SAGE COMPUTER

Technical Manual

SAGE COMPUTER
4905 Energy Way
Reno, Nv. 89502
(702) 322-6868

December 1983

Table Of Contents

I INTRODUCTION 1

II SAGE HARDWARE 4
II.01 THE PROCESSOR 4
II.02 PROCESSOR LIGHT 5
II.03 MEMORY 6
RAM Chip Addresses 9
II.04 PROM STRAPPING OPTIONS 10
II.05 REAL-TIME CLOCK 10
II.06 POWER SUPPLY 11
II.07 PHYSICAL CHARACTERISTICS 12
II.08 COOLING AND ENVIRONMENT 12
II.09 EXPANSION BUS 15
II.10 I/O ADDRESSING 17
II.11 I/O PORTS 17
II.12 MODEM PORT 19
II.13 SERIAL PORTS 22
Serial Channels 26
II.14 FLOPPY DRIVES 28
Referencing floppy drives 28
Floppy Disk Step Times 29
Writing 40 Track Diskettes 30
Floppy Drive Parameters 31
II.15 WINCHESTER DISK 32

III DEVICES, CHANNEL MAPS & PARTITIONS 34
III.01 DEVICE NUMBERS 34
III.02 THE BIOS CHANNEL MAP 34
Physical Device Numbers 35

TABLE OF CONTENTS

p-System Logical Device Numbers 36
Typical p-System Channel Map 38
Auxiliary Channel Device Numbers 39
III.03 WINCHESTER PARTITION OVERVIEW 40
III.04 USING WINCHESTER PARTITIONS 42
Partition Numbering Scheme 42
Factory Installed Partitions 44
Bootstrapping To Partitions 46
Configuring the Bios Channel Map 48
III.05 PLANNING DISK PARTITIONS 50
III.06 PARTITION USAGE CONSIDERATIONS 51
Partition Names 52
Partition Size 52
Worksheet Example 54
III.07 INSTALLING WINCHESTER PARTITIONS 54
FORMATINFO and Partitions 55
WFORMAT and Partitions 55
Installation Example 56
Installing the Partition Map 56
Installing the System files 60

IV **WFORMAT - Winchester Utilities** 62
IV.01 THE BUILD DISKETTE 63
IV.02 THE FORMAT INFORMATION FILE 64
IV.03 FORMATTING AND VERIFICATION 69
IV.04 BAD TRACK MAPPING 71
IV.05 RUNNING WFORMAT 73
IV.06 RECOVERING TRACKS 80

V **p-SYSTEM OPERATING SYSTEM** 81
V.01 FILES REQUIRED FOR BOOTING 82

V.02 THE USER MASTER DISKETTE 84
V.03 REAL ARITHMETIC PRECISION 84
Interpreter Files 85
Realops Files 85
Pascal Files 86
FORTRAN Files 87
BASIC Files 88
V.04 RAM DISK OPERATION 89
MU RAM Disk Operation 91
V.05 p-SYSTEM DEVICES SUPPORTED 92
V.06 p-SYSTEM BREAK 94
V.07 ASYNCHRONOUS I/O 96
V.08 p-SYSTEM WAITER TASK 98
V.09 EXTRA UNITREAD & UNITWRITE INFORMATION 99
CONTROL Parameters 99
Floppy Formatting 100
Remote Channel Control Bits 100
Winchester Formatting 101
V.10 EXTRA UNITSTATUS INFORMATION 101
SAGE UNITSTATUS Information 102
V.11 BIOS CONFIGURATION 104
V.12 SYSTEM CLOCK ACCESS 105
V.13 GENERAL MEMORY ACCESS 108
V.14 SYSTEM MEMORY ALLOCATION 109
V.15 NOTES ON PORTABILITY 109
V.16 CP/M-68K FILES & p-SYSTEM 111
V.17 p-System FILE DESCRIPTIONS 112

VI **SAGE TOOL KIT** 125
VI.01 UNITS 126

TABLE OF CONTENTS

UNIT Code Placement 127
VI.02 STRING I/O UTILITY UNIT 129
VI.03 TIME AND DATE UNIT 135
VI.04 SAGE DATE SETTING UTILITY 140
VI.05 ATTACH IMPLEMENTATION 143
Sleep Request 151
ATTACH/EVENT Numbers 153
VI.06 MNU_Unit - Menu Unit 154
Menu Definition 156
Menu Operation 158
VI.07 CONFIG_SAGE -Configuration Control Unit 163
Record Descriptions 171
Terminal Configuration 172
Remote Configuration 173
Parallel Printer Channel Config. 175
Floppy Configuration 176
RamDisk Configuration 178
System Configuration 179
Operating System Data 179
BIOS Channel Map 179
Winchester Configuration 180
CP/M Device Information 185
BIOS File 185
Procedure Definitions 186
Configuration Control 188
VI.08 WIN_UNIT -Winchester Unit 189
VII **THE IEEE-488 SUPPORT** 191
VII.01 GENERAL BUS OPERATION 191
VII.02 IB_UNIT 193

VII.03 STARTING UP 194
VII.04 IEEE 488 SWITCH OPTIONS: 196
VII.05 BUS INITIALIZATION 198
VII.06 DEVICE DEFINITION 200
VII.07 TALKING 201
VII.08 LISTENING 202
VII.09 USER BUFFER 204
VII.10 SERVICE REQUESTS 205
VII.11 DIRECT REGISTER CONTROL 206
VII.12 PROGRAM EXAMPLE 207
VII.13 BUILDING A USER PROGRAM 210
Example User Program 211
VII.14 IB_BUS DESCRIPTION 212
VII.15 ERROR CODES 220
VII.16 IEEE-488 CONNECTOR 221

VIII **COMPUTER INTERCOMMUNICATION** 223
VIII.01 BACK-TO-BACK COMMUNICATION 224
VIII.02 HARDWARE INTERCONNECTION 224
VIII.03 CHANNEL CONFIGURATION 225
VIII.04 COMMUNICATION HANDSHAKING 226
VIII.05 REMINTEST AND REMOUTTEST 228
VIII.06 SEND AND RECEIVE 229
VIII.07 TEXTIN 231
VIII.08 REMTALK 233
VIII.09 TELETALKER 234
Teletalker Commands 235
Option Menu: 236
Go Option 236
Send Option 237

Record Option 238
Exit Option 241
Break Option 241
Thawed Option 242
7 Option 242
8 Option 242
Lost Carrier 243
Error on Disk Write 243
TELETALKER on Other MODEMS 244
Remote Unit 244
General Operations 253

IX THE SAGE MULTI-USER SYSTEM 254
IX.01 MULTI-USER OVERVIEW 255
The System Manager 255
User Tasks 256
Time Slice and Priority 257
Multiple Operating Systems 258
Static Allocation of Resources 259
Shared Data Access 260
IX.02 RUNNING THE MULTI-USER SYSTEM 262
Terminal Considerations 262
Terminal Baud Rates 262
Terminal Cables 263
Sharing A Terminal 263
Multi-User System Files 264
MU Booting Specifics 267
SAGE II Startup Environment 268
SAGE IV Startup Environment 270
MU Booting Sequence 271

IX.03 BOOTSTRAPPING OPTIONS 272
BOOT Message 273
Initializing RAM Disk 273
Booting Users from RAM Disk 274
Multiple Users on one Device 275
BIOS and User(s) from same Device 277
IX.04 RUNNING THE p-SYSTEM 278

X **MU.UTIL - Multi-User UTILITY** 279
X.01 MENU OPERATION 280
X.02 MU OPERATION CHECKS 282
X.03 MULTI-USER CONFIGURATION 283
Configuring the User Tasks 284
X.04 THE BIOS CHANNEL MAP 285
User Capabilities 287
CP/M Information 288
X.05 CP/M DISK DRIVE CONFIGURATION 289
Operating System Information 290
The Boot Device Parameter 292
Boot Message 292
Boot Control Delay 292
The Shared Terminal Mode 293
The Priority Parameter 294
Number of Comm Buffers 294
Base Memory Addresses 294
The Time Slice Parameter 295
Capability Mask 295
X.06 SERIAL CHANNEL SELECTION 296
Baud Rate 297
Parity 298

Stop bit 299
Data bits 299
Xmit and Receive Buffer Length 300
Type of Terminal 300
Char to Change User 301
BREAK to reboot 302
BREAK to debug 303
Remote Channel 304
XON/XOFF 305
Counts for sending XOFF/XON 305
DSR Polling & Interval 306
Input & Output event numbers 306
Access Control 306
X.07 FLOPPY CONFIGURATION 307
80 vs 40 TRACK DRIVES 307
IBM Format 308
NCI Format 308
SAGE 10 SECTOR 308
Softech Universal Media 308
X.08 RAM DISK CONFIGURATION 309
Initialize RAM Disk Flag 310
X.09 PARALLEL PORT CONFIGURATION 310
Printer mode 310
X.10 WINCHESTER ACCESS 311
X.11 TIME ADJUSTMENT 312
X.12 LOW LEVEL CONFIGURATION 312
Floppy Configuration 313
Shared Floppy Configuration 317
Winchesters Configuration 318
Number of Semaphores 319

Intercept Exceptions 319
Load Terminal Emulator 319
X.13 AUXILIARY DEVICE INFORMATION 322
Winchester Aux Device Info 324
XI **MULTI-USER SYSTEM CONSIDERATIONS** 325
XI.01 USER CHANNEL MAP 325
XI.02 DEVICE ACCESS CONTROL 325
Access Philosophy 326
Access Control Implementation 328
Printer access 331
Floppy Disk Access 332
Winchester Partition Access 332
XI.03 MEMORY ASSIGNMENT 333
XI.04 RAM DISK MEMORY 336
XI.05 MULTI-USER RECOMMENDATIONS 336
Preferred Operation 336
Security 337
XI.06 STANDARD CONFIGURATIONS 338
Standard MU Installations 338
XI.07 DEVICE VISIBILITY DURING BUILD 346
XI.08 MULTI-USER INSTALLATION EXAMPLE 347
Example Environment 348
Total Disk Size 349
Example Partitions 350
Example Assignment of Devices 352
Booting the BUILD disk 354
Example FORMATINFO file 355
Loading partitions 359
Example CONFIGURATION file 362

Users #2 - #5 373
Example of Serial Configuration 376
Floppy configuration example 378
Example of RAM Disk Config 379
Example of Printer Configuration 380
Winchester Access Example 380
Saving the Configuration 383
Zeroing Disk Partitions 384
Bootstrap Load example 385
File load example 386
Multi-User Boot example 389
After installation 389

XII **ADVANCED MU FEATURES** 390
XII.01 RELEASE OF TIME SLICE 390
XII.02 PROGRAMMED DEVICE CONTROL 390
Control Information Block 393
XII.03 GLOBAL SEMAPHORES 395
Semaphore Philosophy 395
Semaphore Implementation 397
Semaphore Functions 399
Semaphore Lockup Protection 403
Semaphore Example 404
XII.04 LOGON HOOKS 404
Read/Write of Boot Device Number 406
Read/Write Capability Mask 406
Reboot User Task 409
XII.05 USER INTERCOMMUNICATION 410
Transmit Queue Detail 411
UNITWRITE Detail 412

UNITREAD Detail 414
UNITSTATUS Detail 415
General Queue Information 416
Read Information: 416
Write Information: 417
UNITCLEAR Detail 418
User Intercommunication Example 418
XII.06 TERMINAL EMULATOR CONFIGURATION 420
The SAGE Terminal Emulator 420
MUTRMSET Overview 421
Running MUTRMSET 421
MUTRMSET Submenus 424
Handling Screen Attributes 440

XIII BIOS INTERFACE 443
XIII.01 PHYSICAL DEVICE NUMBERS 444
XIII.02 BIOS MEMORY LOCATION 445
XIII.03 SHORT CALLING SEQUENCE 446
XIII.04 LONG CALLING SEQUENCE 449
BIOS Functions 450
XIII.05 TECHNICAL NOTES 470
XIII.06 CHANNEL CONFIGURATION CONTROL 471
XIII.07 BIOS CHANNEL ERROR CODES 481
XIII.08 BIOS NOTES 483
Preparing a BIOS 484
BIOS Configuration 484
BIOS Data Storage 484
BIOS Software Interrupt 485
The p-System Bootstrap 486

APPENDIX A: SWITCH SETTINGS 488

TABLE OF CONTENTS

APPENDIX B: FLOPPY DISK BOOT ERRORS 490

APPENDIX C: WINCHESTER BOOT ERRORS 490

APPENDIX D: BIOS PHYSICAL DEVICE NUMBERS 491

APPENDIX E: SHORT CALLS TO BIOS 492

APPENDIX F: LONG CALLS TO BIOS 493

APPENDIX G: FLOPPY ERROR CODES 495

APPENDIX H: WINCHESTER ERROR CODES 495

APPENDIX I: SAGE I/O PARTITIONS 496

APPENDIX I: ASCII CODES 497

INDEX 498

I INTRODUCTION :

The SAGE II/IV TECHNICAL MANUAL provides information that is specific to SAGE II/IV hardware and software. It is intended as a reference guide for moderately technical users and is not a tutorial. If you are a new SAGE user, we recommend that you read the GETTING STARTED manual before looking here. The GETTING STARTED manual will introduce you to the "language" we use to discuss our computer. The Glossary in that manual should be referred to whenever you are in doubt about technical terms. This manual is organized as a collection of topics and is not meant to be read cover-to-cover. To find information about a particular topic, an INDEX is provided at the back of the manual.

This manual also makes reference to other technical manuals available from SAGE:

The **p-System OPERATING SYSTEM MANUAL** documents how the p-System Filer, Editor and other Utilities work.

The **p-System PROGRAM DEVELOPMENT MANUAL** is a technical manual describing specifics of the Softech p-System for programmers and computer specialists.

The **p-System INTERNAL ARCHITECTURE MANUAL** defines how data and programs are stored and accessed by the operating system.

The **SDT/ASSEMBLER MANUAL** documents the operation of the SAGE II/IV PROMs, SAGE DEBUGGING TOOL and the 68000 p-System assembler.

Personal Computing with the UCSD p-SYSTEM by Mark Overgaard and Stan Stringfellow.

Other sources of information available to the SAGE user are:

The UCSD Pascal Handbook by Randy Clark and Stephen Koehler. This book is a guide to the Pascal language as implemented under the p-System.

USUS is an organization of p-System users. They hold regional meetings twice a year and give tutorials in the Pascal language and operation of the p-System. Useful and interesting software packages are in the SOFTWARE library which is available to all members. Members communicate mostly by electronic mail such as TELEMAIL or COMPUSERVE. The p-System OPERATING SYSTEM MANUAL Appendix F page a-18 gives more information and tells how to join USUS.

Some of the important topics covered by this manual are:

SAGE HARDWARE

The major hardware units of the system are described briefly. For more information, refer to the SAGE II/IV SERVICE MANUAL.

WFORMAT & FORMATINFO

WFORMAT formats the Winchester drives and sets up bad track information. It also allows the drives to be divided into partition information.

p-SYSTEM OPERATING SYSTEM

Specific information of how the p-SYSTEM is implemented on the SAGE II/IV is given.

CONFIG_SAGE

CONFIG_SAGE is a PASCAL UNIT which allows access to the configuration. Application programs can use this unit to do all of the configuration functions of SAGE4UTIL under program control. The unit resides in SAGETOOLS.

MULTI-USER INSTALLATION

Details are given on how to install your own special multi-user configuration. The quick installation of standard configurations is given in the GETTING STARTED MANUAL.

BIOS

Details of the SAGE II/IV BIOS (system I/O) are explained in this section.

II SAGE HARDWARE :

II.01 THE PROCESSOR :

The SAGE II/IV processor is an 8mhz , interrupt driven 68000 microprocessor operating at 2 million instructions per second (without wait states). It is a 32-bit machine with a 16-bit data bus and a 24-bit address bus. It will directly address 16 million bytes. It has 17 general purpose registers, each 32 bits long, a 24-bit program counter and a 16-bit status register. The instruction set contains 56 instruction types with 14 different addressing modes. There are 5 data types, bits, BCD, 8-bit bytes,16-bit words and 32- bit long words. The combination of all these means that there are more than 1000 executable 68000 instructions.

