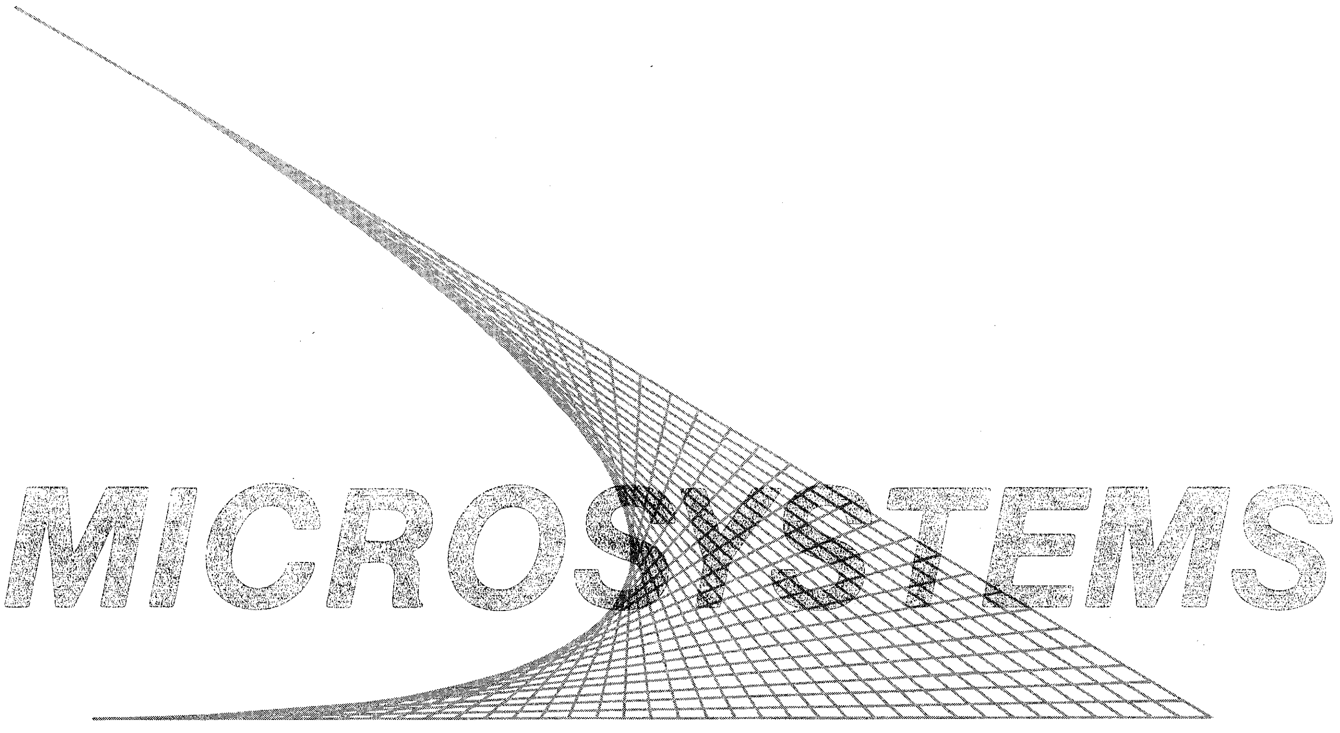
MOTOROLA

M68KVSOM/D1

VME/10 Microcomputer System Overview Manual



QUALITY • PEOPLE • PERFORMANCE

M68KVSOM/D1

FEBRUARY 1984

VME/10 MICROCOMPUTER SYSTEM

OVERVIEW MANUAL

WARNING

THIS EQUIPMENT GENERATES, USES, AND CAN RADIATE RADIO FREQUENCY ENERGY AND, IF NOT INSTALLED AND USED IN ACCORDANCE WITH THE INSTRUCTION MANUAL, MAY CAUSE INTERFERENCE TO RADIO COMMUNICATIONS. AS TEMPORARILY PERMITTED BY REGULATION, IT HAS NOT BEEN TESTED FOR COMPLIANCE WITH THE LIMITS FOR CLASS A COMPUTING DEVICES PURSUANT TO SUBPART J OF PART 15 OF FCC RULES, WHICH ARE DESIGNED TO PROVIDE REASONABLE PROTECTION AGAINST SUCH INTERFERENCE. OPERATION OF THIS EQUIPMENT IN A RESIDENTIAL AREA IS LIKELY TO CAUSE INTERFERENCE, IN WHICH CASE THE USER, AT HIS OWN EXPENSE, WILL BE REQUIRED TO TAKE WHATEVER MEASURES MAY BE REQUIRED TO CORRECT THE INTERFERENCE.

First Edition

PREFACE

Unless otherwise specified, all address references are in hexadecimal throughout this manual.

An asterisk (*) following the signal name for signals which are level significant denotes that the signal is true or valid when the signal is low.

An asterisk (*) following the signal name for signals which are edge significant denotes that the actions initiated by that signal occur on a high to low transition.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

TABLE OF CONTENTS (cont'd)

Page

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

GENERAL INFORMATION

1.1 INTRODUCTION

This manual provides general information, control and indicator descriptions, initialization procedures, and software-related information for the VME/10 Microcomputer System (hereafter referred to as VME/10). Before unpacking the system, powering it up, and performing any software operations described in this manual, refer to the VME/10 Microcomputer System Installation Guide, M68KVSIG.

The VME/10 provides the single user with 8- and 16-bit hardware/software and instrumentation development support and, eventually, 32-bit support. It incorporates the extended performance of the MC68010 MPU, the MC68451 Memory Management Unit (MMU), VMEbus compatibility, and full VERSAdos multitasking real-time operating system support, including high-level languages such as Pascal.

The VME/10 is comprised of a chassis, a keyboard, and a display unit. Refer to Figure 1-1.

1.2 FEATURES

The features of VME/10 are as follows:

. MC68010 16/32-bit Microprocessor Unit (MPU).

. MC68451 Memory Management Unit (MMU).

. Industry-standard VMEbus interface with full bus arbitration logic and software controllable interrupter.

. I/O Channel interface for adding off-board resources such as A/D converters, serial and parallel I/O ports, etc.

. 384K bytes RAM (triple-ported between graphics controller, local bus, and VMEbus).

. Static RAM for storage of user-definable character sets and display attributes.

. Two 28-pin sockets for ROM/PROM/EPROM storage of up to 64K bytes for custom applications.

. Battery backed-up time-of-day clock with 50 bytes of CMOS RAM storage.

. 15-inch video display having the following software-controllable display formats:

  a. 25 lines by 80 characters -- 8 x 10 characters with descenders (10 x 12 font).

  b. 800 x 300 pixel for normal resolution graphics.

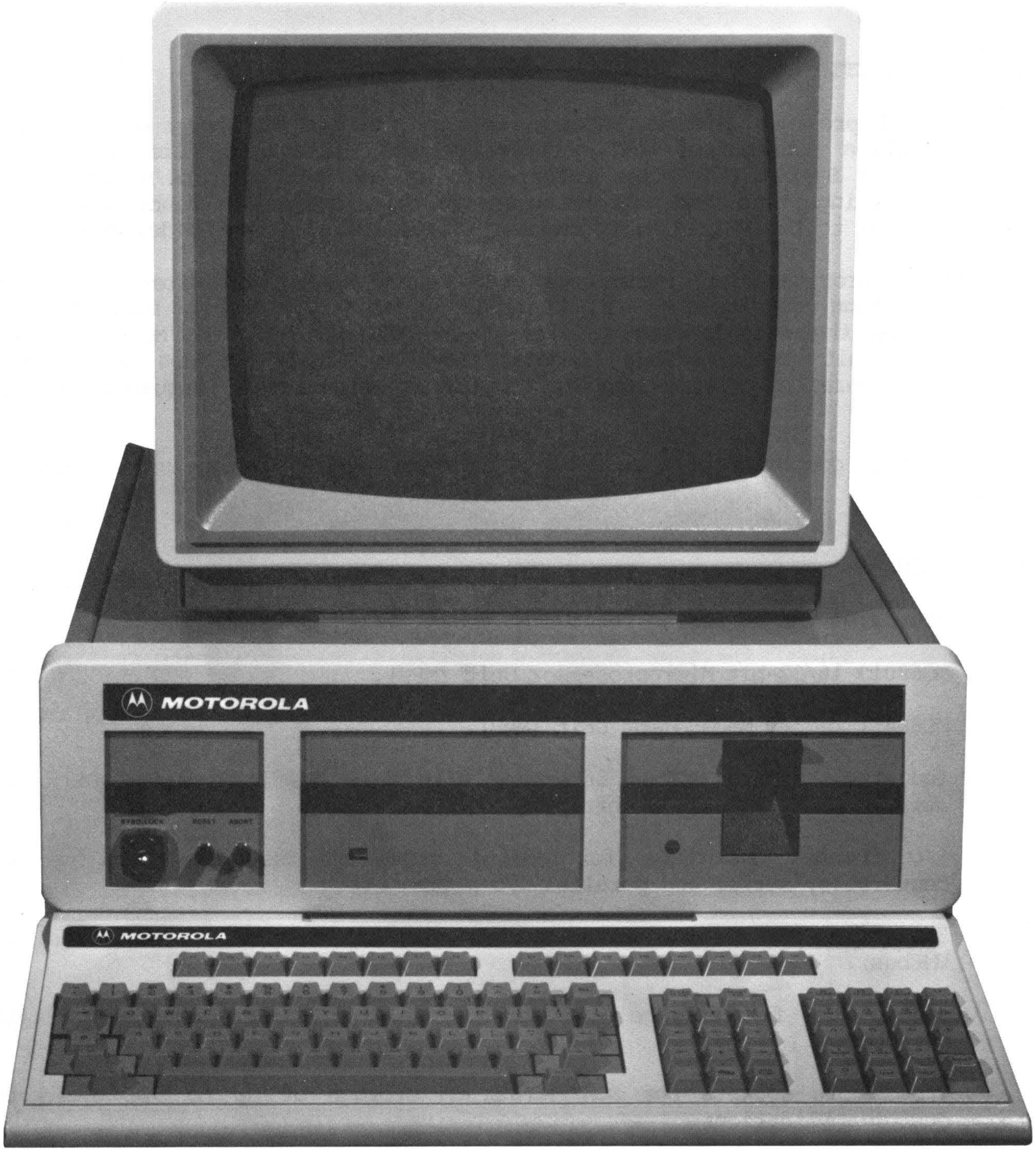


FIGURE 1-1. VME/10 Microcomputer System

c. 800 x 600 pixel for high resolution graphics.

d. Pixel graphics with overlaid character displays.

. Monochrome video display standard, with 7-level gray scaling (color optional).

. Detached full ASCII character set keyboard with cursor control keys, hexadecimal pad, and 16 function keys.

. Mass storage subsystem providing both 5-1/4" floppy disk and 5-1/4" Winchester disk storage units.

Floppy disk

1M-byte unformatted capacity (655K-byte formatted)

Winchester disk

Choice of: (a) 6.38M-byte unformatted capacity (5M-byte formatted)

(b) 19.1M-byte unformatted capacity (15M-byte formatted)

. Card cage options for feature expansion capability.

Choice of: (a) Five I/O Channel card cage slots (with 6.38M-byte Winchester option)

(b) Five VMEbus card cage slots with VMEbus backplane, plus four I/O Channel slots (with 19.1M-byte Winchester option)

. Conformance to ergonomic standards applicable to video display and keyboard.

. TENbug firmware-resident debug/monitor package.

. Firmware-resident power-up/reset and disk-resident module diagnostic self-test.

. VERSAdos real-time multitasking operating system with M68000 macro assembler, plus tools and utilities.

. Capability of hosting hardware development tools.

- HDS-400 for M68000 family 16/32-bit emulation

- HDS-200 for M6800 family 8-bit emulation

- Bus state analyzer for logic analysis functions

1.3 SPECIFICATIONS

Table 1-1 lists the specifications for the VME/10.

TABLE 1-1. VME/10 Microcomputer System Specifications

CHARACTERISTIC	SPECIFICATION	
Microprocessor	MC68010	
MPU clock frequency	10 MHz	
Word size		
Data	1-, 8-, 16-bit	
Address	24-bit	
Memory address capability	16M bytes (8 bits/byte)	
Bus standard	VMEbus	
Clock frequency	16 MHz	
Bus cycle time	200 ns (min.)	
Interrupt control	7-level priority	
Bus arbitration	4-level daisy-chained	
Data	16-bit	
Address	24-bit	
Control	Asynchronous	
Temperature		
Operating	10° to 40° C	
Storage	-40° to 60° C	
Relative humidity	20% to 80% (non-condensing)	
Physical dimensions	Chassis & monitor	Keyboard
Length	22.8 in. (57.9 cm)	8.3 in. (21.1 cm)
Width	19.0 in. (48.3 cm)	19.0 in. (48.3 cm)
Height	20.0 in. (50.8 cm)	2.0 in. (5.1 cm)
Weight	50 lbs. (23 kg)	5 lbs. (2.3 kg)
Power requirements (switching power supply)	90-132 Vac, 47-63 Hz, 500 W	
	180-264 Vac, 47-63 Hz, 500 W	

1.4 EQUIPMENT SUPPLIED

Table 1-2 lists the part number and description for the standard system configurations.

TABLE 1-2. Standard System Configuration

PART NUMBER	DESCRIPTION
M68K101-1	VME/10 Microcomputer System, including the MC68010 Microprocessor Unit, MC68451 Memory Management Unit, 384K bytes dynamic RAM, 655K-byte (formatted) 5-1/4" floppy disk unit, 5M-byte (formatted) 5-1/4" Winchester disk unit, 15-inch monochrome video display, and full ASCII keyboard with cursor control keys, hexadecimal keypad, and 16 user functional keys. For 115 Vac, 60 Hz operation. Expansion card cage incorporates five slots for single wide I/Omodule cards, plus ribbon cable and connectors to provide the I/O Channel interface functions to each card slot. System software includes the VERSAdos operating system, plus M68000 Family Structured Macro Assembler, Symbolic Debugger, CRT Editor, and Linkage Editor. A comprehensive diagnostics package is also included. The software is resident on the Winchester hard disk. System firmware incorporates (a) power-up self-test function,(b) disk bootstrap loader, and (c) TENbug Debug/Monitor package.
M68K102B1	Same as M68K101-1, except as follows: . Expansion card cage provides five slots for double format VMEmodule cards, plus 5-position VMEbus backplane at the rear of the card cage. Also includes four slots for single wide I/Omodule cards, with necessary cabling and connectors to provide the I/O Channel interface to each card slot. . 15M-byte (formatted) 5-1/4" Winchester disk unit replacing 5M-byte unit in M68K101-1).
M68K101-2	Same as M68K101-1, but for 230 Vac, 50 Hz operation.
M68K102B2	Same as M68K102B1, but for 230 Vac, 50 Hz operation.

1.5 I/O CHANNEL AND VME EQUIPMENT OPTIONS

Table 1-3 lists the part number and description for the optional equipment.

TABLE 1-3. Optional Equipment

PART NUMBER	DESCRIPTION
Modular expansion options - VMEbus	
MVME200	64K dynamic RAM with byte parity
MVME201	256K dynamic RAM with byte parity
MVME210	Static ROM/RAM module
MVME300	High performance IEEE-488 GPIB Controller with DMA
Modular I/O expansion options - I/O Channel	
MVME400	Dual RS-232C serial port
MVME410	Dual 16-bit parallel port (see NOTE)
MVME420	SASI adapter
MVME435	Buffered 9-track magnetic tape adapter
MVME600	12-bit analog input module
MVME601	16-channel expander for MVME600
MVME605	12-bit analog output module
MVME610	Opto-isolated 120V/240V ac input
MVME615/616	Opto-isolated 120V/240V ac output
MVME620	Opto-isolated 60 Vdc input
MVME625	Opto-isolated 60 Vdc output
MVME935	Remote I/O Channel extender cable connection module
Remote I/O Channel modules	
M68RAD1	Remote intelligent analog conversion module
M68RIO1	Remote I/O solid state relay module

NOTE: This module recommended for parallel printer interface port applications with the VME/10 system.

1.6 SYSTEMS DEVELOPMENT AND INTEGRATION

The initial stages of developing a microprocessor-based system normally involve two parallel, rather independent, efforts. One is the hardware design -- the other the software design.

The VME/10, when used as the host in conjunction with Motorola's line of hardware development stations, simplifies the design process because of its ability to unite the hardware and software development processes throughout the development cycle.

The hardware development station provides a complete hardware and software development system for Motorola's families of microprocessors. Two major factors contribute to the usefulness of the hardware development station as a systems development tool. The first is its ability to serve as a fully functional substitute for the selected microprocessor or microcomputer in the target system. When plugged into the socket on the prototype hardware, the hardware development station provides efficient testing of hardware as well as software. The second factor is its ability to interface with Motorola's Real-Time Bus State Analyzer (BSA), which speeds the debugging process and allows program code optimization. See Figure 1-2.

The BSA is a development tool which allows simultaneous monitoring of different points in a system. Interfacing through the system bus or the MPU bus, the BSA tracks events occurring on each line of the bus, storing the information for later analysis and interpretation.

The emulator can be configured and software run prior to availability of prototype target system hardware. This allows an early start on the debugging process. As hardware changes occur, software updating and debugging are readily accomplished. Moreover, with the hardware development station, it becomes economically feasible to test alternate design approaches to determine the best solution.

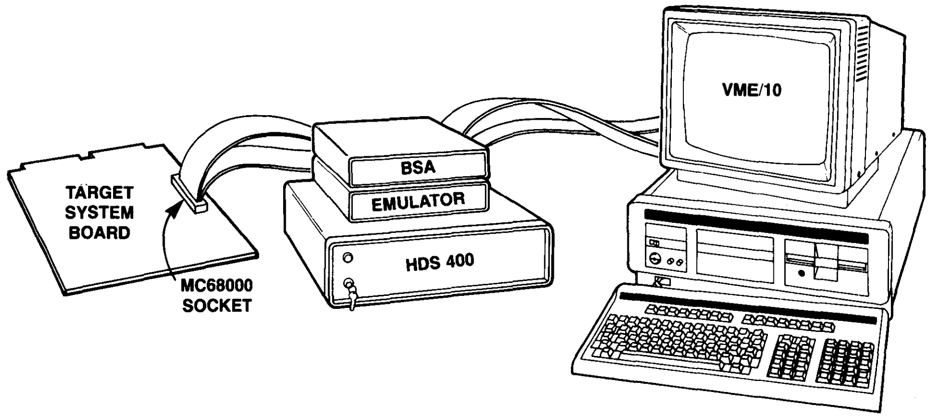


FIGURE 1-2. Typical VME/10 Microcomputer System Development Integration

1.7 GENERAL DESCRIPTION

The VME/10 basic system consists of three assemblies:

- Control unit chassis
- Display unit
- Keyboard

1.7.1 Control Unit Chassis

The control unit chassis, as shown in Figure 1-3, provides the housing for:

- System control module
- Mass storage subsystem, consisting of floppy and Winchester disks
- Expansion card cage
- Power supply
- Cooling fan
- Operator panel

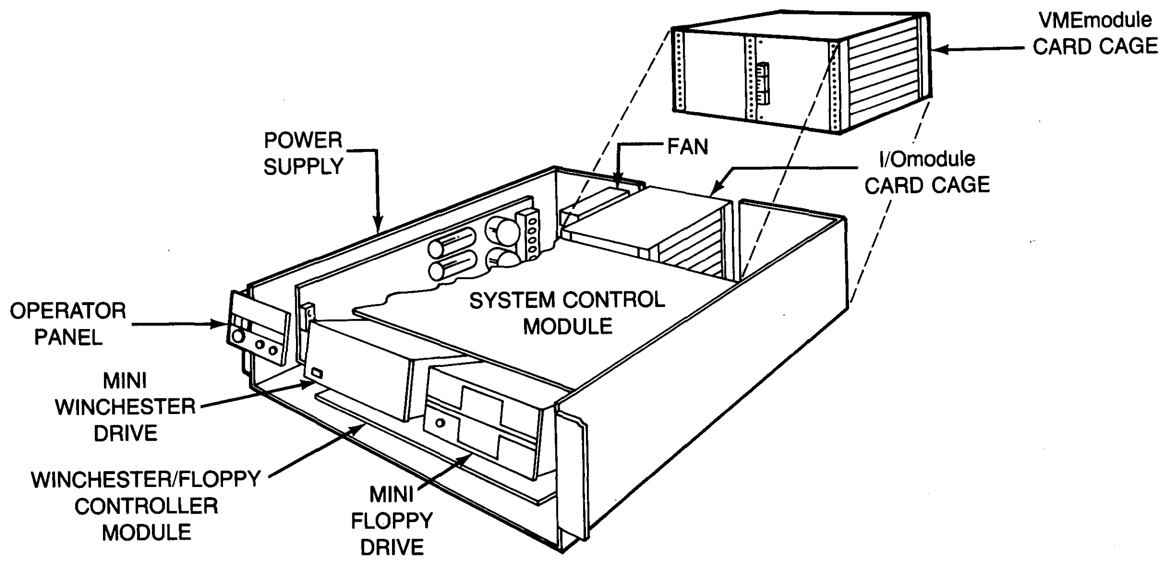


FIGURE 1-3. Control Unit Chassis

1.7.2 System Control Module

The system control module, which contains the central intelligence of the VME/10, consists of two boards -- the processor/MMU board and the graphics/interface board. See Figure 1-4.

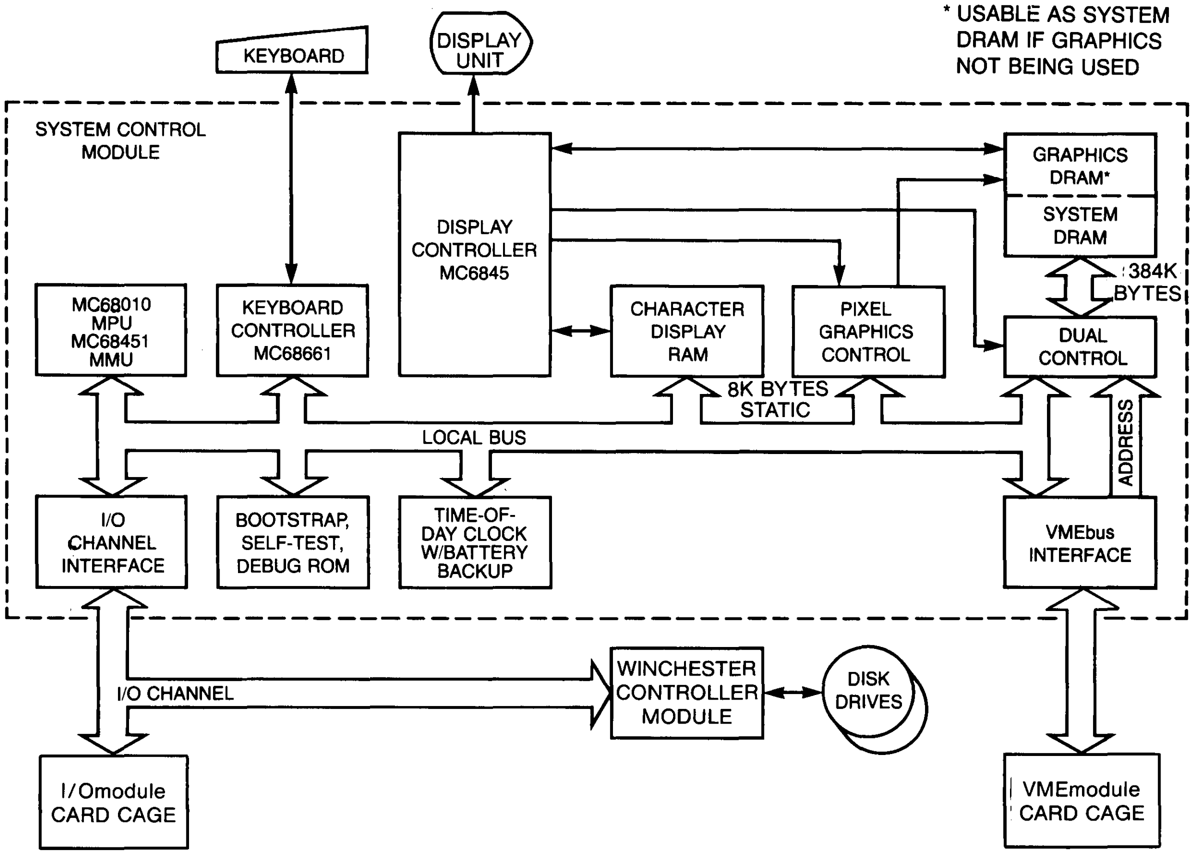


FIGURE 1-4. VME/10 Microcomputer System Block Diagram

Processor/MMU Board

The combination of the MC68010 MPU and MC68451 MMU provides processing power sufficient to permit several development tasks to proceed simultaneously -- editing, program development, system debugging -- with full protection for each task. This fundamental processor/memory architecture also provides designers with the protection features required in multitasking OEM systems where security and protection of both programs and data are essential. The Processor/MMU board contains one MC68451 MMU device which provides up to 32 separate program/data segments and three extra sockets into which additional MMU devices may be optionally installed to allow up to 96 additional segments.

Graphics/Interface Board

The graphics/interface board (VMEC1) contains the major elements of high-speed semiconductor memory in the system, plus the graphics subsystem and interfaces to various off-board devices. The graphics/interface board incorporates the following features:

a. 384K bytes RAM - utilizing 64K X 1 HMOS RAM technology for operating system, support software, user programs, and graphics subsystem display buffer storage. (See item i for detail on the graphics subsystem.) For increased performance and minimum contention, the on-board RAM is multi-ported to allow shared access from the local on-board bus, the VMEbus, and the graphics controller.

b. 16K bytes ROM/PROM/EPROM - used for bootload and test and diagnostic routines for debugging and system start-up and control.

c. Interrupt handler - allows 22 sources of interrupts to the MC68010 Microprocessor.

d. Time-of-day clock (MC146818) - an 8-bit real-time clock including 50 bytes of general-purpose RAM for saving critical information. Both the clock and RAM are battery backed up, allowing up to five days of data retention with fully charged batteries at power down.

e. Keyboard interface (MC68661) - uses RS-422C type buffers (multi-drop, 5-volt, differential communication line) to communicate with the keyboard.

f. Local on-board bus - provides communication between the MPU, ROM, RAM, CRT controller, keyboard interface, battery backed-up time-of-day clock, I/O Channel, and the VMEbus. This architecture allows the on-board processor to continue operating at full speed, while other (optional) VMEbus masters operate simultaneously.

g. I/O Channel interface - links the local on-board bus to the I/O Channel cable for communication to the mass storage subsystem and any optional I/O cards installed.

h. Industry-standard VMEbus compatibility - is supported by three functions:

   1. The VMEbus interface which provides the data and address path from the on-board MPU via the local bus to the VMEbus to allow VMEbus use in a system requiring additional off-board resources such as additional memory, processors, or intelligent device controllers.

   2. The VMEbus arbiter which arbitrates all four bus request priority levels, with operation transparent to software.

   3. The VMEbus requester which is used to gain access to the resources on the VMEbus. Bus requests can be made either indirectly (software transparent) or directly by specific request through corresponding status and control registers under program control. The associated VMEbus interrupter logic permits the MPU to place an interrupt on one of the seven VMEbus interrupt request lines. The related interrupt handler can be software configured to respond to any subset of the seven VMEbus interrupt request lines, thus allowing several boards with interrupt handlers to respond to different interrupt levels from the VMEbus.

i. Graphics Subsystem - provides efficient graphics support hardware at low cost.

The video graphics subsystem generates all video and display synchronization signals required by both monochrome and color display units. Initially, the VME/10 incorporates a monochrome video display.

