178	Compare Integer <=. Compare Tos-1 to to Tos and push true or false.
GEQI	179	Compare Integer >=. Compare Tos-1 to to Tos and push true or false.

Reals (All Over/Underflows Cause a Run-Time Error)

FLT	204	Float Top-of-Stack. The integer Tos is converted to a floating point number.
TNC	190	Truncate Real. The real Tos is truncated and converted to an integer.
RND	191	Round Real. The real Tos is rounded, then truncated and converted to an integer.
ABR	227	Absolute Value of Real. Take the absolute value of the real Tos.
ADR	192	Add Reals. Add Tos and Tos-1.
NGR	228	Negate Real. Negate the Real Tos.
SBR	193	Subtract Reals. Subtract Tos from Tos-1.
MPR	194	Multiply Reals. Multiply Tos and Tos-1.
DUP2	198	Copy Real. Duplicate two words on Tos.
DVR	195	Divide Reals. Divide Tos-1 by Tos.
EQUREAL	205	Compare Real =. Compare Tos-1 to Tos

			and push true or false.
LEQREAL	206		Compare Real (=. Compare Tos-1 to Tos and push true or false.
GEQREAL	207		Compare Real)=. Compare Tos-1 to Tos and push true or false.

Sets

ADJ	199	UB	Adjust Set. The set Tos is forced to occupy UB words, either by expansion (putting zeroes "between" Tos and Tos-1) or compression (chopping of high words of set), and its length word is discarded.
SRS	188		Build Subrange Set. The integers Tos and Tos-1 are checked to insure that 0(=Tos(=4079 and 0(=Tos-1(=4079, a run-time error occuring if not. The set [Tos-1..Tos] is pushed. (The set [ ] is pushed if Tos-1) Tos.)
INN	218		Set Membership. See if integer Tos-1 is in set Tos, pushing true or false.
UNI	219		Set Union. The union of sets Tos and Tos-1 is pushed. (Tos or Tos-1.)
INT	220		Set Intersection. The intersection of sets Tos and Tos-1 is pushed. (Tos and Tos-1.)
DIF	221		Set Difference. The difference of sets Tos-1 and Tos is pushed. (Tos-1 and not Tos.)
EQUPWR	182		Set Compare =.
LEQPWR	183		Set Compare (= (Subset of).
GEQPWR	184		Set Compare)= (Superset of).

Byte Arrays

EQUBYT	185	B	Byte Array Compare =.
LEQBYT	186	B	Byte Array Compare (=.
GEQBYT	187	B	Byte Array Compare)=.

Jumps

UJP	138	SB	Unconditional Jump.
FJP	212	SB	False Jump. Jump is Tos is false.
EFJ	210	SB	Equal False Jump. Jump if integer Tos <> Tos-1.
NFJ	211	SB	Not Equal False Jump. Jump if integer Tos = Tos-1.
UJPL	139	W	Unconditional Long Jump. Jump unconditionally to location with offset W from current location.
FJPL	213	W	False Long Jump. Jump to location with offset W from current location if Tos is false.
XJP	214	B	Case Jump. The first word with offset B in constant word block, W1, is word-aligned and is the minimum index of the table. The next word up, W2, is the maximum index. The case table is the next W2-W1 words. If Tos, the actual index, is in the range of W1..W2 then jump to location with offset W3 from current location. Where W3 is the contents of the word pointed by Tos in the case table.

Procedure and Function Calls and Returns

CPL	144	UB	Call Local Procedure. Cal procedure UB, which is an immediate child of the currently executing procedure and in the same segment. Static link of MSCW is set to old MP.
CPG	145	UB	Call Global Procedure. Cal procedure UB, which is at lex level 1 and in the same segment. The static link of the MSCW is set to base.
CPI	146	DB,UB	Call Intermediate Procedure. Call procedure UB, which is at lex level DB less than the currently executing procedure and in the same segment. Use that activation record's static link as the static link of the new MSCW.