The 68000 is a well-designed processor. Particular emphasis was given to the architecture to make it regular with respect to the registers, instruction types and data types. A consistent structure makes the processor easy to learn and program. It reduces the time required to write a program and the space needed for the program.

Running on an 8 Mhz clock, the MC68000 achieves close to 2 million instructions per sec. Sage II dynamic memory refresh consumes only about 3 percent of processing time.

High level language-oriented instructions such as LINK, UNLK, CHK, etc. ease the implementation of powerful high-level languages such as PASCAL.

II.02 PROCESSOR LIGHT :

A multi-color LED on the front panel indicates the current state of the processor:

GREEN - bus active
RED - bus inactive
other - in process

If the system is reset or just powered-up to the SAGE DEBUGGING TOOL (SDT), the light will be green as the I/O ports are continually being polled.

After booting to an operating system, the color of the light depends on how I/O is done. Operating systems which have an interrupt driven BIOS (such as the p-System) will normally show a RED light as the bus is inactive and the processor is waiting for an interrupt.

Because the SAGE DEBUGGING TOOL (SDT) can run with either polled or interrupt driven drivers, the status of the light will depend on which drivers the programmer is using.

The processor light can also be controlled by user software. Examples of how to do this are given in the SDT/ASSEMBLER MANUAL.

II.03 MEMORY :

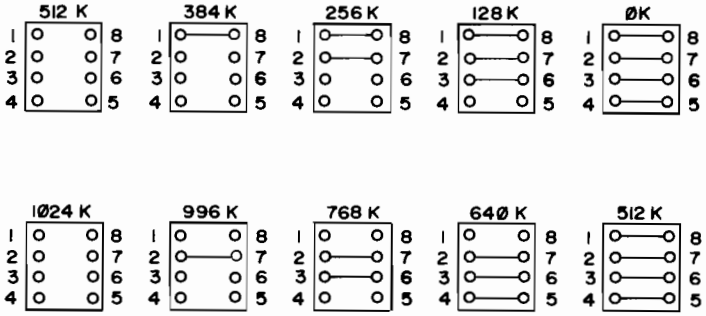
The SAGE II/IV RAM memory is configurable from 128k to 1024K bytes in 128k increments. Memory is implemented in 64k dynamic RAM with 150 nsec access time with no wait states. Parity checking is provided on each byte. Up to 512k bytes may reside on the main processor board (CPU board) and 512K additional bytes may reside on the Winchester board. 16K (64K reserved) EPROM firmware contains self-test, DEBUGGER, and bootstraps.

Memory is laid out on the boards with respect to the chip addressing. On each board there are 4 rows of RAMs, each row providing two 64K byte blocks of memory. Eight chips, (64K x 1) are needed to provide 64K bytes of memory. One more chip provides parity for the other eight. The Figure on the next page shows the location of the RAM CHIPS and their addresses.

Strapping options on the boards must be set for the amount of RAM populated in the board. This option is set for you by the factory and is only mentioned in case you decide to add more memory later. (NOTE: if the SAGE factory does not install and test the memory, we will not guarantee that it will work. Because of variations in quality in chips from the different vendors, we recommend that you have SAGE install and test the correct type of RAM for you.

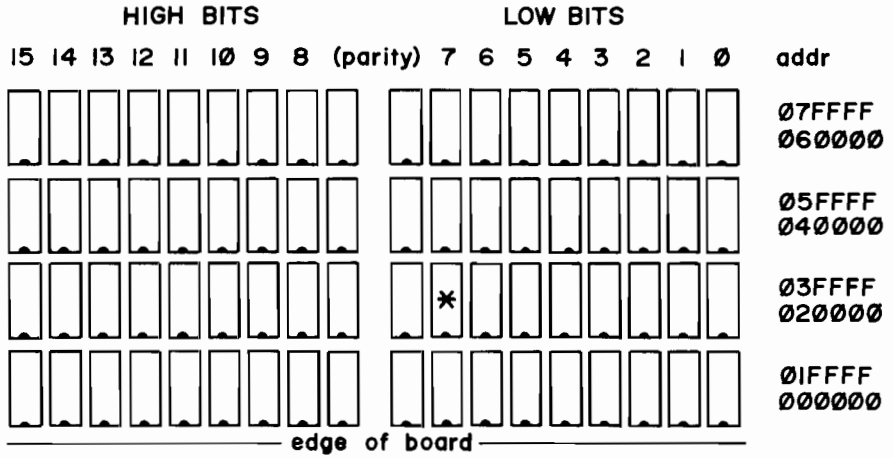
The following figures show the strapping options and their locations.

The possible RAM strapping options for the CPU board are:

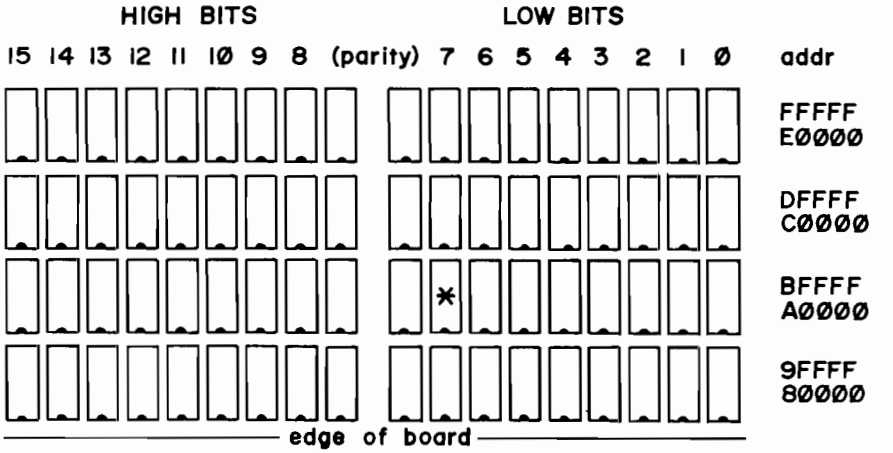


The possible RAM strapping options for the Winchester board are:

This figure gives the addresses for the RAM chip locations:



* The bit that is off (0) is in the chip for bit 7. Note that it would be the same chip for value FF7FFFFF.



* The bit that is off (0) is in the chip for bit 7. Note that it would be the same chip for value FF7FFFFF.

● RAM Chip Addresses

To locate a specific RAM chip you need the address and contents of the address. SAGE II/IV startup test error messages generally give both values.

The figures on the previous page show the layout of the ram chips. The RAM chips are arranged in 4 rows of 18 chips on the board. The two middle chips in each row are the parity chips.

The ADDRESS tells you what row of RAM to look at. It is given as a 8 digit or 6 digit hexadecimal number. If given as a 6 digit number assume the two leading digits are zero. Example:

loc= 0024021 is in row 020000-03FFFF

The CONTENTS identify the bit associated with the chip. The value is usually a LONG WORD of two 16-bit words. Each 16-bit word has 8 HIGH bits and 8 LOW bits (high byte, low byte):

To find the value of a specific bit, convert the hex digits of the words to binary. For example, converting the value FFFFFF7F gives:

```
        High bits   Low bits
FFFF = 1111 1111   0111 1111    High word
FF7F = 1111 1111   0111 1111    Low  word
                   X
```

For example, if an error message indicates that a value should be FFFFFFFF but is FFFFFF7F at loc 024021, then the incorrect bit is in the chip for bit 7. Note that it would be the same bit, different word for value FF7FFFFF.

II.04 PROM STRAPPING OPTIONS :

Various PROMs/ROMs can be used in the SAGE II/IV. The board is currently configured for 8K or 16K byte PROMs.

The strapping traces right above the PROMS on the board allow you to configure for larger or smaller PROMS. The following figures show the options and location of the straps.

PROM SIZE	STRAP		CHANGE
16K bits (4K bytes)	S04	[B] []--[A]	cut B, connect A
	S03	[B] []--[A]	
	S02	[B] []--[A]	
64K bits (8K bytes)	S04	[B]--[] [A]	none
128K bits (16K bytes)	S03	[B] []--[A]	
	S02	[B] []--[A]	
	S04	[B]--[] [A]	
256K bits (32K bytes)	S03	[B]--[] [A]	cut A, connect B
	S02	[B] []--[A]	
	S04	[B]--[] [A]	
	S03	[B] []--[A]	
512K bits (64K bytes)	S02	[B]--[] [A]	cut A, connect B

II.05 REAL-TIME CLOCK :

A 6 byte timer with the least significant bit =1/64000 of a second is kept by the SAGE II/IV for access by the user to time events. The clock can be read using a "LONG CALL 5" to the BIOS. To read the clock in 1/60th of a second use a "LONG CALL 6". See the section on BIOS and your operating system specifics.

II.06 POWER SUPPLY :

The SAGE II and SAGE IV have different power supplies and power requirements:

SAGE II POWER REQUIREMENTS:

Supply Rating	With two floppies
+ 5V at 6.0 A	3.50 A
+12V at 2.5 A	1.00 A
-12V at 0.2 A	.02 A
110 VAC	.60 A

Input from 95 - 135 VAC, 47 - 450 hz
OR 190 - 270 VAC with 220V mod.

SAGE IV POWER REQUIREMENTS:

Supply Rating	with one floppy	with two floppies
+ 5V at 10.0 A	7.80 A	8.30 A
+12V at 4.0 A	2.20 A	2.70 A
-12V at 0.2 A	.05 A	.05 A
110 VAC	.70 A	.80 A

Input from 95 - 130 VAC, 47 - 440 hz
OR 190 - 260 VAC with 220V mod.

The SAGE II/IV comes with a detachable power cord for easy hookup. Note that the SAGE II/IV will operate without modification on either 50 or 60 hz 120VAC. A minor strapping modification is necessary for operation on 220VAC. Systems can be ordered with this modification.

There is a high isolation line filter installed on all SAGE II/IV machines. If power abnormalities affect the system (as in a heavy industrial environment) then installation of an Uninterruptable Power SUPPLY (UPS) is advised. Normally, the filter will provide about 50 db at 1MHZ Common mode and 23 db at 1MHZ differential mode.

II.07 PHYSICAL CHARACTERISTICS :

The SAGE II measures 3.5" x 12.5" x 17" in a sturdy aluminum case. It weighs 15 lb 8 ounces and is compact and easy to service.

The SAGE IV measures 6.5" x 12.5" x 17" and weighs 22 lb 8 ounces.

II.08 COOLING AND ENVIRONMENT :

A quiet 20 brushless DC fan is used for cooling. Air is pulled across the equipment in the case from the right air vents and forced out the bottom in a SAGE II or out the left side in a SAGE IV. Some air also enters through the floppy drive opening(s) in the front. The figures on the next page show this.

Care should be taken that the vents are not blocked.
Temperature:

16 to 35 degrees C operating
-44 to 71 degrees C non-operating

Relative humidity:

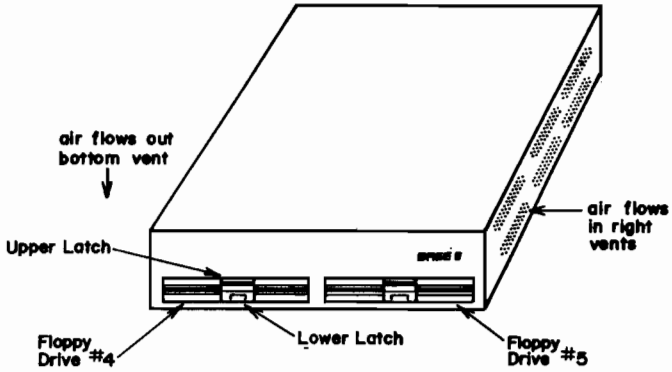
20 - 80 operating (non-condensing)
5 - 95 non-operating (non-condensing)

Altitude:

minimum 304.8 m (500 ft) below sea level
(operating and non-operating)

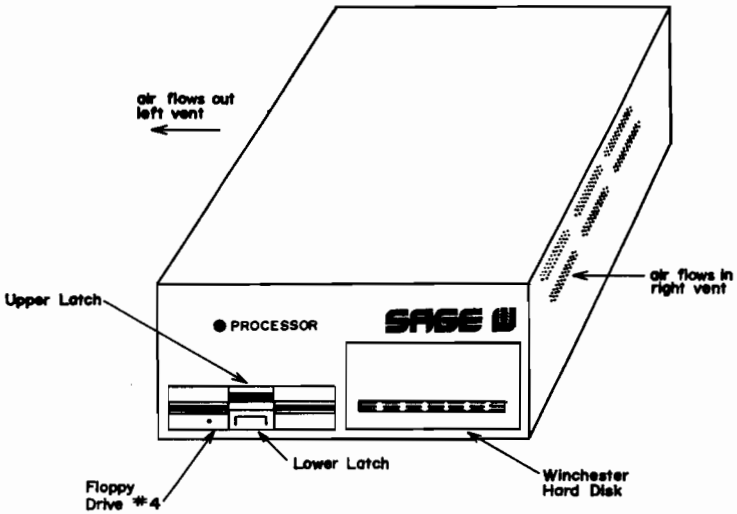
maximum 3,048.0 m (10,000 ft) above sea level
(operating)
15,240.0 m (50,000 ft) above sea level
(non-operating)

Figure of SAGE II air flow.



SAGE II

Figure of SAGE IV air flow.



SAGE IV (With Hard Disk)

II.09 EXPANSION BUS :

The SAGE II/IV bus is a expansion bus provided for interfacing to SAGE products. It is not provided for use as a general application bus. A short description of the bus signals follows which assumes the user is familiar with the general architecture of the 68000. More information is provided in the SAGE II/IV SERVICE MANUAL.

CONN-PIN	SIGNAL	DESCRIPTION
J1-1-49 odd pins	ground	System & logic ground. All odd pins are grounded to reduce signal noise
J1-2-46 even pins	A1-A23	These 23 outputs are cpu address lines buffered with octal drivers.
J1-48	FC2-	This output is (S) the state of the supervisor bit in the Status Reg of the 68000 and is used to control access to I/O locations.
J1-50	PRTY-	This input returns the parity status of an addressed location to the interrupt decoder and causes a level 7 interrupt on a parity error.
J2-1-33 odd pins	ground	System and logic ground on these odd pins reduces noise on the data bus.

CONN-PIN	SIGNAL	DESCRIPTION
J2-2-32 even pins	D0-D15	The 68000 data bus is buffered with octal transceivers gated by the BRW+ and RW read/write signals.
J2-34	SRES-	This output is the soft-reset signal which is generated at power-up by the reset switch or by the software to initialize devices on the bus.
J2-36	8Mhz-	Buffered system clock.
J2-38	RW-	This output is low for a READ cycle and high for a WRITE cycle.
J2-40	LDS+	This output, the LOWER DATA STROBE, controls the transfer of data on the eight low order data lines (D0-D7) for least byte addressing.
J2-42	UDS+	This output, the UPPER DATA STROBE, controls the transfer of data on the eight high order data lines (D8-D15) for high byte addressing.
J2-44	AS+	This output,the ADDRESS STROBE, goes low when a valid address is on lines A0-A23.
J2-46	I2-	This input goes low for a LEVEL 2 interrupt.
J2-48	I3-	This input goes low for a LEVEL 3 interrupt.
J2-50	DTACK-	The input, DATA TRANSFER ACKNOWLEDGE goes low when a data transfer has completed.
J2-35-45 odd pins	+5V	
J2-47	+12V	
J2-49	-12V	

II.10 I/O ADDRESSING :

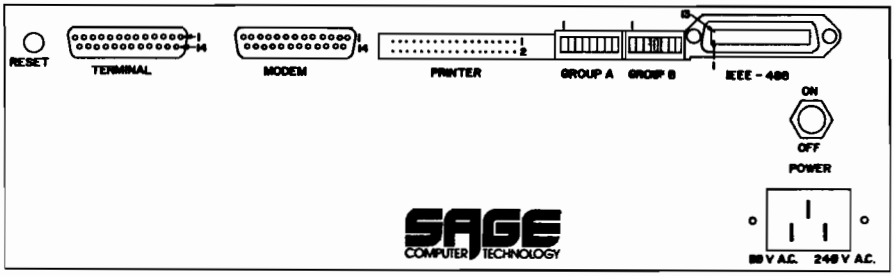
I/O locations for the system devices are given in the SDT-ASSEMBLER manual. Access to I/O must be done in SUPERVISOR mode or a "BUS ERROR" will occur.