Fundamentally, the VME/10 displays characters in a 25-line by 80-column format, or high resolution pixel graphics in an 800 x 600 pixel matrix, or a specialized combination display of both character and pixel graphics simultaneously. Within this basic framework, additional capabilities are provided for (a) alternate normal resolution pixel graphics within an 800 x 300 point field, and (b) user definition through software of specialized character sets, fonts, and display attributes. An MC6845 CRT controller device is used by software to define the video screen.

The 384K-byte RAM in the system control module has the dual purpose of general software storage and graphics data storage. If no pixel graphics capability is being used, all of the RAM on the system control module is available as system RAM. If graphics is being used in the normal resolution mode, the graphics display RAM is located in the high addressed 96K bytes of RAM. Alternately, in the high resolution mode, the high addressed 192K bytes of RAM are dedicated to graphics.

The VME/10 graphics subsystem combines separate character and graphic display memories. The character display memory is 16 bits wide and 2K (4K optional) deep ($F17000-$F18FFF). The 16 bits of each word are defined in Figure 1-5.

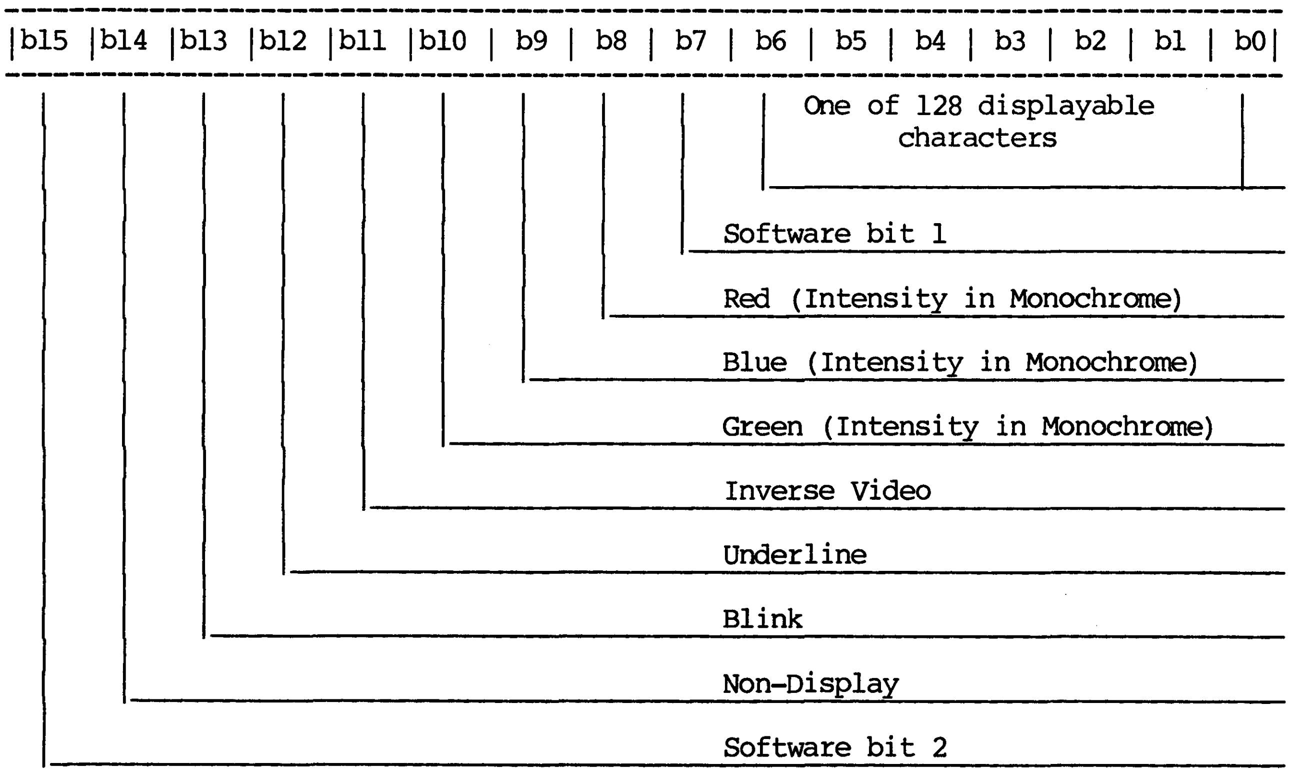

b15
b14
b13
b12
b11
b10
b9
b8
b7
b6
b5
b4
b3
b2
b1
b0
One of 128 displayable characters
Software bit 1
Red (Intensity in Monochrome)
Blue (Intensity in Monochrome)
Green (Intensity in Monochrome)
Inverse Video
Underline
Blink
Non-Display
Software bit 2

FIGURE 1-5. Display Memory Word Format Definition

The character generator RAM is initialized with the ASCII character definitions, but can be modified by the user to define alternate special symbols required by the application. The font is 8 X 16 in a 10 X 24 character field for the high resolution monochrome display, and 8 X 10 in a 10 X 12 character field for the normal resolution/color display.

Two modes of pixel graphics operation are available on the VME/10 -- the group access mode and the pixel access mode. The application determines the most appropriate mode to use. The group access mode utilizes three separate contiguous banks of memory. Each bank represents a primary color for color applications, or an intensity level for monochrome applications. The graphics display buffer RAM block in the memory map is organized so that the first third of the graphics RAM location is bank 1, the second third is bank 2, and the third is bank 3. This organization allows the MPU to change 8 or 16 contiguous pixels on the screen at one time in one color/intensity bank. This mode is particularly useful for drawing bar graphs, color filling an object, or blanking the screen.

The graphics RAM is accessible in another mode called the pixel access mode. In this mode, read/write hardware exists to allow the processor the ability to change one pixel at a time (per memory cycle) in all three banks. In this mode, the processor uses only word accesses, and writes a special "pixel access word".

The organization of the pixel access word is as shown in Figure 1-6. The address range is $E00000-$EFFFFF.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
					MASK1	MASK2	MASK3						BANK1	BANK2	BANK3

FIGURE 1-6. Pixel Access Word Format Definition

The pixel access mode is oriented toward drawing lines or changing a portion of a display. The mask bits allow the user to avoid disturbing the contents of a given plane(s) while changing the contents of another plane(s). This technique minimizes the software design effort required and improves system performance. It further eliminates the necessity to rewrite data into bank addresses when the data remains unchanged.

Mass Storage Subsystem

The mass storage subsystem consists of a disk controller board, plus one 5-1/4 inch floppy disk drive unit and one 5-1/4 inch Winchester disk drive unit. The floppy disk drive provides storage of 1.0M bytes unformatted (655K byte formatted) capacity, incorporating reliable dual-density, dual-sided 96 tracks/inch (TPI) technology. The associated Winchester disk drive unit is available in either of two storage capacities, depending on the specific VME/10 model:

Option #1 - 6.38M bytes unformatted capacity (5M bytes formatted)

Option #2 - 19.1M bytes unformatted capacity (15M bytes formatted)

Expansion Card Cage

An expansion card cage is provided with card plug-in access from the rear panel of the control unit chassis. Either of two card cage options is installed, depending on the specific VME/10 model:

Option #1 - 5-slot single wide card cage with connectors and ribbon cable installed to make the I/O Channel available in each of the five card locations. This card cage is standard with the 6.38M-byte Winchester disk model of the VME/10.

Option #2 - Combination VMEbus and I/O Channel card cage, with 5-slot VMEbus backplane, accommodating up to five double wide VMEmodule boards, plus up to four single wide I/Omodules. All necessary I/O Channel cabling and connectors are installed to serve the four I/O Channel card slots. This card cage is standard with the 19.1M-byte Winchester disk model. Refer to Figure 1-7.

These card cage options permit convenient system expansion and customization through addition of I/O Channel and/or VMEbus compatible cards. Common applications are additional global system memory, serial and parallel ports, analog conversion functions, and an IEEE-488 interface to intelligent instrumentation.

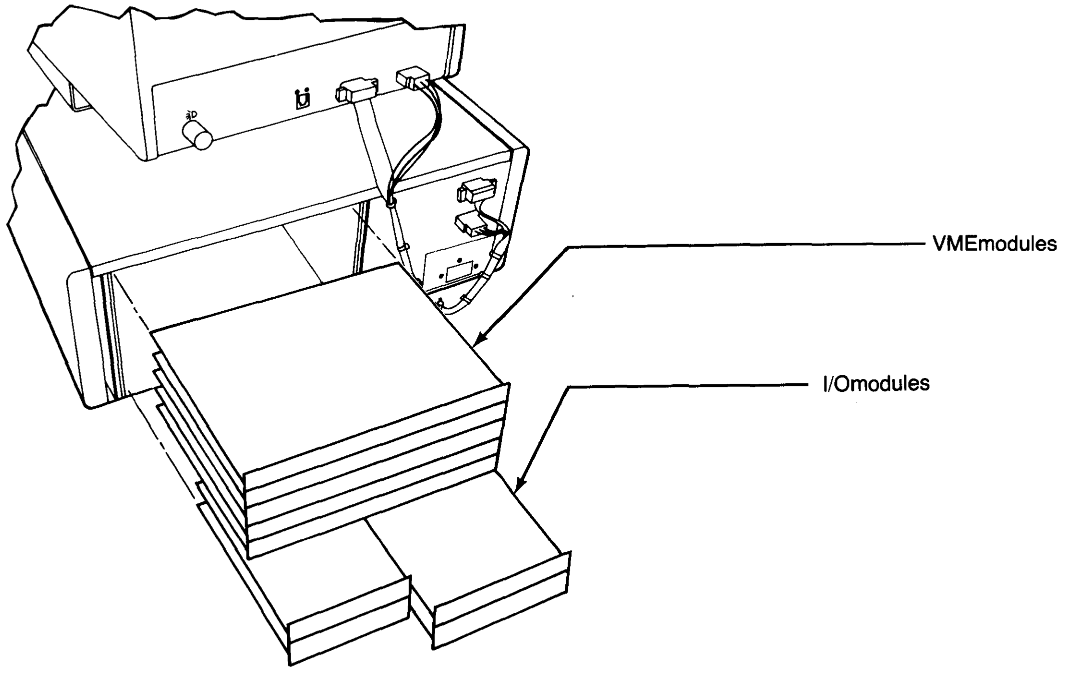


FIGURE 1-7. VMEbus and I/O Channel Module Expansion Card Cage Options

1.7.3 Display Unit

The standard video display is a 15-inch (diagonal) monochrome unit with antiglare, P39 (green) phospor screen. The standard ASCII character set with 8 X 10 characters with lowercase descenders in a 10 X 12 font is used over a 25 line X 80 character display in normal mode. Optionally, 800 X 600 pixel high resolution graphics, 800 X 300 pixel normal resolution graphics, or a combination of characters and graphics may be displayed.

The character set and attributes displayed can be changed by altering the contents of the character/attribute static RAM. Characters or individual pixels can be displayed in any of seven levels of the gray scale on the monochrome display.

A color display monitor of the same size is a planned option.

The display unit is accommodated by a tilt and swivel stand that is mounted to the top of the control unit chassis. The screen is tilted by pressing down at the bottom edge of the screen enclosure; therefore, it cannot be inadvertently moved by leaning on the front top of the screen. This configuration conforms to all ergonomic standards in order to increase user comfort and productivity.

1.7.4 Keyboard

The keyboard is connected to the control unit chassis by a serial link, using a coiled telephone cable with quick disconnect "modular" connectors. The keypad consists of a full ASCII character set with interchangeable keycaps, 16 function keys, a 7-key cursor/tab control pad, and a hexadecimal keypad. The keyboard is also designed to conform to all ergonomic design principles.

1.8 VERSAdos OPERATING SYSTEM AND DEVELOPMENT TOOLS

To achieve most efficient use of the varied system resources provided by the VME/10, efficient system control and management software is required. This need is met by the VERSAdos operating system and its related family of development tools and utilities for the M68000 and M6800 MPU families. VERSAdos incorporates a modular, multilayer design supporting a variety of application environments, and is especially well suited to real-time control system and applications. Because it provides a convenient, friendly interface between the user and the system resources, including a wealth of development support software, VERSAdos has emerged as the operating system environment of choice for increasing numbers of system developers who are incorporating the Motorola 8- and 16-bit microprocessor families into various end applications. VERSAdos is particularly well suited to providing a common host/target environment, thereby minimizing the task of system integration.

As packaged with the VME/10, the complete basic VERSAdos operation environment includes, in addition to the basic operating system, an M68000 family structured macro assembler, symbolic debugger, CRT-oriented text editor, and M68000 linkage family editor. A diagnostics package is also provided for use in helping to isolate suspected hardware system problems. This software is resident on the Winchester hard disk.

Figure 1-8 illustrates the VERSAdos operating system and related utility programs.

1.8.1 VERSAdos Operating System

VERSAdos is a multitasking, multiprogramming system executing on the MC68010 MPU in the VME/10. Programs execute in dynamically-assigned, variable-length segments with read/write privileges. Instructions and data are located in separate memory segments.

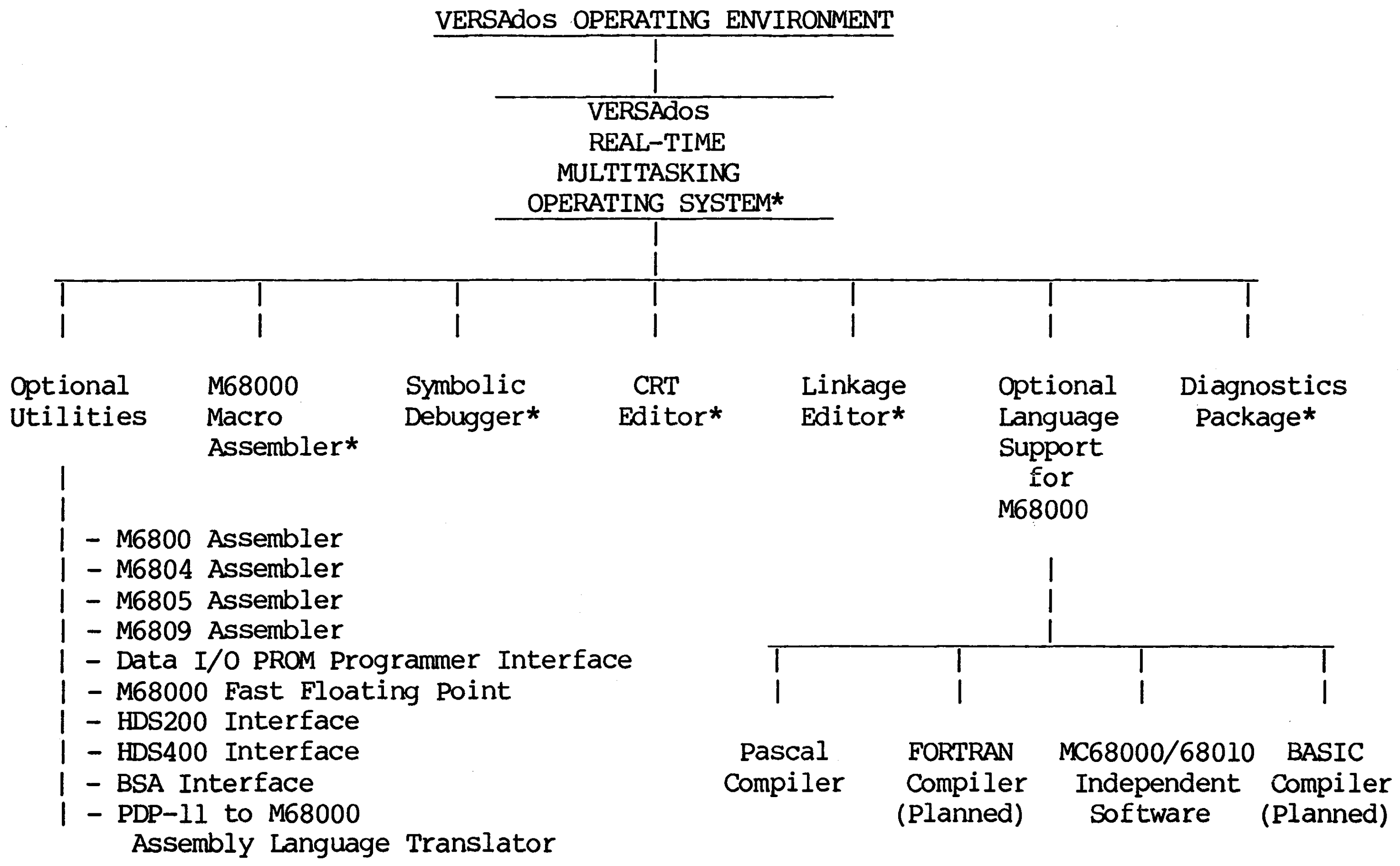
The heart of VERSAdos is the RMS68K Real-Time Executive which provides task services and supports memory management. It also allows inter-task communication, provides exception monitor facilities, and handles system interrupts.

The Input/Output (I/O) subsystem of VERSAdos supports device independence, logic I/O, overlapped computation, and physical I/O. Both sequential and random record access are supported by VERSAdos.

The powerful VERSAdos file management system supports three file structures -- contiguous, sequential, and indexed sequential. Other file system features include disk and file protection, shared file access, dynamic file allocation, and fixed or active protection.

1.8.2 Resident Structured Assembler

The M68000 resident structured macro assembler translates source statements into relocatable machine code, assigns storage locations to instructions and data, performs auxiliary assembler actions designated by the programmer, and optionally produces a cross reference listing. The M68000 resident assembler includes macro and conditional assembly capabilities, plus certain "structured programming" control structures such as "for, repeat, while" loops and "if-then, if-then-else" conditional branches.


VERSAdos OPERATING ENVIRONMENT
VERSAdos REAL-TIME MULTITASKING OPERATING SYSTEM*
Optional Utilities
M68000 Macro Assembler*
Symbolic Debugger*
CRT Editor*
Linkage Editor*
Optional Language Support for M68000
Diagnostics Package*
- M6800 Assembler
- M6804 Assembler
- M6805 Assembler
- M6809 Assembler
- Data I/O PROM Programmer Interface
- M68000 Fast Floating Point
- HDS200 Interface
- HDS400 Interface
- BSA Interface
- PDP-11 to M68000 Assembly Language Translator
Pascal Compiler
FORTRAN Compiler (Planned)
MC68000/68010 Independent Software
BASIC Compiler (Planned)

*Included on mass storage media.

FIGURE 1-8. VERSAdos Operating System and Related Utility Programs

1.8.3 Symbolic Debugger

The symbolic debugger (SYMbug) program is used to debug other programs whose source code is written in Motorola's Structured Macro Assembler for execution on the M68000 Family MPU's. The language processors, in cooperation with the linkage editor, supply information to SYMbug. This permits the user to describe the debugging requirements to SYMbug in terms close to the language in which the source program was written. SYMbug allows the user to debug in symbolic terminology. SYMbug allows the user to perform the following:

- Examine, insert, and modify program elements such as instructions, numeric values, and coded information (i.e., data in all its representations and formats).

- Control execution, including the insertion of breakpoints into a program and request for breaks or changes in elements of data.

- Trace execution by displaying information at designated points in a program.

- Search programs and data for specific elements and sub-elements.

- Create macro commands allowing user-defined formats and commands.

1.8.4 CRT Text Editor

The CRT-oriented Text Editor provides the capability to create and modify source programs. The editor supports both command and page editing, utilizing the cursor control keys, control characters, and function keys of the VME/10 keyboard chassis to insert, alter, or delete characters and lines within a user text file.

1.8.5 Linkage Editor

The Linkage Editor provides the capability of converting one or more separately compiled object units into a loadable object module file. The editor determines segment attributes, calculates address space, searches libraries, resolves external references, relocates object code, and issues error messages. At the end of a linkage process, the editor prints a report that contains a module map, a table of externally defined symbols, and any unresolved or multiply defined symbols.

1.8.6 Diagnostic Package

The diagnostic package verifies the overall functionality of the VME/10 by exposing it to a set of off-line tests. The package provides two levels of diagnostics. The first level is the firmware-resident, power-up/reset test. The second level is comprised of disk-resident diagnostics for more extensive hardware testing. The governing guideline for diagnostics is to provide a comprehensive test package that will isolate a malfunction to a functional block and at least down to the faulty module. For detailed information, refer to the VME/10 Microcomputer System Diagnostics Manual (M68KVSDM).

1.9 SYSTEM MEMORY MAP

Figure 1-9 depicts the 16-megabyte system memory map.

ADDRESS	
\$000000 \$00FFFF	SYSTEM RAM AFTER UNSWAP GOES FROM 0 TO 1 SYSTEM ROM AFTER POWER ON RESET
\$010000 \$05FFFF	SYSTEM RAM AND GRAPHICS RAM (SEE FIGURE 1-10)
\$060000 \$17FFFF	RESERVED FOR RAM EXPANSION
\$180000 \$DFFFFF	VMEbus (see NOTE)
\$E00000 \$EFFFFF	GRAPHICS - PIXEL ACCESS ADDRESSING BLOCK
\$F00000 \$F0FFFF	SYSTEM ROM AFTER UNSWAP BIT GOES FROM 0 TO 1 SYSTEM RAM AFTER POWER ON RESET
\$F10000 \$F1DFFF	I/O (SEE FIGURE 1-11)
\$F1E000 \$FFFFFF	VMEbus
NOTE:	The RAM (384K bytes) on the System Control Module can be accessed by off-board devices via the VMEbus. The base address of the dual-ported RAM on the VMEbus is \$D00000-D5FFFF.

FIGURE 1-9. VME/10 System Memory Map

HIGH RESOLUTION GRAPHICS

\$010000 \$02FFFF	SYSTEM RAM
\$030000 \$05FFFF	GRAPHICS AND SYSTEM RAM

NORMAL RESOLUTION GRAPHICS

\$010000 \$047FFF	SYSTEM RAM
\$048000 \$05FFFF	GRAPHICS AND SYSTEM RAM

FIGURE 1-10. High and Normal Resolution Graphics

ADDRESS	D15-D08 UPPER DATA	D07-D00 LOWER DATA
$F10000 $F13FFF	ILLEGAL	
$F14000 $F14FFF	ILLEGAL	CHARACTER GENERATOR RAM
$F15000 $F15FFF	ILLEGAL	ATTRIBUTE GENERATOR RAM
$F16000 $F16FFF	ILLEGAL	
$F17000 $F18FFF	DISPLAY MEMORY	
$F19000 $F19EFF	ILLEGAL	
$F19F00	VERTICAL GRAPHICS CURSOR REGISTER	
$F19F02	HORIZONTAL GRAPHICS CURSOR REGISTER	
$F19F05	ILLEGAL	CONTROL REGISTER 0
$F19F07	ILLEGAL	CONTROL REGISTER 1
$F19F09	ILLEGAL	CONTROL REGISTER 2
$F19F0B	ILLEGAL	CONTROL REGISTER 3
$F19F0D	ILLEGAL	CONTROL REGISTER 4
$F19F0F	ILLEGAL	CONTROL REGISTER 5
$F19F11	ILLEGAL	CONTROL REGISTER 6
$F19F13	ILLEGAL	GRAPHICS OFFSET REGISTER
$F19F20 $F19F7F	RESERVED	
$F19F85	ILLEGAL	STATUS REGISTER
$F19FA0 $F1A01F	RESERVED	

FIGURE 1-11. VME/10 I/O Memory Map (Sheet 1 of 2)

ADDRESS	D15-D08 UPPER DATA	D07-D00 LOWER DATA	
$F1A021 $F1A023	ILLEGAL	ADDRESS REGISTER INTERNAL REGISTER FILE	MC68A45
$F1A025 $F1A02F	ILLEGAL		
$F1A031 $F1A033 $F1A035 $F1A037	ILLEGAL	TX/RX DATA REGISTERS STATUS REGISTER MODE 1, MODE 2 REGISTERS COMMAND REGISTER	MC2661
$F1A039 $F1A07F	ILLEGAL		
$F1A081 $F1A083 $F1A085 $F1A087 $F1A089 $F1A08B $F1A08D $F1A08F $F1A091 $F1A093 $F1A095 $F1A097 $F1A099 $F1A09B		SECONDS REGISTER SECONDS ALARM REGISTER MINUTES REGISTER MINUTES ALARM REGISTER HOURS REGISTER HOURS ALARM REGISTER DAY OF THE WEEK REGISTER DAY OF THE MONTH REGISTER MONTH REGISTER YEAR REGISTER REGISTER A REGISTER B REGISTER C REGISTER D	MC146818
$F1A09D $F1A0FF	ILLEGAL	BATTERY BACKED UP RAM TIME OF DAY CLOCK	MC146818
$F1A100 $F1A7FF	ILLEGAL		
$F1A800 $F1AFFF	DMA/MMU		
$F1B000 $F1BFFF	ILLEGAL		
$F1C000 $F1DFFF	ILLEGAL	I/O CHANNEL (SEE NOTE)	

NOTE: The standard SYSGEN program dictates that if a printer is included, the MVME410 module should be addressed at $F1C1E1-$F1C1FF. Similarly, if there are serial ports, the MVME400 module should be addressed at $F1C1C1-$F1C1DF. These are recommended addresses reserved for the modules.

FIGURE 1-11. VME/10 I/O Memory Map (Sheet 2 of 2)

CHAPTER 2

HARDWARE/SOFTWARE SYSTEM STARTUP

2.1 INTRODUCTION

This chapter provides system initialization, media backup procedures, and system performance verification for the VME/10. Commands and other input/output (I/O) are presented in this manual in a modified Backus-Naur Form (BNF) syntax. Certain symbols in the syntax are not to be typed; their usage is restricted to the syntactic structure. These symbols and their meanings are as follows:

< >	The angular brackets enclose a "syntactic variable", that is replaced in a command line by one of a class of items it represents.
|	This symbol indicates that a choice is to be made. One of several items, separated by this symbol, should be selected.
[ ]	Square brackets enclose an optional item. The enclosed item may occur zero or one time.
[ ]...	Square brackets followed by periods enclose an item that is optional/repetitive. The item may appear zero or more times.

Operator entries are shown underscored for clarity (the underscore is not typed), and are to be followed by pressing the carriage return key (<--'). When a carriage return is the only required entry, it is shown as (CR).

2.2 SYSTEM POWER-UP

After the VME/10 has been correctly installed as directed in the VME/10 Microcomputer System Installation Guide, M68KVSIG, turn on the system by setting the 0/1 rocker switch on the chassis to the 1 position.

2.3 POWER-UP/RESET SELF-TEST

When the VME/10 is powered up, a firmware-resident power-up/reset (PWRT) self-test is performed to verify the functionality of the system resources necessary to boot and initiate the operating system. During the self-test, messages regarding the progress and results of the self-test are displayed. The PWRT takes about 5 seconds to execute, but note that an additional several seconds is required for the Winchester media to spin up to speed before the PWRT is performed. The total length of time required may be up to a minute.

Upon completion of the PWRT after power-up, control of the system is given either to TENbug or to VERSAdos, depending upon the position of the KYBD LOCK switch (key vertical = locked = VERSAdos, key horizontal = unlocked = TENbug)

The PWRT self-test may also be initiated by the operator with the RESET and ABORT pushbuttons (provided the KYBD LOCK key switch is in the horizontal, unlocked position) when used in the following sequence:

a. Press and hold RESET pushbutton.

b. Press and release ABORT pushbutton.

c. Release RESET pushbutton.

The VME/10 then enters TENbug.

2.4 SYSTEM INITIALIZATION

The VME/10 may be initialized in either of two modes:

a. VERSAdos operating system
b. TENbug (operating system media not required)

NOTE

The VERSAdos operating system and supporting software is resident on the Winchester hard disk. After the system has been initialized, backup copies of this software should be made and stored for safekeeping. This procedure is given in paragraph 2.9, Backup Procedure.