CXL	147	UB1,UB2	Call Local External Procedure. Call procedure UB(2) which is an immediate child of the currently executing procedure and in the segment UB(1).
CXG	148	UB1,UB2	Call Global External Procedure. Call procedure UB(2) which is at lex level 1 and in the segment U(1).
CXI	149	UB1,DB,UB2	Call Intermediate External Procedure. Call procedure UB(2) which is at lex level DB less than the currently executing procedure, and in the segment UB(1).
CPF	151		Call Formal Procedure. Tos contains segment number and procedure number and Tos-1 contains static link for the called procedure.
RPU	150	B	Return From User Procedure. Static l link if discarded, MP is reset from MSDYN, IPC is also reset from MSIPC. If segment number is not zero, segment pointer is set from segment dictionary. Stack pointer is decremented by B.
LSL	153	DB	Load Static Link Onto Stack. DB indicates the number of static link to traverse to get the static link to load.

System Control

SIGNAL	222		Signal. Tos is a semaphore address signal on this semaphore.
WAIT	223		Wait on Semaphore. Tos is a semaphore address wait on this semaphore.
LPR	157		Load Processor Register. Tos is a Reg # (If it is positive it is one of the TIB registers. If not -1 is current task pointer, -2 is segment dic. pointer and -3 is ready queue pointer.) Load contents of this register on top of stack.
SPR	209		Store Processor Register. Tos-1 is a register number (same definition as LPR). Store Tos in this register.

Debugger

BPT	158	Break Point.
RBP	159	Return From Breakpoint.

Miscellaneous

NOP	156	No Operation.
SWAP	189	Swap Top-of-Stack with Next to Top-of-Stack.

Appendix B.6 ASSEMBLER SYNTAX ERRORS

Version 3

This section lists all the general errors found in the ERRORS file, specific machine errors are found in the sections below dealing with machine specifics.

1: Undefined label
2: Operand out of range
3: Must have procedure name
4: Number of parameters expected
5: Extra garbage on line
6: Input line over 80 characters
7: Not enough ifs
8: Must be declared in ASECT before use
9: Identifier previously declared
10: Improper format
11: EQU expected
12: Must EQU before use if not to a label
13: Macro identifier expected
14: Word addressed machine
15: Backward ORG not allowed
16: Identifier expected
17: Constant expected
18: Invalid structure
19: Extra special symbol
20: Branch too far
21: Variable not PC relative
22: Illegal macro parameter index
23: Not enough macro parameters
24: Operand not absolute
25: Illegal use of special symbols
26: Ill-formed expression
27: Not enough operands
28: Cannot handle this relative
29: Constant overflow
30: Illegal decimal constant
31: Illegal octal constant
32: Illegal binary constant
33: Invalid key word
34: Unexpected end of input - after macro
35: Include files must not be nested
36: Unexpected end of input
37: Bad place for an include file
38: Only labels & comments may occupy column one
39: Expected local label
40: Local label stack overflow
41: String constant must be on 1 line
42: String constant exceeds 80 chars
43: Illegal use of macro parameter
44: No local labels in ASECT
45: Expected key word
46: String expected
47: Bad block, parity error (crc)

48: Bad unit number
49: Bad mode, illegal operation
50: Undefined hardware error
51: Lost unit, no longer on-line
52: Lost file, no longer in directory
53: Bad title, illegal file name
54: No room, insufficient space
55: No unit, no such volumn on-line
56: No file, no such file on volumn
57: Duplicate file
58: No closed, attempt to open an open file
59: No open, attempt to access a closed file
60: Bad format, error in reading real or integer