II.11 I/O PORTS :

I/O implementation is simplified with I/O memory-mapped assignment. All input/output is interrupt-driven but can be optionally polled. The user I/O connectors are located on rear panel:

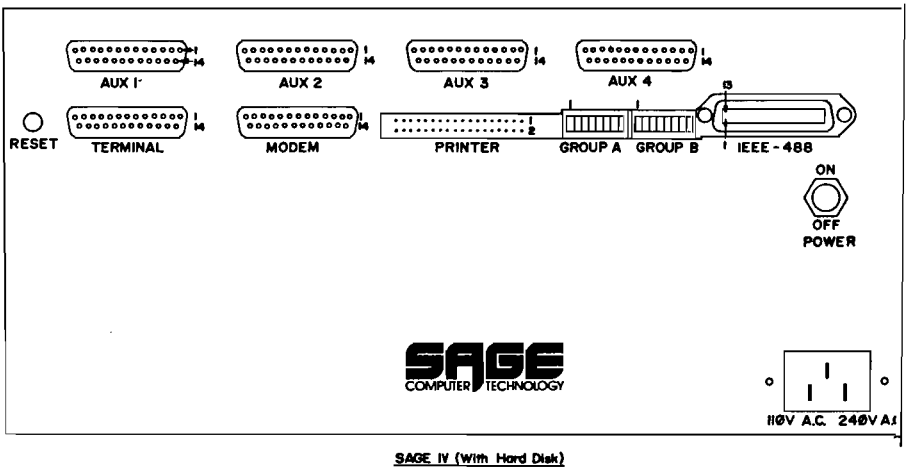
TERMINAL	RS232-C port
MODEM	RS232-C port
PRINTER	parallel port
GROUP-A	dipswitch
GROUP-B	dipswitch
IEEE-488	GPIB bus

SAGE II BACK PANEL



SAGE II

SAGE IV BACK PANEL



SAGE IV (With Hard Disk)

II.12 MODEM PORT :

A second RS-232C serial port is available to connect to a modem or other RS-232C device. (Refer to the GETTING STARTED MANUAL for more information on using and connecting a MODEM.) It operates as Data Terminal Equipment. Modem control lines including ring detect are fully supported. A DB-25 connector is located at the rear of the machine and is marked MODEM. Pin-outs for the signals are:

Modem port is considered Data Terminal Equipment

J9 - 1	Protective Ground	
2	Transmitted data	(to modem) --->
3	Received data	(from modem) <---
4	Request-to-send	(to modem) --->
5	Clear-to-send	(from modem) <---
6	Data set ready	(from modem) <---
7	Signal Ground	
8	Carrier detect	(from modem) <---
9	+12 VDC	
10	-12 VDC	
20	Data terminal ready	(to modem) --->
22	Ringing detect	(from modem) <---

For the modem port to be able to send Transmitted Data, the Clear to Send (pin 5) must be asserted (>+3V). If no Clear to Send signal is available (such as when connected to another DTE device), the Clear to Send input (pin 5) may be connected to pin 9 (+12V) to hold it in the asserted state.

If the computer hangs when transmitting to the Remote serial channel, two possibilities exist. First, the Clear to Send line may be inactive (< +3V). Second, if the DSR polling option is selected (see SAGE4UTIL) the Data Set Ready signal may be inactive (< +3V). If the computer does not hang up on a transfer of over 255 characters then the data is probably being transmitted by the SAGE modem port. In cases where transmission seems successful but the data is not received, check that the Transmitted data from pin 2 is going to the proper input pin on the receiving device (see

the GETTING STARTED MANUAL for more information on RS232 protocol). Also verify that the proper baud rate, number of stop bits, and parity are consistent between the SAGE Remote channel and the receiving device.

If a problem occurs in receiving data from another device check these same possibilities in the opposite direction. The Clear to Send of the transmitter for DTE devices must be asserted. The Received Data line to pin 3 must be connected to the transmitters output signal. The baud rate, number of stop bits, and parity must match between the SAGE Remote channel and the transmitting device.

The 8251A USART chip used for the Remote Channel serial interface has the protocol of double sending a character if the device is held up by an external Clear to Send signal. The USART contains a Transmit register from which data is being sent, and a Transmit Buffer which holds the next byte to be sent. If both the register and buffer contain data (the normal case) when the Clear to Send signal is negated, the chip goes ahead and completes sending both bytes before stopping. Unfortunately the circuit seems to expect that if you negate the Clear to Send signal you will ignore any further data and so when you make Clear to Send active again it re-sends the byte that was in the Transmit Buffer. Most receiving devices however will accept a few more characters after they tell the transmitting end to stop so they receive the same character twice.

The Remote Channel Transmit Driver has an option to use the Data Set Ready for controlling the transmission. Refer to the SAGE4UTIL section of the GETTING STARTED MANUAL. The driver checks the DSR signal before each character is output from the interrupt routine. If DSR is not asserted, a DSR polling routine is scheduled to keep checking to see if data can be output.

II.13 SERIAL PORTS :

The Winchester board supports 4 serial ports labeled AUX1-AUX4 on the back of the computer. As the CPU board has two, this gives the SAGE IV a total of 6 serial ports. These ports can be used to connect to terminals, printers or almost any device which supports RS232 asynchronous communications protocol. AUX1-AUX4 do not have the extra handshake lines for modem control although "smart modems" may work on the ports. The Modem port on the CPU board is the only serial port with modem handshaking.

The auxiliary serial channels are implemented as Data Communication Equipment (DCE) since the most common use for them will be to connect to terminals and printers. The signals available are:

AUX1- AUX3 PINOUTS

PIN#	SIGNAL	DESCRIPTION	Direction
1	GND	Protective Ground	-----
2	RXD+	Transmitted Data	from terminal <----
3	TXD+	Received Data	to terminal ---->
4	RTS+	Request to send	from terminal <----
5	CTS	Clear to send	to terminal ---->
6	DSR+	Data Set Ready	to terminal ---->
7	GND	Signal Ground	-----
9	+12V		-----
10	-12V		-----

The Data Set Ready output is tied high (+12v) to indicate to the terminal that the SAGE is on-line. Data Terminal Ready (pin #20) is not used, the Terminal will be considered on-line at all times.

The Clear to send (CTS+ on pin #5) is tied high (+12V) internally to indicate the SAGE is always ready for data.

Typically, the Request to send (RTS+ on pin #4) is tied to high (+12V on pin 9 in the connector) so that the SAGE will always expect the terminal to be ready. If connecting RTS+ high cannot be done inside your connector, straps are available inside the computer. Cut the strap at A and connect to B. (Strap E7 for AUX 1, E8 for AUX 2 and E1 for AUX 3). This ties RTS+ high (+12V).

If the device to be connected is a printer or one that expects handshaking operations, connect the incoming handshake line to the auxiliary channel's RTS+ pin 4. This implements a "one-way" handshake with the terminal informing the SAGE by way of RTS of its status.

> IMPORTANT NOTE: This implementation assumes that the SAGE will always be able to keep up with the terminal. However, if any auxiliary ports are connected to smart terminals capable of sending pages of data in a burst, XON/XOFF protocol will be required to be output from the SAGE. If missing characters occur when receiving at high baud rates simultaneously on several channels, reduce the baud rate.

The AUX4 port can be connected in the same way as AUX1-AUX3 as the straps are normally installed for RS232 operation but does have other capabilities. The transmit and receive clocks are brought out to the edge connector for the user. This allows for synchronous communications at the hardware level. No system software is currently provided for synchronous communication but the user can write his own driver to implement the protocol he needs.

AUX4 RS232 PINOUTS

PIN	SIGNAL	DESCRIPTION	DIRECTION
1	GND	Protective Ground	-----
2	RXD+	Transmitted Data	from terminal <----
3	TXD+	Received Data	to terminal ---->
4	RTS+	Request to send	from terminal <----
5	CTS	Clear to send	to terminal ---->
6	DSR+	Data Set Ready	to terminal ---->
7	GND	Signal Ground	-----
9	+12V		-----
10	-12V		-----
11	RXC+	Receive clock. Strap E6 to A (standard)	
13	TXC+	Transmit clock. Strap E5 to A (standard)	

The last port (AUX 4) can be operated identically to ports AUX1 AUX2 and AUX3. It can be also be optionally strapped to support RS422 protocol. This is provided for the user who may have a RS422 device.

The RS422 +5V differential drivers are capable of transmitting a longer distance than the RS232 +12V drivers. If additional RS422 ports are needed, adapters which convert RS232 to RS422 signal levels are readily available.

AUX4 RS422 PINOUTS

PIN	SIGNAL	DESCRIPTION	DIRECTION
3	TXD+	Received Data + Cut Strap E4 to A, connect B.	to terminal ---->
16	TXD-	Received Data -	to terminal ---->
2	RXD+	Transmitted Data Cut strap E3 to A, connect B.	from terminal ---->
17	RXD-	Transmitted Data	from terminal <----
11	RXC+	Receive Clock+ Cut Strap E6 to A, connect B.	
12	RXC-	Receive Clock-	
13	TXC+	Transmit Clock+ Cut Strap E5 to A, connect B.	
14	TXC-	Transmit Clock-	

● Serial Channels

On the p-System the four extra serial channels are assigned different volume numbers depending on how many subsidiary volumes are allowed and the number of the first subsidiary volume. Briefly volume numbers are assigned in this order:

Standard devices
Subsidiary volumes
Extra Serial channels

Thus if the number of real devices allowed is increased, the subsidiary volumes and extra serial channels acquire higher volume numbers. If the number of subsidiary volumes increase, the extra serial channels acquire higher volume numbers.

The p-System Filer will show the different volume numbers for the extra serial channels depending on how many subsidiary volumes are on-line. The Filer will always list the extra serial channels after the subsidiary volumes.

The number of devices and subsidiary volumes allowed is contained in SYSTEM.MISCINFO. The SYSTEM.MISCINFOs on the SAGE distribution disks have 12 devices and 16 subsidiary volumes. The p-System program SETUP.CODE can be used to change them. Note that the number of devices allowed is one less than the volume number assigned to the first subsidiary volume. This table shows the relationships of channels and volumes for the extra serial ports under a standard SYSTEM.MISCINFO.

Single User Physical Channel	Multi-User Physical Channel	p-System Logical Channel	p-System Volume Number	p-System device name
13	13,14	28	#29:	SERIALA:
14	15,16	29	#30:	SERIALB:
15	17,18	30	#31:	SERIALC:
16	19,20	31	#32:	SERIALD:

Note that the Multi-user System defines both an INPUT and and OUTPUT physical channel for the extra serial ports.

Because the volume number varies according to the user's SYSTEM.MISCINFO, programs written for commercial distribution should never embed absolute volume numbers. They should always reference them by the names assigned to them: SERIALA:, SERIALB:, SERIALC: and SERIALD:. The actual p-System volume number for the names can be determined by using the D_Dir_List function of the DIRINFO unit.

UNITREAD and UNITWRITE commands use the p-System numbers.

Note that a rate of 200 baud is not supported on any of these extra serial channels.

II.14 FLOPPY DRIVES :

● Referencing floppy drives

There are several ways that floppy disks can be physically installed in a SAGE II/IV system. A floppy drive is referenced either as the LEFT drive (#4:) or as the RIGHT drive (#5:) depending on how it was installed. The SAGE4UTIL program in particular makes reference to LEFT and RIGHT drives.

● On a SAGE II with two floppy drives, The drives are referenced:

LEFT (#4:)	RIGHT (#5:)

● SAGE IV systems with one floppy drive have the single floppy mounted on the left so it is properly called the left drive (#4:).

LEFT (#4:)	WINCHESTER

● SAGE IV systems with two floppy drives have the drives mounted one above the other. The top drive will be LEFT (#4:) and the bottom drive will be RIGHT (#5:).

LEFT (#4:)	WINCHESTER
RIGHT (#5:)	

● Floppy Disk Step Times

Due to variations in floppy disk drive models and vendors, the step time for all drives has been set up to a default value of 6 milliseconds. The low profile (half height) Mitsubishi drives which are normally shipped in SAGE II/IV systems will run at a step time of 4 milliseconds.

The step time for these drives may be modified down to 4 milliseconds using the SAGE4UTIL configuration utility program. Note that some users have successfully gone as low as 2 milliseconds but this exceeds the specifications of the drive. Turnkey distributors should always configure the BIOS for 6 millisecond step time to insure that the system will work on all possible drives.

Refer to the SAGE4UTIL section of the GETTING STARTED MANUAL for information on how to change the setting.

● Writing 40 Track Diskettes

Reading or writing a 40 track diskette on an 80 track drive is not supported by the drive manufacturers. However because reading a 40 track diskette on an 80 track drive generally works, a mode was provided to double step the drive to allow this capability. Writing a 40 track diskette on an 80 track drive is even more susceptible to problems. An early version of the BIOS made the access read only. Experiments by users and SAGE have shown that writing 40 track diskettes on 80 track drives was fairly secure depending on the drive.

The current BIOS has removed the write lockout and now allows users to write 40 track diskettes **at their own risk!** Note that this only seems to work on new (or bulk erased) diskettes which do not have information recorded on the unused tracks.

This feature is often useful when porting software between different types of machines. SAGE does not recommend it for distributing commercial packages as the reliablity is highly questionable and possibly drive dependent. Note that SAGE4UTIL menu screens still indicate that write access is not allowed.

● Floppy Drive Parameters

Fast, reliable, 5 1/4" floppy drives access either 320K bytes (48 TPI) or 640K bytes (96 TPI) of diskette storage per drive. A 10K program is typically loaded into memory in 1/2 second. The motors are turned off automatically when not in use to increase head and media life, save power and generate less heat. The SAGE II/IV determines when it should turn off the motor by tracking previous useage of drives. This makes the time to de-select the drive history-dependent so it may vary from one programming session to another. Normal operation of drives is interrupt-driven although they can optionally be polled.

The disk drives will operate upright, horizontally or vertically, although horizontal operation of SAGE II is standard.

Pin-outs for floppy drive connector signals are:

J4-	-10	Select drive 0
	12	select drive 1
	16	Motor on
	22	write data
	2,4,6,14,34	unused
	24	write gate
	32	head select
	18	direction
	20	step
	26	track 0
	28	write protect
	8	index
	30	raw data
	1-33	all odd pins grounded.

More information on the floppy drives is given in the SAGE SERVICE MANUAL.

II.15 WINCHESTER DISK :

SAGE II/IV Winchester disk drives are all low power devices fitting in a 5 1/4" space. The different sizes of drives may have a different number of heads (platters) and cylinders.

The following table shows the currently supported drives and (in order) the total number of cylinders, heads, tracks, blocks, and bytes available on the disk. The last column (User Blocks) shows the number of blocks available for user partitions. This is the total minus the allowance for bad track mapping is done (76 blocks per head) and minus 19 blocks for the track used for the partition map.

Disk Type	Tot Cyl	Tot Hds	Total Trks	Total Blocks	Total Bytes	User Blocks
6M	306	2	612	11,628	5,953,536	11,457
12M	306	4	1224	23,256	11,907,072	22,933
18M	306	6	1836	34,884	17,860,608	34,409
40M	512	8	4096	77,824	39,845,888	77,197

Generally, 4 cylinders per head are reserved for mapping bad tracks. Typically, drive manufacturers guarantee no more than 4 bad tracks per head. Users who have drives with less than the reserved number of tracks may want to utilize this area and can adjust the size accordingly, though this is **not** recommended.