Prior to performing these procedures, refer to Chapter 3 for identification and function of the VME/10 switches and keys.

2.4.1 VERSAdos Operating System

The VERSAdos operating system may be entered by either of two methods. Method 1 is used when the system is initially powered up. Method 2 is used after the system has entered TENbug.

Method 1 (enter VERSAdos operating system at system initial power-up)

NOTE

When using the power-up boot process, described as follows, VERSAdos is booted from device 0, controller 0. To boot a different file or boot from a different device/controller, the boot must be initiated from TENbug. Refer to TENbug's BO command described in Chapter 9.

a. On chassis operator panel, set the KYBD LOCK key switch to the locked (vertical) position. (Note that the chassis operator panel pushbutton switches RESET and ABORT are inoperative.)

b. Set the power switch to the on position (1). When power is applied, the power-up/reset self-test (PWRT) is initiated. It may require up to one minute for the disk drive to attain running speed and to perform the self-test. The following messages are displayed on the monitor:

```
Power-up test in progress
Waiting for disk to spin up
```

c. After the self-test concludes, assuming no errors have been indicated, the following is displayed:

```
Power-up test complete
```

d. The VME/10 boots the VERSAdos operating system into memory from the Winchester disk media and VERSAdos identifies itself. Unlock the keyboard (KYBD LOCK switch in horizontal position) and press the "uppercase lock" key ( ) on the keyboard. Make the responses indicated, using current date and time:

```
VERSADOS VERSION: n.nn mm/dd/yy        xxxxxxxxxx
ENTER DEFAULT SYSTEM VOLUME:USER NO.=SYS:0
ENTER DATE (MM/DD/YY)=3/31/83
ENTER TIME (HR:MIN)=7:00
7:00:01 3/31/83 START SESSION 0001 USER 0
```

Two informative chain files, which may be modified by the user, are executed and their messages displayed.

The operating system may have been generated with an automatic logon as volume SYS:, user 0, and session 0001. In this case, current time and date are displayed. They may be changed by the user, with the DATE and TIME commands.

e. To exit the VERSAdos operating system, enter one of the following on the keyboard:

```
=LOG OFF
 14:00:00 3/31/83 END SESSION 0001 USER 0

or

=OFF
 14:00:00 3/31/83 END SESSION 0001 USER 0

or

=BYE
```

f. If the power has not been turned off, VERSAdos can be reentered after logoff by pressing the CLEAR/BREAK key and responding to the system prompts.

Method 2 (enter VERSAdos operating system from TENbug)

a. When TENbug is running on the VME/10, ensure that the VERSAdos operating system media is available and that the KYBD LOCK switch is in horizontal position.

b. Enter BO command from keyboard to load VERSAdos from the fixed hard disk.

```
TENbug x.x > BO
```

Go to step d. of Method 1.

NOTE

When booting operating system from floppy disk drive, enter BO 2 or BO 2,0 [<filename>]. Refer to TENbug BO command described in Chapter 9.

2.4.2 TENbug

TENbug may be entered by either of two methods. Method 1 is used when the system is initially powered up, and method 2 is used to return to TENbug from VERSAdos.

Method 1 (enter TENbug at system initial power-up)

a. Set the KYBD LOCK key switch to the unlocked (horizontal) position.

b. Set the power switch to the on position (1). When power is applied, the PWRT self-test is initiated.

c. If PWRT self-test indicates no errors, and there is no MVME400 (Dual RS-232C Serial Port) present in the VME card cage, the following will be displayed on the monitor:

```
TENbug x.x >
```

d. If PWRT self-test indicates no errors, and there is an MVME400 present, the following is displayed on the monitor:

```
TENbug
```

e. Select the terminal to serve as the default console device, and depress any key on its keyboard. The following will be displayed:

```
TENbug x.x >
```

Method 2 (enter TENbug from VERSAdos operating system)

a. When the VERSAdos operating system is running on the VME/10, TENbug is entered by pressing the RESET pushbutton (provided the KYBD LOCK key switch is in the unlocked (horizontal) position and the vectors in location 0-7 have not been destroyed).

b. Go to step c. of Method 1.

2.5 DISK-RESIDENT MODULE DIAGNOSTICS

After the VME/10 has been powered up, extended tests can be performed on the system by executing the disk-resident module diagnostics (DRMD) package. It is recommended that the first-time user perform these extended tests to verify system performance. For detailed information, refer to the VME/10 Microcomputer System Diagnostics Manual, M68KVSDM.

2.6 BACKUP PROCEDURE

The software supplied on the fixed Winchester hard disk should not be exercised until the following procedure has been performed to create a complete backup copy. The backup version should be labeled and stored for safekeeping. Note that when backing up from the fixed disk to floppy diskettes, several diskettes will be required.

The following steps initialize and configure diskettes and create backup diskettes from the fixed hard disk. User-entered responses are shown underlined and are to be followed by a carriage return.

a. After entering VERSAdos as shown in paragraph 2.7.1, insert a double-sided blank or scratch diskette in the floppy drive.

b. Call the INIT program and make the entries shown below:

```
=INIT #FD02;V
OK TO INITIALIZE #FD02  (Y/N) ? Y
Data Density of media (S-single,D-double) D > C
DO YOU WANT TO FORMAT DISK (Y/N) ?  Y
START FORMAT
ENTER NEW VOLUME NAME VME1
ENTER USER NUMBER 0
ENTER DESCRIPTION (MAX 20 CHARACTERS) VME/10 BACKUP
DO YOU WANT THE BOOT STRAP (Y/N) ? Y
FILE NAME IS: SYS:0000..IPL.SY
ENTER NEW NAME IF NEEDED (CR)
THE CURRENT LOAD ADDRESS IS  $00000E00
ENTER NEW LOAD ADDRESS $E00
DO YOU WANT A DUMP AREA (Y/N) ? N
DO YOU WANT TO VALIDATE SECTORS (Y/N)? Y
   VALIDATING SECTORS ....
   0 BAD SECTORS ENCOUNTERED
=
```

NOTE

Because this first diskette must be bootable, the V option must be entered on the INIT command line. When the bootstrap question is answered with a Y, the user will be allowed to enter a LOAD address. (The LOAD address for the VME/10 is $E00).

c. Call the BACKUP program to copy files from the hard disk to the floppy. The file names are listed as the files are copied:

```
=BACKUP #HD00,#FD02
STARTING FILE-BY-FILE BACKUP PROCESS
COPY ALL FILES, SELECT FILES, OR QUIT (A/S/Q) ? A
DUPLICATE FILE - OK TO COPY (Y/N/Q) ?   VME1:0000..IPL.SY N
VME1:0000..ARESTRRG.HT
  .
  .
  .
VME1:<user no.>.<catalog>.<file name>.<ext>
** OUTPUT DISK FULL **    CONTINUE (Y/N) ? N
=
```

d. Note the full user number, catalog, file name, and extension of the last file copied.

e. Remove the diskette, label it, and set it aside.

f. Use the DISMOUNT utility:

```
= DISMOUNT #FD02
DISMOUNT version xxxxxx x
=
```

g. Insert another double-sided blank or scratch diskette into the floppy drive.

h. Initialize and format the floppy and continue backing up the hard disk as follows. (Note that BACKUP's S option must be used to back up these successive floppies.).

```
=INIT #FD02
OK TO INITIALIZE #FD02  (Y/N) ? Y
Data Density of media (S-single,D-double) D > C
DO YOU WANT TO FORMAT DISK (Y/N) ?  Y
START FORMAT
ENTER NEW VOLUME NAME VME2
ENTER USER NUMBER 0
ENTER DESCRIPTION (MAX 20 CHARACTERS) VME/10 BACKUP
DO YOU WANT THE BOOT STRAP (Y/N) ? N
DO YOU WANT A DUMP AREA (Y/N) ? N
DO YOU WANT TO VALIDATE SECTORS (Y/N)? Y
   VALIDATING SECTORS ....
   0 BAD SECTORS ENCOUNTERED
=BACKUP #HD00,#FD02;S
STARTING FILE-BY-FILE BACKUP PROCESS
COPY ALL FILES, SELECT FILES, OR QUIT (A/S/Q) ? A
ENTER RESTART FILENAME (INCLUDING USER NUMBER)
 <user no.>.<catalog>.<filename>.<ext>        (Enter full file name of
 VME2:<user no.>.<catalog>.<filename>.<ext>    last file copied to
                                               previous diskette, but
 .                                             do not enter the volume
 .                                             name. The files will be
 .                                             listed as they are
                                               copied.)
```

i. Each time the ** OUTPUT DISK FULL ** message appears, enter N and repeat the procedure from step d. above until all files are copied from the hard disk (system will return to the VERSAdos "=" prompt without issuing the ** OUTPUT DISK FULL ** message). Use a different volume name for each diskette.

Note that in the preceding examples, the utility DISMOUNT is not required. However, for most routine operations with diskettes that have been initialized previously (have a volume name), the diskettes must be MOUNTed and DISMOUNTed. For examples of using BACKUP, INIT, MOUNT, and DISMOUNT for routine copying of files on the fixed disk to floppy diskettes, and from floppy diskettes to the fixed disk, refer to Chapter 4 or to the VERSAdos System Facilities Manual, M68KVSF.

CHAPTER 3

CONTROLS AND INDICATORS

3.1 INTRODUCTION

This chapter provides control and indicator descriptions for the VME/10 chassis, display unit, and keyboard.

3.2 CHASSIS

The chassis has an operator panel (Figure 3-1) located at the bottom left corner on the front of the chassis.

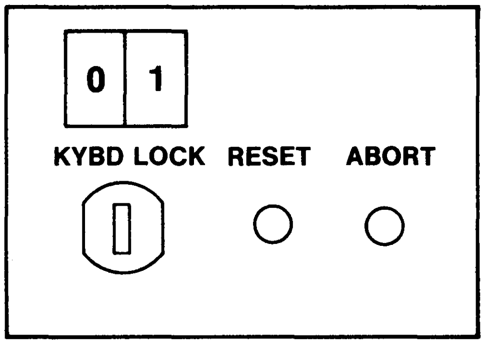


FIGURE 3-1. Operator Panel

The controls perform the following functions:

a. | 0 | 1 | - The power on/off rocker-arm switch is used to turn on power to the VME/10. The '0' represents the off position; the '1' represents the on position.

b. KYBD LOCK - The KYBD LOCK key switch controls a bit in a register which is monitored by TENbug. When the key switch is in the locked position, the VME/10 enters the VERSAdos operating system. When the key switch is in the unlocked position, the VME/10 enters TENbug. Also, when the key switch is in the locked position, the keyboard keys and the front panel pushbutton switches RESET and ABORT are inoperative. This feature provides protection from inadvertent interrupts during system usage.

c. RESET - When this momentary-action pushbutton switch is pressed, it resets the VME/10 logic circuits. If the VME/10 is in the VERSAdos operating system, TENbug is entered by pressing RESET (provided the KYBD LOCK key switch is in the unlocked position).

d. ABORT - When this momentary-action pushbutton switch is pressed, the VME/10 enters TENbug, but the VME/10 logic circuits are not reset. After an abort, the user can enter 'G' to continue execution of the current program prior to the abort.

There are two indicators located at the front of the chassis. When either the Winchester or floppy disk drive is accessed, the respective indicator becomes illuminated.

3.3 DISPLAY UNIT

The display unit has a rear panel which contains a rotary adjustment control (⁼D) used for varying the screen intensity.

3.4 KEYBOARD CONSOLE

The keyboard console is partitioned into four basic functional groups:

a. Typewriter keyboard
b. Cursor control keypad
c. Hex/edit keypad
d. User function keys (F1-F16)

NOTE

The exact action taken when a key is depressed depends upon the port configuration and the program communicating to the keyboard display unit. Refer to the Data Management Services Manual (RMS68KIO) for detailed information.

Refer to Figure 3-2 for keyboard assembly.

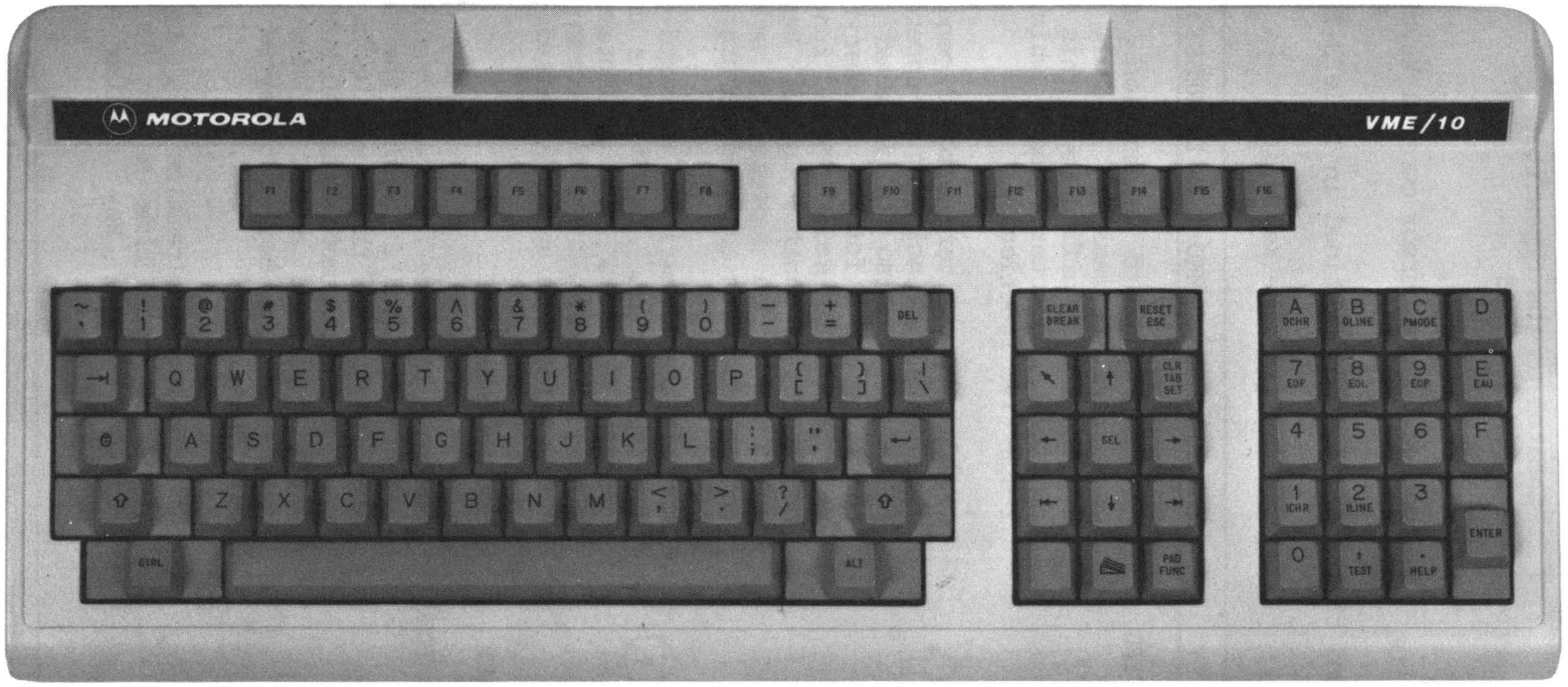


FIGURE 3-2. Keyboard Assembly

3.4.1 Mode Keys

There are six mode keys on the keyboard console. The mode keys and their functional respective locations are defined as follows:

a. CTRL b. SHIFT c. CAPS LOCK d. ALT	Located on the typewriter keyboard keypad
e. PAD/FUNCTION f. CLACKER CONTROL	Located on the cursor control keypad

The six mode keys and their functions are defined in Table 3-1.

TABLE 3-1. Mode Keys

FUNCTION	KEY	FUNCTION PERFORMED WHEN DEPRESSED
Control (1)	CTRL	Enables keyboard generation of ASCII control characters. Depress and hold the control key, then depress the specified key. This action generates a control character code which performs the respective function.
Shift (1)	⇧	Enables typewriter keyboard generation of shifted characters (including uppercase alphabetic characters). Also is involved in generation of character codes for the 16 user function keys, and the CLEAR/BREAK and RESET/ESC function keys located on the cursor control keypad. Depress and hold the shift key, then depress the selected key. This action generates the shifted character code.
Uppercase (1) lock	🔒	Enables the typewriter keyboard generation of all capital letters. Depress the uppercase lock key (remains down), then depress the selected alphabetic key. The uppercase lock key remains down until it is again depressed.
Alternate	ALT	Not implemented.
Pad/function control	PAD/FUNC	Controls the mode for the hex/edit keypad (refer to paragraph 3.4.3). When the key is depressed (remains down), the hex/edit keypad functions in the hexadecimal mode. The PAD/FUNC key remains down until depressed again. When the key is not depressed, the hex/edit keypad functions in the edit mode.
Clacker control	≡	When released (remains up), a 'soft click' is generated each time a key is depressed. The clacker control key remains up until it is depressed again.

NOTE: (1) If none of the above modes is activated, the keyboard is considered to be in the normal mode, as referenced in Table 3-2.

3.4.2 Typewriter Keyboard

The typewriter keyboard contains the following numerics, alphabetic, symbol and special character selection:

a. Numerics (0-9)
b. Alpha characters (A-Z)
c. Symbol characters
d. Special characters (delete, carriage return, and forward tab)

Table 3-2 lists the characters and codes generated when a key is depressed when the typewriter keyboard is in a specific or multi-mode of operation.

3.4.2.1 Numerics (0-9). Numerics are obtained by depressing the respective numeric key when in the normal mode. Refer to Table 3-2.

3.4.2.2 Alphabetic Characters (a-z). Lowercase alphabetic characters are obtained by depressing the respective alphabetic key when in the normal mode. To obtain uppercase characters, depress the CAPS LOCK key (remains down) or depress and hold the shift key, then depress the respective alpha key. Refer to Table 3-2.

3.4.2.3 Symbol Characters. Symbols are obtained by depressing the respective symbol key when in the normal or shift mode. Refer to Table 3-2.

3.4.2.4 Special Characters. The special character keys are defined as follows:

a. The DEL (delete) key erases the last character typed on the line.

b. The <--' (carriage return) key moves the cursor to the beginning of the next line and signals the end of a line typed to the computer.

c. The -->| (forward tab) key moves the cursor to the next tab position in certain programs (e.g., the editor).

TABLE 3-2. Standard Typewriter Keyboard Character Code

		MODE			
KEY	DESCRIPTION	NORMAL	CAPS LOCK	SHIFT	CTRL
A	Alphabetic A	a	A	A	$01
B	Alphabetic B	b	B	B	$02
C	Alphabetic C	c	C	C	$03
D	Alphabetic D	d	D	D	$04
E	Alphabetic E	e	E	E	$05
F	Alphabetic F	f	F	F	$06
G	Alphabetic G	g	G	G	$07
H	Alphabetic H	h	H	H	$08
I	Alphabetic I	i	I	I	$09
J	Alphabetic J	j	J	J	$0A
K	Alphabetic K	k	K	K	$0B
L	Alphabetic L	l	L	L	$0C
M	Alphabetic M	m	M	M	$0D
N	Alphabetic N	n	N	N	$0E
O	Alphabetic O	o	O	O	$0F
P	Alphabetic P	p	P	P	$10
Q	Alphabetic Q	q	Q	Q	$11
R	Alphabetic R	r	R	R	$12
S	Alphabetic S	s	S	S	$13
T	Alphabetic T	t	T	T	$14
U	Alphabetic U	u	U	U	$15
V	Alphabetic V	v	V	V	$16
W	Alphabetic W	w	W	W	$17
X	Alphabetic X	x	X	X	$18
Y	Alphabetic Y	y	Y	Y	$19
Z	Alphabetic Z	z	Z	Z	$1A

TABLE 3-2. Standard Typewriter Keyboard Character Code (cont'd)

		MODE			
KEY	DESCRIPTION	NORMAL	CAPS LOCK	SHIFT	CTRL
~	Tilde			~	
'	Grave accent	'	'		
!	Exclamation point			!	
1	Digit 1	1	1		
@	Commercial at			@	$00
2	Digit 2	2	2		
#	Number sign			#	
3	Digit 3	3	3		
$	Dollar sign			$	
4	Digit 4	4	4		
%	Percent sign			%	
5	Digit 5	5	5		
^	Circumflex			^	$1E
6	Digit 6	6	6		
&	Ampersand			&	
7	Digit 7	7	7		
*	Asterisk			*	
8	Digit 8	8	8		
(	Opening parenthesis			(	
9	Digit 9	9	9		
)	Closing parenthesis			)	
0	Digit 0	0	0		
_	Underline			_	$1F
-	Hyphen (minus)	-	-		
+	Plus sign			+	
=	Equals	=	=		
DEL	Delete	$7F	$7F	$7F	
-->|	Forward tab	$09	$09	$09	
{	Opening brace			{	$1B
[	Opening bracket	[	[		

TABLE 3-2. Standard Typewriter Keyboard Character Code (cont'd)

		MODE			
KEY	DESCRIPTION	NORMAL	CAPS LOCK	SHIFT	CTRL
}	Closing brace			}	$1D
[	Closing bracket	]	]		
'	Apostrophe			'	$1C
\	Backward slant	\	\		
:	Colon			:	
;	Semicolon	;	;		
"	Double quotation			"	
'	Single quotation	'	'		
<--|	Carriage return	$0D	$0D	$0D	
<	Less than			<	
,	Comma	,	,		
>	Greater than			>	
.	Period	,	.		
?	Question mark			?	
/	Forward slant	/	/		

NOTE: A blank entry indicates no character generated.

There are four other multi-modes available. Following are the multi-modes and their respective character sets generated:

MODE	CHARACTER SET
CAPS LOCK AND SHIFT	Same as SHIFT
CAPS LOCK AND CTRL	Same as CTRL
CAPS LOCK, SHIFT, CTRL	Same as CTRL
SHIFT AND CONTROL	Same as CTRL

3.4.3 Cursor Control Keypad

The cursor control keypad provides the following:

a. Cursor control
b. Functions: CLEAR/BREAK, RESET
c. Special character: ESC

3.4.3.1 Cursor Control. The cursor control keys are used in special programs (e.g., the editor). Table 3-3 indicates the cursor control keys and their respective character codes and functions.

TABLE 3-3. Cursor Control Keys

KEY	FUNCTION	CHARACTER CODE	FUNCTION PERFORMED WHEN DEPRESSED
<--	Cursor left	$08	Moves cursor left one column.
-->	Cursor right	$0C	Moves cursor right one column.
↑	Cursor up	$0B	Moves cursor up one line in same column.
↓	Cursor down	$0A	Moves cursor down one line in same column.
|<--	Backward tab	$DB	Moves cursor left to previous tab position.
-->|	Forward tab	$09	Moves cursor right to next tab position.
SEL			Not implemented.
CLR TAB SET			Not implemented.
↖	Home	$C0	Moves cursor to left-most column in top line.

3.4.3.2 Functions (CLEAR/BREAK, RESET). The function keys are described as follows:

a. The shifted value of the CLEAR/BREAK key (CLEAR) causes all positions in the display to be filled with spaces. The cursor moves to the home position.

b. The non-shifted value of the CLEAR/BREAK key (BREAK) generates a 'special condition' signal which is recognized by the VERSAdos operating system and allows the user to log on.

c. The shifted value of the RESET/ESC key (RESET) initializes the screen to the power-on condition.

3.4.3.3 Special Character (ESC). The non-shifted value of the RESET/ESC key (ESC) generates an ASCII escape character ($1B).

3.4.4 Hex/Edit Keypad

The hex/edit auxiliary keypad performs two modes -- hexadecimal keyboard entry and editing functions -- which are controlled by the PAD/FUNC key (refer to paragraph 3.4.1). The ENTER key (carriage return) generates a character code $0D which moves the cursor to the beginning of the next line (left margin). The ENTER key is not affected by the PAD/FUNC key.

3.4.4.1 Hexadecimal Mode. When the PAD/FUNC key is depressed, the keypad can be used as a hexadecimal keypad utilizing characters 0-F and also a comma (,) and a period (.).

3.4.4.2 Edit Mode. When the PAD/FUNC key is in the normal position (not depressed), the keypad can be used for editing purposes when in special programs (e.g., the editor). Table 3-4 defines the editing notations and the character codes generated.

TABLE 3-4. Edit Mode Keys

FUNCTION	KEY	CHARACTER CODE	FUNCTION PERFORMED
Delete character	DCHR	$D1	The delete character key deletes the character on which the cursor is positioned. The characters to the right of the cursor on that line are moved left one column. The right-most column becomes a blank.
Delete line	DLINE	$D7	The delete line key deletes the line on which the cursor is positioned. The lines below the cursor are moved up. The last line becomes blank.
	PMODE		Not implemented.
	EOF		Not implemented.
Erase to end of line	EOL	$D5	The end of line key erases all positions from the cursor position to the end of the line.
Erase to end of page	EOP	$D4	The end of page key erases all positions from the cursor position to the end of the display.
	EAU		Not implemented.
Insert character	ICHR	$D0	The insert character key moves the character under the cursor and all characters to the right on the same line right one column. The character position under the cursor becomes blank.
Insert line	ILINE	$D6	The insert line key moves all lines, starting with the line on which the cursor is positioned, down one line. The line the cursor is on becomes blank.
	TEST		Not implemented.
	HELP		Not implemented.

3.4.5 User Function Keys (F1-F16)

There are 16 user function keys on the keyboard. A program may be written to monitor the character values generated by these function keys and perform corresponding functions. Refer to Table 3-5 for character values generated by the user function keys during the normal and shift modes.

TABLE 3-5. User Function Key Character Code

FUNCTION KEY	NORMAL	SHIFT
1	$A0	$B0
2	$A1	$B1
3	$A2	$B2
4	$A3	$B3
5	$A4	$B4
6	$A5	$B5
7	$A6	$B6
8	$A7	$B7
9	$A8	$B8
10	$A9	$B9
11	$AA	$BA
12	$AB	$BB
13	$AC	$BC
14	$AD	$BD
15	$AE	$BE
16	$AF	$BF

3.4.6 ASCII Character Set

Table 3-6 lists the ASCII character set and the methods by which each ASCII character can be generated from the keyboard.