Appendix B.7 American Standard Code for Information Interchange

Version 3

0	000	00	NUL	32	040	20	SP	64	100	40	@	96	140	60	`
1	001	01	SOH	33	040	21	!	65	101	41	A	97	141	64	a
2	002	02	STX	34	042	22	"	66	102	42	B	98	142	62	b
3	003	03	ETX	35	043	23	#	67	103	43	C	99	143	63	c
4	004	04	EOT	36	044	24	$	78	104	44	D	100	144	64	d
5	005	05	ENG	37	045	25	%	69	105	45	E	101	145	65	e
6	006	06	ACK	38	046	26	&	70	106	46	F	102	146	66	f
7	007	07	BEL	39	047	27	'	71	107	47	G	103	147	67	g
8	010	08	BS	40	050	28	(	72	110	48	H	104	150	68	h
9	011	09	HT	41	051	29	)	73	111	49	I	105	151	69	i
10	012	0A	LF	42	052	2A	*	74	112	4A	J	106	152	6A	j
11	013	0B	VT	43	053	2B	+	75	113	4B	K	107	153	6B	k
12	014	0C	FF	44	054	2C	,	76	114	4C	L	108	154	6C	l
13	015	0D	CR	45	055	2D	-	77	115	4D	M	109	155	6D	m
14	016	0E	SO	46	056	2E	.	78	116	4E	N	110	156	6E	n
15	017	0F	SI	47	057	2F	/	79	117	4F	O	111	157	6F	o
16	020	10	DLE	48	060	30	0	80	120	50	P	112	160	70	p
17	021	11	DC1	49	061	31	1	81	121	51	Q	113	161	71	q
18	022	12	DC2	50	062	32	2	82	122	52	R	114	162	72	r
19	023	13	DC3	51	063	33	3	83	123	53	S	115	163	73	s
20	024	14	DC4	52	064	34	4	84	124	54	T	116	164	74	t
21	025	15	NAK	53	064	35	5	85	125	55	U	117	165	75	u
22	026	16	SYN	54	066	36	6	86	126	56	V	118	166	76	v
23	027	17	ETB	55	067	37	7	87	127	57	W	119	167	77	w
24	030	18	CAN	56	070	38	8	89	130	58	X	120	170	78	x
25	031	19	EM	57	071	39	9	89	131	59	Y	121	171	79	y
26	032	1A	SUB	58	072	3A	:	90	132	5A	Z	122	172	7A	z
27	033	1B	ESC	59	073	3B	;	91	133	5B	[	123	173	7B	{
28	034	1C	FS	60	074	3C	<	92	134	5C	\	124	174	7C	|
29	035	1D	GS	61	075	3D	=	93	135	5D	]	125	175	7D	}
30	036	1E	RS	62	076	3E	>	94	136	5E	^	126	176	7E	~
31	307	1F	US	63	077	3F	?	95	137	5F	_	127	177	7F	DEL