Typically, the drives are divided into smaller areas called partitions (subdevices). Up to 16 partitions can be specified, with 15 available to the user(s). Refer to the section on WINCHESTER PARTITIONS page 40 for more information.

The Winchester controller will support up to 4 Winchester drives.

III DEVICES, CHANNEL MAPS & PARTITIONS :

One of the most important features to understand about the SAGE operation is how DEVICES, CHANNEL MAPS and the hard disk PARTITIONS relate to one another. Multi-user System Managers will find this particularly important.

III.01 DEVICE NUMBERS :

Each different software operating system has a method of referring to the physical hardware devices. In the p-System each hardware device is given a number and/or a name. Serial device names are permanently assigned by the p-System while names for the block structured storage devices (diskettes and disks) are assignable by the user and recorded on the device.

Normally an operating system will have a fixed relationship between its device numbers, called "Logical Device Numbers" and the physical devices. The SAGE Single and Multi-User BIOS (Basic I/O System) programs also refer to the devices by numbers called "Physical Device Numbers". Different operating systems may have varied numbering requirements (the p-System always numbers the printer as #6, etc.) so it is generally impossible to make the operating system's Logical Device Numbers always match the same Physical Device Numbers assigned by the BIOS.

III.02 THE BIOS CHANNEL MAP :

The SAGE BIOS programs have therefore defined a BIOS Channel Map to allow a configurable assignment of the Physical Devices to the operating system's Logical Device Numbers. Not only does this handle the problem of different operating system Logical Device Number assignments, it may be useful when using the same operating system as will be shown later.

Physical Device Numbers

Following is a table of Physical Device Numbers used by the SAGE Single and Multi-User BIOS programs. Notice that there are some differences in the assignments between the two BIOS programs (extra serial ports) and that not all the Multi-User devices (extra RAM Disks) are available from the Single-User BIOS.

BIOS PHYSICAL DEVICE NUMBERS

Single User#	Multi User#	Device	Connector (if any)
0	0	no device	
1	1	Keyboard #1	Terminal
2	2	Terminal #1	Terminal
3	3	reserved	
* 4	4	Left floppy	
* 5	5	Right floppy	
6	6	Parallel Printer Port	
7	7	Keyboard #2 (Remote serial input channel)	Modem
8	8	Terminal #2 (Remote serial output channel)	Modem
* 9	9	Winchester drive partitions	
*10	10	Reserved	
*11	11	RAM Disk #1	
*12	12	Reserved	
13	13	Keyboard #3	Aux 1
13	14	Terminal #3	Aux 1
14	15	Keyboard #4	Aux 2
14	16	Terminal #4	Aux 2
15	17	Keyboard #5	Aux 3
15	18	Terminal #5	Aux 3
16	19	Keyboard #6	Aux 4
16	20	Terminal #6	Aux 4
*	21	RAM Disk #2 (MU only)	
*	22	RAM Disk #3 (MU only)	
*	23	RAM Disk #4 (MU only)	
		User defined devices (by SAGE)	
128	128	Device configuration control	
129	129	Reading system clock	
130	130	General Memory Access	
131	131	Scheduled event control	
	132	Multi-User advanced features	
	133	Multi-User Inter Communications	

* Block structured devices.

p-System Logical Device Numbers

As was stated earlier, each operating system has an assignment of Logical Device Numbers which are intended to correspond with the physical devices on the system. Following is a table showing the Logical Device Numbers for the p-System and their intended usage.

Logical Device	Usage
0	unused
1	Keyboard input
2	Terminal output
3	unused
4	Block storage device
5	Block storage device
6	Printer
7	Remote input (serial)
8	Remote output (serial)
9	Block storage device
10	Block storage device
11	Block storage device
12	Block storage device
13...27	extra Block storage devices
28	Extra serial channel #1
29	Extra serial channel #2
30	Extra serial channel #3
31	Extra serial channel #4
128..133	User defined devices

It is important to note that although there are some similarities between the p-System's Logical Device Number assignments and the Volume numbers that are displayed in the FILER, they are not the same numbers. The Logical Device Numbers are the numbers provided by the interpreter/BIOS interface routine when the BIOS is called. Following is a discussion of the differences to clear up the confusion.

The p-System Volumes #1: and #2: are both used for terminal input and output. The difference is that the characters read on Volume #2: are not echoed while those read on Volume #1: are echoed. Logical Device 1 is only used on reads of keyboard input by the interpreter interface. Logical Device 2 is only used for terminal output from the interpreter. For the implementation of p-System Volume #1:, the interpreter uses Logical Device 1 to read the character and then echoes the character back on Logical Device 2.

The p-System Volume numbers for volumes #4: to #12: are passed straight through to the BIOS as Logical Device Numbers.

Logical Devices 13 to 27 are only used if the program SETUP.CODE is used to raise the value of the First Subsidiary Volume in SYSTEM.MISCINFO. If this number is increased, the lower p-System Volume numbers are passed straight through to the BIOS as the Logical Device Numbers.

The Extra Serial Channels are always passed the Logical Device Numbers 28 to 31, independent of the p-System Volume numbers. The p-System Volume numbers for the Extra Serial channels will always be allocated just above the Subsidiary Volumes. Since the First Subsidary Volume and the number of Subsidiary Volumes is configurable by SETUP, the Volume numbers may vary from installation to installation. The p-System UNITIO procedures (UNITREAD, etc.) use the Volume number as their device number argument so care should be taken to insure that a program using the Extra Serial Channels will not fail on a different SYSTEM.MISCINFO configuration.

● Typical p-System Channel Map

The Single or Multi-User BIOS Channel Map sets up a relationship between the Logical Device Numbers given by the p-System interpreter and the Physical Device Numbers known to the BIOS. Following is a typical Channel Map assignment of Logical to Physical device numbers for a single user SAGE IV.

Logical Device	Physical Device
0 - unused	0 - unused
1 - Keyboard	1 - Keyboard #1
2 - Terminal output	2 - Terminal #1
3 - unused	0 - unused
4 - Block Storage Device	4 - Left Floppy
5 - Block Stroage Device	5 - Right Floppy
6 - Printer	6 - Parallel Printer Port
7 - Remote in	7 - Keyboard #2 (modem port)
8 - Remote out	8 - Terminal #2 (modem port)
9 - Block Storage Device	9 - Winchester drive
10 - Block Stroage Device	0 - unused
11 - Block Storage Device	11 - RAM Disk
12 - Block Storage Device	0 - unused
13 - Block Storage Device	0 - unused
27 - Block Storage Device	0 - unused
28 - Extra Serial Channel #1	13 - Keyboard/Terminal #3 (Aux 1)
29 - Extra Serial Channel #2	14 - Keyboard/Terminal #4 (Aux 2)
30 - Extra Serial Channel #3	15 - Keyboard/Terminal #5 (Aux 3)
31 - Extra Serial Channel #4	16 - Keyboard/Terminal #6 (Aux 4)

Note: The Physical Device 9 does not completely define the Winchester drive. See the discussion of Winchester Partitions later in this manual.

● Auxiliary Channel Device Numbers

There is a difference in the Physical Device Numbers associated with the Extra Serial Channels between the Single-User BIOS and the Multi-User BIOS. The p-System uses a single Logical Device Number for its Extra Serial Channels for both the input and output directions. Unfortunately, if an Extra Serial Channel is to be used as another user's terminal port, the channel needs to be treated as two different devices.

Therefore the Multi-User BIOS defines the four Extra Serial Channels as pairs of devices with different Physical Device Numbers. Now it is easy to assign the Extra Serial Channels to the Keyboard input, Terminal output, Remote input, and Remote output channels. To assign a Physical Extra Serial Channel (with two device numbers) to a Logical Extra Serial Channel (with one device number), use the Physical Device Number for either the input (keyboard) or output (terminal) channel. Both Physical Device Numbers will work equally well.

III.03 WINCHESTER PARTITION OVERVIEW :

Removable media devices such as floppy diskette drives provide a great deal of flexibility in that an unlimited amount of information can be made available to the system. The user can just change the diskette on the same device and different information becomes available. For the p-System, the Logical Device Number (such as #4 for the floppy) stays the same while the Volume Name, which is recorded on the diskette, changes. A completely different operating system may be started by loading and rebooting the system from a different diskette.

Fixed media devices such as a Winchester disk drive can provide a larger amount of storage on-line than a floppy diskette but the total available storage is limited to the size of the device. It is therefore important to make the most approriate use of the available on-line storage provided by a Winchester drive. Since there are a variety of different uses for a computer system, a fixed Winchester storage device could potentially limit the application of the system. SAGE has provided a method of avoiding this problem by allowing the user to partition the Winchester drive into areas which each look like a separate device. The Winchester Partition feature provides flexibility but does require that the user understand how to allocate and control the space on the disk.

There are several reasons for having separate partitions.

1. If several people are using the system, (even without multi-user) they will each need at least one work area, possibly more.

2. It is often convenient to assign partitions to different software projects or applications. Example, partition #2 is for word-processing, #3 is for statistical analysis, #4 sales records, etc.

3. Different operating systems MUST have their own partitions. It is possible to have the p-System, CP/M 68K, and other operating systems all on the same disk but not in the same partition.

4. Each user under the SAGE Multi-user system must have at least one partition that is the user's "boot" partition containing all of the necessary system files and a bootstrap program. A description of "Boot" partitions is on page 46 .

5. The p-System limits the number of blocks per device to 32K. This means that the larger disks (over 16 megabytes) MUST be split into partitions. This also means that no one p-System partition can be greater than 16 Megabytes. Other operating systems do not have this restriction but may have other restrictions of their own.

6. Under the Multi-user system it may be desirable to assign a partition to a special program, such as a print spooler or data collector. This program will act like another user (beyond the maximum of 6 which have terminals assigned to them) and can be set up to start immediately when the Multi-user system is loaded.

III.04 USING WINCHESTER PARTITIONS :

In order to use Winchester Partitions, a few facts are necessary. The following sections cover topics on the Winchester Partition numbering scheme, what Partitions are available on new systems, how to bootstrap to Partitions, and how to make Partitions available to an operating system.

● Partition Numbering Scheme

The SAGE IV BIOS software allows each Winchester drive to be divided into 16 partitions. As there can be up to 4 Winchester drives on the system, this gives a maximum of 64 partitions.

The SAGE IV BIOS has assigned each of the partitions a "Subdevice" number. In this way, each partition looks like a separate disk drive to the software that calls the BIOS.

SAGE4UTIL and MU.UTIL maintain the BIOS Channel Maps which relate the operating system's Logical Device Numbers to the Physical Device Numbers. For Winchester disk drives, the Physical Device Number is comprised of a normal device Number (always 9) and a Subdevice number which corresponds to a disk partition.

The following table provides a quick cross-reference between the partitions on the disk drives and their Subdevices number. The 64 partitions are numbered as Subdevices 0-63. The partitions are shown by the four drives (0-3), each with 16 partitions given in hexadecimal 0-9, A-F.

Note that it is convenient to think of partitions in hexadecimal because the Debugger boot commands (discussed later) usually require hexadecimal arguments.

Drive 0

Partition#	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Subdevice	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Drive 1

Partition#	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Subdevice	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Drive 2

Partition#	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Subdevice	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47

Drive 3

Partition#	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Subdevice	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

If more partitions are needed, remember that the p-System supports a Subsidiary Volume feature which allows a logical device to exist within a file. The SAGE IV normally is shipped configured for 16 subsidiary volumes.

Factory Installed Partitions

The Winchester disk is partitioned by the factory into these device areas:

Volume	Name	No. of blocks	Description
# 9:	SYS:	[1280]	System files
#10:	SCRATCH:	[1280]	Empty, zeroed directory.
#12:	WORK:	rest of disk	Empty, zeroed directory.

SYS: Partition 1 contains the system files necessary to boot the p-System. If the computer terminal was ordered from SAGE, it contains the proper SYSTEM.MISCINFO and GOTOXY installed to run the terminal. It also has the wordprocessing, communications and spreadsheet programs. The GETTING STARTED MANUAL includes instructions on how to "boot" to this partition and run the programs.

Otherwise, a "vanilla" system is built and the user must install his own terminal setup. See the INSTALLATION sections of the GETTING STARTED MANUAL.

SCRATCH: Partition 2 is set up with 1280 blocks which is the standard size of a SAGE 80 track floppy. This area may be used for transferring an entire floppy onto (and back off of the hard disk). It is often faster to use this area to build an image of a distribution diskette and then copy it to a floppy than it is to build onto the floppy itself. An example of how to use this area to copy a diskette is given in the GETTING STARTED MANUAL.

WORK: Partition 3 is a large partition containing all the blocks on the disk not used by the boot, SYS: and SCRATCH: partitions. Its size depends on the size of the disk ordered.

Note that partitions 2 and 3 may be reconfigured without disturbing partition 1. For example, splitting WORK: up into smaller partitions does not effect the information in partitions 1 and 2 as long as they remain the same size.

Another alternative to using the large area in WORK: which is available under the p-System, is to create a number of Subsidiary Volumes on the device. This will allow a user to group together common files and to get around the 77 file per directory restriction.

Note that on drives larger than 18 Meg, the space remaining for WORK: will be greater than the 16 Meg allowed by the p-System. Additional partitions with up to 16 Meg each will be allocated and named WORK1:, WORK2:, etc. Check your FORMATINFO file (described on page 64) for specific details on the assignment.

● Bootstrapping To Partitions

All partitions (except the special case 0) are capable of being booted to. A bootable partition must have the **SYSTEM files necessary to run that operating system.** These files can be loaded onto the partition from a floppy.

It is convenient to give partitions Boot Names that identify their functions. The Boot Names are installed with the WFORMAT program as described in the next section. Note that the Boot Name is independent of any Volume name set up by the operating system.

You must be in the SAGE Debugger to boot to a partition . To get to the debugger from the p-System command line type **"H"**. A **">"** should appear on your screen. Now you can use any of these commands:

IH Boots to the default partition, drive 0, partition 1. This is the same partition that will boot on power-up or RESET if SW6 and SW5 (port A) are set up and down respectively.

IHx Where x is drive number 0 through 3. This boots to partition 1 on the drive specified.

IHx #n Where x is the drive number 0 through 3 and n is the partition number 1-9,A,B,C,D,E,F where A through F represent partitions 10 through 15 respectively.

IH NAME Defaults to drive 0, where NAME is the partition Boot name.

IHx NAME Where x is the drive number (0-3) and NAME is the Boot Name of the partition.

IHRx #n The RAMDISK boot command IHR may be used once the system has already been booted to a partition once. It is generally used to recover information in RAMDISK if the user accidently left his operating system. IHR does not clear RAM Disk (the IH commands do). The system will boot to the RAMDisk using the BIOS and INTERPRETER on the partition given but will not change any of the information in RAM Disk. Note that if the partition was left due to a program crash, RAM Disk may no longer contain valid files and the boot may fail.

IFx There are boot commands for the floppy drives which are similar to the Winchester boot commands. They are defined in the SDT/ASSEMBLER MANUAL.

If you have many partitions, it may be more convenient to set port A switches SW6 and SW5 down so that a power-up or RESET will always boot to the Debugger, allowing you to start up whatever partition you need.

Note also that the p-System cannot currently boot to a device which has a logical channel number greater than 12. This is because until SYSTEM.MISCINFO is read in, the p-System assumes the maximum available volume is twelve.

Configuring the Bios Channel Map

The BIOS Channel Map for a Single-User system is maintained by the SAGE4UTIL program. Each user in a Multi-User system has a different BIOS Channel Map.