TABLE 3-6. ASCII Character Set

CHARACTER	COMMENTS	KEYBOARD IMPLEMENTATION OF CHARACTER (1)	HEX VALUE
NUL	Null or tape feed	2^C	00
SOH	Start of Heading	A^C	01
STX	Start of Text	B^C	02
ETX	End of Text	C^C	03
EOT	End of Transmission	D^C	04
ENQ	Enquire (who are you, WRU)	E^C	05
ACK	Acknowledge	F^C	06
BEL	Bell	G^C	07
BS	Backspace	<-- or H^C	08
HT	Horizontal Tab	-->| or I^C	09
LF	Line Feed	↓ or J^C	0A
VT	Vertical Tab	↑ or K^C	0B
FF	Form Feed	--> or L^C	0C
RETURN	Carriage Return	<--', ENTER, or M^C	0D
SO	Shift Out	N^C	0E
SI	Shift In	O^C	0F
DLE	Data Link Escape	P^C	10
DC1	Device Control 1	Q^C	11
DC2	Device Control 2	R^C	12
DC3	Device Control 3	S^C	13
DC4	Device Control 4	T^C	14
NAK	Negative Acknowledge	U^C	15
SYN	Synchronous Idle	V^C	16
ETB	End of Transmission Block	W^C	17
CAN	Cancel	X^C	18
EM	End of Medium	Y^C	19
SUB	Substitute	Z^C	1A
ESC	Escape, prefix	ESC or $[^C$	1B
FS	File Separator	$\backslash^C$	1C
GS	Group Separator	$]^C$	1D
RS	Record Separator	6^C	1E
US	Unit Separator	$-^C$ (hyphen)	1F
SP	Space or Blank	Space bar	20

TABLE 3-6. ASCII Character Set (cont'd)

CHARACTER	COMMENTS	KEYBOARD IMPLEMENTATION OF CHARACTER (1)	HEX VALUE
!	Exclamation point	1^{s}	21
"	Quotation marks (dieresis)	'^{s}	22
#	Number sign	3^{s}	23
$	Dollar sign	4^{s}	24
%	Percent sign	5^{s}	25
&	Ampersand	7^{s}	26
'	Apostrophe (acute accent, closing single quote)	'	27
(	Opening parenthesis	9^{s}	28
)	Closing parenthesis	0^{s}	29
*	Asterisk	8^{s}	2A
+	Plus sign	$=^{s}$	2B
, (2)	Comma (cedilla)	,	2C
-	Hyphen (minus)	-	2D
. (2)	Period (decimal point)	.	2E
/	Slant	/	2F
0 (2)	Digit 0	0	30
1 (2)	Digit 1	1	31
2 (2)	Digit 2	2	32
3 (2)	Digit 3	3	33
4 (2)	Digit 4	4	34
5 (2)	Digit 5	5	35
6 (2)	Digit 6	6	36
7 (2)	Digit 7	7	37
8 (2)	Digit 8	8	38
9 (2)	Digit 9	9	39
:	Colon	;^{s}	3A
;	Semicolon	;	3B
<	Less than	,^{s}	3C
=	Equals	=	3D
>	Greater than	.^{s}	3E
?	Question mark	/^{s}	3F
@	Commercial at	2^{s}	40

TABLE 3-6. ASCII Character Set (cont'd)

CHARACTER	COMMENTS	KEYBOARD IMPLEMENTATION OF CHARACTER (1)	HEX VALUE
A (2)	Uppercase letter A	A^S	41
B (2)	Uppercase letter B	B^S	42
C (2)	Uppercase letter C	C^S	43
D (2)	Uppercase letter D	D^S	44
E (2)	Uppercase letter E	E^S	45
F (2)	Uppercase letter F	F^S	46
G	Uppercase letter G	G^S	47
H	Uppercase letter H	H^S	48
I	Uppercase letter I	I^S	49
J	Uppercase letter J	J^S	4A
K	Uppercase letter K	K^S	4B
L	Uppercase letter L	L^S	4C
M	Uppercase letter M	M^S	4D
N	Uppercase letter N	N^S	4E
O	Uppercase letter O	O^S	4F
P	Uppercase letter P	P^S	50
Q	Uppercase letter Q	Q^S	51
R	Uppercase letter R	R^S	52
S	Uppercase letter S	S^S	53
T	Uppercase letter T	T^S	54
U	Uppercase letter U	U^S	55
V	Uppercase letter V	V^S	56
W	Uppercase letter W	W^S	57
X	Uppercase letter X	X^S	58
Y	Uppercase letter Y	Y^S	59
Z	Uppercase letter Z	Z^S	5A
[	Opening bracket	[	5B
\	Reverse slant	\	5C
]	Closing bracket	]	5D
^	Circumflex	6^S	5E
_	Underline	$-^S$ (hyphen)	5F

TABLE 3-6. ASCII Character Set (cont'd)

CHARACTER	COMMENTS	KEYBOARD IMPLEMENTATION OF CHARACTER (1)	HEX VALUE
\`	Quotation mark	'	60
a	Lowercase letter a	A	61
b	Lowercase letter b	B	62
c	Lowercase letter c	C	63
d	Lowercase letter d	D	64
e	Lowercase letter e	E	65
f	Lowercase letter f	F	66
g	Lowercase letter g	G	67
h	Lowercase letter h	H	68
i	Lowercase letter i	I	69
j	Lowercase letter j	J	6A
k	Lowercase letter k	K	6B
l	Lowercase letter l	L	6C
m	Lowercase letter m	M	6D
n	Lowercase letter n	N	6E
o	Lowercase letter o	O	6F
p	Lowercase letter p	P	70
q	Lowercase letter q	Q	71
r	Lowercase letter r	R	72
s	Lowercase letter s	S	73
t	Lowercase letter t	T	74
u	Lowercase letter u	U	75
v	Lowercase letter v	V	76
w	Lowercase letter w	W	77
x	Lowercase letter x	X	78
y	Lowercase letter y	Y	79
z	Lowercase letter z	Z	7A
{	Opening brace	[^{s}	7B
|	Vertical line	\\$^s$	7C
}	Closing brace	]^s	7D
~	Tilde	\`^{s}	7E
DEL	Delete	DEL	7F

NOTES:
(1) For implementation on keyboard, c = Control; s = Shift
(2) This key is located on the main typewriter keyboard and also on the hex/edit keypad. On the hex/edit keypad, this key is activated when the PAD/FUNC key is depressed (remains down).

CHAPTER 4

SOFTWARE DESCRIPTION

4.1 INTRODUCTION

The VME/10 Microcomputer System package includes the VERSAdos operating system and associated development system software furnished on the fixed Winchester media. VERSAdos consists of a powerful set of file-handling utilities, security capability, real-time multitasking kernel, a system generation facility, an M68000-family assembler and linkage editor, a CRT-oriented text editor, diagnostics, and both symbolic and non-symbolic debuggers.

Optionally available are Pascal and FORTRAN compilers and various cross assemblers, cross linkers, and a cross Pascal compiler; the latter make it possible to assemble or compile and link programs for 8-bit "target" systems using the VME/10 as the "host" design station.

The VME/10 firmware contains a resident monitor/debugger, TENbug, useful not only as the "bootstrap" of the operating system or other program, but as a simple, easy-to-use debug tool.

This chapter and those that follow provide not only general descriptions of the features and functions of various components of the VME/10 software, but step-by-step examples which can be performed by the new user for familiarization with the system.

NOTE

Before using the software furnished on the Winchester disk, a backup copy should be made and stored for safekeeping, as directed in Chapter 2.

System firmware error messages are listed in the TENbug Debugging Package User's Manual, M68KTENBG. Operating system error messages are described in the VERSAdos Error Messages Manual, M68KVMSG.

4.2 VERSAdos

4.2.1 Functional Overview

VERSAdos is a modular, multilayered operating system that provides a convenient and friendly interface between the user and system hardware. It provides a solution to general-purpose program generation requirements associated with the development of microprocessor-based systems, as well as the execution requirements of dedicated, real-time, multitasking application systems. The modular nature of the operating system permits configuration of the VME/10 for a variety of host/target applications. This flexibility reduces the costs and problems normally encountered during system integration by permitting extensive debugging to be performed on a compatible hardware/software configuration prior to the integration process.

VERSAdos operations are task oriented. A task is a program, complete with its associated data area, that performs a functional unit of work. Application programs are performed as tasks and are executed according to their priorities, scheduling requirements, and availability of required resources.

VERSAdos is responsible for accepting, checking, interpreting, and expediting user application requests. During execution of a task, the operating system may request assistance from various operating system support routines not directly accessible to the application program. These support routines assist in operator control, memory management, task segmentation, and input/output control for various hardware subsystems. This permits execution of more than one task at a time, thereby allowing several application programs to be operating independently on the system. This also relieves the application program from the necessary chore of direct interaction with the system hardware. Instead, application programs communicate their input/output requests to the system via the operating system using an established protocol.

The operating system is divided into four major layers, with each layer further subdivided into other layers. The four major layers are: the Real-Time Multitasking Executive (RMS68K) layer, the I/O layer, the File Management layer, and the Session Management layer. This structure is shown in Figure 4-1.

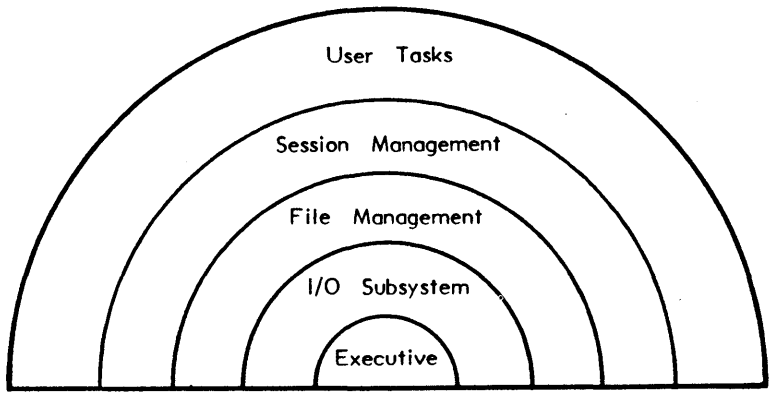


FIGURE 4-1. VERSAdos Structure

RMS68K, the Executive, is the nucleus of the VERSAdos operating system. It has responsibility for servicing all hardware and software generated interrupts and dispatching the interrupts to the proper tasks for processing. RMS68K also acts as the arbiter to resolve conflicts that result when competing tasks vie for processor time. Facilities that permit inter-task communications and task synchronization are also supported. RMS68K protects user applications while providing diagnostic feedback during error conditions. Refer to the M68000 Family Real-Time Multitasking Software User's Manual, M68KRMS68K.

The layered design of VERSAdos provides a degree of system flexibility, while maintaining a straight-forward structure that is easy to understand and use. The layered structure also provides the unique ability of combining the normally diverse functions of time-sharing software development with real-time system control or else tailoring an operating system to the user's requirements. The high degree of modularity inherent with the major programs of VERSAdos permits each user to add specific functions for his individual requirements with a minimum of time and effort. With the incorporation of Intelligent Peripheral Controllers to optimize I/O functions, CPU overhead is significantly reduced, thereby providing more computing power for each system user and permitting connection of higher data rate I/O devices.

Real-time I/O processing capabilities are provided that allow directly connected or processing-generated interrupts to be serviced. In addition, multi-programming of real-time tasks can be accommodated. All tasks can be scheduled on a priority basis. Inter-task communications are included to pass parameters and/or control between tasks and/or the operating system. A special control, the semaphore, is used to provide task synchronization and to coordinate the use of shared resources. Operating system I/O operations are device independent; that is, they refer to the logical properties of operation rather than to the physical characteristics or file formats. I/O operations are performed by the Input/Output Services (IOS) portion of VERSAdos. File management services provide transfer control between memory and logical devices or files. Contiguous, sequential, and indexed sequential files are supported. File handling operations are performed by the File Handling Services (FHS) portion of VERSAdos. For further information on IOS and FHS, refer to the VERSAdos Data Management Services and Program Loader User's Manual, RMS68KIO.

4.2.2 Operational Overview

From the user's standpoint, VERSAdos is made up of a collection of utility programs used to manipulate files, directories, and peripherals; a versatile CRT-oriented text editor for the creation and editing of ASCII files; a set of "session control" commands for direct communication with the system; batch and chaining capabilities which allow operation in an interactive, on-line, foreground mode as well as in background mode; a job spooling function; a multi-level security package; an M68000 Family structured macro assembler and module linker; two software debugging programs, DEbug (non-symbolic) and SYMbug (symbolic); and a "system generation" program, SYSGEN, which simplifies user-modification of the operating system to suit a particular hardware/software configuration.

4.2.3 VERSAdos File Name Format

All files used under VERSAdos, including those program files (or load modules) which comprise the operating system, are fully described as follows:

```
<volume>:<user number>.<catalog>.<file name>.<extension>(<protection>)
```

where:

volume	is the name of the media on which the file resides (e.g., the fixed disk might be named SYS, FIX, VOL0, etc.; floppies might be named VOL1, VOL2, FL1, FL2, etc.). Up to four alphanumeric characters may be used (first character must be alpha).
user number	is from one to four decimal digits, and establishes "ownership" of catalogs and files. User 0 is referred to as the system administrator; user 0 validates numbers assigned to other users of the system.
catalog	is a name of up to eight alphanumeric characters, or it may be null (blank). A volume may contain many catalogs, and many files may be grouped under the same catalog name. For instance, files pertaining to a particular project may share the same catalog name.
file name	is up to eight alphanumeric characters. It identifies a particular file.
extension	is an extension of two alpha characters which further identifies a file by type. Certain extensions have meaning to VERSAdos; for instance, SA identifies ASCII source files, SY identifies operating system files, LO identifies executable load modules, XX and NW identify news files.
protection	is an optional user-selectable access permission code consisting of two, three, or four of the characters A-P. If specified when the file is created, it must be matched whenever the file is accessed. If not specified, it defaults to PPPP (any user may read or write to the file).

Default values for <volume>, <user number>, and <catalog> are those established at logon, although they can be changed by the user. Default file name extension varies according to the utilities that use the file. For example, ASM, E, and LIST assume an extension of SA; LOAD, PATCH, and SYMbug assume an extension of LO; and LINK expects an extension of RO or RX. Default values need not be typed. For example, assuming that the logon values are a volume identification of VOL0, user number 0, and a catalog name of TEST, the following ASCII source file names are generally equivalent:

```
VOL0:0.TEST.FILE01.SA
TEST.FILE01
FILE01
```

4.2.4 Session Management

When the VME/10 System is booted up, the Session Control task is initiated. This task enables several useful functions which are executed with a set of commands (see Table 4-1).

Functions performed as part of the Session Control task include batch and chain file processing, user identification, system security, system date and time setting, and dissemination of system news. Certain of the Session Control capabilities can be deleted from the operating system and a new system created with SYSGEN, if the functions are not needed.

4.2.4.1 Sessions. At power-up, the operator will typically log on as user number 0. User 0, also referred to as the system administrator, is allowed several "privileged" activities, including the establishing of a system password and the assigning of user numbers to other authorized personnel. The first session after booting is session number 0001. Each logon after a logoff initiates another sequentially numbered session, until the system is rebooted.

The values established at boot-up for volume identification, user number, and catalog remain in effect until changed with the Session Control USE command, or by logging off the system and logging back on with different values. The DEF command is used to display current defaults.

TABLE 4-1. Session Control Commands

COMMAND	DESCRIPTION	
OFF	Terminate session.	) IDENTICAL SESSION TERMINATION
LOG OF[F]	Terminate session.	) IDENTICAL SESSION TERMINATION
LOGOF[F]	Terminate session.	) IDENTICAL SESSION TERMINATION
BYE	Terminate session.	
BATC[H]	Submit batch job.	
CANC[EL]	Cancel selected or all batch jobs.	
ELIM[INATE]	Cancel all batch jobs (privileged).	
QUER[Y]	Request status of batch job.	
CHAI[N]	Execute chain file.	
RETR[Y]	Restart execution of aborted chain file at current record.	
PROC[EED]	Restart execution of aborted chain file at next record.	
OPT[ION]	Set chain conditional processing option.	

TABLE 4-1. Session Control Commands (cont'd)

COMMAND	DESCRIPTION
R?	Display contents of chain conditional processing pseudo registers.
END	Terminate chain processing.
LOAD	Call the loader.
STAR[T]	Begin execution of user task.
CONT[INUE]	Restart execution of user task.
STOP	Stop execution of user task.
TERM[INATE]	Terminate execution of user task.
BSTO[P]	Stop all tasks on Break.
BTER[M]	Terminate all tasks on Break.
USE	Enter file descriptor defaults.
DEF[AULTS]	Display default values.
ARG[UMENTS]	Enter/display new arguments.
NOARG[UMENTS]	Clear argument list.
DATE	Display time and date or (privileged) change date.
TIME	Display time and date or (privileged) change time.
^	Place the session control task in the dormant state.
PASS[WORD]	Specify or change user password.
SWORD	Specify or change system security word (privileged).
SECURE	Specify level of system security (privileged).

During session 0001, the system clock is set with the correct date and time, if necessary, with the DATE and TIME commands.

Three furnished information files can be edited by user 0 to convey appropriate messages to users. The files and their functions are as follows:

SYS:0.PRIV.BULLETIN.NW	This file's contents are displayed at successful logon, and provide pertinent information to system users.
SYS:0.PRIV.REJECT.NW	This file announces an unsuccessful logon attempt, when system security is in effect.
SYS:0.PRIV.NEWS.NW	This file is designed to contain information of interest to system users, and is accessed by any user by entering the session control command, NEWS. This causes the file's contents to be scrolled on the screen.

Two chain files can also be created/edited by the administrator so that a previously selected set of processes can be performed automatically at logon. They are:

SYS:0.PRIV.UPSYSTEM.CF	This chain file may optionally be activated at the completion of session 0001. A typical use of this file is to establish security levels for the system (paragraph 4.2.4.2).
SYS:0.PRIV.USER.CF	This chain file may be created to perform any desired functions, and to be activated any time a user logs on as user 0. Additionally, any user may create his own PRIV.USER.CF under his own user number to contain desired commands to be activated whenever he logs on.

4.2.4.2 Security. VERSAdos' security package can be SYSGEN'd into the operating system or excluded from it. When part of the system, there are four levels of security available:

LEVEL	SECURITY
0	None. Any user number may be entered at logon at any session.
1	System security. A system password has been installed by the system administrator. No one may log on to the system without entering the security word.
2	User number/password security. A list of valid user numbers, with or without individual passwords, has been entered into a password file maintained by the administrator. Only valid user numbers may be entered at logon, and if a password has been established for a particular user number, it must also be entered.
3	Both system and user number/password security. At logon, a valid user number, a system security password, and (if established) an individual password must be entered.

As supplied, the boot load file VERSADOS.SY establishes level 0 security. During session 0001, the system administrator can select any other level for himself or for all other users with the session control commands SECURE and SWORD. User 0 can also change the contents of the chain file executed when VERSAdos is booted, PRIV.UPSYSTEM.CF, to automatically establish security levels for all subsequent sessions.

Levels 2 and 3 are dependent upon a password file having been created by the system administrator, using the ACCT utility program. ACCT and the VALID, NOVALID, and PASS utilities are used to maintain these security levels. Refer to the VERSAdos System Facilities Manual, M68KVSF, for the use of these utilities.

4.2.4.3 Examples. The following examples assume that the VME/10 has just been powered up, VERSAdos is running, and that the default logon values established are the volume identification of SYS:, a user number of 0, and a null catalog, and it is session 0001.

```
=DEF
 SYSTEM VOLUME = SYS:
 USE DEFAULT VOLUME = SYS:0..
 USER NUMBER = 0
 USER TASK =
 SESSION = 0001
 TERMINAL = CNSL
 OPTION(S) SET =
=DATE
 14:21:52 9/8/83
 ENTER DATE (MM/DD/YY) = 09/15/83
=TIME
 14:22:05 9/15/83
 ENTER TIME (HR:MIN) = 07:25
=
```

If date and time do not need to be changed, the CLEAR/BREAK key can be used to return to the VERSAdos prompt. If the user is logged on as other than user 0, no invitation to enter the date or time is issued.

```
=USE 0.PRIV
 SYSTEM VOLUME = SYS:
 USE DEFAULT VOLUME = SYS:0.PRIV.
 USER NUMBER = 0
 USER TASK =
 SESSION = 0002
 TERMINAL + CNSL
 OPTION(S) SET =
=E BULLETIN.NW                          (Call the editor utility; the present
 VERSADOS EDITOR RELEASE x.xx            contents of the file are displayed
 COPYRIGHT BY MOTOROLA xxxx              on the screen, and the editor will
       .                                 accept direct commands.)
       .
       .
> DEL 0-9999                            (Delete all lines of text in the
                                         file.)
```

(Press the Fl key to enter the CRT page edit mode. The cursor appears at the top of the screen.)

```
> (CR)
> (CR)
> (CR)
> (CR)
> WELCOME TO THE VME/10 MICROCOMPUTER SYSTEM.
> (CR)
>
```

(Press the Fl key to return to command mode. The cursor appears at the bottom of the screen.)

```
> QUIT                                  (Exit the editor program.)
 EDIT DONE
=
```

The following example edits the 0.PRIV.UPSYSTEM.CF chain file to install a system security word. Note, however, that the security level established in this chain file will not be in effect for "session 0001" -- in other words, whenever VERSAdos is rebooted. Security level will be in effect only in subsequent sessions after 0001 -- when VERSAdos has not been rebooted. For security level 1, 2, or 3 to be in effect at system boot time, the IPL.SY file must be modified (refer to the VERSAdos System Facilities Reference Manual).

```
=E UPSYSTEM.CF
(Editor identifies itself and the present contents of the chain file are
displayed.)
> DEL 0-9999
(Press the Fl key to enter CRT page edit mode.)
> =SECURE
> 1                                      (Establish security level 1.)
> 1
> =SWORD
> SECRET                                 (Enter system security word "SECRET"
                                          or other password of your choice.)
> =END
(Press Fl function key to return to command mode.)
```

```
> QUIT
EDIT DONE
=E REJECT.NW
(Editor identifies itself, and the present contents of the chain file are
displayed. Press the F1 key to enter CRT page edit mode.)
> DEL 0-9999
(Press the uppercase lock key to return to uppercase and lowercase mode.)
> So sorry, you are not an authorized user.
> Please see your system administrator for the logon procedure.
>
(Press the F1 key to return to command mode.)
> QUIT
EDIT DONE
=OFF


07:45:20 9/15/83 END SESSION 0001 USER 0
(Press the CLEAR/BREAK key.)

VERSAdos VERSION x.xx          xxxxxxxxxx
ENTER USER NO. = 21
ENTER SECURITY WORD XX                   (Entries are not echoed to the screen.)
ENTER SECURITY WORD YY
ENTER SECURITY WORD ZZ

So sorry, but you are not an authorized user.
Please see your system administrator for the logon procedure.

LOGON REJECTED, LOGGED OFF

(Press CLEAR/BREAK.)

VERSAdos VERSION x.xx          xxxxxxxxxx
ENTER USER NO. = 10
ENTER SECURITY WORD SECRET               (Or other security password selected
                                          when editing the UPSYSTEM.CF file;
                                          security word is not echoed to the
                                          screen.)
07:58:15 9/15/83 START SESSION 0002 USER 10

WELCOME TO THE VME/10 MICROCOMPUTER SYSTEM.

=NEWS
*
*       FILE:     PRIV.NEWS.NW
*

This file is similar to the PRIV.BULLETIN.NW file in that it can be used to
provide important information to the user; however, unlike PRIV.BULLETIN.NW, the
user must request the PRIV.NEWS.NW file if one desires to read its contents.
=LOGOFF
07:59:00 9/15/83 END SESSION 0002 USER 10
```

4.2.5 Utilities

The VERSAdos utility program set is used for physical manipulation of files, storage media, and intersystem data transfer.

VERSAdos utilities are invoked by simply entering the name of the load image file (a file with an extension of .LO) which performs the function. The Session Control Task calls the loader to create a task and load the file, and then starts the function. Because user-created files of type LO can also be loaded and started by simply specifying their names, any which perform a useful function may also be considered utility commands. These are created through use of the LINK command (linkage editor) on a program which has been typed into a file using the Editor, and then assembled or compiled.

It is sometimes useful to load a utility but, rather than have execution begin automatically, start execution manually at the desired time. For example, this two-step operation will allow a diskette to be changed between loading and execution, which can be helpful in a two-drive system. The session control command LOAD, used in conjunction with the session control command START, allows this to be done. If the utility to be loaded requires arguments, these must be specified on the LOAD command line.

Most VERSAdos utilities allow variation in their functions by the specification of options. Many also are "interactive"; i.e., they allow a subset of commands which enable the system operator to supply parameters or otherwise control operation of the utility "on-line".

The general format used to load and run a utility program is:

```
    =<utility name> <input field>,<output field>;<options>
```

Paragraph 4.2.5.1 lists the VERSAdos utilities alphabetically and gives brief descriptions of their functions. Paragraph 4.2.5.2 provides a set of examples of several of the more commonly used utilities. These examples can be performed as practice exercises by new VERSAdos users.

Full descriptions of most of the utilities can be found in the VERSAdos System Facilities Reference Manual, M68KVSF. Those not covered in the facilities manual -- the Editor, Assembler, Linker, DEbug, SYMbug, and SYSGEN, as well as the optional Pascal and FORTRAN compilers and the TENbug monitor -- are described in Chapters 5 through 9 of this manual, and more extensively in individual manuals.

Additionally, the VERSAdos Reference Card, MVDOSCARD, is available for both new and experienced VERSAdos users.

4.2.5.1 Descriptions. VERSAdos utilities are listed alphabetically, with descriptions of their usage, in the following paragraphs.

ACCT

The Account utility may be used by the system administrator to open a password file and an accounting file, and then to monitor individual and collective usage of the system. This utility will probably not be needed when the VME/10 is used as a single-user system.

BACKUP

BACKUP provides two methods of transferring data from one disk to another: track-to-track mode and file transfer mode. Since track-to-track mode requires that the source and destination disks be of the same type, the file transfer mode will be used for the standard configuration VME/10. File transfer mode must be specified with BACKUP options A or R. Sub-options allow several variations in copying. Individual files or families of files can be selected for transfer. File descriptor fields information can be specified on the destination disk. Indexed sequential files can be packed to reclaim internal file space. Files can be packed together to reclaim disk space. A starting point at which file transfer should begin can be specified on the source disk. Files can be selected by date range and/or file/family, or can be selected one at a time. When source data exceeds capacity of the destination disk, the file transfer mode permits insertion of additional destination disk(s). All destination disks must have been initialized previously with the INIT utility.

BUILDS

The BUILDS utility transforms a binary load module into a file of ASCII-encoded information which may then be transported to another system for further use. The format of the records in the file is Motorola S-record, so called because each record begins with a byte containing the code for an ASCII "S" -- for start of record.

CONNECT

The CONNECT utility allows the user at a terminal on a VERSAdos system to communicate with a second computer which is connected to a second port. It produces the same effect as physically disconnecting the terminal from the VERSAdos system and connecting it to the second computer, without having to move any cables. When the L=n option is specified, CONNECT performs the following functions on the terminal from which it was invoked before connecting the terminal to the other port:

a. Resets the display screen.

b. Sets up the virtual screen (the area which scrolls while in CONNECT mode) as lines 1 through 1-n.

c. Displays the message indicating successful connection.

COPY

The COPY utility copies a file onto the same volume under a new file name, or onto another volume under the same or a new name. Options allow a file to be appended to the end of an existing file, packing of data in an indexed sequential file, character-by-character comparison of existing files with display of byte differences within records, and character-by-character comparison of a copied file and the original with display of byte differences within records. Output can be sent to a printer if part of the system, or to the display terminal for a quick look at the contents of a file.

DEL

The Delete utility removes a file name from a disk directory and frees all space allocated to that file. Options allow a list of files or a "family" of files with like parameters (e.g., same catalog or same extension) to be deleted with one command, and/or to direct a list of files deleted (normally displayed on the CRT) to an output file or to a printer.

DIR

Each VERSAdos disk contains a Volume Identification Directory (VID), established when the disk was initialized. Information describing the disk space allocation, location, and attributes of each file contained on the disk is stored in this directory. Part or all of the information entered for each file can be obtained by using the DIR utility. Options provide greater detail.

DISMOUNT

This utility, used in conjunction with the MOUNT utility, enables the VME/10 to handle disks of unlike formats. DISMOUNT performs the complementary function of the MOUNT utility. It forces VERSAdos to release control of a mounted floppy disk and to reject input/output requests to a new disk until the MOUNT command has been reissued. Before using DISMOUNT, the floppy must be off-line -- i.e., the floppy drive door must have been opened.

DISPATCH

The use of this utility is privileged; i.e., only logon user 0 may use it. It is used in conjunction with BATCH job processing, to change dynamically the number of batch jobs that are able to execute.