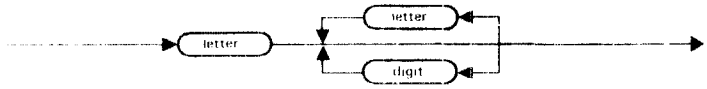
APPENDIX C

UCSD PASCAL SYNTAX DIAGRAMS

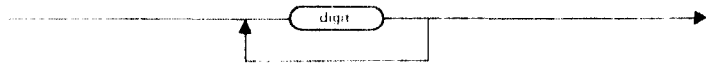
Appendix C UCSD Pascal Syntax Diagrams

UCSD PASCAL SYNTAX DIAGRAMS

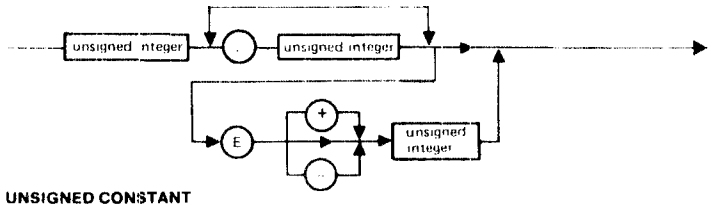
IDENTIFIER



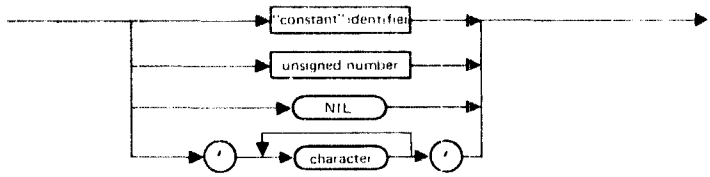
UNSIGNED INTEGER



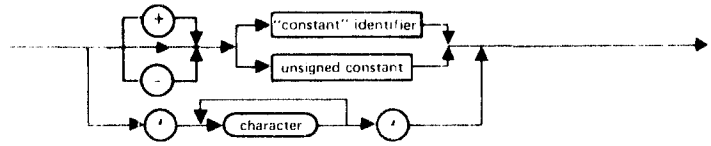
UNSIGNED NUMBER



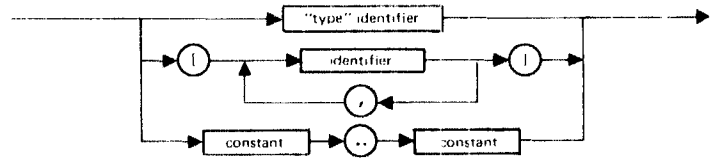
UNSIGNED CONSTANT



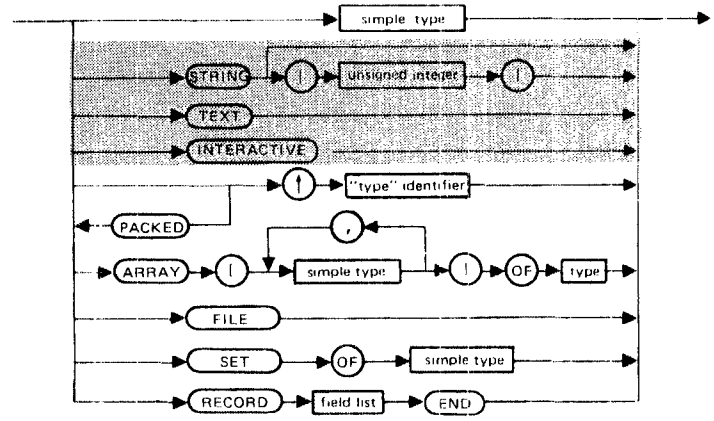
CONSTANT



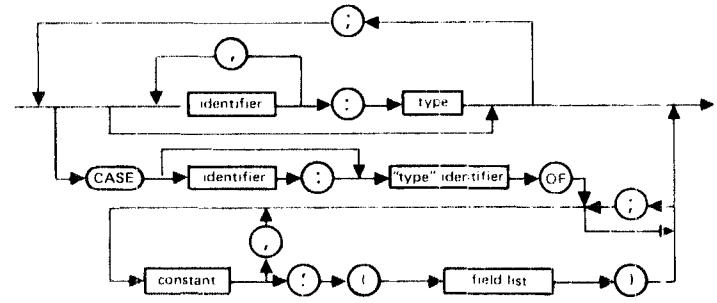
SIMPLE TYPE



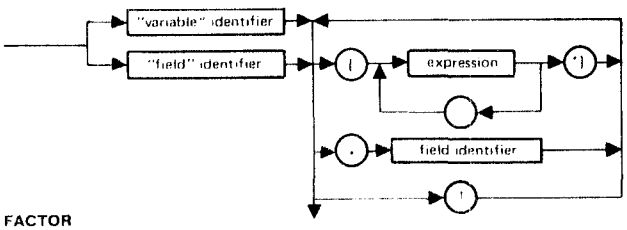
TYPE



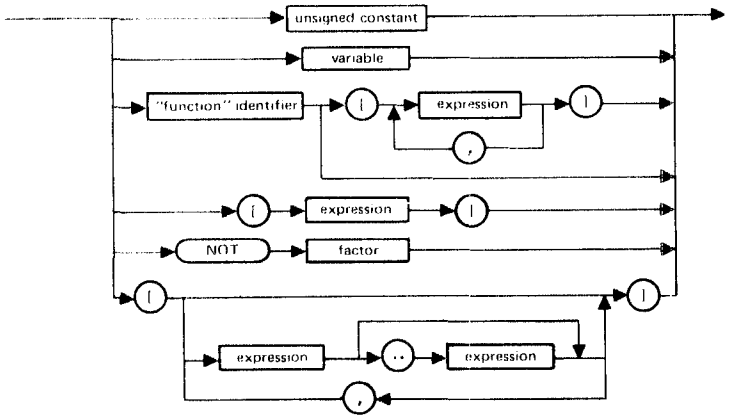
FIELD LIST



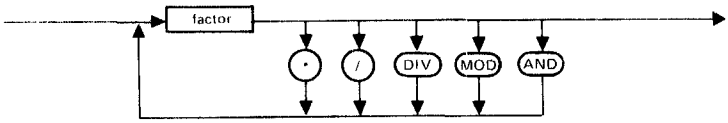
VARIABLE



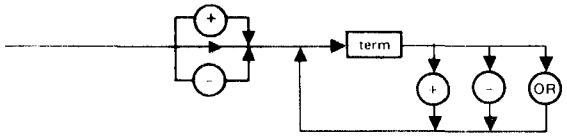
FACTOR



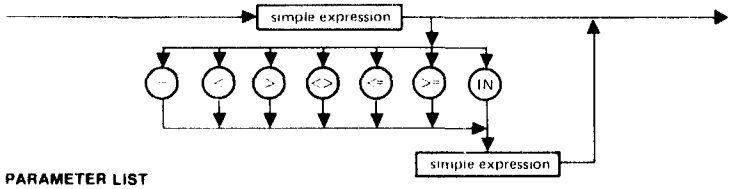
TERM



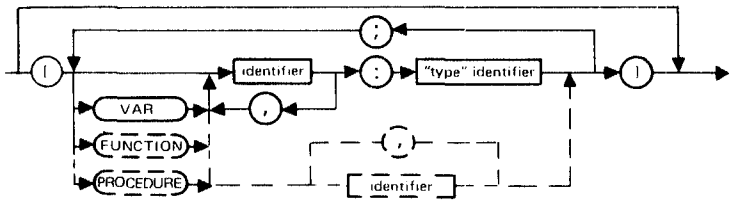
SIMPLE EXPRESSION



EXPRESSION

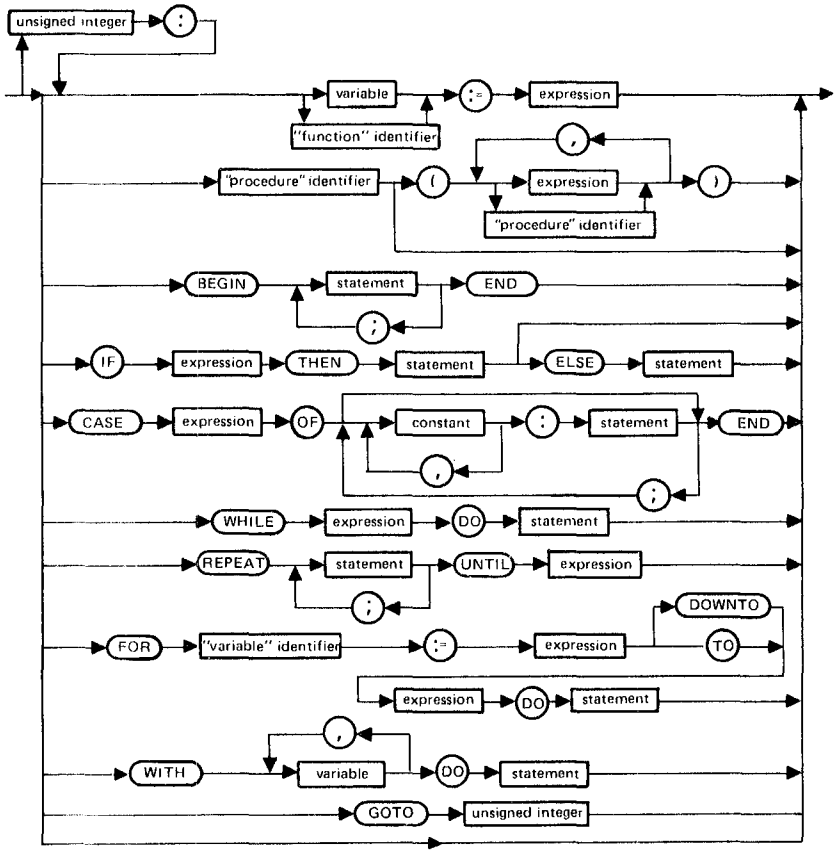


PARAMETER LIST