These maps are maintained using the MU.UTIL program. Both SAGE4UTIL and MU.UTIL have split control of the 32 Logical Device Numbers assignments into two menu screens (0 to 15, and 16 to 31). Under SAGE4UTIL the BIOS Channel Map is reached under the "Low Level Configuration" menu. Under MU.UTIL the BIOS Channel Map is reached from the User Configuration menu. A sample menu screen is shown below.

```
                    BIOS Channel Map (0 to 15)

A - Channel  0 device =   0     Q - Channel  0 subdevice =  0
B - Channel  1 device =   1     R - Channel  1 subdevice =  0
C - Channel  2 device =   2     S - Channel  2 subdevice =  0
D - Channel  3 device =   3     T - Channel  3 subdevice =  0
E - Channel  4 device =   4     U - Channel  4 subdevice =  0
F - Channel  5 device =   5     V - Channel  5 subdevice =  0
G - Channel  6 device =   6     W - Channel  6 subdevice =  0
H - Channel  7 device =   7     X - Channel  7 subdevice =  0
I - Channel  8 device =   8     Y - Channel  8 subdevice =  0
J - Channel  9 device =   9     Z - Channel  9 subdevice =  1
K - Channel 10 device =   9     0 - Channel 10 subdevice =  2
L - Channel 11 device =  11     1 - Channel 11 subdevice =  0
M - Channel 12 device =   9     2 - Channel 12 subdevice =  3
N - Channel 13 device =   9     3 - Channel 13 subdevice =  4
O - Channel 14 device =   9     4 - Channel 14 subdevice =  5
P - Channel 15 device =   9     5 - Channel 15 subdevice =  6

Select Menu item <CR exits, ! aborts>:
```

Note that there are two columns, one labeled "Channel x device =" and the second labeled "Channel x subdevice =". The "Channel x" portion indicates the Logical Device Number position in the menu. The "device=" in the first column refers to the Physical Device Number which is currently assigned to the Logical Device Number. The "subdevice =" in the second column refers to the Winchester Partition number for any Winchester device (Physical Device equal to 9). For all other Physical Devices the subdevice assignment is ignored.

In order for a Parition to be visible to an operating system, the partition must be included in the BIOS Channel Map. For single-user systems this means that the SYSTEM.BIOS file on the operating system's boot device must be configured with the desired Partitions. Remember that each single-user boot partition will contain a copy of SYSTEM.BIOS (potentially with different configurations).

For Multi-User systems this means that each User Task's Channel Map must be independently configured to include the desired Partitions.

III.05 PLANNING DISK PARTITIONS :

The Winchester disk is set up by the Factory with 3 partitions per drive. One is the system partition, the other two are work areas. Refer to the section on "FACTORY INSTALLED PARTITIONS" for details. Once you have determined that your system is working properly, you may want to define your own partitions.

> NOTE: If you do not feel comfortable using the Filer and Editor yet, do not attempt this. The GETTING STARTED MANUAL contains pointers for becoming familiar with the system.

Partitioning is done with the WFORMAT program and is fairly easy. You are simply telling the system how many pieces or "partitions" the disk is divided up into and how large they are. However, although it is easy to assign the partitions, it is time-consuming to load and unload the data from the different areas being changed. The normal process is to:

1. Back-up all the files by transferring the data onto floppys or other media.

2. Change the partitions with WFORMAT. The following sections show how to do this.

3. Re-transfer the information from the back-up into the new areas.

If the initial partitioning is thought out with care, you can avoid a somewhat tedious process later on.

III.06 PARTITION USAGE CONSIDERATIONS :

The Winchester Partition Overview listed most of the reasons that a disk might be split into different areas. The two most important reasons to partition a Winchester disk are to support large (greater than 16 Megabyte) disks under the p-System and to support multiple user areas when using the Multi-User system.

The p-System has a Subsidiary Volume facility which may be used to internally subdivide a device by treating files as p-System Volumes. This is convenient because the restriction on the number of files in a directory, 77, would otherwise limit the usefulness of a large device. Individual users may use the Subsidiary Volume feature rather than the Winchester Partition feature to arrange their files into groups.

When a Single-User system is used by more than one person, it is often desired that each user should have a separate area for file storage. The p-System Subsidiary Volume feature may be used for this purpose, however it might be safer to keep the user areas totally separate by using different Winchester Partitions.

Different operating systems may be supported on the same Winchester drive providing that they use a SAGE developed BIOS or use the standard partitioning technique. Some operating systems may not run together under the SAGE Multi-User BIOS, however, they may still be stored on the same disk and run separately.

The SAGE Multi-User BIOS has many factors which affect the way that the system is used. Such aspects as the information sharing requirements and the application program capabilities to handle shared data will affect the number of required partitions. Sophisticated users should digest the information contained in the chapter on Multi-User operation to determine the Winchester Partition requirements.

Partition Names

You can assign a **BOOT NAME** to each partition. This name can be up to 8 characters long and will allow you to boot to that partition using the IH command of the PROM Debugger. The name is optional as not all partitions need to be bootable and you do not have to boot by name.

Note that this name is completely independent of any other name given to the partition by an operating system.

Partition Size

Next, decide **HOW BIG** you want each partition. Partition size is defined by the number of BLOCKS, where each block is 512 bytes.

You cannot create partitions to the EXACT number of blocks you may want. This is because partitions are defined on disk TRACK boundaries. A track holds 9.5K bytes, which is 19 blocks. WFORMAT will round upwards to the nearest 19 block boundary, so you can expect your size to increase by at most 18 blocks. It is best to make the size a multiple of 19 in the first place.

The only area of the disk which cannot be part of a user partition is TRACK 0. The first user partition can start at TRACK 1. TRACK 0 is part of Partition 0 and contains the Partition Map and Bad Track Map.

> NOTE on the BAD TRACK MAP: If you are not familiar with the concepts involved in "bad track" mapping, do not be concerned. Your disk is not bad. Disk Manufacturers cannot afford to produce perfect media, so software techniques are used to map around media bad spots. The section on "BAD TRACK MAPPING" explains the SAGE implementation of this technique.

Because it contains the maps, Partition #0 must not be changed. Partition #0 also defines the WHOLE disk and overlaps all of the other partitions on it. It is used for formatting and testing the disk.

This table shows the total size in Cylinders, Heads, Tracks, Blocks and Bytes for the currently supported drives. The last column, User blocks is the number of blocks available to the user after an allowance is subtracted for Bad Tracks and 19 blocks for the Map itself.

Disk Type	Tot Cyl	Tot Hds	Total Trks	Total Blocks	Total Bytes	User Blocks
6M	306	2	612	11,628	5,953,536	11,457
12M	306	4	1224	23,256	11,907,072	22,933
18M	306	6	1836	34,884	17,860,608	34,409
40M	512	8	4096	77,824	39,845,888	77,197

Four tracks per head (76 blocks per head) are typically reserved as spares for bad tracks. Typically, drive manufacturers guarantee no more than 4 bad tracks per head. Users who have drives with less than the reserved number of tracks may want to utilize this area and can adjust the size accordingly. However, SAGE does not recommend this as several possible methods of automatic error recovery could be done using the extra tracks.

● Worksheet Example

This table shows an example of a typical "worksheet" for planning 3 partitions for a 12M (12 megabyte) drive.

Drive#	Partition#	Name	Size in blocks	
0	0	-----	19	(other blocks overlap the user areas)
0	1	SYSTEM	3876	(204 x 19 tracks)
0	2	SALLY	11324	(596 x 19 tracks)
0	3	JOHN	7733	(407 x 19 tracks)
reserved for spare tracks			304	(16 x 19 tracks)

			23256	Total

This information (in slightly different form) will be needed for the FORMATINFO file described in the WFORMAT section. Remember that the number of blocks which should be reserved for spares is:

Reserved Blks =(Hds) x (4 spare trks/hd) x (19 blks/trks)

III.07 INSTALLING WINCHESTER PARTITIONS :

The previous sections have described what a partition is, how large they can be, naming conventions and the operation of "booting".

Once you have finalized your definition of how the disk is partitioned, the BIOS program and the operating system must be "configured" for your choices. This is also called "installing" the partitions.

This section details the steps of the installation process. The MULTI-USER INSTALLATION EXAMPLE on page 347 may also be helpful.

However, before we itemize the major steps in installation, a few definitions are needed, mainly what is the FORMATINFO.TEXT file and WFORMAT.CODE.

● FORMATINFO and Partitions

The original FORMATINFO.TEXT file contains all of the information about YOUR hard disk. It defines the standard factory partitions and also any bad blocks that your specific system "maps" around. (See page 71 for an explanation of this technique.) Because the file is specific for each system, we recommend keeping an exact copy of it. The MASTER BUILD diskette which comes with your system has a copy of this file so be sure to make backup of it. (See the GETTING STARTED MANUAL.)

NOTE: If you have more than one SAGE IV system, you can find the correct set of MASTER diskettes by comparing the serial number on the bottom of the machine with the serial number on the label of the diskette.

A complete definition of FORMATINFO.TEXT starts on page 64.

● WFORMAT and Partitions

WFORMAT.CODE is a utility program for the Winchester disks. It performs several "house-keeping" functions for the Winchester such as formatting, bad track mapping, etc. It is described in detail starting on page 62.

The major function needed for installing partitions is 'Update Device Maps' (see page 77). Once the FORMATINFO.TEXT file is created, this option of WFORMAT reads the file and loads the information on the first track of the disk where it is used by the BIOS.

Installation Example

At this point you should be somewhat familiar with FORMATINFO.TEXT, WFORMAT, SAGE4UTIL and/or MU.UTIL before attempting to install the new partitions.

Although, there are several steps shown below, there are really only two **major** parts to the installation process; creating a new FORMATINFO.TEXT file and loading it onto the disk with WFORMAT.CODE.

This example assumes that you are using the standard p-System editor.

Installing the Partition Map

Save your existing Winchester disk files (if any) on floppies or another disk drive.

Copy the current FORMATINFO.TEXT file to a new name with the Filer to create a new FORMATINFO file as shown.

```
SCREEN SHOWS:                  YOU TYPE

C(ommand.....................  F
F(iler.......................  T
Transfer what file?..........  FORMATINFO.TEXT <cr>
To where?....................  NEWINFO.TEXT <cr>
F(iler.......................  G
Get what file?...............  NEWINFO.TEXT <cr>
Text file loaded
F(iler.......................  Q
C(ommand.....................  E
>Edit........................  (edit commands)
```

Edit it for the new partitions you have planned. The section on "THE FORMAT INFORMATION FILE" page 64 explains how to set up the file.

Quit the editor and save the file, Execute WFORMAT, give it the name of the new FORMATINFO file:

```
SCREEN SHOWS:                    YOU TYPE

Edit.............................  Q
(UPDATE MENU)....................  U
C(ommand.........................  F
F(iler...........................  S
Save as NEWINFO.TEXT?............  Y
F(iler...........................  N
Throw away current workfile?.....  Y
Work file cleared
F(iler...........................  Q
Command..........................  X
Execute what file?...............  WFORMAT

Winchester formatter Version x
File with Device Maps:      ....   NEWINFO
```

WFORMAT will check the file. If there were mistakes, use the Editor to correct them. If the file was ok, then:

```
SCREEN SHOWS:                    YOU TYPE

Winchester Drive Selection

   A - Drive 0
   B - Drive 1
   C - Drive 2
   D - Drive 3

Select Menu item..... A

     Function Selections

A - Format complete disk
B - Format selected tracks
C - Verify complete disk
D - Verify selected tracks
E - Update Device Maps
F - Display partition map
G - Display bad track map
H - Translate block to track
I - Format parameters

Select Menu item..... E
```

Use the "Update device maps" option to store the new file data to the map area. Quit WFORMAT by typing a **<cr>** .

Installing the System files

Now that the new Partition Map has been stored on track 0 of the disk, any properly configured BIOS will be able to access your new partitions. To continue the installation process, however, you must configure the BIOS channel Map in your on-line BIOS to make the new partitions available to it.

This means using SAGE4UTIL (via the On-Line Configuration, Low Level Configuration, and BIOS Channel Map) to configure an available logical device Number (such as #12) so that the new partition will be visible.

Once this has been done the new Volume (partition) may be zeroed (as #12:) and accessed normally by the p-System Filer. Now transfer on the desired files from floppy diskette or other visible Winchester Partition.

If the new partition is to be used as a boot device, a copy of SYSTEM.BIOS must be placed on the Partition and a Winchester bootstrap must be installed. Remember also to configure the BIOS Channel Map in the SYSTEM.BIOS copied to your new Partition so that it will also be able to access the new partition. Otherwise it will return a disk error when trying to boot because it will think that the device (partition) does not exist.

The following example shows how to copy a Winchester bootstrap to your new Partition (visible as Volume #12:).

```
SCREEN SHOWS:                          YOU TYPE

Command: E(dit,R(un,F(ile.............. X
Execute what file?..................... SAGE4UTIL <cr>

SAGE4UTIL version 1.3
Initializing....

     SAGE4UTIL
A On-line Configuration
B BIOS file Configuration
C Floppy Formatter
D Boostrap Copy
E Prepares Drives for shipping

Select menu item ...................... D

Source file or device <just CR quits>?.... SAGE.WBOOT.CODE
Ready to load bootstrap from file ..... Y
Bootstrap of 1008  bytes read successfully
Destination file or device<just CR quits>? #12:
Ready to load bootstrap on volume #12:.... Y

Bootstrap data written successfully
Type <space> to continue.............. <space>

Select menu item <cr exits, !aborts>:..... <cr>
```

For the p-System boot device a copy of SYSTEM.PASCAL, SYSTEM.MISCINFO, and SYSTEM.INTERP are also required on the device. Once the appropriate files have been placed on the new boot device the new system may be tried by rebooting to the new partition (See page 46).

IV WFORMAT - Winchester Utilities :

In addition to formatting the complete disk, WFORMAT allows several other functions including selective track formatting, verification (Bad Track Scan), updating the device and bad track maps, and several utility functions.

The List Partition Map function outputs a list of the drive partitions and their sizes in blocks.

The Translate Block to Track function allows the user to determine the physical cylinder and head number of a block within a partition. This is typically used for setting up a new bad track or reformatting a bad track. Note that in each case a block is 512 bytes of data.

Formatting, Verification and Mapping have already been done for you by SAGE and should rarely need to be redone. Re-structuring the partitions on the drive should be done very carefully making sure that all current files on the disk have been saved on floppies or another disk drive. See the section on "PLANNING DISK PARTITIONS" for detailed information.

All of the functions of WFORMAT work from data contained in the file: FORMATINFO.TEXT.

Because WFORMAT can destroy all data on the disk, it should be run with caution. It is generally executed from a bootable floppy. See the section on "BUILDING A WFORMAT DISK".

WFORMAT is written in USCD PASCAL. It uses the MNU_Unit, Config_Sage Unit, SIO_Unit, and Win_Unit which have been loaded into WFORMAT to make it stand-alone. These units are also available in SAGETOOLS.

IV.01 THE BUILD DISKETTE :

SAGE ships a "BUILD" diskette with every system. This disk is convenient for building the Multi-User system or reformatting your hard disk (if necessary).

You should make a copy of the original diskette to work with. Also, the SYSTEM.MISCINFO file and GOTOXY may not be correct for your terminal, as the standard distribution assumes the SAGE terminal. The GETTING STARTED MANUAL explains how to back up the diskette.

NOTE: The script files for changing to Televideo 925 or Freedom 100 operation are available on the BUILD diskette.

You will want to use your BUILD diskette to keep a copy of the FORMATINFO.TEXT files you may have customized for your system.

IV.02 THE FORMAT INFORMATION FILE :

When the WFORMAT program is first started, it requests the name of a file containing the Bad Track and Device Map (partition) information. This file is supplied by SAGE with each system under the name FORMATINFO.TEXT. Note that each system's FORMATINFO file may be different because of the possible different bad tracks and the different user partitions.

If the FORMATINFO.TEXT file is not specified, WFORMAT will default to the current configuration as read from the first track on the disk. If no configuration is present, it will prompt the user for the number of heads.

If you have more than one SAGE, you can find the correct factory FORMATINFO file by comparing the serial numbers on the MASTER diskettes with the number on the bottom of the SAGE.

If the FORMATINFO.TEXT file is not specified (type just a <CR>) then WFORMAT will default to the current configuration as read from the first track of the disk. If no configuration is present, it will prompt the user for the number of disk heads.