DUMP

DUMP is a utility that allows examination and/or modification of one or more sectors of disk data. The basic command provides a display of the contents of a disk, a file, or a portion of a file, in hexadecimal; alternatively, the dump may be directed to a printer or into another file. Specifying the interactive option allows certain sectors of the disk or file to be read into a change buffer in memory; bytes may be individually examined, changed, and read back to the disk to replace the original version.

EMFGEN

This utility allows the user to add error messages and/or alter existing messages in the error message file, ERRORMSG.SY, which is used by VERSAdos' error message handler to issue most system messages.

FREE

Knowledge of unallocated space on a disk is often needed for file creation or editing, or before copying a file. The FREE utility determines and displays the total number of available sectors and the size of the largest available block of contiguous sectors in decimal and hexadecimal representation for a specified volume.

INIT

All blank diskettes for use on the VME/10 must be formatted and initialized with the INIT utility before their first use. Formatting establishes a sector/track pattern on the diskette which is compatible with the VME/10 and VERSAdos. Initializing creates a Volume Identification Block (VID) on the diskette which can be recognized by VERSAdos. The VID includes a user-supplied volume I.D., description, and ownership. A disk file directory is also created by INIT. If directed to do so, INIT will check the disk for bad sectors; if any are found, INIT will write their locations into the Sector Lockout Table (SLT) so data cannot be written to them.

Used diskettes can also be initialized with INIT to clear the file directory. (Disks containing wanted files should not be initialized, as their directory entries will be altered so as to be unrecognizable by VERSAdos, and new data will overwrite their contents.) The formatting function need not be performed when initializing a used VERSAdos disk. (Note: Formatting destroys all data on a disk.)

An option allows specification of the address of the bootstrap file. For the VME/10, the VERSAdos bootstrap file is named SYS:0..IPL.SY, and it must be located at location $E00.

Although INIT can be used on hard disks, the fixed hard disk furnished with the VME/10 was formatted and initialized at the factory and contains all operating system files. It should not be re-initialized unless these files are to be replaced.

LIB

The Library utility makes useful software routines available for use by more than one program or more than once in a program. These routines, or program modules, are created in assembly or high-level language; put into a file using the editor; assembled or compiled; and combined into a "library" file or files with the LIB utility. These user-created library files, along with those supplied with the system and with optional high-level languages, can then be linked and made accessible to application programs. LIB offers several interactive commands to aid in manipulation of the modules while creating library files.

LIST

Using the LIST utility, all or part of an ASCII disk file can be displayed, written to a separate file, or (if a printer is part of the system) printed. Selectable options allow specification of beginning and/or ending lines; numbering of lines; prompt for wider or narrower line length and longer or shorter page length specification; prompt for heading; and interactive mode. In interactive mode, if the heading prompt option or non-standard length and width prompt option were specified, these parameters can be supplied. Lines to be listed can also be specified while in interactive mode.

MBLM

Object files which were assembled using the M68000 Family Cross Macro Assembler are in S-record format. These files cannot be linked into load modules, but can be transported to the VME/10 and then converted to loadable and executable files by means of the MBLM utility.

MIGR

ASCII programs filed on MDOS-format diskettes can be converted to VERSAdos format with the MIGR utility. MDOS is the resident operating system for Motorola's EXORciser computer. Because EXORciser's standard drives are typically EXORdisk 8" floppies, and VME/10's floppy drive is 5-1/4", an EXORdisk must be available to the VME/10 in order to use MIGR.

MOUNT

MOUNT allows VERSAdos to access disks of differing media formats. It must be used before performing I/O operations to a floppy diskette on the VME/10 (except for the first diskette accessed after powerup). In turn, the DISMOUNT utility must be used after the diskette has been taken off line, to release the device. If the diskette is of VERSAdos format (contains a VERSAdos V.I.D.), entering the MOUNT command and the device designation is all that is required. If the diskette is of foreign format, however, it may be accessed after MOUNTing when configuration data has been supplied by the user during MOUNT's interactive dialog.

NOVALID

If system security level 2 or 3 is in effect, and a user password file exists, NOVALID is used to delete specified user number records from the file.

PATCH

Changes can be made to executable load module files with the PATCH utility. Interactive subcommands allow the display and change of portions of a file after it has been read into memory. This makes it possible to make changes to a program without having to change the source and reassemble it. PATCH includes a one-line disassembler and a one-line assembler.

PRTDMP

The Print Dump utility, PRTDMP, allows dumping part or all of memory to a file after an abort of a load module. The file or a portion of it can then be displayed or routed to a printer for examination. To use this utility, the load module must have been linked with the linker's D option. Interactive commands vary the type of output.

RENAME

This utility is used to change the name of a file and/or its catalog name. The system administrator (logon user 0) may also change a file's user number. User 0 or the volume owner may change a file's protection key.

REPAIR

REPAIR is an interactive utility used to repair the various logical structures of disks and files if they have become damaged. These structures include:

VID	Volume I.D. block
SAT	Sector allocation table
CFGA	Configuration area (media format)
SDB	Secondary directory block (catalog list)
SDE	Secondary directory entry (catalog entry)
PDB	Primary directory block (file name list)
PDE	Primary directory entry (file name entry)
FAB	File allocation block (list of data blocks)
DB	Data block (list of sequential records)
HDR	Header
SLT	Sector lockout table
DTA	Diagnostic test areas

REPAIR can be used to recover a deleted file, if the file's DB and FAB have not been reallocated.

SCRATCH

This utility quickly erases the VID of a used diskette so that it can be reused. Only the disk's owner or logon user 0 can SCRATCH a disk. The disk also may be reformatted with SCRATCH. After using this utility, the disk must be reinitialized by INIT.

SESSIONS

The SESSIONS utility is used to detemine the current online sessions and the batch jobs in queue for execution. Information is displayed by device number (terminal) and sessions number for online sessions and by user number and session number for batch jobs.

SPL/SPOOL

VERSAdos offers a spooling capability whereby a particular volume can be designated as storage media for a queue of files awaiting time-consuming background tasks such as batch and chain processing and printing. This frees the system for foreground operations. The operating system must be SYSGEN'd to add a printer or an auxiliary storage device. SPL must be installed in session 0001. SPOOL may then be accessed whenever needed in subsequent sessions. SPOOL includes a list of subcommands for initiating, monitoring, and cancelling spooling functions.

SYSGEN

The SYSGEN facility makes it possible to customize the operating system, deleting unwanted parameters and adding others to accommodate additional peripheral equipment. Furnished with VERSAdos are SYSGEN command files and chain files for several specific system types which facilitate this process. The files for a particular system are identified by their catalog name; e.g., the command file for the VME/10 is named VMES10.SYSCMD.SA. This file reflects the exact configuration of the VERSAdos software furnished for VME/10 uses. By examining this file, the user can determine whether any of several system attributes should be redefined. If changes are made to SYSCMD, SYSGEN must then be used on the file to ceate a usable system.

SYSANAL

SYSANAL is an interactive operating system debugging utility. It provides a means of examining system tables in RMS68K, the nucleus of VERSAdos, and at any part of memory while VERSAdos is running. Output is to the display screen or to a printer if one is available.

TRANSFER

The ASCII file transfer utility allows up- or downloading of files such as source code or S-records between the VME/10 and another system. The systems may be connected directly between serial ports, or by phone lines/modems. Both systems must be configured for the same baud rate and character makeup. TRANSFER uses two associated programs, ULOAD and DLOAD.

UPLOADS

UPLOADS is used to migrate S-records from some external source to a VERSAdos system. The S-records must be received through an MVME400 dual port serial module or the VME/10 I/O Channel which is connected to the source system via a direct RS-232C hardware configuration.

4.2.5.2 Examples. Following are some typical examples of some frequently-used utilities. They may be used for familiarity with the system.

Boot VERSAdos as described in Chapter 2. Insert a blank diskette in the floppy drive, and make the following entries:

```
    =INIT #FD02
    OK TO INITIALIZE #FD02 (Y/N) ? Y
    Data Density of media (S-single,D-double) D > C
    DO YOU WANT TO FORMAT DISK (Y/N) ? Y
    START FORMAT
    ENTER NEW VOLUME NAME VOL1
    ENTER USER NUMBER 0
    ENTER DESCRIPTION (MAX 20 CHARACTERS) PRACTICE ONE
    (prints only for user 0)
    DO YOU WANT THE BOOTSTRAP (Y/N) ?  N  (Prints only for user 0)
    DO YOU WANT A DUMP AREA (Y/N) ? N
    DO YOU WANT TO VALIDATE SECTORS (Y/N) ? Y
       VALIDATING SECTORS ....
       0 BAD SECTORS ENCOUNTERED
    = COPY 0..PATCH.LO,VOL1:0..PATCH.LO
    =
```

Remove the diskette and dismount it by entering:

```
    =DISMOUNT #FD02
    DISMOUNT Version xxxxxx x
```

Insert another blank diskette, repeat the INIT #FD02 sequence above, giving this diskette a volume name of VOL2 and a description of PRACTICE TWO.

At the VERSAdos prompt, enter:

```
    =COPY 0.*.*.XX,VOL2:0.*.*.XX
    COPY ALL OR SELECT FILES (A/S) ? A
     .
     .
     .
    FILES COPIED = n

    =COPY 0.*.*.NW,VOL2:0.&.&.NW
    COPY ALL OR SELECT FILES (A/S) ? A
     .
     .
     .
    FILES COPIED = n
    =
```

These commands will copy various news and instructional files to the diskette. The asterisk, or "wild card", selects all files on the default volume (the Winchester) with a blank catalog name and extensions of XX and NW. The files are listed as they are copied.

```
    =DIR #FD02
    DEVICE FD02 IS VOLUME VOL2
    USER NUM= 0000       DESC= VOLUME TWO
    =DIR VOL2:0.*.*.*;S
    DIR VERSION xxxxxx x  mm/dd/yy  hh:mm:ss
      .
      .
      .
```

All files on the diskette are listed alphabetically on the CRT screen.

Change default volume to the floppy with the USE session control command:

```
    =USE VOL2:
    SYSTEM VOLUME =SYS:
    USE DEFAULT VOLUME = VOL2:0..
    USER NUMBER = 0
    USER TASK =
    SESSION = 0001
    TERMINAL = CNSL
    OPTION(S) SET =
    =DIR ;E
    DIR VERSION xxxxxx x  mm/dd/yy  hh:mm:ss
```

Each file on VOL2 is listed, with all directory information as to file type, size, location, and protection.

```
    =COPY TRANSFER.XX,#
```

The contents of the ASCII File Transfer instructional file are displayed on the screen. To halt display, press the CTRL and W keys. To resume display, type any key.

```
    =LIST TRANSFER.XX;L=1
```

The file contents are listed on the screen in LIST's format, with page heading and line numbers.

```
    =DEL TRANSFER.XX
    DELETED VOL2:0000..TRANSFER.XX
    =DIR
      .
      .
      .
    (Note that TRANSFER is no longer listed in the volume directory.)
```

Use the FREE utility to ascertain how much space is left on the diskette (dddd = decimal, $hhh = hexadecimal):

```
    =FREE
    VOLUME VOL2:
        dddd/$hhh        TOTAL SECTORS AVAILABLE
        dddd/$hhh        LARGEST CONTIGUOUS SECTORS
              xx%        OF SECTORS ARE AVAILABLE
    =USE SYS:
      .
      .
      .
```

Remove the diskette and dismount it:

```
    =DISMOUNT VOL2:
    DISMOUNT Version xxxxxx x
```

Insert the diskette VOL1 and mount it:

```
    =MOUNT #FD02
    MOUNT Version xxxxxx x
    VOL1 has been mounted
    Total Vdos sectors       2552
    =DUMP VOL1:0..PATCH.LO,#;I
    DUMP VERSION xxxxxx x
    >D $7,$8
```

The I option instructs DUMP to enter the interactive mode, and the # in the output field calls for output to the console screen. In interactive mode, the D subcommand asks for a dump of sectors $7 and $8. They are displayed on the screen in hexadecimal, with printable ASCII characters at the right-hand side.

```
    >QUIT
    =OFF
    09:52:15  9/15/83  END SESSION 0001 USER 0
```

4.3 SOFTWARE DEVELOPMENT

A user may custom-configure an operating system to suit a particular application by using the VERSAdos System Generation Facility (SYSGEN) to modify any of several system attributes, including:

- Type and number of devices
- Number of logical units per user
- Amount of memory space for:
  - Global segment table
  - Trace table
  - Device connection queue
- Number of files

A file must be created to contain a series of commands from the SYSGEN command set. In addition, utility programs not containing interactive dialog may be invoked from within this "command file", allowing a utility or selected portions of a utility to be run as if it had been called directly.

Furnished on the VERSAdos media are command files which represent the configurations of the furnished VERSAdos versions, along with chain files which can be used to perform the SYSGEN. These command files can be listed to identify their parameters, and if a different configuration is required, the command files can be modified and a new operating system generated with SYSGEN.

The following paragraphs contain concepts to be considered when designing an operating system, and a brief listing of the SYSGEN command set. For full information on the SYSGEN process -- the SYSGEN command set, user-changeable parameters, and the SYSGEN command syntax -- refer to the System Generation Facility User's Manual, M68KSYSGEN, which includes a typical example of a SYSGEN command file for the VME/10.

Refer also to the M68000 Family Real-Time Multitasking Software User's Manual, M68KRMS68K, for a more detailed discussion of design concepts.

4.3.1 Designing A System

The software development of an operating system can be broken into four phases. These phases are not necessarily distinct phases carried out in a particular sequence, but will probably overlap and be re-conceived as changes in one phase impact the others.

Analysis Phase.

Many types of systems can be built using real-time operating system concepts, including industrial process control systems, operations control systems, data acquisition systems, management information systems, and development systems. As the first step, the designer should consider some general questions such as:

- What are the basic functional requirements of the system?
- What basic type of system can satisfy these requirements?
- What basic hardware components are needed to satisfy these requirements?
- What basic software components are needed to satisfy these requirements?

The basic functional requirements of the system must be clearly defined at the outset. Some common configurations are: (a) a complete bootstrap-loadable system, in which an entire user system is located in non-resident memory or on a peripheral mass storage device, and is loaded into system RAM at start-up time; (b) a ROM-resident initializer and RMS68K executive, which would load user tasks into RAM during initialization; and (c) a complete ROM-resident system, where the initializer, the RMS68K executive, and the user tasks are all located in ROM.

Design Phase.

In this phase, the basic components needed to satisfy the functional requirements are defined.

A top-down structured methodology will typically be used to define the system functions, making it easier to define the necessary tasks. Once a certain level of functional modularity has been determined, modules can be grouped together to form tasks.

Implementation Phase.

After user tasks have been coded and assembled into relocatable object modules, the final system can be created. The three main steps involved are:

- Build the tailored RMS68K module. The complete RMS68K package is very extensive. A given application may not require its full set of capabilities; therefore, unneeded functions may be omitted by deleting the appropriate directives and reassembling and linking the selected/modified object modules.

- Build application modules, assemble, and link them into load modules. If necessary, the supplied system initializer may also be modified, reassembled, and linked.

- Use the SYSGEN utility to combine the RMS68K load module and the applications load modules.

Testing and Debugging Phase.

Use the TENbug program to test and debug the new operating system. TENbug, the firmware resident monitor, offers versatile commands which facilitate debugging. Refer to the TENbug Debugging Package User's Manual, and to the M68000 Family Real-Time Multitasking Software User's Manual for helpful tables.

4.3.2 SYSGEN Command Set

PARAMETER	Contains the name of a SYSGEN parameter followed by its value. The value is in effect throughout the remainder of the SYSGEN process and cannot be redefined.
PC	Adjusts the location counter maintained during SYSGEN execution.
TASK	Defines the beginning of a task stream which is of the type that results in a task being included in the output file. Also marks the end of the previous task or process if it was not completed by an END statement.
PROCESS	Defines the beginning of a process stream which is of the type that results in a process being included in the output file. Also marks the end of the previous task or process if it was not completed by an END statement.
EXCLUDE	Specifies a segment of a process or task that will not be loaded with the process or task.
SEGMENT	Defines the beginning of a segment stream which is of the type that results in a process being included in the output file. Also marks the end of the previous task or process if such was not completed by an END statement.
END	Ends previous task or process.
MSG	Causes an operator message to be displayed at the relevant terminal.
PAUSE	SYSGEN execution halts until any character is depressed.
SUBS	Indicates source file(s) in which the actual values of SYSGEN defined parameters are substituted for the parameter names.
ASM	Specifies an assembler command line which causes ASM to be invoked.
LINK	Specifies a source file that contains input to linkage editor, and invokes LINK.
IFxx	(Where xx is EQ, NE, GT, LT, GE, or LE), will initiate conditional processing.
ENDC	Terminates the conditional processing associated with its associated IFxx directive.

=<progname>[<legal args>]

Invokes a utility program (where <progname> is the name of the utility and <legal args> represents any command line input which is allowable for that utility). The utility cannot carry on an interactive dialog. This capability is used in the SYSGEN command file to invoke the COPY utility with the append option to produce a single listing of all assemblies and links.

* Comment; everything following the asterisk is treated as a comment and will be listed but not processed.

4.4 OPTIONAL SOFTWARE

Currently available for use on the VME/10 are compilers for the high-level languages, Pascal and FORTRAN. Also available are cross assemblers and linkers which enable programmers working at a VME/10 to assemble and link programs for the MC6800, MC6804, MC6805, and MC6809 microprocessors, as well as a cross Pascal compiler which allows development on the VME/10 of Pascal programs for Motorola's 8-bit microprocessors.

4.4.1 Pascal

Pascal is a high-level, user-oriented language for the MC68000 family of microprocessors, based on the language as defined by Niklaus Wirth. Pascal is a highly structured language which promotes good programming techniques, is self-documenting, and its user-oriented statement forms simplify program writing. Extensions provided by Motorola include address specification for variables, alphanumeric labels, string types, exit, non-decimal integers, runtime error checking, runtime file assignment, and separate compilation and linking. The optimizer produces compact efficient code. Library routines include both IEEE floating point and a single precision fast (multiply: 44 microseconds) floating point.

4.4.2 FORTRAN

The FORTRAN compiler translates source programs written in FORTRAN into MC68000 Family machine language, object code. FORTRAN is a high-level programming language widely used for scientific and engineering problem solving with features also useful for certain business-related applications. The FORTRAN compiler is the 1977 ANSI subset standard. Also included are extensions designed specifically for microprocessor applications such as bit manipulation and assembly language interface capabilities.

4.4.3 Cross Products

The Motorola 8-Bit Cross Macro Assembler Series provides assembly language programming capability for the Motorola MC6809, MC6805, MC6804, MC6801, and MC6800 microprocessor families on the VME/10 System. The assemblers are available to support the individual programming requirements of each processor family. Each assembler features macro instruction capability, evaluation of complex expressions, and inclusion of input from other disk files; and high-level operators that allow the programmer to write structured assembly language. In addition, the assemblers may optionally produce a symbol cross reference listing.

The 8-Bit Cross Linkage Editor also runs on the VME/10 System, and takes the relocatable object module disk file produced by the 8-Bit Cross Macro Assembler Series as its input. The output of the 8-Bit Cross Linkage Editor is an absolute load module file in Motorola's S-record format as well as a comprehensive listing. The S-record load module file can then be transmitted to Motorola EXORciser, EXORset, or Hardware Development Station products for system integration.

The cross Pascal compiler allows Pascal programs for the MC6809 microprocessor to be developed on the VME/10. It is similar to the resident M68000-family Pascal compiler, processing code in two phases. Cross Pascal programs are linked with furnished library routines by the cross linker, and may be linked with assembly language subroutines.

4.4.4 PROM Programmer

Interface software for programming PROM's and EPROM's is available that is compatible with VERSAdos on the VME/10. This software is designed to be used with Data I/O's System 22 PROM Programmer.

With this hardware/software combination, almost any PROM or EPROM in 16-, 18-, 20-, 24-, or 28-pin sizes can be programmed. The Data I/O programmer can be connected to the VME/10 through an RS-232C cable.

4.4.5 Independent Software

A broad range of independent software suppliers support Motorola's 16/32-bit microprocessors. See the latest issue of the "Motorola Microprocessor Software Catalog" for a list of applicable software and addresses of independent suppliers.

CHAPTER 5

CRT TEXT EDITOR

5.1 INTRODUCTION

VERSAdos provides a CRT text editor program, E, to simplify the creation of ASCII text files - such as program source - and modify these files.

Two simple-to-use modes of operation are offered by the editor - CRT screen editing and line editing. In CRT mode, page editing is accomplished by positioning a cursor by means of the cursor keys and special function keys, and/or by an extensive and versatile set of commands. In line mode, insert and command levels are used.

Multiple files may be merged using the editor, and portions of files may be written to new files while editing, and/or copied to other locations within the file, either saving or deleting the original text.

The output of the editor may be in "indexed sequential" files (type ID), or "sequential" files (type S). The default file type when creating a new file is indexed sequential, but sequential can be selected by specifying the S option on the command line. When editing an old file, its type determines the type of the output file, except when otherwise specified by the S or I option.

Sequential files to be edited are placed in a temporary scratch file, and saved to disk after editing by using the editor's QUIT command.

If an edited version of a sequential file is not to be saved on disk, specifying the A option after typing QUIT will delete it from the scratch file, leaving only the original, unedited version on the disk. If no changes are made to the contents on an existing file after it has been opened by E, only the original file is saved. Because indexed sequential files are edited directly, the A option after typing QUIT has no effect.

5.1.1 Command Line

The editor is called from VERSAdos as follows:

```
    E <fname1>[,<fname2>][;<options>]
```

where:

fname1 Is a VERSAdos file descriptor (explained in paragraph 4.3.3).

It either describes an old (existing) file or a new (non-existing) file. A default value of .SA is assumed for the extension.

If existing, <fname1>'s contents are made available for editing. If <fname2> is not specified, the contents of the edit file will be stored under <fname1> on completion (upon issuing a SAVE or QUIT command with no arguments), overwriting its previous contents.

If non-existing, <fname1> is newly created to receive the contents of the edit file on completion.

fname2 Is a VERSAdos file descriptor of a new (nonexisting) file.

If <fname2> is specified, <fname1> must be an existing file. <fname2> will receive the contents of the edit file upon completion, and the contents of <fname1> will be left unchanged.

A default value of .SA is assumed for the <extension>.

options Is one or more of the following options (multiple options are entered with no intervening spaces):

L Enter the line mode of operation. All other options described below are valid for use in the line mode.

The default condition is the CRT mode, except when E is called from a chain file.

Line mode is the normal condition for chain file editing; i.e., the L option need not be specified on the command line within a chain file.

K This option allows the viewing of a file but allows no changes to be made. Records greater in length than 79 characters are not trucated as would be the case without the K option. Viewing of the first 79 characters in each record is provided. Although no changes to a file can be made, the editor scroll functions and the DOWN, FIND, LIST, QUIT, RANGE, UP, and VERIFY commands are available. The K option may only be used alone or in combination with the L option.

I Creates the output file (<fname2> or new <fname1>) in indexed sequential format.

S Creates the output file (<fname2> or new <fname1>) in sequential format.

A Sets assembler tabs (columns 11, 18, 37).

C Sets COBOL tabs (columns 6, 9, 12).

F Sets FORTRAN tabs (column 7).

P Sets Pascal tabs (columns 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46, 49, 52, 55, 58, 73).

NOTE

When no tab option is specified, a default format of tab stops every 10 columns is assumed.

5.1.2 E Commands

The commands available for use when editing, and a summary of their functions, follows. Note that the following terms are equivalent:

line = record
current line = line at which cursor is positioned
column = character position
page = full screen display

Commands to E may be in uppercase or lowercase.

CRT & LINE MODE

ADUP	Duplicate records (lines) from the file and place them in the XTRACT buffer, appending them to any records already in the buffer. The records still remain in the file. A vertical range may be specified in the format ADUP 1-10 (lines 1 through 10). If range is not specified, default is the current record only. Vertical range may also be in the format ADUP *-100, where * is current line and 100 is the next 100 lines. If number of lines is not specified, only the current line is used.
AMOV[E]	Move records from the file and place them in the XTRACT buffer, appending them to any records already in the buffer. The records moved are deleted from the file. A vertical range may be specified in the formats AMOV 60-100 (line numbers specified) or AMOV *-100 (number of records specified), with defaults as specified for ADUP. The XTRACT command is used to copy the deleted data back into the file.
C[HANGE]	Change strings within records. The following may optionally be specified: vertical range of records, horizontal range with each record (number of characters within each line), a "transparent" charcter (to be inserted in the new string to cause any characters in the old string in the same columns to be ignored, or left unchanged), occurrence within a record in which the change is to be made, old string and new string, and number of lines in which the change will be made. For example, C 5-9:10-12; a/JMP/JSR/a changes the characters JMP to JSR every time they appear in columns 10 through 12 in lines 5 through 9 in the file being edited.
DEL[ETE]	Delete records or parts of records from the edit file. Vertical and horizontal ranges may be specified, with defaults as specified for CHANGE. For example, DEL deletes the line at which the cursor is pointing. DEL *-20:10-50 deletes all data in columns 10 through 50 from the next 20 lines, beginning with the current line.

D[OWN]	Move the record pointer downwards. For example, D 50 moves the cursor from its current location to the 50th line following.
DUP[LICATE]	Duplicate records and place them a newly created XTRACT buffer, replacing any previous XTRACT buffer. The records still remain in the file. A vertical range may be specified. For example, DUP copies only the current line into the XTRACT buffer; DUP *-15 copies the next 15 lines, beginning with the current line.
EX[TEND]	Append data to the end of records. Any string may be appended, and a vertical range may be specified. For example, EX 20-29 "HDS400" appends the string HDS400 to the ends of the 10 lines beginning with line 20.
F[IND]	Find a string, or position record pointer at a record, the format of the FIND command is the sample as for the CHANGE command, except that only one string may be specified. For example, F 3-5:2-7;a/HDS400/a locates every occurrence of the string HDS400 in character positions 2 through 7 of records 3 through 5 and displays them on the screen. F 0 moves the cursor to the beginning of the file; F 200 moves the cursor forward to the 200th line in the file.
LINE	Display the line number of the current record. After LINE is entered, the number of the record at which the cursor is positioned is displayed at the bottom of the screen. For example, CURRENT LINE IS 18.
MERGE	Copy a file or a portion of a file resident on disk into the file being edited. This allows combining files or copying of data within a vertical range from another file to the desired location in the edit file. For example, MERGE VOL1:..TEST.SA inserts the entire contents of the file TEST.SA on a disk named VOL1, into the edit file above the current record. MERGE 1-3 OLDFILE.SA copies lines 1 through 3 from the file OLDFILE.SA on the default disk, to the edit file above the cursor location.
MOVE	Move records into a newly created XTRACT buffer, replacing any previous XTRACT buffer. The records moved are deleted from the file. This command works exactly as AMOV, except that the records moved replace any previously stored XTRACT buffer contents. Records moved are deleted from their current location and can be restored to the desired location with the XTRACT command. For example, MOV 5-10 deletes 6 records from their current location; after the cursor is moved to the desired location, typing XTRACT copies them from the XTRACT buffer to the edit file above the cursor location.