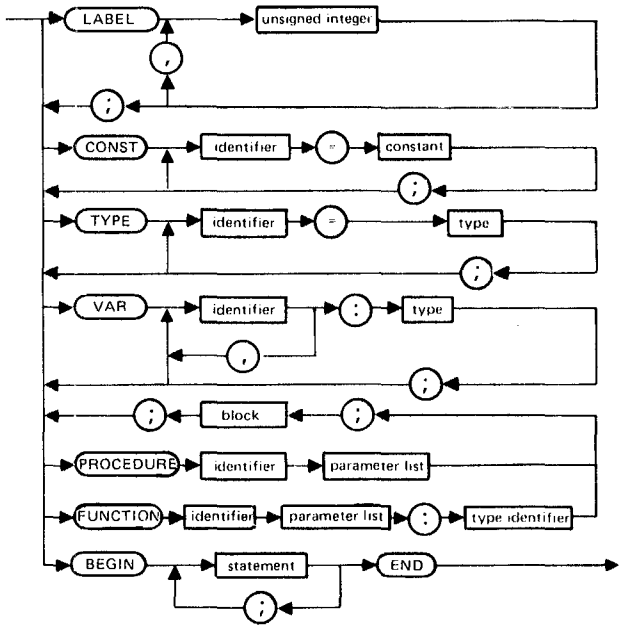


UCSD PASCAL SYNTAX DIAGRAMS

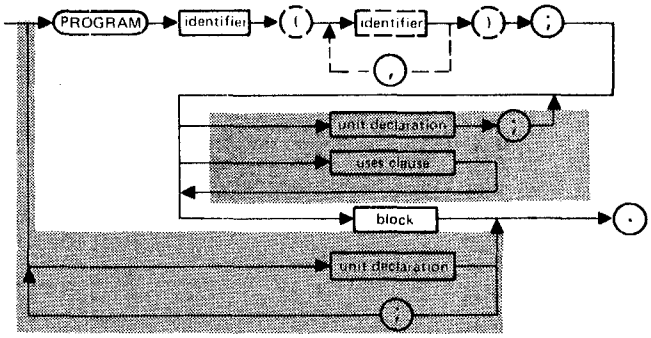
STATEMENT



BLOCK



COMPILATION



DASHED LINES (– –) ARE NOT INCLUDED
SHADED AREAS REPRESENT UCSD EXTENSIONS

APPENDIX D

GLOSSARY

ARRAY
An ordered arrangement of characters; a PACKED ARRAY OF CHAR.

BACKUP FILE
A copy of a file created for protection in case the primary file is destroyed unintentionally.

BAD BLOCK
A defective block on a storage medium, such as a disk, that produces a hardware error when attempting to read or write data to this block.

BASE SEGMENT
The portion of a segmented program that is always memory resident.

BLOCK
A group of characters or bytes transmitted as a unit; one disk block of 512 bytes.

BOOLEAN VARIABLE
A variable which, when evaluated, produces either a true or false result.

BOOTSTRAP
A routine whose first instructions are sufficient to load the remainder of the routine into memory from an input device. Normally, it starts a complex system of programs.

BREAKPOINT
A program point indicated by a breakpoint instruction that has been inserted by the Compiler to interrupt the program so that the program can be checked before continuing to completion; an aid for the Debugger.

BUFFER

A storage area used to hold information temporarily when it is being transferred between two devices or between a device and memory; often a specially designated area of memory.

CODE FILE

A file containing code to be executed; has the suffix of ".CODE".

COMMAND or COMMAND NAME

A word, mnemonic or character, by virtue of its syntax in a line of input, causes a predefined operation to be performed.

COMMAND STRING

A line of input that includes, generally, a command, one or more file specifications, and optional qualifiers.

COMPILE

The production of binary code (machine-readable) from symbolic instructions written in a high-level language.