The FORMATINFO file is a standard .TEXT file created and maintained using the standard p-SYSTEM editor. WFORMAT checks the file when it is read in, giving error messages if mistakes are found. If correct, the new information can be loaded to the map area (TRACK 0) with a WFORMAT option.

The option in WFORMAT, 'Update Device Maps' loads the information from FORMATINFO.TEXT to the first track of the disk. See page 77.

The FORMATINFO.TEXT file can contain information on up to four drives.

Example of the FORMATINFO file:

```
Drive 0, Heads 4
  Bad Track Map
      Cylinder 201, Head 1
      Cylinder 202, Head 1
      Track=406
  Device Map
    Partition 0: Cylinder 0, Head 0  to  Cylinder 305, Head 3
    Partition 1: Cylinder 0, Head 1, Blocks = 1280
                 Name = Pascal
                 System = UCSD
    Partition 2: Blocks = 1280
                 System = UCSD
    Partition 3: Cylinder 34, Head 1 to Cylinder 301, Head 3
                 System = UCSD
```

Either Upper of Lower case is acceptable.
All commas and Equal signs are optional.
Indentation is not required.

The "Bad Track Map" and "Device Map" are unique phrases with single spaces between words of phrase.
The "Bad Track Map" (if present) must precede "Device Map".

Bad Tracks may be specified with these assignments:
"Cylinder=x, Head=x"
or "Track= x"
Partitions may be specified with these assignments:
"Cylinder=x, Head=x to Cylinder=x, Head=x"
or "Cylinder=x, Head=x Blocks=x"
or "Blocks=x"

The drive definition starts with a line which identifies which drive it is and how many heads it has:

```
Drive = 0, Heads 4      ( The equal sign is optional.)
```

Next comes "Bad Track Map". This section identifies bad areas of the disk drive. This area is set up for you by the SAGE factory and will usually never need changed. Refer to the section on BAD TRACK MAPPING on page 71 for details. Bad Tracks may be specified with these assignments:

```
    Cylinder=x, Head=x
or Track= x
```

This table shows the sizes of the currently supported drives. Note that:

(Trks per drive) = (Cylinders) x (No. of heads)

Disk Type	Tot Cyl	Tot Hds	Total Trks	Total Blocks	Total Bytes	User Blocks
6M	306	2	612	11,628	5,953,536	11,457
12M	306	4	1224	23,256	11,907,072	22,933
18M	306	6	1836	34,884	17,860,608	34,409
40M	512	8	4096	77,824	39,845,888	77,197

Track numbers are assigned sequentially according to the number of heads on a cylinder. A track number can be calculated by the equation:

TRK = (Cylinder * HeadsPerCylinder) + Head.

After the Bad Track definition comes "Device Map". This section identifies the partitions, their names, size and operating system. Partitions may be specified in any of three ways:

```
   Cylinder=x, Head=x to Cylinder=x, Head=x
or Cylinder=x, Head=x    Blocks=x
or Blocks=x
```

If the partition size is specified in blocks, the partition is started at the end of the previous partition. The first partition (#1) will start at Cylinder 0, Head 1.

If partition #0 is specified, it must include the whole disk size as given in cylinders and heads in the BIOS configuration. If not specified, it will default to the size configured in the BIOS.

The "Name" field is optional. Up to 8 characters can be specified in the name which can be used when bootstrapping the system from PROM. If the partition is not a boot partition, a name is not needed.

The optional "System" field indicates the type of Operating System resident on the partition. This field may be used by a Multi-User Logon program.

SYSTEM NAME	REPRESENTS OPERATING SYSTEM:
UCSD	p-System operating system
CPM	CP/M-68K operating system
MODULA	Volition's Modula 2 environment
HYFORTH	HYPERFORTH Plus environment
PDOS	PDOS operating system
MIRAGE	Mirage operating system
BOS	BOS operating system
IDRIS	IDRIS operating system

Either Upper or Lower case is acceptable. All commas and Equal signs are optional. Indentation is not required. The two fields, " Bad Track Map" and "Device Map" are optional if no map information is provided. Any combination of the four drives may be specified.

IV.03 FORMATTING AND VERIFICATION :

Normally, the Winchester will only need to be formatted once during production. However, if hardware or software problems cause one or more tracks to become unreadable, re-formatting will be required. Therefore, it is advisable to keep a bootable diskette which contains the WFORMAT utility and a file containing the Drive Partition Assignments and the Bad Track Map.

Although the Winchester Controller does not utilize traditional sector formatting information, a formatting process is still required.

The formatting process exercises the seek mechanism and then delays before starting to format. The Winchester driver always reads a track before it writes back new information. This is necessary to verify the seek and to perform partial track updates.

The track number is recorded at the beginning of each track's data and a CRC is stored at the end of the data.

During formatting, each track is written with a data pattern (typically all E5H bytes). Also at the time of formatting a bad track map and a device partition map are stored on track 0 of the drive for reference by the BIOS and the PROM boot drivers.

FORMATTING AND VERIFICATION

The formatting and verification passes of WFORMAT each output a '.' for each track that is processed. If an error occurs, a special character will be printed out (instead of a dot) to indicate the problem.

```
*   Error on a mapped out track
X   CRC error
?   Unexpected or unknown error type
0   Could not initialize VCO
1   Seek/Recalibrate failure
2   Drive not ready
3   Timeout waiting for data
4   Unused
5   Write protected
6   Block address out of range
7   Wrong cylinder found
8   Bad Device number
```

The routine will attempt to format and verify tracks which have been indicated as bad but it will put an '*' for the mapped tracks if an error occurs. At the completion of the process the routine will ask if the new bad track information should be displayed. If the user replies with a 'Y', the cylinder and head of each new bad track (not already in the map file) will be presented.

Each format process will automatically verify the area which was formatted. The format process will also query the user to insure that he understands that data will be destroyed. The Bad Track and Device Maps will be written to the disk automatically at the end of a successful format and verify of the complete disk. Alternately, this data may be written to the disk via the Update Device Maps command in the main menu.

IV.04 BAD TRACK MAPPING :

All Winchester drive manufacturers sell drives that contain a limited number of known bad tracks. This is because the production of perfect media would cost too much. Computer manufacturers are expected to "map around" these bad tracks in their system software. The drive manufacturer provides a list of known bad tracks. Track 0 is always guaranteed to be good so there is a common place to store the bad track information.

There are many ways of mapping bad disk areas. SAGE chose to skip the bad tracks and re-number the remaining tracks in order to keep the disk transfer speed up. Your system comes already mapped and formatted. As tracks do not generally go bad in the field you may never have to re-format or update your bad track map.

One implication of this type of mapping is that adding a bad track moves all of the following logical track numbers forward. For Example, this figure represents the disk before mapping:

```
Physical trk   ..100 101 102 103 104 105 ... End Spare1 Spare2..
Logical  trk   ..100 101 102 103 104 105 ... End
DATA           .. Da  Db  Dc  Dd  De  Df ...  D*
```

If physical track 103 was mapped out, the disk is represented:

```
Physical trk   ..100 101 102 103 104 105 ... End Spare1 Spare2..
Logical  trk   ..100 101 102 *** 103 104 ... ... End
DATA           .. Da  Db  Dc  Dd  De  Df ...  D*  xx
```

Note that data Dd is lost and data De and Df now reside on a logical track one off from their previous track.

It would be possible to write a utility to move the data back to conform with the correct track. (That is, move D* to xx, ... De to Df, Dd to De etc.) Some user interaction would still be required to account for the lost data. At this time, no such utility exists. Users should back up their disk files to floppies or another disk drive, map the bad track and reload their files.

IV.05 RUNNING WFORMAT :

To run WFORMAT type:

"X" for Execute then "WFORMAT"

The screen will display:

```
Winchester Formatter Version 1.2

File with Bad track and Device Maps:
```

At this point, you must type in the name of a FORMATINFO file. This can be one that you have created with the Editor or the original file built for you by SAGE. This original file is on the BUILD diskette shipped with the system and will be called FORMATINFO.TEXT.

If the FORMATINFO file has been incorrectly set up, the incorrect line will be shown with one of the following error messages below it. A "^" will point to the mistake in the line.

```
Illegal value
Command syntax error - Number expected
                       Head expected
                       Name expected
                       Illegal system name
                       Track expected
                       Blocks expected
                       Illegal command
                       Bad track Map must precede Device Map
```

If an error occurs, quit WFORMAT and correct the file using the Editor. Re-execute WFORMAT.

If the FORMATINFO.TEXT file is not specified (type just a <cr>), then WFORMAT will default to the current configuration as read from the first track of the disk. If no configuration is present, it will prompt the user for the number of disk heads.

Once the FORMATINFO file has been accepted, this menu will appear: (NOTE: WFORMAT uses the same menu unit MNU_UNIT described under SAGE4UTIL)

```
Winchester Drive Selection

A - Drive 0
B - Drive 1
C - Drive 2
D - Drive 3

Select Menu item <CR exits, ! aborts>:
```

Select one of the four physical drives that can exist on the SAGE IV. The Function Menu will display:

```
      Function Selections

A - Format complete disk
B - Format selected tracks
C - Verify complete disk
D - Verify selected tracks
E - Update Device Maps
F - Display partition map
G - Display bad track map
H - Translate block to track
I - Format parameters

Select Menu item <CR exits, ! aborts>:
```

Some options (= and /) do not appear on the screen. These options are used to print the screens or write them to a file. Refer to the section on menu operation in the SAGE4UTIL section of the GETTING STARTED MANUAL or the MU.UTIL section, page 154.

A... Format complete disk

The Winchester disk is shipped formatted from the factory. Rarely will you need to do this. All data on the disk will be destroyed so you must save your files on floppy (or to another physical drive) before doing the format if possible.

When "A" is typed WFORMAT will ask:

```
Do you really want to destroy information on Winchester Drive 0,
Cylinder x Head x to Cylinder x Head x?
```

Type "Y" to begin the formatting process.

B... Format selected tracks

Usually, it is not necessary to reformat the entire disk if a problem has occurred in one area. This option allows you to reformat selected areas of the disk. See the notes on "RECOVERING TRACKS". Define the area to be reformatted by typing the letter of the parameter and then the value. Close with a carriage return.

```
      Selective Format

A - Starting cylinder    306
B - Starting head          0
C - Ending cylinder      306
D - Ending head            0
E - Do the format

Select Menu item <CR exits, ! aborts>:
```

When "E" is typed WFORMAT will ask:

```
Do you really want to destroy information on Winchester Drive 0?
Cylinder x Head x to Cylinder x Head x?
```

C... Verify complete disk

The verify option reads the disk and displays a code (see page 70) depending on the results of the read. The Verify will not harm any data on the track. The display will look like:

```
Verify Pass
<trk> ........................................
<trk> ........................................

Verify successful
Type <space> to continue
```

If errors occurred during the verify, the physical cylinder and head numbers of the tracks are displayed on the screen. If an output file is open (menu option "= filename "), they are written to the file.

D... Verify selected tracks

This option verifies only selected tracks. It works like option "C" except that the starting and ending tracks must be defined. Type "E" to start the verify.

```
      Selective Verify

A - Starting cylinder    306
B - Starting head          0
C - Ending cylinder      306
D - Ending head            0
E - Do the verify

Select Menu item <CR exits, ! aborts>:
```

E... Update Device Maps

This is used to load the new map information to the disk map area (track 0) from the FORMATINFO file specified at the start of WFORMAT.

F... Display partition map

The current partition information is displayed. If an output file is open, then it is written to that file:

```
Drive 0:  23237 usable blocks

                                                  Logical track range
Partition  Name          Blocks    Category     Cyl Hd  to  Cyl Hd

    0                    23256                  0  0      305  3
    1      PASCAL        23237       UCSD       0  1      305  3
```

The number of usable blocks is calculated from the total disk size minus 19 blocks for the maps in track 0 and 19 blocks for each bad track.

Note that the partition boundaries are shown by LOGICAL cylinder/head not by physical cylinder/head.

Also, the number of usable blocks shown indicates the total amount of disk blocks minus track zero and the number of bad tracks actually specified. It includes other spare tracks which should be reserved for future bad tracks as described on page 52.

G... Display bad track map

If the drives have any bad tracks they will be displayed to the screen and written to the output file (if open). The bad tracks are shown by PHYSICAL cylinder/head.

H... Translate block to track

The "H" option is used to translate a partition block number to a PHYSICAL cylinder/head number. For example, if the Bad Block check of the p-System Filer told you block 500 was bad, you could convert:

```
Partition number:1    (a number from 1-9,A-F)
LOGICAL block number:500

Physical Track is 27  (Cyl 6 head 3 )
```

Now you can use the Physical track number to try recovering the track. See "RECOVERING TRACKS".

I... Format parameters

This option shows two format parameters:

```
        Format Parameters

A - Data pattern          E5
B - Retries                0

Select Menu item <CR exits, ! aborts>:
```

The formatter writes the specified data pattern to every byte in the area being reformatted. Certain operating systems (such as CP/M) expect an initialization pattern. The E5 pattern works with all operating systems supported by SAGE.

The number of retries should not be changed by the user. If an area cannot be formatted and verified with no retries, it should be considered suspect and marked in the BAD TRACK MAP.

IV.06 RECOVERING TRACKS :

Normally, bad disk tracks are identified by the disk drive manufacturer and are due to defects in the disk surface. If proper shipping and handling precautions are taken, disk tracks do not normally go bad in the field. This sections describes what steps to take should this happen.

Usually the first indication of a bad area is the failure to read a file. If your operating system has a verification test built in, run it to determine what logical block has the problem. For the p-System and Modula II run the "Bad block" option of the Filer.

Execute WFORMAT. Type in a <cr> for the name of your standard FORMATINFO file.

If you do not know the number of the partition, type "F" to display the current partition map. The partition number is expressed in hexadecimal (1-9, A-F). Type a <cr> to return to the Function menu.

Use option "H" to convert the block number to the physical track number.

Use option "B" to reformat JUST that track. Specify the cylinder/head of the bad track as both the start and end of the format area. Note that the information in this track (19 blocks) will be destroyed by the format process.

Verify the track with option "D" a few times. If the verify works, the track does not have to be entered into the BAD TRACK MAP but can be used as normal although the data was lost.

See the section on "BAD TRACK MAPPING" if the track must be mapped out.

V p-SYSTEM OPERATING SYSTEM :

Development of the p-System started in 1973 at the University of California, San Diego on the school's Burroughs B6700 computer. It grew from a simple support package for an introductory Pascal programming class to a user-oriented interactive system available on many different systems and sold by SOFTECH MICROSYSTEMS. Two more languages, FORTRAN and BASIC are now supported by the operating system.

The heart of the p-System is the concept of a "p-machine". The system compilers convert the source language text to "p-code" which is code for this abstract "p-machine". The 68000 INTERPRETER emulates the p-machine for the SAGE II/IV at run time.

The INTERPRETER and the BIOS are the only two parts of the entire p-System that must be re-written for each new machine. With this concept, the p-System can be easily put on any machine although its performance will, of course, be better on a 16-bit machine than on an 8-bit machine.

Programs written in Pascal, FORTRAN, or Basic on one p-System are easily moved to a different machine with the p-System. This "portability" is one of the major advantages of the p-System. Many SAGE II/IV users have ported their software in a matter of hours - with most of the time spent transferring their program over the serial port from their old machine to the SAGE. (The section on COMPUTER COMMUNICATION page 223 explains how to do this.)

The p-System supports a powerful screen-oriented EDITOR which has many word-processing features such as "Filling" where lines of text are combined together to make the right edge of the text as nearly even as possible. The p-System OPERATING SYSTEM MANUAL explains all the commands of the Editor, the Filer, and other various utility programs.

Most sections of this manual will give specific information about how the p-System relates to the topic being discussed in the section.

V.01 FILES REQUIRED FOR BOOTING :

For a standard **single-user** system, there are certain files which must exist on a device which is used for Booting up the p-System on a SAGE II/IV computer. These files are:

SYSTEM.BIOS	Hardware device drivers.
SYSTEM.INTERP	p-code Interpreter.
SYSTEM.PASCAL	Operating System.
SYSTEM.MISCINFO	System configuration information.