PRINT	Output records to the printer. A vertical range of records may be specified; default is the entire file. The records may optionally be printed out double- or triple-spaced by specifying the option D or T. For example, PRINT prints the entire edit file, single-spaced, on printer #PR. PRINT 100-199 #PR2 D prints records 100 through 199, double-spaced, on printer #PR2.
QUIT	Save the edit file in a VERSAdos file, terminate the edit session, and return control to VERSAdos. For example, QUIT closes the edit file and writes it to the disk under the output file name. If no changes were made during editing, no output file is created. If, after editing, the changes made are not wanted, exiting with QUIT A prevents the output file from being created. Note, however, that the A option on QUIT is only valid if the output file was to have been sequential (either because the input file was sequential or because the S option was specified on the E command line).
R[ANGE]	Establish default values for the vertical and horizontal ranges of the CHANGE, FIND, PRINT, and SAVE commands. The original defaults are entire record and/or entire file. RANGE is used to change these defaults. For example, R 1-100 changes the vertical range for the FIND, CHANGE, PRINT, and SAVE commands. R :10-30 leaves the vertical range unchanged, but sets the default horizontal range to columns 10 through 30. Entering R or RANGE alone returns defaults to entire record/entire file.
SAVE	Save part or all of the edit file in a VERSAdos file, and continue editing. For example, SAVE *-100 VOL1:0.CAT1.FILE2.SA creates a new file named CAT1.FILE2.SA on VOL1, and copies 100 lines into it from the edit file, beginning at the current line. SAVE FILEDUP.SA copies the entire edit file into a new file named FILEDUP on the current volume.
TAB	Specify tab stops. The default tab stops are set at every 10th column. The column numbers of desired tab locations to be added to existing settings can be specified, or an option letter (A, C, F, or P) can be specified to set tabs at locations convenient for Assembler, COBOL, FORTRAN, or Pascal source programming. When an option letter is specified, previous tab settings are cleared. Specifying TAB with no tab stops or option letter sets tabs at the default settings, or at settings specified on the E command line, if any. For example, TAB 25,35 adds stops at columns 25 through 35, but does not change any settings already in effect. TAB A sets tab stops at columns 11, 18, and 37, convenient for Assembly language source code.

U[P]	Move the record pointer upwards. For example, U 25 moves the cursor to the 25th line preceding the current line.
X[TRACT]	Copy the records from the XTRACT buffer placed there by ADUP, DUP, AMOV, or MOVE, and insert them above the current record in the file. The records still remain in the buffer. XTRACT may also be used simply to empty the buffer. For example, XTRACT or X copies the contents of the buffer into the edit file, above the current line. XTRACT A or X A does not copy the contents of the buffer into the edit file, but deletes them from the buffer.

LINE MODE ONLY (;L OPTION OR WITHIN CHAIN FILES)

COLM	Display the ruler of column spacings.
DTAB	Delete tab stops.
I[NSERT]	Enter insert level, for adding records to the edit file.
L[IST]	List records in the edit file.
STAB	Specify tab stops.
V[ERIFY]	Display records that are altered or record pointer changes.

5.1.3 Examples

The following simple examples are intended to illustrate various functions of the editor.

With VERSAdos running, insert the diskette VOL1 created in the example in Chapter 4 (or use another formatted, initialized diskette) and enter:

```
= MOUNT #FD02
  .
  .
  .
=
```

Change defaults to the diskette (if volume name is not VOL1, substitute the actual volume name for VOL1):

```
= USE VOL1:0
  .
  .
  .
=
```

Create a new file on the diskette:

= E TESTFILE.SA

The new file is opened and the editor is in CRT page edit mode. Fill four lines with random text, beginning with an asterisk and a tab (*-->|) such as the following:

```
*       KLJLKSDF AJFLKSDJK SJFKDJFJS WUROEURIXS SJLKWUIOSKFLDKJA SDFJLSJSKJ
*       SDFLKJKJAF SJFLKDKA LKJKJE SAJLJKJ;LLLFS AKJLJ  AJDLFJKD E UOULF
*       DSJFLSDJK ADJFLDJKF ADFJLDJF AADFJLKJA FJSLFKJ ADFJLJ A SFJDLJ A
*       SFJKDJFKLA A FSJFLJ AJDLFJSK A FSLJFJA SJFLDJF A
```

Press ↖ key to move cursor to upper left hand corner. Press the F1 key to enter command mode. Make the following changes to the file:

> C *-999;a/F/$/a

(In all lines from the cursor location to the end of the file, change all occurrences of F to $.)

```
*       KLJLKSD$ AJ$LKSDJK SJ$KDJ$JS WUROEURIXS SJLKWUIOSK$LDKJA SD$JLSJSKJ
*       SD$LKJKJA$ SJ$LKDKA LKJKJE SAJLJKJ;LLL$S AKJLJ  AJDL$JKD E UOUL$
*       DSJ$LSDJK ADJ$LDJK$ AD$JLDJ$ AAD$JLKJA $JSL$KJ AD$JLJ A S$JDLJ A
*       SFJKDJFKLA A FSJFLJ AJDLFJSK A FSLJFJA SJFLDJF A
```

> DEL 1-4:2-9

(In lines 1 through 4, delete all data in character positions 2 through 9. Note that the spaces inserted by the tab key are deleted.)

```
*KLJLKSD$ AJ$LKSDJK SJ$KDJ$JS WUROEURIXS SJLKWUIOSK$LDKJA SD$JLSJSKJ
*SD$LKJKJA$ SJ$LKDKA LKJKJE SAJLJKJ;LLL$S AKJLJ  AJDL$JKD E UOUL$
*DSJ$LSDJK ADJ$LDJK$ AD$JLDJ$ AAD$JLKJA $JSL$KJ AD$JLJ A S$JDLJ A
*SFJKDJFKLA A FSJFLJ AJDLFJSK A FSLJFJA SJFLDJF A
```

Press F1 to return to page edit mode. Note that the cursor is still positioned at line 4.

Press the EOL key on the right hand key pad. Line 4 is deleted. Press the carriage return key (<--') 6 times, moving the cursor to line 10. Verify this by pressing the F1 key, then typing:

```
> line
CURRENT LINE IS      10
>
```

Press F1, then fill 2 lines with dollar signs (hold $ and ⇧ keys down until 2 lines are filled.)

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$
```

Press F1 to return to command mode. Enter:

```
> DUP 10-11       (Fill XTRACT buffer with contents of lines 10 and 11,
> XTRACT          then copy the contents of the XTRACT buffer to the file,
> XTRACT          inserting it at the cursor location, four times.
> XTRACT          As the lines are added, the display is scrolled upwards.)
> XTRACT
```

```
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$
```

```
> find 10
```

Move the cursor to the last full line of dollar signs with the ↓ key and ask for line number:

```
> line
CURRENT LINE IS   19
```

Change all occurrences of $ in columns 25 through 50 in lines 10 through end of file to dots:

```
> c 10-999:25-50;a/$/./a
```

(This takes several seconds, and then the display changes.)

```
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$
```

Change those dots in columns 30 through 45 in lines 12 through 17 to asterisks:

```
> C 12-17:30-45;a/./*/a
```

```
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$
```

Copy part of the file to a new file and exit the editor.

```
> save 10-19 NEWFILE.SA
> quit
EDIT DONE
=
```

Open the newly created file. The editor is in command mode, and the lines copied to this file are displayed.

```
= E NEWFILE.SA
  •
  •
  •
```

Press F1 to enter page edit mode. Using the cursor arrow keys, move the cursor through the display and make a few changes to the file by typing over the existing characters; e.g.:

```
$$$$$$$$$$$$$$$$$$$$$$$$$........VME/10............$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$....****************......$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$....****************......$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$....****************......$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$....****************......$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$....****************......$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$....****************......$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$$......MICROCOMPUTER.......$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

Note that in page edit mode, characters entered replace those over which they are typed. To insert space for new text, use the ICHAR and ILINE keys to insert character space and insert line space. Use the DCHAR and DLINE keys to delete characters and lines.

Position the cursor at the last character in the last line in the file and press <--|, then press F1.

Merge part of the old file with this file:

```
> Merge 10-19 TESTFILE.SA
```

The contents of lines 10 through 19 of the old file are inserted at the cursor position and displayed.

Change the last line:

```
> C 20:34-39//SYSTEM/
```

```
$$$$$$$$$$$$$$$$$$$$$$$$........VME/10............$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$......MICROCOMPUTER.......$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.....****************.....$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$..........................$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$$$$$$$$$$$$$$$$.........SYSTEM...........$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
```

If part of the display has scrolled off the top of the screen, press the F4 key to view page 1, or press F6 repreatedly until the first line in the file can be seen ("BOF OR EOF ENCOUNTERED" appears at the bottom of the screen).

Four of the function keys (Fn) can be used to advance or reverse the display through a file, as follows:

F3	Display next page
F4	Display previous page
F5	Move display up one line at a time
F6	Move display down one line at a time

In all cases, the display will advance or retreat only until BOF or EOF is reached.

If a printer is part of the system, get a printout of the file's contents, then return to VERSAdos:

```
> PRINT #PR
> QUIT
EDIT DONE
=
```

CHAPTER 6

ASSEMBLER

6.1 INTRODUCTION

VERSAdos' assembler, the M68000 Family Resident Assembler, is used to translate M68000 family assembly language source programs into relocatable object code which, after linking, is "machine-readable" by the MC68010 microcomputer. The assembler's capabilities include handling the following:

. Structured syntax
. Complex expressions
. Macros
. Conditional assembly
. Absolute or relocatable code generation
. Symbol table listing
. Cross-referencing

Source programs are written in MC68000-family assembly language, entered into a file on disk using the VERSAdos CRT Text Editor, assembled, then linked with the VERSAdos linkage editor and assigned absolute memory addresses or relocatable addresses. The output file produced by the linker may be a load module (an executable program) or a relocatable object module, which can then be linked with other assembly, Pascal, or FORTRAN programs.

6.2 SOURCE PROGRAMS

MC68000 family assembly language source programs consist of a series of statement lines arranged in a logical way to perform predetermined tasks. Source statement lines may be any of the following:

. Executable instructions
. Assembler directives
. Macro invocation
. Comment

6.2.1 Coding

Source statement lines are comprised of four fields, separated by a space or spaces:

. Label
. Operation
. Operand
. Comment

If the first character (column 1) in a statement is an asterisk (*), the entire line is treated as a comment -- ignored by the assembler, but printed in listings for documentation of the program.

Source statement lines must not be numbered. The assembler will number the lines while assembling the program.

Label. The first field of a source statement is the label field. A label is generally a symbolic name representing a numerical value or an address (location) in memory. Labels are allowed on all instructions and directives which define data structures (FOR, IF-THEN). Labels are required in the assembler directives which define symbol values -- SET, EQU, REG. Labels also are required on MACRO definitions and on the IDNT directive.

Operation. This may be any of the following:

. Instruction -- the MC68000 instruction set, with the following additions for the MC68010:

MOVE CCR,<effective address>	Move from condition codes register to effective address.
MOVEC Rc,Rn, or Rn,Rc	Move from control register Rc to register Rn or from register Rn to control register Rc.
MOVES <effective address>,Rn or Rn,<effective address>	Move from effective address to register Rn or from register Rn to effective address.
RTD #<displacement>	Return from subroutine with displacement (2's complement 16-bit integer, sign extended to 32 bits).

Certain instructions allow a "quick" and/or an "immediate" form when immediate data within a restricted size range appears as an operand. These abbreviated forms are normally chosen by the assembler, when appropriate. However, it is possible for the programmer to "force" such a form by appending a "Q" or "I" to the mnemonic opcode (to indicate "quick" or "immediate", respectively). The instructions are: ADDI, ADDQ, CMPI, EORI, MOVEQ, ORI, SUBI, SUBQ.

Some instructions also have "address" variant forms (which refer to address registers as destinations); these variants append an "A" to the instruction mnemonic. This variant will be chosen by the assembler without programmer specification, when appropriate to do so; the programmer need specify only the general instruction mnemonic. However, the programmer may "force" such a variant form by appending the "A". They are ADDA, CMPA, MOVEA, SUBA.

The CMP instruction also has a memory variant form (CMPM) in which both operands are a special class of memory references. The CMPM instruction requires postincrement addressing of both operands. The CMPM instruction will be selected by the assembler, or it may be specified by the programmer.

. Directive -- pseudo-operation codes for controlling the assembly process.

. Macro call -- invocation of a previously described macro.

Some instructions and directives can operate on more than one data size. Size is specified by one of the following suffixes to the operation mnemonic:

.B = Byte (8-bit data)
.W = Word (16-bit data; usually default)
.L = Long word (32-bit data)

Examples:

LEA 2(A,0),A1	Long word size is default; loads effective address of first operand into register A1.
ADD D1,D2	Word size is default; adds low order word of D1 to low order word of D2.
ADD.L A3,D3	Long word size; adds entire 32-bit contents of A3 to D3.
DC.B 10,5,7	Byte size; defines constants 10, 5, and 7 (decimal) in three contiguous bytes of memory.

Operand. When used, specified generally in the format <opcode source>,<opcode destination>; e.g., MOVE D1,D2 moves the contents of D1 to D2.

Comment. This optional field is used to document the code. It appears in listings, but otherwise is ignored by the assembler.

6.2.2 Symbols and Expressions

Symbols recognized by the assembler consist of one or more valid characters, the first eight of which are significant. The first character must be an uppercase letter (A-Z) or a period (.). Each remaining character may be an uppercase letter, a digit (0-9), a dollar sign ($), a period (.), or an underscore (_).

Numbers recognized by the assembler include decimal, hexadecimal, octal, and binary values. Decimal numbers are specified by a string of decimal digits (0-9); hexadecimal numbers are specified by a dollar sign ($) followed by a string of hexadecimal digits (0-9, A-F); octal numbers are specified by an "at" sign (@) followed by a string of octal digits (0-7); binary numbers are specified by a percent sign (%) followed by a string of binary digits (0-1).

One or more ASCII characters enclosed by apostrophes (') constitute an ASCII string. ASCII strings are left-justified and zero-filled (if necessary), whether stored or used as immediate operands. This left justification will be to a word boundary if one or two characters are specified, or to a long word boundary if the string contains more than two characters.

Expressions are composed of one or more symbols, which may be combined with unary or binary operations. Legal symbols in expressions include:

a. User-defined labels and their associated absolute or relocatable values.

b. Numbers and their absolute values.

c. The special symbol "*" always identifies the value of the program counter at the beginning of the DC directive, even when multiple arguments are specified (e.g., DC.B 1,2,3,*-3). The program counter may be either absolute or relocatable.

Operators recognized by the assembler include the following:

Operator	Definition	Operator precedence
+	Addition	6
-	Subtraction	6
*	Multiplication	5
/	Division	5
-	Unary minus	2
>>	Shift right	3
<<	Shift left	3
&	Logical AND	4
!	Logical OR	4
( )	Parentheses	1

6.2.3 Registers

The following MC68010 register mnemonics are recognized by the assembler:

D0-D7	Data registers.
A0-A7	Address registers.
A7, SP	Either mnemonic represents the system stack pointer of the active system state.
USP	User stack pointer.
CCR	Condition code register (low 8 bits of SR).
SR	Status register. All 16 bits may be modified in the supervisor state. Only low 8 bits (CCR) may be modified in user state.
PC	Program counter. Used only in forcing program counter-relative addressing.
VBR	Vector base register. Supports multiple vector table areas during exception processing. Accessed by the MOVEC instruction.
SFC	Alternate function code source register. Accessed by the MOVEC instruction.
DFC	Alternate function code destination register. Accessed by the MOVEC instruction.

6.2.4 Macros

A set of instructions that are to be repeated can be defined as "macros"; values of variables within the macro routines can be specified later when the macro is called into use. When the macro is called, the generated instructions that comprise the macro are "expanded" -- executed inline in the normal flow of the program.

A macro definition consists of header, body, and terminator:

<label> MACRO	header; <label> must be a unique identifier, the <name> by which the macro is called.
<statement> . . . <statement>	body; may be instructions, directives, calls to other macros; parameter arguments may be included.
ENDM	terminator.

Statements within the macro can call out argument substitution, using the designations \0 through \9 and \A through \Z, e.g., MOVE.L \3,LOCATN. When the macro is expanded after being invoked, the third parameter specified on the macro call will be moved to LOCATN.

Macros are invoked from a program in the following form:

```
[<label>] <name>[.<qualifier>] [<parameter list>]
```

where:

label	is another unique label.
name	is the <label> defined with the MACRO directive.
.qualifier	is the \0 size/displacement qualifier, such as .B, .W, or .L.
parameter list	is the list of parameters to be substituted into the macro, separated by commas. Null parameters must also be set off by commas --e.g., <param1>,<param2>,,,<param5>. Up to 36 parameters may be specified.

6.3 INVOKING THE ASSEMBLER

The command line format for the assembler is:

```
        ASM   <source file>[,[<object file>][,<listing file>]][;<options>]
```

Only the <source file> is required. The default extension on the <source file> is SA. If the <object file> and/or <listing file> are not specified, they will default to the same file name as the <source file>, but with extensions of RO and LS, respectively.

Multiple source files may be assembled by separating these input files with a slash (/). In the case of multiple source files, the first file name is used for the default object and listing file names. The listing may be output to the CRT or the printer during assembly by specifying the appropriate mnemonic in place of the listing file; e.g., the command ASM TEXT,,#PR will print the listing.

The assembler recognizes the following options on the command line:

OPTION	DEFAULT	FUNCTION
C	C	Produce object code.
D	D	Produce symbolic debug symbol table file; file name will be same as that of the relocatable object file, with an extension of RS.
L	L	Produce listing.
M	M	List macro expansions.
P=<processor>	P=68000	Accept MC68000 instruction set. P=68010 is required for VME/10, unless OPT P=68010 is used within program.
R	-R	Inhibit production of cross-reference.
S	-S	Inhibit listing of structured control expansions.
W	W	Enable warning messages during assembly (default).
Z=n	Z=37	Increase data area size to n Kbytes.

If the P=<processor> or Z=n option is followed by another option, a separating comma must be used. Otherwise, separating commas are not required for multiple options. For all options except P=<processor> and Z=n, a minus sign before the option letter produces the opposite effect.

6.4 DIRECTIVES

Assembler directives (pseudo-ops), with the exception of DC and DCB, are instructions to the assembler rather than instructions to be translated into object code. Following the descriptions and examples of the basic forms of the most frequently used assembled directives.

ASSEMBLY CONTROL

ORG	Absolute origin
INCLUDE	Include second file as if it was inline
SECTION	Relocatable program section
OFFSET	Define offsets
MASK2	Assembler for Mask2 (R9M, MC68000 mask)
END	Program end

SYMBOL DEFINITION

EQU	Assign permanent value to label
SET	Assign temporary value to label
REG	Define register list for MOVEM instructions

DATA DEFINITION/
STORAGE ALLOCATION

COMLINE	Command line
DC	Define constants
DS	Define storage
DCB	Define constant block

LISTING CONTROL
AND OPTIONS

PAGE	Top of page
LIST	Enable the listing
NOLIST or NOL	Disable the listing
FORMAT	Enable the automatic formatting
NOFORMAT	Disable the automatic formatting
SPC n	Skip n lines
NOPAGE	Disable paging
LLEN n	Set line lengths $72 \leq n \leq 132$
TTL	Up to 60 characters of title
NOOBJ	Disable object output
OPT	Assembler options
FAIL	Programmer-generated error

LINKAGE EDITOR CONTROL

IDNT	Relocatable identification record
XDEF	External symbol definition
XREF	External symbol reference

STRUCTURED CONTROL

ELSE	ENDW	REPEAT
ENDF	FOR	UNTIL
ENDI	IF	WHILE

6.5 ASSEMBLER OUTPUT

Assembler outputs include an assembly listing, a symbol table, a symbolic debug symbol table file, and a relocatable object program file.

The assembly listing includes the source program, as well as additional information generated by the assembler. Most lines in the listing correspond directly to a source statement. Lines which do not correspond directly to a source line include:

- Page header and title
- Error and warning lines
- Expansion lines for instructions over three words in length

The last page of the assembly listing is the symbol table. Symbols are listed in alphabetical order, along with their values and an indication of the relocatable section in which they occur (if any). Symbols that are XDEF, XREF, REG, in named common, or multiply defined are flagged. If option CRE has been specified in the program, the cross-reference listing will identify the source lines on which the symbol was defined or referenced (definitions appear first, flagged with a "-").

LISTING CONTROL AND OPTIONS	
PAGE	Top of page
LIST	Enable the listing
NOLIST or NOL	Disable the listing
FORMAT	Enable the automatic formatting
NOFORMAT	Disable the automatic formatting
SPC n	Skip n lines
NOPAGE	Disable paging
LLEN n	Set line lengths $72 \leq n \leq 132$
TTL	Up to 60 characters of title
NOOBJ	Disable object output
OPT	Assembler options
FAIL	Programmer-generated error

LINKAGE EDITOR CONTROL	
IDNT	Relocatable identification record (label required)
XDEF	External symbol definition
XREF	External symbol reference

6.6 LINKAGE

The relocatable object module produced by the assembler must be processed by the M68000 family linkage editor, or linker, in order to resolve relocation and cross referencing needs. The output of the linker, in turn, can be another relocatable object module, a file in "S-record" format which can be transmitted to another computer, or a "load module" which can be executed directly on the VME/10. A relocatable object module may be linked with other assembly language programs, as well as FORTRAN and Pascal programs; in these cases, the user should be aware of FORTRAN and Pascal requirements. Libraries of commonly used routines may also be linked with assembly language programs.

"Relocation" refers to the process of binding a program to a set of memory locations at a time other than during the assembly process. For example, if subroutine "ABC" is to be used by many different programs, it is desirable to allow the subroutine to reside in any area of memory. One way of repositioning the subroutine in memory is to change the "ORG" directive operand field at the beginning of the subroutine, and then to re-assemble the routine. A disadvantage of this method is the expense of re-assembling ABC. An alternative to multiple assemblies is to assemble ABC once, producing an object module which contains enough information so that another program (the linkage editor) can easily assign a new set of memory locations to the module. This scheme offers the advantages that re-assembly is not required, the object module is substantially smaller than the source program, relocation is faster than re-assembly, and relocation can be handled by the linkage editor (rather than editing the source program and changing the ORG directive).

In addition to program relocation, the linkage editor must also resolve inter-program references. For example, the other programs that are to use subroutine ABC must contain a jump-to-subroutine instruction to ABC. However, since ABC is not assembled at the same time as the calling program, the assembler cannot put the address of the subroutine into the operand field of the subroutine call. The linkage editor, however, will know where the calling program resides and, therefore, can resolve the reference to the call to ABC. This process of resolving inter-program references is called "linking".

Program sections provide the basis of the relocation and linking scheme. Each of these sections may also have a variable number of named common sections associated with it, with each common section having a unique name. These relocatable sections are passed on to the linkage editor, which concatenates all sections with the same number in the different modules to be linked. Each of the 16 relocatable sections may contain data and/or code; in addition, named common sections may be defined within any relocatable section.

Absolute sections are unnumbered (and, therefore, unlimited in number); they are specified by the ORG directive.

6.7 EXAMPLES

Following is an example of creating an absolute load module using the assembler and the linkage editor.

a. At the VERSAdos prompt (=), invoke the CRT text editor by entering the following:

```
=E PROGNAME
```

PROGNAME represents the name of the source file being created; SA is the default extension, and is usually used for ASCII source files.

After the user program has been entered, depress the F1 function key to return the editor prompt (>) to the lower left portion of the screen, and exit the editor as follows:

```
>QUIT
```

b. Assemble the program:

```
=ASM PROGNAME
```

The relocatable object module created by the assembler will be written to a file named PROGNAME, with the default extension of RO. A list of the source with the hex representation will be written to PROGNAME.LS.

c. Link the program. In this example, an assembly language subprogram (created and assembled as above) is linked with the main program into an executable load module (the file PROGNAME.LO):

```
=LINK PROGNAME/SUBPROG
```

RO is the default extension for the input files.

An assembly language program may also be linked with a Pascal or FORTRAN program which is to call it.

d. Alternatively, a chain file such as the following could have been used to assemble (step b.) and link (step c.) the assembly program.

First, call the editor:

```
=E CHAINASM.CF
```

Then enter the following command lines into the edit file:

```
=ASM PROGNAME
>=ASM SUBPROG
>=LINK PROGNAME/SUBPROG
>=END
```

NOTE

The VERSAdos prompt (=) is required on a chain file command which calls a utility.

Exit the editor by depressing the F1 key and typing:

```
>QUIT
```

The chain file is executed by entering its name to VERSAdos:

```
=CHAINASM.CF
```

e. The user has the option of either listing the program on the screen or printing hard copy to investigate and correct errors.

   1. To list the program on the screen, enter:

      ```
      =LIST PROGNAME.LS
      ```

      The entire program will scroll on the screen (line by line) and the compilation errors will be indicated as they occurred. To stop the scrolling in order to investigate an error, press CTRL-W (hold the CTRL key, then press the W key). To continue scrolling, press any key.

   2. To list the program on the printer, enter:

      ```
      =COPY PROGNAME.LS,#PR
      ```

      The entire program will be printed, indicating the total number of errors and also where each error appeared in the program listing.

f. Use the text editor to correct the errors, then call the chain file again. When no assembly errors exist, a valid absolute load module will automatically be created, provided the linkage editor encounters no problems.

g. The absolute load module is now ready to be examined or modified by utilizing one of the two debugging programs -- DEbug or TENbug. To use DEbug, refer to the SYMbug/A and DEbug Monitors Reference Manual for the procedure to load an absolute load module into memory. To use TENbug, refer to the BO command in the TENbug Debugging Package User's Manual for the procedure to load an absolute load module into memory.

h. A completely debugged load module can be executed by entering only its name to the VERSAdos prompt (=):

```
=PROGNAME
```

CHAPTER 7

LINKAGE EDITOR

7.1 INTRODUCTION

After user-written source files have been assembled (if assembly language) or compiled (if high-level language such as Pascal or FORTRAN) into "relocatable object module" files, they must be "linked" by the VERSAdos Linkage Editor (the "linker") to convert them to absolute binary load modules. These load modules are programs which are executable under VERSAdos. Furnished or user-created library files may also be linked with the object modules, if their routines are to be utilized.

One or two other forms of output files may be selected instead of a load module: relocatable object modules can be linked to form a larger relocatable object module, or they can be converted to Motorola S-record format modules, which can easily be downloaded from the VERSAdos system to another computer. During the downloading to a target system, the S-records are converted to machine-readable code. An appendix to the VERSAdos System Facilities Reference Manual, MM68KVSF, describes Motorola S-records.

The task of the linker is to examine and gather information from the relocatable object module(s) associated with independently compiled or assembled source code module(s) and, based upon this information, to allocate memory to code and data, to relocate according to this allocation, and to resolve all references to symbols assumed to be global to one or more modules.

The linker requires two passes (the input is read twice) before it can create an output module. During the first pass, the linker gathers information about externally referenced and externally defined symbols, building a symbol table in the process. It also keeps track of what sections are assigned -- their names, lengths, and starting addresses. Finally, it determines what modules from the library file(s) are required. During pass one, no attention is paid to the actual code/data in the relocatable object modules that are input.

After pass one, if an S-record module or absolute load module is being generated, the linker assigns each section to an absolute address in memory. This address is where the section will be loaded when the absolute load module is executed. After allocation of memory, it is known how much space is required for the resulting load module or S-record module. At this time, the output file is allocated.

If a relocatable object module is being generated, the linker simply computes the total size of each section in use; it opens the output file and outputs the necessary information about each section and symbol to it. (Note: Each section will always be started on a word boundary.)

The linker then proceeds to pass two, where the relocatable object modules read in pass one are re-read in the same order. However, at this time, the data/code in each module is relocated and reference resolution is performed, and the data/code is then written to the output file. If a relocatable object module is being produced, however, the input is not relocated, but any references between the input modules are resolved.

At the completion of pass two, the linker outputs its final listings. The listings produced depend on which options were specified in the invoking command line.

7.2 INVOKING THE LINKER

When the linker is called from VERSAdos, several variations in processing can be specified as options on the command line. Alternatively, certain parameters and options may be specified interactively by user commands after the linker has been started. The options are described in paragraph 7.2.1 and the user commands are described in paragraph 7.2.2.