COMPILER

Translates high-level language (Pascal or Basci) into machine code.

CONFIGURATION

A particular selection of hardware devices or software routines or programs that function together.

CONSOLE

The terminal that acts as the primary interface between the computer operator/user and the system; used to initiate and direct overall system operation.

CONSTANT

A value that remains the same throughout a distinct operation; as compared to a variable.

CONTROL CHARACTER

Controls an action rather that passing on data to a program.

CREATE

To open, write data to, and close a file for the first time.

DATA FILE

A file containing data to be manipulated by a program.

DEBUGGING

Searching for, and eliminating, sources of error (bugs) in a program.

DEFAULT

The value of an argument, operand or field assumed by a program if a specific assignment is not specified by the user.

DEVICE

A hardware unit such as an I/O peripheral (e.g., disk, video terminal);

the physical unit as opposed to VOLUME, the logical unit.

DIRECTORY
: A table that contains the names of, and pointers to, files on a mass-storage device.

DISASSEMBLER
: A program that translates object code back to source statements.

EXPRESSION
: A combination of commands and operands that can be evaluated to a distinct result.

FILE
: A logical collection of data treated as a unit; may be work, code, text, foto or data file.

FILE SPECIFICATION
: A name that identifies uniquely a file maintained in any system; must contain, at a minimum, the file name; may also contain the volume number and name.

FUNCTION
: A routine that returns a value.

HEXADECIMAL
: Whole numbers in positional notation using 16 as a base.

HIGH-LEVEL LANGUAGE
: A problem-oriented language rather that a machine-oriented one.

INITIALIZE
: Setting all hardware and software controls to starting values at the beginning of a new program.

INTERRUPT
: The suspension of the normal programming routine to handle a sudden request for service. After completion of interrupt service, the program is resumed where it left off.

KEYBOARD ENTRY DEVICE
: A device with a keybord (e.g., teletypewriter, video terminal) used by the system operator to control the system; CONSOLE.

LIBRARY
: A collection of programs or subprograms contained as segments in a library file; normally contains frequently needed routines that may be accessed by other programs.

LISTING
: A hard copy generated by a line printer.

LITERAL
: The explicit representation of character strings.

LOAD

To store a program or data in memory.

LOGICAL DEVICE NAME

An alphanumeric name assigned by the user to represent a physical device; used synonymously with the physical device name/number in the logical program.

MACHINE LANGUAGE

Instructions in binary code that can be operated on by the computer; as compared with high-level languages that can be read and understood by the user.

MAIN MEMORY

A set of storage locations connected directly to the processor.

NESTING

Routines enclosed within larger routines but not necessarily a part of the larger; a series of looping instructions may be nested.

OBJECT CODE

Relocatable machine-language code.

OBJECT PROGRAM

The source language program after it has been translated into machine language, output of the Compiler.

ON-LINE

Equipment and devices directly connected to, and controlled by, the central processing unit.

OVERLAY SEGMENT

A segment of code treated as a unit that can overlay code already in memory and be overlaid by other segments.

OVERLAY STRUCTURE

An overlay system consisting of a root segment and, optionally, one or more overlay segments.

PACK

To compress data in storage.

PROCEDURE

A routine that does not return a value.

QUALIFIER

A parameter specified in a command string that modifies some other parameter.

SOURCE LANGUAGE

A system of symbols and syntax easily understood by people that is used to describe a procedure that a computer can execute.

STACK

A block of successive memory locations accessible from one end on a LIFO basis (last-in-first-out).

SUBSCRIPT

A numerically valued expression or expression element that is appended to a variable name to uniquely identify elements of an array.

SWAPPING

Copying areas of memory to mass storage and back in order to use the memory for two or more purposes.

UTILITY

Any general-purpose program included in an operating system to perform common functions.

VARIABLE

The symbolic representation of a logical storage location that can contain a value that changes during a discrete processing operation, as compared to constant.

The MICROENGINE Company

3128 REDHILL AVENUE, BOX 2180
NEWPORT BEACH, CA 92663
(714) 557-3550, TWX 910-595-1139

PRINTED IN U.S.A.