Also required is a copy of the System Bootstrap in device block zero which does not show up in the directory as a file. This area may be transferred with a 'device to device transfer' by the Filer or by using the SAGE II/IV System utility SAGE4UTIL.CODE.

Note: When booting to RAM Disk, the files SYSTEM.BIOS and SYSTEM.INTERP must be on the floppy but need not be copied to RAM Disk (they can follow the ENDBOOT file). The files SYSTEM.PASCAL and SYSTEM.MISCINFO must be copied to the RAM Disk for successful operation.

Note also that the p-System cannot currently boot to a device which has a logical channel number greater than 12. This is because until SYSTEM.MISCINFO is read in, the p-System assumes the maximum available volume is twelve.

boot a p-System partition under the SAGE **Multi-User** system, the user's system device must contain all the files listed above except the file SYSTEM.BIOS. The BIOS function is replaced by the Multi-User Environment which has already been loaded. The user's system device must contain the operating system specific bootstrap code (MU.PBOOT.CODE for the p-System) installed in the bootstrap area. The same Multi-User p-SYSTEM bootstrap works for both floppy and Winchester boot devices.

Other files which are not required for booting, but if used, are accessed from the boot device are:

SYSTEM.LIBRARY
SYSTEM.SYNTAX
USERLIB.TEXT

The following major system files can be found by the operating system on any on-line file structured device.

SYSTEM.FILER
SYSTEM.EDITOR
SYSTEM.COMPILER
SYSTEM.ASSMBLER
SYSTEM.LINKER

V.02 THE USER MASTER DISKETTE :

The BUSINESS MASTER is a good example of a USER MASTER diskette. It contains only those system files needed to boot and run the programs on it. The GOTOXY, SYSTEM.MISCINFO and BOOT have been installed.

The INSTALLATION section of the GETTING STARTED MANUAL describes how to build a USER MASTER and the files normally needed.

V.03 REAL ARITHMETIC PRECISION :

The p-System currently supports either 2 word (32 bit) or 4 word (64 bit) real arithmetic, but not at the same time. The user must determine which version of arithmetic to use when configuring the system diskette. The 2 word arithmetic runs faster and requires less storage space but does not maintain as many significant digits. The distribution diskette is shipped configured for 4 word arithmetic.

Several steps must be done for the correct installation of the chosen real arithmetic. A checklist for the common installations is given in this section. Specifically, the interpreter, compiler, library and REALOPS must all be two word or four word or the error "EXECUTIVE ERR#17" may occur.

Refer also to the p-SYSTEM OPERATING SYSTEM MANUAL, APPENDIX E, page a-6, 'FLOATING POINT PACKAGES'.

Interpreter Files

The Interpreter contains the **primitive** real arithmetic routines. There are the interpreter files:

INTERP.0.CODE	no real arithmetic
INTERP.2.CODE	two word reals
INTERP.4.CODE	four word reals (Standard)

The correct file must be transferred to SYSTEM.INTERP to become effective. The SYSTEM.INTERP on the distribution diskette is a copy of the file INTERP.4.CODE. Enough room is allocated in the system memory assignment for any of the interpreters so no memory is saved by using the smaller file. The user does not need the INTERP.x.CODE files on his USER MASTER.

Realops Files

The more complex real arithmetic routines, such as transcendental functions, are contained in the files:

REALOPS.2.CODE	two word arithmetic
REALOPS.4.CODE	four word arithmetic

One of these files resides in the SYSTEM.PASCAL operating system. The LIBRARY program is used to install a different file. The standard distribution has REALOPS.4.CODE installed. The USER does not need the REALOPS files on his USER MASTER.

Pascal Files

There is only one version of the p-System Pascal compiler distributed as SYSTEM.COMPILER. This compiler defaults to generating code according to the precision of the arithmetic of the installed interpreter. Code may be generated for a specific precision by using the (*$R2*) or (*$R4*) compiler options.

If using another compiler such as FORTRAN or BASIC, SYSTEM.COMPILER should be renamed PASCAL.CODE.

A two word PASCAL system must be set up:

```
INTERP.2.CODE   --->  SYSTEM.INTERP
REALOPS.2.CODE  in    SYSTEM.LIBRARY
                      SYSTEM.COMPILER
```

A four word FORTRAN system must be set up:

```
INTERP.4.CODE   --->  SYSTEM.INTERP
REALOPS.4.CODE  in    SYSTEM.LIBRARY
                      SYSTEM.COMPILER
```

FORTRAN Files

Pascal programs do not require a special runtime library. However, the FORTRAN 77 and BASIC languages each have different runtime library routines which must be made available to the system to run their generated programs. These routines are dependent on the real arithmetic. The runtime library files for FORTRAN 77 are :

```
FORTLIB2.CODE   2 word FORTRAN library
FORTLIB4.CODE   4 word FORTRAN library
```

One of these library files must be installed in SYSTEM.LIBRARY using the program LIBRARY.CODE if FORTRAN programs are to be run. An example of using LIBRARY is given in the GETTING STARTED MANUAL and also in the p-System Operating System Manual.

The FORTRAN 77 and BASIC compilers each have 2 and 4 word versions. The FORTRAN compiler files are:

```
FORTRAN2.CODE   two word compiler
FORTRAN4.CODE   four word compiler
```

A two word FORTRAN system must be set up:

```
INTERP.2.CODE   --->  SYSTEM.INTERP
REALOPS.2.CODE  in    SYSTEM.LIBRARY
FORTRLIB2.CODE  in    SYSTEM.LIBRARY
FORTRAN2.CODE   --->  SYSTEM.COMPILER
```

A four word FORTRAN system must be set up:

```
INTERP.4.CODE   --->  SYSTEM.INTERP
REALOPS.4.CODE  in    SYSTEM.LIBRARY
FORTRLIB4.CODE  in    SYSTEM.LIBRARY
FORTRAN4.CODE   --->  SYSTEM.COMPILER
```

BASIC Files

The runtime library files for BASIC are:

```
BLIB.R2.CODE    2 word BASIC library

BLIB.R4.CODE    4 word BASIC library
```

The appropriate runtime library for the language and arithmetic being used should be installed into the SYSTEM.LIBRARY file with the LIBRARY program. The user does not need the FORTLIB or BLIB files on his USER MASTER.

The BASIC compiler files are:

```
BASIC.R2.CODE  2 word BASIC compiler
BASIC.R4.CODE  4 word BASIC compiler
```

A two word BASIC system must be set up:

```
INTERP.2.CODE   --->  SYSTEM.INTERP
REALOPS.2.CODE  in    SYSTEM.LIBRARY
BLIB.R2.CODE    in    SYSTEM.LIBRARY
BASIC.R2.CODE   --->  SYSTEM.COMPILER
```

A four word BASIC system must be set up:

```
INTERP.4.CODE   --->  SYSTEM.INTERP
REALOPS.4.CODE  in    SYSTEM.LIBRARY
BLIB.R4.CODE    in    SYSTEM.LIBRARY
BASIC.R4.CODE   --->  SYSTEM.COMPILER
```

V.04 RAM DISK OPERATION :

Under RAM Disk operation, an area of RAM memory may be set up with a directory just like a disk device. Files may be transferred to the RAM Disk device and in general it acts like a diskette except that the information is lost on a power down or hardware reset. The RAM Disk is accessed through device #11:. The configuration option under SAGE4UTIL is used to enable or disable the RAM Disk feature.

The SAGE4UTIL program may also be used to set up a single-user system for booting to the RAM Disk as the primary System Device. (Select RAM Disk then turn the Boot option ON.)

Under this option the bootstrap program creates a directory in RAM with the name RAMDISK: and with the maximum size available in the RAM disk memory (minimum required for booting is 64K bytes, 128 blocks). Then the bootstrap routine copies files from the diskette to the RAM Disk device until the RAM Disk is full or a file called ENDBOOT is found. The bootstrap then finds the operating system (SYSTEM.PASCAL) on RAM Disk and uses this device as the primary System Device. The floppy may then be removed if no other files are required.

A file called ENDBOOT may be used to prevent all floppy files from filling the RAM Disk device on booting. The only files required on the RAM Disk are SYSTEM.PASCAL and SYSTEM.MISCINFO. Make sure that these files occur near the beginning of the diskette so that they won't be left off during the copy. All other files may be left only on diskette or copied to RAM Disk as desired. To create the one block file called ENDBOOT use the M(ake command in the Filer with a filename of ENDBOOT[1].

The RAM Disk device (#11:) may be used as a storage device without booting to it. Device #11 may be zeroed and the number of blocks set using the Filer. This gives the user program access to up to another 340K (680 blocks) of RAM using file I/O. The amount of RAM generally available for RAM Disk on a single user system is given below:

with 1024K	RAM Disk size	= 1704 blocks
768K		= 1192 blocks
512K		= 680 blocks
384K		= 424 blocks
256K		= 168 blocks
128K		= ram disk not available

NOTE: Floppy base SAGE IV systems will have about 20 blocks more RAM Disk.

When using RAM Disk remember to back up any files written there as they will not be saved on power-down or reset. If any system problem causes a processor EXCEPTION Error back to the Debugger, the **RAM Disk files may be recovered** by using **IHR** or **IFR** to reboot. The directory and files will probably still be available on device #11: as long as you do not reset the SAGE II/IV.

MU RAM Disk Operation

In the Multi-User System, RAM may be split into four RAM DISKS. The size of each RAM Disk is defined using MU.UTIL.

Note that there may not be enough memory equipped on the machine to have **ANY** RAM DISKS. RAM DISKs are created from the memory left over after all of the user memory areas have been defined.

The MU.UTIL program is very similar to SAGE4UTIL. Selecting the RAM disks option brings up a preliminary menu which allows selection of one of the four possible RAM Disk devices. Then the user must assign the bottom and top limits of the RAM Disk area.

A RAM Disk which has a non zero Top specification greater than the base of the BIOS and system tables will be automatically disabled without any warning message. Therefore if a RAM Disk access returns a non- existent device error, it may be configured but require more space than is available.

V.05 p-SYSTEM DEVICES SUPPORTED :

The table on page 35 shows the standard system device numbers (Physical channel number) and their usage for all operating systems.

The table on the next page shows the p-System device assignments as set up by the SYSTEM.MISINFOs on the distribution diskettes.

Normally, the p-System is set up in the SYSTEM.MISCINFO file to allow only 6 block structured devices. The floppy disk drives, hard disk partitions and RAM Disk are all block structured devices. This means that they handle information in groups of 512 bytes at a time. (A block is 512 bytes in the p-System.) This is in contrast to the TERMINAL, REMOTE and auxiliary ports which are all serial devices which handle one byte at a time.

The physical channels (devices) must be assigned to "Logical channels" in the system using SAGE4UTIL or MU.UTIL.

The SYSTEM.MISCINFOs on the SAGE distribution disks are set up for 12 normal devices and 16 subsidiary volumes. The p-System program SETUP.CODE can be used to change this. The number of normal devices allowed is determined by the volume number assigned to the First Subsidiary Volume.

Note that if the system does not have two floppies or RAM DISK, the unused device channels may be configured to a Winchester Partition instead. This also gives access to more partitions.

P-SYSTEM DEVICE ASSIGNMENTS

Single User Physical Channel	Multi User Physical Channel	p-System Logical Channel	p-System Volume Number	p-System device name
1	1	1	# 1:	CONSOLE:
2	2	2	# 2:	SYSTERM:
3	3	3	# 3:	unassigned
4	4	4	# 4:	(name of volume):
5	5	5	# 5:	(name of volume):
6	6	6	# 6:	PRINTER:
7	7	7	# 7:	REMIN:
8	8	8	# 8:	REMOUT:
9	9	9	# 9:	(name of volume):
10	10	10	#10:	(name of volume):
11	11	11	#11:	(name of volume):
12	12	12	#12:	(name of volume):
			#13:	(name of subvol):
			|	
			#28:	(name of subvol):
13	14	28	#29:	SERIALA:
14	16	29	#30:	SERIALB:
15	18	30	#31:	SERIALC:
16	20	31	#32:	SERIALD:

Note that because the standard MISCINFO allows 16 subsidiary volumes, the SERIALA: through SERIALD: volumes have higher volume numbers. Allowing for more subsidiary volumes will push these numbers out further. See page 26 for more information.

V.06 p-SYSTEM BREAK :

The p-System Break key may be used to stop a program before the program would normally complete. Note that this key is normally not the key labeled BREAK on the terminal. The terminal's BREAK key normally will generate a low level serial channel framing error which is detected by the BIOS and either ignored or used to enter the PROM resident Debugger (see the terminal configuration under SAGE4UTIL in the GETTING STARTED MANUAL).

The p-System Break key is defined to the system by the file SYSTEM.MISCINFO and is maintained using the program SETUP.CODE. It is normally defined to be a NUL (byte value of 0) for most terminal configurations. A NUL character may usually be generated by typing CTRL @ (or in some cases CTRL space).

The p-System Break key may be inhibited by a program which desires to not allow a user to interrupt it. This is useful for instance in a program like the p-System Editor where an accidental Break could cause the loss of a lot of valuable data in memory.

The method for inhibiting Break is to set the variable SYSCOM©.MISCINFO.NOBREAK equal to TRUE in the p-System KERNEL. KERNEL variables are accessed by including a USES KERNEL; statement at the beginning of a Pascal program.

Note that this technique is dependent on the layout of variables in the operating system and may change in future versions of the system. When the p-System Break key has been inhibited in this manner, the Break character is passed through to the program.

Another method of disabling the p-System Break altogether is to define the character (with SETUP.CODE) as a character which cannot be received. In typical systems where the character mask is 7FH (127 decimal), the most significant bit of a received character byte can never be a one.

Therefore if the p-System Break character is defined as a value from 128 to 255 (decimal) it will never match a received character. This of course prevents Breaks in all programs.

Unfortunately, the p-System Break is currently not handled well by programs which use asynchronous processes. This includes the situation where the p-System Print Spooler has been enabled using SETUP.CODE. Typing a p-System BREAK will sometimes force the user to RE-BOOT the system. In order to prevent this RE-BOOT problem, the system has been modified to only recognize the p-System Break when the main program (Task) is running. IF other concurrent processes (Tasks) are running at the time the Break is typed, the Break will be ignored.

V.07 ASYNCHRONOUS I/O :

The asynchronous I/O feature for UNITREAD and UNITWRITE is currently only supported for the Winchester and Floppy drivers. Thus, bit 0 of the Control Word for other devices is ignored. The functions UNITBUSY and UNITWAIT are not implemented (even for the Winchester and Floppy drivers) The function UNITBUSY always returns false. The procedure UNITWAIT always returns immediately.

To specify an asynchronous transfer for the Winchester or Floppy, bit 0 of the control word (fifth parameter of UNITREAD or UNITWRITE) must be set on. This will cause the driver to initiate the transfer and then return to the caller before the transfer is complete.

Two methods are provided for determining transfer completion. If the user has attached to event number 48 (defined as ATT_Win in ATT_Unit), each Winchester disk transfer completion will signal the attached semaphore. Event number 49 (defined as ATT_Floppy) is signaled for each floppy transfer completion. The driver may also be polled to find out if it is still busy (see UNITSTATUS below).

At the completion of an asynchronous transfer the user should check the transfer error code. The Winchester and Floppy error codes are available via UNITSTATUS in a word at byte offset 52 of the result area. A zero word indicates no error. If the word is non-zero, the low byte of the word contains the error reply code from the BIOS (not p-System errors). Byte offset 50 contains the busy flag which will be zero if the driver is inactive and non-zero if the transfer is not yet complete. One should not attempt to start another transfer until the error from the first transfer has been read. Otherwise the error information may be lost.

Although the asynchronous transfer protocol is not available for the Keyboard, Terminal output, Printer output, Remote input and output, and Auxiliary Serial Ports, it should be noted that they all use interrupt driven drivers with 255 byte buffers. The procedure UNITSTATUS may be used to read back the number of characters in the interrupt buffers. On input from the Keyboard or Remote input channels, the UNITSTATUS procedure may be used to find if any characters exist before using UNITREAD which would otherwise wait for a character. On output to the Terminal, Printer, Remote, or Auxiliary Serial channel, UNITSTATUS can be used to determine if characters may be output with UNITWRITE without hanging to wait for the device.