7.2.1 Command Line

VERSAdos will bring the linker into memory and begin its execution in response to a LINK command. Parameter or option information provided with the LINK command line is saved for use by the linker. The format for this command line is as follows:

```
    LINK  [<input file>][,[<output file>][,<list file>]][;<options>]
```

<input file> is the name of a disk file containing one or more relocatable object modules. As many input files as desired may be specified on the command line, separated by a slash. Extensions, if not given, default to RO. These files are processed first before any files specified by INPUT commands.

If input files are not specified, the A option is forced on to allow entry of user commands. Files may then be specified by means of the INPUT command.

<output file> is the name of the disk file which will be used to contain the output of the linker. This will be a file containing a load module, a relocatable object module, or an S-record module (depending on what option(s) are used). If a load module or an S-record module is being created, this file name need not be specified, in which case the linker will assume the name of the first input file processed, but with an extension of LO or MX, respectively. If a relocatable object module is being created, an output file name that is different from the input file name(s), <u>must</u> be specified. The default extension for this file name is LO, RO, or MX, depending upon whether a load module, relocatable object module, or S-record module is being produced, respectively.

<listing file> is the name of the disk file, with default extension of LL, which will be used to contain the listings produced by the linker.

If # or #PRn is specified instead of <list file>, the listings will be directed to the user's console or line printer, respectively.

If no list file or device is specified, but options requesting listings are, the listing will be directed to the default output file/device (usually the user's console).

<options> may be one or more of the following options.

OPTION	DEFAULT	FUNCTION
A	-A	Accept user commands from the command input device. If no input files are specified in the command line, this option is forced on.
B	-B	In the listing produced by the assembler, each relocatable section in a module starts at relative address zero. However, each actual starting address (offset) is wherever the linker locates a section within a memory segment. Therefore, to form actual addresses for a section, this offset must be added to each relative address in the listing. The B option forces each relocatable section from each module to start on a page ($100-byte) boundary. The offset then appears as $xxxx00, which -- being a multiple of $100 -- makes it easier to work with and remember. This option does not affect an absolute section, which is always placed at the address indicated by its ORG directive. If a START user command -- following after a B option -- defines a starting address that is not on a page boundary, the particular section or sections will start at the first page boundary after that address. This option may be used only if a load module or an S-record file is being created.
D	-D	Create a symbolic debug file. It will have the same name as the first file processed for input, with an extension of DB.
H	-H	Include in the listing file the information found in the header record of each input object module.
I	-I	List the command line and all user commands, if any, on the listing file.
L=<libfile>	-L	Search the specified library files in the order listed if any references are unresolved at the end of pass 1. Process any modules which contain definitions satisfying any unresolved references. More than one <libfile> may be specified, separated by a slash. If this is not the last option in the command line, it must be followed by a comma.
M	-M	List a map of the resulting module on the listing file.
O	-O	Create an absolute binary load module. If no output file name was specified in the command line, the load module will have the same name as the first file processed for input, but with an LO extension. Note that this option and the R and Q options are mutually exclusive.

OPTION	DEFAULT	FUNCTION
P	P or -P	Search default libraries at the end of pass 1 if unresolved external references. There is one default library file supplied for each language supported by VERSAdos (e.g., 0.&.FORTLIB.RO or 0.&.PASCALIB.RO). This option defaults to on (P) if a load module or an S-record module is being created (O or Q option is on). Otherwise, it defaults to off (-P). The L option, if specified, is executed first, in order to load any user-written modules before default library modules.
Q	-Q	Create an S-record output module. If the output file name is not specified, its name defaults to that of the first input file, plus the MX extension. When Q is specified, the user commands TASK, MONITOR, PRIORITIES, OPTIONS, ATTRIBUTES, and COMLINE may not be used, but the IDENT command may be used to specify identification to the S0 record. Note that this option and the O and R options are mutually exclusive.
R	-R	Create a relocatable object module. This option requests that the relocatable object modules input be combined to create another relocatable object module, rather than an S-record module or an absolute load module. All references between the modules input will be resolved. Only those external references that cannot be resolved among the input modules will be included in the output module. All the external symbol definitions encountered in the input modules will be included in the output module unless an XDEF user command is specified. When the R option is used, an output file name, different from the input file name(s), <u>must</u> be specified on the command line; otherwise, an error will result. Note that this option and the O and Q options are mutually exclusive.
S	-S	When the S option is used on the LINK command line, segments without user-specified starting addresses are allocated sequentially, on page boundaries, after the segment having a user-specified starting address. Allocation occurs in the order segments are defined in SEGMENT commands. For example, when only one segment is given a user-specified starting address, that segment will be allocated at that address, and all remaining segments will be allocated immediately after it. When more than one user-specified starting address is given, segments without user-specified starting addresses are allocated after the segment having the highest user-specified starting address.

OPTION	DEFAULT	FUNCTION
		In the event that no user-specified starting addresses are specified, the S option has no effect. The segments will be allocated sequentially, starting at memory address 0.
		When -S is in effect, segments without user-specified starting addresses are allocated sequentially in the order in which they are encountered by the linker, on a "first-fit" basis.
		This option may be used only if a load module or an S-record module is being created.
U	-U	If any unresolved references exist at the end of pass 1, list the references. Allow the user to specify additional commands to resolve the references. This option is forced off if the command input device is not the user console.
		IMPORTANT: If this option is specified, the linker will not proceed to pass 2 until all unresolved references are resolved.
W=n	W=24	Valid <n> may be 24, 28, or 32. Specify the bit width of the addressable memory space. Some target processors may provide for a 28-bit or 32-bit memory space (e.g., the MC68010 or the MC68020). Use of the W=28 or W=32 option allows the user to create an S-record output module which contains 28-bit or 32-bit addresses. Note that W=28 or W=32 may be used only when the Q option is on.
X	-X	List the external definition directory on the listing file.
Z=n	Z=35	Allocate a stack and heap segment of at least <n>K bytes (1K = 1024). This segment is used by the linker for storage of the symbol table. If the linker aborts with a Pascal runtime abort code of $1008, $1010, or $1011 (see VERSAdos Messages Reference Manual or M68000 Family Resident Pascal User's Manual), it may be possible to perform the link successfully by invoking it with a larger Z option.

7.2.2 User Commands

The linker will accept direct commands when any of the following conditions exist:

a. Input files were not specified on the command line.

b. The A option was specified on the command line.

c. External references were not resolved at the end of pass 1, and the U option was specified on the command line.

The linker prompts for entry of a user command by displaying a right angle bracket (>). After each command entry, the linker issues its prompt until an END, QUIT, or ABORT command is entered. The linker then either continues prompting or returns control to VERSAdos, as appropriate.

The available commands are:

ABORT	All open files are closed and control is returned to VERSAdos.
ATTR[IBUTES] [<list>]	Used only when output file is a load module. Sets up task attributes in resultant load mode. <list> may be zero, any, or all of the following: S = identifies task as a system task. D = if task is aborted, do a task dump. F = assigns logical unit number 8 to task when loaded. P = makes task position independent.
COML[INE] <name>[,<length>]|[(<segment>)]<address>[,<length>]	Specifies where in memory the command line used to invoke the resultant user program will be stored. If linker output will be a relocatable object module, only <name> (previously externally defined in input module) may be specified. If linker output will be a load module, <name> or <address> may be specified. COML may not be used if an S-record module is being created.
DEF[INE] <symbol>,<value>	Places <symbol> name in external symbol definition table of resultant relocatable object module. Ignored if the XDEF command is also given.
END	Signals end of user command entry. During linkers first pass, a search is made for unresolved external references. If found and P option was specified, default libraries are searched. If found and U option was specified, they are displayed and the linker prompts for further user entries. If found and U option was not specified, the linker aborts. If no unresolved references remain, pass 2 processing will begin. The QUIT command also effects these results.
ENTRY <name>|[(<segment>)]<address>	Identifies entry point (beginning execution location) of linker output module. Only <name> may be specified if output will be relocatable object module. If output will be load module or S-record module, <name> or <address> may be specified.

IDENT <name>,<version>,<revision>[,<description>]

Identifies module name, version, and revision, and optionally supplies a description, to a relocatable object module or S-record module. If relocatable object module, this is the name by which it will be referenced in any future linking. If S-record module, the IDENT information is put in the S0 record.

IN[PUT] (filename>[,<modname1>[,<modname2>]...]

Extends or replaces the input file name specification on the command line. The term <modname> represents a module name specified in assembler IDNT, Pascal PROGRAM or SUBPROGRAM, or linker IDENT directives. More than one series of file name and module name(s) can be specified on an IN directive line.

LIB[RARY] <libfile> [<libfile>]...

Allows specifying one or more library files, which will then be searched in the order listed for unresolved external references. LIB may be used in place of the L=option on the command line.

LIST and <filename>|#|#PR[n]
LISTM|U|X

The first form of the LIST command establishes whether, when the second form of the command is used, the listings will be output to a file, to the screen (#), or to a printer. Default is to the screen. LISTM asks for an immediate listing of the load map; LISTU asks for an immediate list of unresolved references; and LISTX asks for an immediate list of external definitions.

MON[ITOR] <name>[,<session number>]

Used to specify name and session number for a monitor task for the task being created (load modules only).

OPT[IONS][M][P]

Two directive options, specified in any order, are available (load modules only). M instructs the linker that a monitor task is specified; P directs the linker to use the monitor of the loading task (propagate the monitor).

PRIO[RITIES] <initial priority>,<limit priority>

Establishes a task's initial priority and its limit priority (load modules only).

QUIT

Ends input of interactive user commands. Works exactly the same as END.

SEG[MENT] <segname>[(<attributes>)]:<sec#>[,<sec#>]...[<start>[,<end>]]

Allows defining a particular segment of the memory management unit when the output or the linker is a load module or an S-record module. Segment names may be up to four characters. Attributes may be any of the following: R=read only segment; L=locally shareable segment; and G=globally shareable segment. One section number or a range of section numbers may be specified. Starting or starting and ending addresses of the segment may be specified. Refer to the M68000 Family Linkage Editor User's Manual for a more detailed explanation of memory allocation.

START <section#>[,<section#>]...<start address>

Defines the starting address at which a particular section or sections, or a range of sections of a segment will be stored in memory. The linker's output must be a load module or an S-record module.

TASK <name>[,<session number>]

When the output load module is to be executed as a "task", this command is given to the linker to give the task a name and, optionally, specify a session number. If TASK is not used, the resulting load module's task name will be the first four characters of its file name -- session number defaults to zero.

XDEF <symbol>[,<symbol>]...

Specifies which externally defined symbols in the input relocatable object module(s) that have already been processed will be externally defined in the relocatable object module the linker is producing. If not used, all externally defined symbols encountered will be defined as XDEF's in the new module.

7.3 LINKER OUTPUT

The following paragraphs describe the listings produced by the linker, and also the formats of the linker's output modules -- relocatable object modules, load modules, and S-record files. Additionally, the format of "debug files" created optionally by the linker, which can then be accessed by VERSAdos' symbolic debugger program, SYMbug, is described.

7.3.1 Listing Types

At the end of pass 1 when unresolved references still exist, at a fatal error, at the end of pass 2, or immediately in response to user listing commands, the linker will print listings. Some of the listings at the end of pass 2, and at fatal errors, occur only as a result of options specified in the LINK command line. The listings are directed to the listing output file or device.

Fatal Errors

When the linker encounters a fatal error and cannot complete the link, the following information is generated: Linker version identification, options in effect, unresolved external references, multiply defined symbols, error count, and a statement that the output module has not been created. The command line entered and any user commands entered are also part of the listings if the I option was specified.

Immediate Listings

The following information is immediately output when the LIST directives are given to the linker:

LISTM = Load map
LISTU = Unresolved external references
LISTX = Externally defined symbols

End of Pass 1

If any unresolved references still exist after pass 1, they are listed.

End of Pass 2

The listing produced always includes: linker version identification, options in effect, unresolved external references, multiply defined symbols, error count, and a message as to whether a new output module has been created or a previous one has been replaced by this module. Depending upon options specified, the following may be part of the listing:

I	= command line and user directives
H	= object module header information
M	= load map
O,Q	= length of segments and module, in bytes
X	= externally defined symbols

7.3.2 Relocatable Object Module Format

Under the VERSAdos operating system, relocatable object modules are stored in sequential files with fixed length records of 256 bytes.

Within each 256-byte record are stored a variable number of variable length relocatable object records. Each one of these records consists of a one-byte byte count followed by the actual data of the relocatable record. The byte count indicates the number of data bytes that follow in that record. The byte count may contain any value between 0 and 255, inclusive. A byte count of zero indicates a relocatable object record with no data bytes (a record of this type is ignored by the linkage editor). Thus, the length of one relocatable object record is limited to a total of 256 bytes -- one byte for the byte count and a maximum of 255 data bytes.

Each 256-byte fixed length record is totally filled before continuing on to the next 256-byte record. Thus, it is possible for a variable length record to be divided between two fixed lengths.

Any space not used in the last 256-byte fixed length record of a relocatable object module file is filled with binary zeroes. This effectively fills out the rest of the file with relocatable object records that have zero bytes of data (which will be ignored by the linkage editor).

There are four basic types of relocatable object records. The type of a record is indicated by the first byte of data (the byte immediately after the byte count) in the record. Record types may be: identification, external symbol definition, object text, or end.

Each relocatable object module must contain an identification record as the first record in the module. It is this record which indicates the beginning of a relocatable object module. The identification record contains general information about the reloctable object module, such as its name, version and revision, what language processor was used to create the module, what source files was used to create the module, the time and date the module was created, and a description of the module.

Each external symbol definition record contains a variable number of external symbol definitions (ESD's), each of which defines a relocatable section, a common section, an absolute section, an externally defined symbol, an externally referenced symbol, or a command line address. The type of an ESD within an external symbol symbol definition record is indicated by a one-byte value at the beginning of the ESD.

Several ESD entries may be included in one ESD record.

In order that they may be easily referenced later in the relocatable object module, each section (relocatable, common, and absolute) and each external reference in a relocatable object module is assigned an index. This index is called an external symbol definition index (ESDID).

ESD's which describe externally defined symbols and command line addresses are not assigned indices. This is because these types of ESD's do not need to be referred to later in the relocatable object module.

ESDID's are assigned anew for each relocatable object module processed.

Object text records define the actual code and data which is to be put in the resulting load module (or relocatable object module). Each object text record contains absolute code along with relocation data for computing relocated code. A bit map is employed to indicate what data is absolute code and what data is relocation data.

An end record indicates the end of a relocatable object module and must be the last record in every module. It also contains information about the starting execution address of the module.

7.3.3 Load Module Format

Under the VERSAdos operating system, load modules are stored in contiguous files. Each load module consists of a header block followed by a variable number of memory image blocks. Each block is 256 bytes long.

The first block in a load module is known as the Loader Information Block (LIB). It is also sometimes called the header block. The LIB contains all the necessary information about the load module except the actual data. The LIB consists of three major sections: the header, the segment allocation descriptors, and the memory image descriptors.

The header part of the LIB occupies the first 48 bytes of the LIB and contains information about the task that is created when the load module is loaded into memory by the VERSAdos loader.

Immediately following the header section of the LIB are the segment allocation descriptors (SAD's). There are eight SAD's where each one occupies 16 bytes. This makes for a total of 128 bytes. The SAD's occupy the 49th through 176th bytes in the LIB. Each SAD describes a memory management unit (MMU) segment that is to be set up when the module is loaded. Currently, a task may have a maximum of four MMU segments allocated to it.

Immediately after the segment allocation descriptors in the loader information block are the memory image descriptors (MID's). There are 20 MID's in a LIB and each MID occupies 4 bytes, which makes for a total of 80 bytes. The MID's occupy the 177th through 256th bytes of the LIB. Each memory image descriptor defines a logical address space in memory into which data in the load module is to be located.

For each memory image descriptor, there is a contiguous block of data in the memory image blocks of the load module which corresponds to the address space defined by the MID. The data corresponding to the MID's appears in the load module in the order in which the MID's appear.

Immediately following the LIB in a load module are a variable number of contiguous memory image blocks. Each memory image block contains 256 bytes of code/data which is to be loaded into memory, without alteration, when the task is loaded. The number of memory image blocks in a load module depends upon the number of memory image descriptors in the LIB and the address spaces defined by them. Note that MID's define memory images in multiples of pages (256 bytes). Therefore, all the data in any given memory image block belongs to one and only one memory descriptor.

7.3.4 S-Record File Format

An S-record file consists of a sequence of specially formatted ASCII character strings. There are several fields within these records in which groups of characters must be interpreted as hexadecimal values of one to four bytes in length. An S-record will be less than or equal to 70 bytes in length. Since each S-record requires 10 to 14 bytes in fixed overhead for the type, byte count, address and checksum fields, the variable length data field may be allocated, at most, 60 bytes. This translates to 60 characters or 30 character pairs or bytes of data per data record from the user viewpoint.

The order of S-records within a file is of no significance, and no particular order may be assumed in the S-record file output by the linker.

S-record types output by the linker may be:

S0	The type of record field is 'S0' ($5330). The address field is unused and will be filled with zeros ($30303030). The header information within the data field is supplied by the user by means of the interactive user command IDENT.
S1	The type of record field is 'S1' ($5331). The address field is interpreted as a two-byte address. The data field is composed of memory loadable data.
S2	The type of record field is 'S2' ($5332). The address field is interpreted as a three-byte address. The data field is composed of memory loadable data.
S3	The type of record field is 'S3' ($5333). The address field is interpreted as a four-byte address. The data field is composed of memory loadable data.

S7 The type of record field is 'S7', 'S8', or 'S9' ($5337, $5338, or
S8 $5339), respectively. The address field contains the starting
S9 execution address specified by the user by means of the interactive user command ENTRY. If no ENTRY command is specified, the first entry point encountered in the object module's input will be used. If no starting address is encountered, the beginning address of the first segment will be used. If none of these methods is used to specify the starting address, this field will be set to zeroes. The address field of the 'S7', 'S8', and 'S9' records is four, three, and two bytes, respectively. There is no data field.

7.3.5 Debug File Format

The format of debug files, used by the symbolic debugger SYMbug, is similar to that of relocatable object module files in that they are stored in sequential files with fixed length records of 256 bytes. Records not completely filled with information are padded with $FF to fill 256 bytes.

A debug file contains information taken from three sources: the relocatable object modules used as input to the linker, their associated .RS files (if they exist), and the load module information block. This information is organized into a .DB header record and one or more additional records per relocatable object module included in the link. The module information is organized into a module header record, zero or more module symbol records, and zero or more module index records per module.

CHAPTER 8

PASCAL COMPILER

8.1 INTRODUCTION

Source programs written in Pascal for the VME/10 are compiled with the M68000 Family Pascal Compiler and then linked with applicable library routines by the M68000 Family Linkage Editor to create an executable load module. They may be linked, also, with other Pascal subprograms and assembly language subroutines.

The Pascal compiler consists of three separate programs which are run sequentially. The first and third programs are required; the second program, which is optional, is an optimizer which reduces code size and increases its efficiency, thereby increasing the speed at which the finished Pascal program executes. The three programs are named PASCAL, POPTIM, and PASCAL2, and are also referred to respectively as Phase 1, Phase 1.5 (or optimizer), and Phase 2. Each program processes its input file in a single pass and generates the input file for the next program.

8.2 SOURCE PROGRAMS

8.2.1 Pascal Source Programs

Various options can be specified from within the Pascal program that affect the compiler's source, listing, and object output, control runtime checks, change stack and heap size, and call for fast floating point arithmetic.

These options are specified in the source file as a Pascal comment, with an additional symbol which informs the compiler that the comment is an "option comment". Two forms may be used:

```
  {$<option>} or (*$<option>*)
```

Options are specified as alphabetic characters, followed immediately by a plus, minus, or equal sign. More than one option can be specified within the same comment, separated by commas but not spaces. The option comments generally may be specified anywhere a comment is normally allowed.

Many of the options may alternatively be specified on the Pascal command lines (paragraph 8.3). The option comment characters and their meanings are:

OPTION	DEFAULT	FUNCTION
A=n	A=4	Specify the number of bytes used for integer arithmetic.
C-	C+	Generate an input file for the optimizer or Phase 2. Eliminating this file reduces the time necessary to generate the listing and any errors.

OPTION	DEFAULT	FUNCTION
D+	D-	This combines the K and R options to (1) generate code to perform runtime checks which verify that array indices and subrange type variables are in range, and (2) include executable unit numbers in the executable object code.
E	none	Page eject for Phase 1 listings.
F=<filename>		Include the file specified by <filename> in the source. Immediately after the line which contains this comment option, Phase 1 will start obtaining its source input from the file indicated by <filename> (which must conform to the rules for specifying a file name for the operating system). When the end of the "include file" is encountered, Phase 1 will return to getting its source from the original source file at the point it left off.
G+	G-	Keep object files output by the compiler or optimizer which contain errors (normally deleted).
H=n	H=4096	Specify the size of the program heap in bytes.
I-	I+	Pass any external files specified on the command line to the program at start-up.
K+	K-	Include executable unit numbers in the executable object code. The executable unit numbers relate to statements and are found on the source listing.
L-	L+	Generate a source listing in the Phase 1 listing file, on the printer, or on the CRT.
O+	O-	Enter source statements as comments in the Phase 2 input.
P+	P-	Include executable unit numbers in the executable object code, but only at function/procedure entry and exit points.
Q+	Q-	Use fast floating point.
R+	R-	Generate code to perform runtime checks which verify that array indices and subrange type variables are in range.
S=n	variable	The value specified by n will be the default stack/heap size in bytes used by the program. If specified, n must be at least 768.
W+	W-	Generate a warning during Phase 1 processing if non-standard Pascal features are used. Standard Pascal comprises only the language features proposed by Jensen and Wirth.

In Motorola Pascal, program's code size and data size are limited only by the amount of memory in the user's system. The size of a component of a file type is limited to 32767 bytes due to the nature of the Pascal input/output utilities.

Motorola extended data types include a dynamic string type which may include character strings up to 32764 characters long. Extended floating point types include double and extended precision reals in addition to the normal precision reals.

String constants are limited to a maximum of 132 characters. Strings are limited to 32766 bytes (32764 bytes of data and a two-byte current length word). Sets are fixed at eight bytes.

The subranges of case statement index expressions and array index expressions may be any subrange which can be expressed using four-byte integers.

8.2.2 Pascal Subprograms

Level one procedure and function declarations may have their declaration and statement parts replaced by the Pascal directive forward, and then have their specific program parts treated as external and compiled separately within a subprogram. The subprograms are linked to the main program when the load module is created by the linker.

The form of a subprogram is similar to the form of a Pascal program. It consists of a subprogram heading and declaration part. However, it does not contain a statement part.

The subprogram heading contains, in the following order: (1) the symbol subprogram, (2) a subprogram name identifier, and (3) a subprogram parameter list. The subprogram parameters should be the same as the program parameters in type, number, and order.

The declaration part of a subprogram consists of the declaration of variables, procedures, and functions, and the definition of constants and types. In order to preserve recognition of global identifiers, all variables in the subprogram variable declaration part must agree in type, number, and order with those appearing in the program's variable declaration part.

Constants and types declared in the Pascal program may be duplicated in a subprogram.

Among the procedures and functions declared in the subprograms are those which were declared as external (forward) in the Pascal program. These externally-referenced procedures and functions must be declared at level one.

The final end statement of the final procedure or function declared in the subprogram is followed by a period (.), which terminates the subprogram.

8.2.3 Assembly Language Subroutines

An assembly language routine may be called externally by a Pascal program using normal Pascal argument passing. Such a routine may perform a function not available in Pascal or a function to be used repetitively in a real-time environment, a shorter and faster routine than might be possible in Pascal.

There are two requirements which must be satisfied in order to include an assembly language subroutine in a Pascal program. The first is to declare the external assembly language routine in the Pascal program. This is done by declaring a level 1 procedure or function, contained in the main program or a subprogram, using the forward directive. These declarations should appear before the first non-external procedure heading.

For example:

```
         FUNCTION   SUMTHREE(I,J,K:INTEGER):INTEGER; FORWARD;
```

The external assembly language subroutine may then be called just as any Pascal procedure or function. Calling an assembly language routine is identical in format -- and its runtime requirements are identical in system usage -- to a regular function or procedure call in Pascal. The call might be:

```
    BEGIN
      A:=SUMTHREE(3,5,7);
```

The second requirement concerns the file which contains the assembly language routine. This file must have an entry point, which has been declared external with an XDEF, with the same name (truncated to 8 characters) as the procedure or function in the Pascal program. The assembler must be informed that the subroutine is to be included in section 9. A 'SECTION 9' directive at the beginning of the assembly language subroutine file accomplishes this.

For example:

```
                XDEF       SUMTHREE
                SECTION    9
    SUMTHREE    EQU        *
```

Control may be returned to the Pascal program by means of either a return from subroutine instruction or a jump indirect through an address register which contains the return address. Refer to the M68000-family Resident Pascal User's Manual for stack requirements of the assembly language subroutine.

After it is assembled, an assembly language routine is linked with a Pascal program and appropriate libraries by means of the linkage editor.

8.2.4 Runtime Libraries

When Pascal programs are linked, they are automatically linked also with the default library of runtime routines, PASCALIB.RO. Libraries of routines written by the user can be created by using the LIB VERSAdos utility, and then linked with the Pascal programs. The name of the user library must be specified to the linker, either on the LINK command line or with the interactive LIB command.

Other Pascal libraries are furnished for use with other Motorola computers, and for use when only the VERSAdos kernel, RMS68K, is used. These are described in the M68000-Family Resident Pascal User's Manual.

8.3 INVOKING THE COMPILER

Following are the command lines used to invoke the three Pascal compiler programs, PASCAL (Phase 1), POPTIM (Phase 1.5), and PASCAL2 (Phase 2), and brief descriptions of their operation. The output of Phase 1 may be input directly to Phase 2, or it may be input to the optimizer, Phase 1.5. Phase 2 accepts the output of either Phase 1 or 1.5, and produces a relocatable module, ready to be linked.

8.3.1 Phase 1 - PASCAL

Phase 1 processes a Pascal source program, checking the syntax of each statement it encounters. If any errors are detected, they are brought to the attention of the user. These errors should be eliminated and Phase 1 should again be invoked to compile the modified program. When no errors are reported, Phase 1 processing is complete. The file output by Phase 1 is an intermediate file. If errors are detected, this file is automatically deleted. A listing of the file can optionally be requested; any errors found are flagged in the listing.

The command line to invoke the PASCAL Phase 1 program is:

```
    PASCAL <source file [,[<output file>][,<list file>][;<options>]
```

More than one source file may be specified, separated by a slash (/) character. If the output and/or list files are not specified, appropriate default filenames will be created, based upon the first input filename. The listing may also be directed to the CRT screen or a printer by specifying # or #PR, respectively, instead of a file name. Options are similar to the source file option comments. Except for the Q option, option comments override command line options. Command line options are:

OPTION	DEFAULT	FUNCTION
C	C	Generate an intermediate code file.
D	-D	Generate runtime range checking code; include executable unit numbers in object code.
E	-E	Disable progress counter updating during compilation.
G	-G	Retain the intermediate code file in the event an error is detected.
I	I	Pass external files from the command line.
K	-K	Include executable unit numbers in object code.
L	L	Generate a Phase 1 listing.
O	-O	Include source statements in Phase 2 input.
P	-P	Include executable unit numbers in object code at function/procedure entry and exit points.

OPTION	DEFAULT	FUNCTION
Q	-Q	Use fast floating point. When using separate compilation, the same state of this option (Q or -Q) must be used with each compilation.
R	-R	Generate runtime range checking code.
W	-W	Warn if non-standard Pascal features are used.
Z=n	Z=40	Set stack/heap (symbol table) size used by this compiler phase to nK. Value of n must be at least 40 (the default value, 40K bytes).

8.3.2 Phase 1.5 -- POPTIM

When an error-free intermediate file is produced by Phase 1, it can be input to POPTIM, the optimizer, at the user's option, or skipped.

POPTIM provides machine-independent optimization of the pseudo-code produced by the compiler by reducing the number of pseudo-codes and providing more information to the machine-dependent code generator about the program in general and about variable usage.

The output file produced by Phase 1.5 becomes, in turn, the input file for Phase 2. However, if the optimizer encounters errors, this file is automatically deleted.

The optimizer is called from VERSAdos as follows:

```
    POPTIM <intermediate file>[,<output file>][;<options>]
```

Options may be:

OPTION	DEFAULT	FUNCTION
E	-E	Disable progress counter updating during optimization.
G	-G	Retain the optimized intermediate code file in the event an error is detected.
O=n	O=1	Perform levels of optimization up to n. The default level is 1. (Levels 2 and 3 are not yet implemented.)
Z=n	Z=32	Set stack/heap size used by this compiler phase to nK. Value of n must be at least 32.

8.3.3 Phase 2 -- PASCAL2

Phase 2 of the compiler processes the intermediate code produced by Phase 1 or the optimized intermediate code produced by Phase 1.5, and generates an object module that can be link edited to create a load module. It then generates, in the form of a relocatable object module, the machine code equivalent of the corresponding group of intermediate instructions. One object module is generated for the entire input file.

Phase 2 is called as follows:

```
    PASCAL2 <intermediate file>[,[<object file>][,<list file>]][;;<options>]
```

Options may be:

OPTION	DEFAULT	FUNCTION
E	-E	Disable progress counter updating during code generation.
G	-G	Retain the relocatable object output file in the event an error is detected.
L	-L	This option enables generation of the listing file specified by <list file>.
J	J	This option causes a JSR to an F-line trap simulator to be generated before each floating point instruction generated when the standard version of floating point is being used (-Q). This is the default condition. Entering -J as a command line option suppresses generation of the JSR's to the F-line trap simulator. If -J is used, the user must supply his own floating point initialization routine, F-line trap handler, and memory access routine at linkage edit time. When using separate compilation, the same state of this option, J or -J, must be used with each compilation.
Z=n	Z=48	Set stack/heap size used by this compiler phase to nK. Value of n must be at least 48.

8.4 COMPILER OUTPUT

The final output file of Phase 2 of the compiler is a relocatable object module file that is compatible with the M68000 Family Linkage Editor. Also produced at this time, if requested with the L option on the PASCAL2 command line, is a "pseudo assembly" listing which facilitates later debugging.

Following is a pseudo assembly listing of lines 55 through 66 of a Pascal program. For comparison, the same lines from the Phase 1 listing of the program are shown below it.

Phase 2 Output (Pseudo Assembly Listing)

```
                                      *             BEGIN                                              55
                                      *                                                                56
                                      *               {read in the numbers, one by one}                57
                                      *                                                                58
                                      *               FOR i := 1 TO array_size DO                      59
00000092-0094 3B7C 0001 B1CA                  MOVE    #1,-20022(A5)
00000098-009A 3B6D B1CE B1C2                  MOVE    -20018(A5),-20030(A5)
0000009E-00A0 60**                            BRA     L6
000000A0-00A4                         L5      EQU     *
                                      *                 BEGIN                                          60
                                      *                    write (output,'Input number ',i:3,': ');    61
000000A0-00A4 486D FFF0                       PEA     -16(A5)
000000A4-00A8 4EAB ****                       JSR     .PLDCS-.PLJSR(A3)
000000A8-00AC 000D                            DC.W    13
000000AA-00AE                                 DC.B    'Input number '
000000B8-00BC 4267                            CLR     -(A7)
000000BA-00BE 4EAB ****                       JSR     .PWRS-.PLJSR(A3)
000000BE-00C2 7203                            MOVEQ   #3,D1
000000C0-00C4 302D B1CA                       MOVE    -20022(A5),D0
                                              XREF    8:.PWRI
000000C4-00C8 4EAB ****                       JSR     .PWRI-.PLJSR(A3)
000000C8-00CC 2F08                            MOVE.L  A0,-(A7)
000000CA-00CE 2F3C 00023A20                   MOVE.L  #145952,-(A7)
000000D0-00D4 4267                            CLR     -(A7)
000000D2-00D6 4EAB ****                       JSR     .PWRS-.PLJSR(A3)
                                      *                    force (output);                             62
000000D6-00DA 2F08                            MOVE.L  A0,-(A7)
000000D8-00DC 4E93                            JSR     (A3)
000000DA-00DE ********                        DC.L    USER1-*
                                      *                    readln (input,number_array[i])              63
                                      *                 END; {FOR}                                     64
000000DE-00E2 302D B1CA                       MOVE    -20022(A5),D0
000000E2-00E6 E540                            ASL     #2,D0
000000E4-00E8 41ED B1CC                       LEA     -20020(A5),A0
000000E8-00EC 43F0 0000                       LEA     0(A0,D0),A1
000000EC-00F0 41ED FFF8                       LEA     -8(A5),A0
                                              XREF    8:.PRDJ
000000F0-00F4 4EAB ****                       JSR     .PRDJ-.PLJSR(A3)
000000F4-00F8 4EAB ****                       JSR     .PRLN-.PLJSR(A3)
000000F8-00FC 526D B1CA                       ADDQ    #1,-20022(A5)
000000FC-0100 69**                            BVS     L7
000000FE-0104                         L6      EQU     *
000000FE-0104 302D B1C2                       MOVE    -20030(A5),D0
00000102-0108 B06D B1CA                       CMP     -20022(A5),D0
00000106-010C 6C**                            BGE     L5
00000108-0110                         L7      EQU     *
                                      *                                                                65
                                      *               {now sort the numbers - use a bubble sort}       66
```

Phase 1 Output

```
55          0)C-          BEGIN
56          0)--
57          0)--            {read in the numbers, one by one}
58          0)--
59        8 0)--            FOR i := 1 TO array_size DO
60          0)D-              BEGIN
61        9 0)--                write (output,'Input number ',i:3,': ');
62       10 0)--                force (output);
63       11 0)--                readln (input,number_array[i])
64          0)-D              END; {FOR}
65          0)--
66          0)--            {now sort the numbers - use a bubble sort}
```

8.4.1 Relocatable Object Modules

The relocatable module file contains information which, when extracted by the linker, makes possible the combination of separate programs and the automatic inclusion of necessary system routines. The location of every level 1 procedure is recorded in the object file in an external definition record. A list of all modules referenced by the program, either explicitly requested by the user or determined by Phase 2 to be needed, is included in an external reference record. An indication of the memory occupied by the program is provided, along with a request for space to be used by the Pascal program for data storage in a stack/heap.

The code itself is also stored in the object module. Phase 2 creates code that is position independent, as well as relocatable. The linking process will preserve the position-independence so that Pascal programs may theoretically be loaded into any memory address space. A special feature of this code is that it includes a pseudo long relative branch facility that enables any instruction to be reached with six bytes of code. Routines obtained from the runtime library may always be reached with a four-byte instruction.

8.4.2 Pseudo Assembly Listing Description

If a Pascal program does not perform as expected, debugging may be necessary. The most convenient way to perform this activity is by including facilities in the program to inform the user of its progress, reporting the values of critical variables at appropriate times. Occasionally it might be desirable to conduct debugging of individual machine instructions rather than source statements. The pseudo assembly listing output at the end of Phase 2 processing greatly facilitates this activity.

This listing contains the following information:

a. Pascal source statements are present if the O option was selected when Phase 1 processing was requested. To the right of the source statement appears a statement number that matches the statement number appearing at the beginning of each line of the Phase 1 listing. This makes it easy to find a specific source statement in the pseudo assembly listing.

b. Between source statements appears a representation of the code that was stored in the object file. This appears in a similar format to that which would be produced by an assembler. Machine code for instructions which cannot be shown in final form (instructions containing forward references and instructions requiring linkage for completion) is indicated by asterisks (**).

c. An assembly language instruction equivalent to the machine code representation appears on the right side of the pseudo assembly listing. This code may serve as a basis for users desiring to modify code generated by Phase 2, but will not, in general, assemble correctly.

d. In certain situations, addresses have not been determined at the time the listing is generated. In the Phase 2 listing, unknown addresses jumped to or branched to are indicated by asterisks. Instruction addresses which are uncertain at this time are shown as ranges in which they will fall -- e.g., 00000054-005C. This uncertainty results from forward references to labels and Phase 2's attempt to reach the label using a short branch. Phase 2 does not know whether a short branch will be adequate until sometime after the pseudo assembly listing has been output.

8.5 LINKAGE

The relocatable object module produced by Phase 2 of the Pascal compiler must then be processed by the M68000 Family Linkage Editor. The output of the linker may be either an executable Pascal program (a load module), a file that can be downloaded to a target system for execution (an S-record module), or another relocatable module, to be linked with other relocatable object modules (Pascal or assembly).

If procedures and functions in a Pascal program are written in assembly languages, these subroutines must be linked with the program.

The linker also links the runtime library, PASCALIB, with the program so that input/output routines called by the program can be accessed.

Pascal supports separate compilations so that the user may group one or more procedures or functions into a subprogram. The linker can combine as many subprograms as desired, and can resolve references between the program and subprogram or between two subprograms. Modules using standard floating point may not be linked with modules using fast floating point.

According to the linker's default processing, memory will be allocated in two segments. Segment SEG1, the program segment, will contain the runtime routines, the Pascal code section, and assembly language routines. Segment SEG2, the data segment, will contain the Runtime Maintenance Area (RMA), Pascal stack/heap, and the Pascal exception vectors.

8.6 LOAD MODULES

When the Pascal program has been finally linked with all necessary subprogram modules, assembly language modules, and libraries into an executable load module, it may then be run by typing its name to VERSAdos.

Filename assignments can be made on the command line when entering the program name, and the Z=n option can be specified on the command line to extend the stack/heap size.

The following command line to VERSAdos associates filenames with file variable specifications on a program statement in a Pascal program named COMPUTE, and enlarges the stack/heap size to 32K bytes:

```
  = COMPUTE MATH.SA,TRIG.SA,ARITH.LS;Z=32
```

where the program statement was:

```
  program compute(input,output,source,object,listing)
```

The input and output file specifications required by Pascal ("I=" and "O=") were omitted on the command line, as they default to the user's terminal (I=#, O=#).

8.7 EXAMPLES

Following is an example of creating an absolute load module using the Pascal compiler and the linkage editor.

a. At the VERSAdos prompt (=), invoke the CRT text editor by entering the following:

```
=E PROGNAME
```

PROGNAME represents the name of the source file being created; .SA is the default extension, and is usually used for ASCII source files.

After the user program has been entered, press the F1 function key to return the editor prompt (>) to the lower left portion of the screen, and exit the editor as follows:

```
>QUIT
```

b. Compile the program into a relocatable object file and link the file with other files called from the Pascal library to create an absolute load module. Each Pascal phase (1, optional 1.5, and 2) and the linkage editor can be called separately, or the user can create a chain file that will automatically execute the two (or three) phases of Pascal and the linking function to create the absolute load module, as in the following example.

First, call the editor and create a new file:

```
=E CHAINPAS.CF
```

Then enter the following command lines into the edit file. The first three lines call Phase 1, Phase 2, and the linker, in that order. The Pascal library, PASCALIB, is linked by default.

```
>=PASCAL \1
>=PASCAL2 \1
>=LINK \1
>=END
```

NOTE

The VERSAdos prompt (=) is required on a chain file command line which calls a utility.

Exit the editor by pressing the F1 key and typing:

```
>QUIT
```

c. The chain file can be called for compilation and linking by entering the following:

```
=CHAINPAS.CF PROGNAME
```

Note that in this example, the program name is specified on the CHAIN command line, rather than within the chain file.

d. When the execution of Pascal, Phase 1, is completed, the number of errors is indicated.

e. The following events occur during execution of the chain file: Phase 1 creates an intermediate code file, PROGNAME.PC, and a listing file, PROGNAME.PL; Phase 2 creates the relocatable object module, PROGNAME.RO; the linker creates the absolute load module, PROGNAME.LO.

   The user then has the option of either listing the program on the screen or printing hard copy to investigate and correct errors.

   1. To list the program on the screen, enter:

      =<u>LIST PROGNAME.PL</u>

      The entire program will scroll on the screen (line by line) and the compilation errors will be indicated as they occurred. To stop the scrolling in order to investigate an error, press CTRL-W (hold the CTRL key, then press the W key). To continue scrolling, press any key.

   2. To list the program on the printer, enter:

      =<u>COPY PROGNAME.PL,#PR</u>

      The entire program will be printed, indicating the total number of errors and also where each error appeared in the program listing.

f. Use the text editor to correct the errors, then call the chain file again. When no compilation errors exist, a valid absolute load module will automatically be created, provided the linkage editor encounters no problems.

g. The absolute load module is now ready to be examined or modified using either TENbug or DEbug. To use DEbug, refer to the SYMbug/A and DEbug Monitors Reference Manual for the procedure to load an absolute load module into memory. To use TENbug, refer to the BO command in the TENbug Debugging Package User's Manual for the procedure to load an absolute load module into memory.

h. A completely debugged load module can be executed by entering only its name to the VERSAdos prompt (=):

      =<u>PROGNAME</u>

CHAPTER 9

DEBUG CAPABILITY

9.1 INTRODUCTION

A load module often requires debugging to overcome deficiencies which come to light when the program runs in an actual application. Supplied with VERSAdos are two debug monitor programs -- DEbug and SYMbug. In addition to these, a firmware-resident debug monitor program, TENbug, is supplied in the ROM of the VME/10 System.

9.2 TENbug

TENbug is the resident firmware monitor and debugging package for the VME/10. The 16K-byte firmware (stored in two 8Kx8 ROM or EPROM devices) provides a self-contained programming and operating environment. TENbug may be entered directly at system power-up or from VERSAdos. These two methods are given in Chapter 2.

TENbug interacts with the user through predefined commands that are entered via the terminal. The commands fall into five general categories:

a. Commands which allow the display or modification of memory.

b. Commands which allow the display or modification of the various internal registers of the MC68010.

c. Commands which allow execution of a program under various levels of control.

d. Commands which control access to the various input/output resources on the board.

e. Commands which allow selection of video graphics resolution.

An additional function called the TRAP #15 I/O handler allows the user program to utilize various routines within TENbug.

For complete information on TENbug, refer to the TENbug Debugging Package User's Manual, M68KTENBG.

9.2.1 Command Set

TENbug's debugging functions are performed in response to the entering of simple "primitive" commands, with or without associated parameters and options. Several of the commands are set and reset pairs -- the reset function is specified by preceding the command with NO. The entry of a command line is always followed by pressing the carriage return key (<--'). TENbug checks each entry for validity, returning an error message if incorrect, or processing the command and displaying an interpretation of the parameter values if correctly entered.

Table 9-1 lists the primitive commands supported.

TABLE 9-1. TENbug Commands by Type

COMMAND MNEMONIC	DESCRIPTION
MD	Memory display
MM	Memory modify
.A0-.A7	Display/set address register
.D0-.D7	Display/set data register
.DFC	Display/set destination function code
.PC	Display/set program counter
.SFC	Display/set source function code
.SR	Display/set status register
.SSP	Display/set supervisor stack pointer
.USP	Display/set user stack pointer
.VBR	Display/set vector base register
DF	Display formatted registers
.R0-.R6	Display/set relative offset register
OF	Display offsets
BR	Breakpoint set
NOBR	Remove breakpoint
GO	Execute program
GT	Go until breakpoint
GD	Go direct execute program
TR	Trace
TT	Trace to temporary breakpoint
PA	Printer attach
NOPA	Detach printer
BH	Bootstrap halt
BO	Bootstrap operating system
VM	Video map

9.2.2 TENbug Examples

The following example assumes that the system has been initialized and the furnished software has been backed up, as directed in Chapter 2.

a. Ensure that the KYBD LOCK key switch on the VME/10 chassis is in the unlocked (horizontal) position.

b. Press the on/off switch to the "1" position and wait for the hard disk to spin up.

c. Press and release the RESET pushbutton on the VME/10 chassis. TENbug will take control and display its prompt (if system includes an MVME400, press any key on the keyboard after pressing RESET, to get the full prompt):

```
TENbug x.y >
```

d. Display and alter the contents of MC68010 registers by typing in the commands shown underscored (underscore is not to be typed), following each entry with a carriage return. (The initial register values displayed will differ from these.)

```
TENbug x.y > DF                     (Display formatted registers)
PC=00F02C9E SR=2704=.S7..Z.. USP=FFFFFFFF SSP=000007C4 VBR=00000000 SFC=2 DFC=2
D0-7 00300030 00000804 00000000 00000000 4D505520 00000020 00000000 00000000
A0-7 00F1A031 00F0133C 00F008AA 00000458 0000049A 00000536 00000536 000007C4
  PC=F02C9E

TENbug x.y > .R1 3000               (Set register R1 offset to 3000)

TENbug x.y > OF                     (Display offset registers)
R0-7 00000000 00003000 00000000 00000000 00000000  0000000 00000000 00000000

TENbug x.y > .PC 40000              (Change value in program counter)

TENbug x.y > .SSP C00               (Set supervisor stack pointer)

TENbug x.y > DF                     (Display formatted registers)
PC=00040000 SR=2704=.S7..Z.. USP=FFFFFFFF SSP=00000C00 VBR=00000000 SFC=2 DFC=2
D0-7 00300030 00000804 00000000 00000000 4D505520 00000020 00000000 00000000
A0-7 00F1A031 00F0133C 00F008AA 00000458 0000049A 00000536 00000536 00000C00
  PC=03D000+R1

TENbug x.y >
```

e. Return to VERSAdos with the boot command:

```
TENbug x.y > BO
=
```

9.3 DEbug

DEbug is a VERSAdos-resident monitor program, used to debug other programs whose source code is written in assembly language for execution on the MC68010. The language processor and the linkage editor suppy information to the DEbug monitor.

DEbug allows the user to examine, insert, and modify program elements such as instructions, numeric values, and coded data.

Execution can be controlled by DEbug, via the insertion of breakpoints into a program.

DEbug uses an extensive set of primitive commands for manipulation and examination of foreground tasks. A set of task-level commands may be used on foreground or background tasks and are applicable to both the single and multitasking modes of operation.

9.3.1 Command Line

The DEbug program is invoked as follows:

```
  = DEBUG [<program name>]
```

Specifying the name of the load module to be debugged enters single task mode. The first four letters of the program name are then included in the DEbug prompt. Typing DEbug without a program name enters multitask mode. The maximum number of tasks to be debugged must then be specified, and either the LOAD or ATTA command must be used before any of DEbug's primitive commands can be used.

9.3.2 Primitive Commands

Table 9-2 lists DEbug's primitive commands.

TABLE 9-2 DEbug Primitive Commands

COMMAND SYNTAX	DESCRIPTION
AS[<address>] [<value>l	Address stop
[NO]BR[<address>]...	Set/reset breakpoint
DE	Default to attach/detach printer
DF	Display format
G[O]	Execute target task
HE[LP]	Display commands
MD <address> [<count>]	Memory display
MS <address> <byte 1> [<byte 2> <byte 3>]...	Memory set
OF [<register) <value>]	Offset
Q[UIT]	Quit (go to VERSAdos)
T[R] [<count>]	Trace target task
ATTA <task name>[m<terminal>|#*]	Attach task
DETA [<task name>]	Detach task
EVEN [<task name>],<exception #>	Event definition
LOAD <file name>[<command line>]	Load (task)
MASK [<task name>],<exception #>	Mask exception
STAR [<task name>] |ALL]	Start task(s)
STAT [<task name>],<status>]	Status definition
STOP [<task name>] |ALL]	Stop task(s)
TASK <task name>[,<note level>]	Task notify
TERM <task name>	Terminate task
WAIT	Wait task
.A0-.A7	Display/change address register
.D0-.D7	Display/change data register
.MC	Display/change maximum count (software register)
.OP	Display/change execution options (software register)
.PC	Display/change program counter
.SR	Display/change status register
.ST	Display/change task state
.VA	Display/change value (software register)
.VL	Display/change value location (software register)
.VM	Display/change value mask (software register)
.XM	Display/change execption mask
BREAK	Abort command
CTRL-S	Redisplay line
CTRL-H	Delete character
CTRL-W	Suspend output (See NOTE)
CTRL-X	Cancel command line
CR (Carriage Return)	Send line for execution

NOTE: After CTRL-W has been used, the entry of any character will cause the output display to continue.

9.4 SYMbug/A

SYMbug/A, referred to here as SYMbug, is a VERSAdos-resident multitasking utility that allows a user to debug application program(s) in terms close to the actual program itself. That is, unlike other debuggers that allow only absolute memory accesses, SYMbug generates information about the program that is available to the user during debug. Information is kept concerning assembler symbol names, module names, and section numbers. SYMbug will automatically evaluate this type of symbolic information to absolute addresses for user. It is not necessary to reference a current link map to debug a program. Instead, knowledge of module names and symbols is sufficient to calculate relative offsets and debug the program by reference to an assembler listing. Without the overhead of user responsible address resolution the task of debugging a program becomes faster and easier with a reduction in the chance for error.

To utilize the symbolic referencing capability of SYMbug, a relocatable symbol file (RS extension) is created during assembly by specifying the D option. The RS file is then changed into a debug file (DB extension) during linking by specifying the D option. This debug file is in optimized form to increase the symbolic referencing speed of SYMbug.

At present, the Pascal compiler does not provide the option of creating a relocatable symbol (RS extension) file for input to the linkage editor. Therefore, the symbolic referencing capabilities of SYMbug cannot be used to access a point in a compiled module represented by a label in the source file. However, provided the D option was set during assembly and linkage, symbolic referencing can be used to access symbolic locations within <u>assembled</u> modules of a load module. Access to relative offsets within <u>compiled modules</u> is also provided.

SYMbug's functional kernel is DEbug. SYMbug interfaces with the VERSAdos operating system to provide complete debug control to the user. User interface is via a powerful set of "primitive" commands. These commands allow the user to:

a. Examine/modify registers and absolute and program relative memory addresses specified in a number of ways:

   . Directly
   . In an expression
   . As an effective address
   . Symbolically
     (also allows control of display/modification formats)

b. Control program execution by allowing the user to:

   . Insert breakpoints into the program
   . Trace program execution
   . Monitor data changes

c. Direct multitasking functions by allowing the user to:

   . Modify task scheduling/information handling
   . Modify task attributes/status

d. Expand debugger functions through user generation of:

   . User "macros" built of a series of primitive commands
   . In line command/command block repeat functions
   . Default input/output format modifications

e. Access information outside of SYMbug so that the user may save and restore previously defined information:

   . Save and load program(s) to and from disk
   . Save and load symbolic information (macro names/local symbols) to and from disk
   . Generate debug session echo to printer

SYMbug error messages are informative and precise. The SYMbug HELP command can be used to display a brief syntax summary for all commands.

9.4.1 Symbol Table Creation

In order to use SYMbug as a symbolic debugger, a symbol table must have been created using the assembler's and the linker's D options when assembling and linking each module.

For example, the program created in Chapter 6 might have been assembled as follows:

```
 = ASM PROGNAME;D
```

Three files are created -- the relocatable object module PROGNAME.RO, the listing file PROGNAME.LS, and the relocatable symbol file, PROGNAME.RS.

The subprogram would also have to be assembled with the D option:

```
 = ASM SUBPROG;D
```

Again, three files are created -- all with the same filename but with the extension of RO, LS, and RS, respectively.

The files are then linked to create an executable load module. The D option must be used on the linker's command line:

```
   = LINK PROGNAME/SUBPROG;D
```

The linker's output consists of a load module named PROGNAME.LO, and a symbolic debug file named PROGNAME.DB. This file is dependent upon the contents of the RS files created during assembly, and contains the absolute address specifications for the modules.

SYMbug may alternatively be used as an absolute debugger, similar to DEbug, if the D option was not used when assembling or linking.

9.4.2 Command Line

To use as a symbolic debugger, SYMbug is called as follows:

```
  = SYMBUG [<filename>[<arguments>]]
```

If the name of the file to be debugged is not given on the command line, it can be loaded with the LOAD command after SYMbug is running. <arguments> are any that are allowable for the file being debugged. For example, a new file-copying program named CPY.LO might be loaded as a task performing a copying operation:

```
  = SYMBUG CPY FILEA.SA,FL6:21.CAT1.FILEA.SA;B
```

SYMbug automatically locates the corresponding symbolic debug file.

9.4.3 SYMbug Commands

SYMbug primitive commands are listed in the following table.

TABLE 9-3. SYMbug Primitive Commands

COMMAND SYNTAX	DESCRIPTION
AS [<address> [<value>[;<mask>]]]	Address stop
BF <address1> <address2> <data>[;<length>]	Block fill
BM <address1> <address2> <address3>	Block move
[NO]BR [<address>[;<count>]]...	Set/reset breakpoint
BS <address1> <address2> <data>	Block search
CR [<count>]	Command repeat
DC <expression>	Define constant or Data convert
DE [<default option>]	Defaults
DF	Display formatted registers
FR <file name>	File read
FS <file name>	File save
G[O] [<address>]	Go (execute)
HE[LP] [<command>]	Display commands
[NO]IT <address1> <address2>	Set/reset inside trace
[NO]MA [<name>]...	Set/reset macro define
MD <address> [<count>][;<option>]	Memory display
MM <address>[;<option>]	Memory modify
MS <address> <data>	Memory set
OF	Display Offset register
[NO]OT <address1> <address2>	Set/reset outside trace
Q[UIT]	Quit (go to VERSAdos)
[NO]SD [<local> [<value>]]	Set/reset symbol define
T[R] [<count>]	Trace
ATTA <task name>[,<terminal>|#*]	Attach task
DETA [<task name>]	Detach task
EVEN [<task name>],<exception #>	Event definition
LOAD <file> [<command line>]	Load (task)
MASK [<task name>],<exception #>	Mask exception
STAR [<task name>|ALL]	Start task(s)
STAT [<task name>,<status>]	Status definition
STOP [<task name>|ALL]	Stop task(s)
TASK <task name>[,<note level>]	Task notify
TERM <task name>	Terminate task
WAIT	Wait task
BREAK	Abort command
CTRL-S	Redisplay line
CTRL-H	Delete character
CTRL-W	Suspend output (See NOTE'
CTRL-X	Cancel command line
CR (Carriage Return)	Send line for execution

NOTE: After CTRL-W has been used, the entry of any character will cause the output display to continue.

SUGGESTION/PROBLEM REPORT

MICROSYSTEMS
QUALITY • PEOPLE • PERFORMANCE

Motorola welcomes your comments on its products and publications. Please use this form.

To: Motorola Inc.
Microsystems
2900 S. Diablo Way
Tempe, Arizona 85282
Attention: Publications Manager
Maildrop DW164

Product: ______________________ Manual: ______________________

COMMENTS: ____

____

____

____

____

____

____

____

____

____

____

____

Please Print

Name ______________________ Title ______________________

Company ______________________ Division ______________________

Street ______________________ Mail Drop __________ Phone __________

City ______________________ State ______________ Zip __________

For Additional Motorola Publications
Literature Distribution Center
616 West 24th Street
Tempe, AZ 85282
(602) 994-6561

Four Phase/Motorola Customer Support, Tempe Operations
(800) 528-1908
(602) 438-3100

MOTOROLA