V.08 p-SYSTEM WAITER TASK :

When asynchronous processes are used in a p-System program, and all the user tasks are dormant (waiting on semaphores), the system runs the lowest priority task in the system (resident in the Kernel) called Waiter.

Unfortunately, the Waiter task is normally compute bound because there is no definition in the interpreter/machine interface as to how to stop the processor. For most systems this is adequate because there is nothing to do anyway. However, in the SAGE Multi-User environment, idle programs should not be compute bound or the total system thruput will be affected.

The system has therefore been modified to detect the Waiter process and to stop the processor (or individual user) until the next Break or Event. This is done in the interpreter/BIOS interface using a BIOS read request on device number 131. The read call will return only when an Event or Break is to be processed. No data is actually transferred by the read.

Implementors of other operating environments should note that it is necessary to disable events before doing the read request in order to not miss an event which occurs before the routine has completely started. Likewise p-System users should rely on the built-in scheme and not issue a direct UNITREAD to device 131.

One will notice that the p-System Print Spooler no longer keeps the LED processor indicator green while waiting for keyboard input. A system using the Print Spooler will Attach to the Keyboard Event (#19) and wait on the semaphore instead of issuing a read request to the BIOS. Therefore when the system is awaiting keyboard input it is actually running the Waiter Task which originally was compute bound.

V.09 EXTRA UNITREAD & UNITWRITE INFORMATION :

Additional information can be given to various devices using UNITREAD and UNITWRITE. Refer to the p-System INTERNAL ARCHITECTURE GUIDE and the UCSD Pascal Handbook for more details.

CONTROL Parameters (FLAG)

The CONTROL parameter to UNITREAD, UNITWRITE, and UNITSTATUS is a word used to pass special information to the interpreter/BIOS interface and the BIOS regarding the handling of the I/O request.

CONTROL Word Format for UNITREAD and UNITWRITE

Bit 0 ASYNC Set implies asynchronous I/O request. Reset implies synchronous I/O request. This bit should always be reset unless doing asynchronous operations. Refer to page 96 .

Bit 1 PHYSSECT Set implies "physical sector mode" for disk I/O. Reset implies "logical block mode" for disk I/O.

Bit 2 NOSPEC Set implies "no special character handling" such as DLE expansion, alpha lock toggle, and EOF processing. Reset implies normal "special" character handling.

Bit 3 NOCRLF Set implies that no line-feeds (LFs) are to be appended to carriage returns (CRs) during serial channel I/O. Reset implies LFs are to be appended to CRs during serial channel I/O.

Floppy Formatting

UNITWRITE requests for the floppy diskettes will format a track if bit 13 of the Control Word is set. The high byte of the Logical Block Number is the cylinder address to be formatted. The low byte of the Logical Block Number is the head address to be formatted. The Memory Buffer Area contains the ID field data to be recorded during formatting (4 bytes per sector - cylinder, head, sector, and bytes per sector code). Note that before formatting, the Gap 3 parameter must be modified using a write to special channel 128.

Remote Channel Control Bits

For the Remote Out channel, the Control Word bits 12 and 13 are used to specify control of the Data Terminal Ready, the Request to Send, and the Transmit Break signal. When bit 12 is set in the Control Word, the one byte of data from the buffer contains the signal bits to be cleared. When bit 13 is set in the control word, the one byte of data from the buffer contains the signal bits to be set. The Data Terminal Ready signal bit is 2H, the Request to Send signal bit is 20H (32 decimal), and the Break signal is 8H. Following is a sample Pascal sequence to strobe the Break signal.

```
CONST
  TurnOn = 8192; { Turn on  data bits }
  TurnOff= 4096; { Turn off data bits }
VAR
  Data:PACKED ARRAY[0..1] OF 0..255;

BEGIN
  Data[0]:=8; { Define Break bit }
  UNITWRITE(8,Data[0],1,0,TurnOn);  { Turn on Break signal }
  { Insert statements to generate appropriate delay }
  UNITWRITE(8,Data[0],1,0,TurnOff); { Turn off Break signal }
```

● Winchester Formatting

Bit 13 of the Control word for Winchester UNITREAD and UNITWRITE's is used for formatting. It inhibits the pre-read of the track on writes. It also bypasses using the Bad Track Map and does not force an attempt to read in the previous configuration information from track 0.

V.10 EXTRA UNITSTATUS INFORMATION :

The UNITSTATUS procedure returns information about device status as described in the p-System INTERNAL ARCHITECTURE GUIDE. The UNITSTATUS procedure returns information in a 30 word (60 byte) area. The additional information on the next page is returned by the SAGE II/IV BIOS at the end of the data areas to leave room for standard p-System expansion.

The CONTROL parameter to UNITSTATUS passes special information to the RSP/IO and BIOS regarding the handling of the I/O request.

CONTROL Word Format for UNITSTATUS

Bit 0 IODIR Set implies the status of the input channel is to be returned. Reset implies the status of the output channel is to be returned.

SAGE UNITSTATUS Information

```
Channel 1 (Keyboard)

      Offset: 54.- Framing error count (word).
      Offset: 56.- Parity error count (word).
      Offset: 58.- Overrun error count (word).

Channel 6 (Printer)

   Offset: 48.- Printer mode (word).
                1 - use Remote Serial output channel.
                2 - Parallel (interrupt operation).
                3 - Parallel (polled operation).
   Offset: 50.- Printer port flags (word).
                Bit 4 is one if printer is busy.
                Bit 5 is one if Out of Paper.
                Bit 6 is one if printer is Selected.
                Bit 7 is zero for a printer Fault.
   Offsets 52. to 59. are only defined if the Remote channel
                is being used as the printer (See channel 7)

Channel 7 (Remote input)

   Offset: 52.- Ringing and Carrier Detect (byte).
                Bit 2 is zero if ringing detected.
                Bit 3 is zero if carrier detected.
   Offset: 53.- Data Set Ready (byte).
                Bit 7 is one if DSR is asserted.
   Offset: 54.- Framing error count (word).
   Offset: 56.- Parity error count (word).
   Offset: 58.- Overrun error count (word).
```

EXAMPLE: Accessing Ringing and Carrier Detect

```
program RemoteTest;
VAR
  Status:PACKED RECORD
           Dummy1:ARRAY[0..25] OF INTEGER;
           Dummy2:0..127;
           DataSetReady:BOOLEAN;
           Dummy3:0..3;
           NoRinging:BOOLEAN;
           NoCarrier:BOOLEAN;
           Dummy4:0..15;
           Dummy5:ARRAY[0..2] OF INTEGER;
         END;
BEGIN
  UNITSTATUS(7,Status,1);
  WITH Status DO
    BEGIN
      IF NoRinging THEN WRITELN('Ringing not detected');
      IF NoCarrier THEN WRITELN('Carrier not detected');
      IF DataSetReady THEN WRITELN('Data Set Ready');
    END;
END.
```

V.11 BIOS CONFIGURATION :

The SAGE I/O implementation provides a special method for on-line configuration of the BIOS features. Read/Write access is provided to the I/O configuration information via UNITREAD and UNITWRITE to device 128. The Control word passed to the UNITREAD or UNITWRITE must contain the device number being accessed. The size and logical block number fields are ignored. The amount of information passed is predefined and possibly different for each device. For a more detailed breakdown on the configuration information see the BIOS Configuration writeup.

> NOTE: p-System users will typically want to use the CONFIGSAGE Unit where configuration record layouts are provided and routines are available to make most changes.

EXAMPLE: Set up parallel port with scheduled polling

```
VAR
  Configuration:PACKED RECORD
                  PrtTimeout:INTEGER;
                  PrtPollTime:INTEGER;
                  PrtDummy:0..255;
                  PrtMode:0..255;
                END;
BEGIN
  UNITREAD(128,Configuration,0,0,6);
  Configuration.PrtMode:=3;
  UNITWRITE(128,Configuration,0,0,6);
```

Note that all these configuration changes only affect the BIOS in memory, not on the diskette. Any permanent changes must be installed in the SYSTEM.BIOS file with SAGE4UTIL.

V.12 SYSTEM CLOCK ACCESS :

SYSTEM CLOCK ACCESS

The standard p-System TIME procedure returns a two word time counter value in increments of 1/60th of a second. This value is derived from the standard SAGE system clock which maintains a two word second counter and a one word counter for 1/64000ths of a second. This standard SAGE system clock can be read using UNITREAD on device 129. The size and block number are ignored. The Control word should be set to zero to read this "raw" clock value from the system.

```
Example:  System clock access

PROGRAM TestTime;
VAR
  SysTime:RECORD
            PartSeconds, {1/64000ths}
            LowSeconds,
            HighSeconds:INTEGER;
          END;
  LastTime:INTEGER;
BEGIN
  LastTime:=1;
  REPEAT
    UNITREAD(129,SysTime,0,0,0); {Get raw time}
    WITH SysTime DO
      IF LastTime <> LowSeconds THEN
        BEGIN
          WRITELN('Low Seconds = ',LowSeconds);
          LastTime:=LowSeconds;
        END;
  UNTIL (LastTime MOD 20) = 0;
END.
```

Note that all three numbers should be considered as unsigned values The PartSeconds value does not count completely to the end but wraps back to zero after 64000 counts.

The BIOS also supports the setting and reading back of a 32 bit second count which is based on the "raw" clock value. This feature is used by the Time Utility Unit covered in a later section. The Time Utility Unit sets the Real Time with the number of seconds after the beginning of January 1, 1970. This Time Bias plus the current "raw" clock value is saved by the BIOS. When the real time is read back, the BIOS calculates the difference in "raw" clock values between when the time was set and the current time and then adds this to the Time Bias to result in the current number of seconds after January 1, 1970. Note that the "raw" system clock is never modified, thus keeping all relative time measurements accurate.

The Real Time is set using a UNITWRITE to device 129 with the data buffer containing the 32 bit second count (past the base date). The size, block number, and Control word for this UNITWRITE are ignored. The Real Time second count is read back into a similar buffer using a UNITREAD to device 129 with a Control word of 1. The Time Bias may be read back using a UNITREAD to device 129 with a Control word of 2. The "raw" clock second count when the time was last set may be read back using a UNITREAD to device 129 with a Control word of 3.

The crystal used to generate the timing signals for the SAGE IV computer does not have the accuracy necessary to maintain perfect time over a period of many days. Therefore a time correction factor may be set up using the Configuration options of SAGE4UTIL. This factor is applied to the difference between the current and set times to generate a correction to the returned Real Time value (control word = 1).

```
Example:  Second Count access

  TYPE
    TimeValue = RECORD
                  LowTime,
                  HighTime:INTEGER;
                END;
  VAR
    TimeCheck,OldTime,NewTime:TimeValue;
  BEGIN
    UNITREAD(129,TimeCheck,0,0,2);  {Read Time Bias}
    IF (TimeCheck.LowTime<>0) OR (TimeCheck.HighTime<>0) THEN
      UNITREAD(129,OldTime,0,0,1); { Old time is valid if
                                     Time Bias previously set}
    UNITWRITE(129,NewTime,0,0,0); {Set a new time     }
```

V.13 GENERAL MEMORY ACCESS :

The current p-System implementation (IV.13) supports 64k bytes of data space and 64k bytes of program space. In addition the p-System Interpreter and the BIOS can be placed outside of the program and data spaces. Two possibilities exist for using the additional SAGE II/IV memory from a p-System program.

- Define the extra memory as a mass storage device (RAMDISK: #11). The user may access the memory through the standard file I/O but with a much faster access time than a floppy.

- The second method of accessing memory is by using UNITREAD and UNITWRITE to device 130. The logical block number is treated as the low word address for the access. The Control word is treated as the high word address for the access. The buffer type is not checked by these low level I/O procedures so it is easy to mix reading and writing of different types of information. It is of course up to the user to keep track of the memory assignments and prevent writing over the BIOS, Interpreter, and p-System program and data areas.

EXAMPLE: Get directory of 2048 bytes at 1D400H

```
VAR
  directory:RECORD
              -------------
            END;
BEGIN
  UNITREAD(130,Directory,2048,-11264,1);
```

EXAMPLE: Read one byte from Group B DIP Switch at 0FFC023H.

```
VAR
  TwoBytes:PACKED ARRAY[0..1] OF 0..255;
BEGIN
  UNITREAD(130,TwoBytes[0],1,-16349,255);
  WRITELN('Dip Switch B = ',TwoBytes[0]);
```

V.14 SYSTEM MEMORY ALLOCATION :

The SDT-ASSEMBLER MANUAL gives a detailed description of memory allocation.

V.15 NOTES ON PORTABILITY :

The p-System has addressed several areas of system portability quite well. Device directories and .TEXT files are readable on all processors. Code files are executable on all processors. There are still, however, areas of data file portability which have not been solved.

Data which is not stored in an ASCII textual representation is generally not portable to all processors. A 'FILE OF INTEGER' or any other type will store the processors internal binary representation of the data into the file. The major problem that occurs is the 'byte sex' difference between processors. Some processors store word sized values with the least-significant byte first and others store the most-significant byte first. Text files are stored as a sequence of individual bytes so there is no 'byte sex' confusion.

Also contributing to the data portability problem is the fact that the internal representation of REAL numbers and Long INTEGERS is processor dependent. Real constants in code files are stored in a processor independent fashion and must be converted to the processor's internal representation by the system.

This conversion costs some time when the program is loaded. If a routine with real constants is heavily overlayed it may be desirable to sacrifice the portability for speed. In this case the routine REALCONV.CODE may be used to change the portable representation into the internal storage format for the processor. This routine is documented in the p-System OPERATING SYSTEM MANUAL.

Programs which store non-real and non-long-integer values in data files may want to develope their own routines for

handling the 'byte sex' conversion. If a known integer value of 1 is stored and it is read back by another system as 256, then the 'byte sex' of all the word sized values is reversed. A routine may be written to optionally swap the byte order for each of the word sized values in the data. Real and Long Integer values may be stored in a textual form by writing them to a .TEXT file. A time penalty will be paid for the conversion to and from the textual format.

Because the Real numbers and Long Integers may have different internal representations, the system routines REALOPS and LONGOPS should not be used in a different processor. LONGOPS is actually an extension of the interpreter and contains assembly code for a specific processor. These routines are generally located in the files SYSTEM.PASCAL or SYSTEM.LIBRARY. Care should be taken if either of these files is moved to another processor to check for and replace the REALOPS and LONGOPS with the implementation for the target processor. Use the program LIBRARY.CODE to view the segment names and move the routines.

V.16 CP/M-68K FILES & p-SYSTEM :

The file TOPSYS.CODE is a utility which will read a SAGE CPM-68K disk and selectively transfer files to a p-SYSTEM disk. "DATA" files are transferred as images so code files can be moved across.

V.17 p-System FILE DESCRIPTIONS :

ANSI.CODE	SCREENOPS Unit for ANSI standard terminals.
ANSIGOTOXY.CODE	GOTOXY routine for ANSI standard terminals.
ANSI.MISCINFO	SYSTEM.MISCINFO for ANSI standard terminals.
ASMLOAD.CODE	Stand-alone assembly code loader.
BASIC.R2.CODE	BASIC Compiler for generating code using 2 word real arithmetic.(not in standard release)
BASIC.R4.CODE	BASIC Compiler for generating code using 4 word real arithmetic.(not in standard release)
BLIB.R2.CODE	Runtime support library for BASIC using 2 word real arithmetic.
BLIB.R4.CODE	Runtime support library for BASIC using 4 word real arithmetic.
BOOTER.CODE	Program to copy p-System bootstraps.
CHKSUMOPS.CODE	Unit used to generate checksums of code which are used with QUICKSTART.CODE
COMMANDIO.CODE	Interface section for the operating system's COMMANDIO Unit.

COMPRESS.CODE	Program to apply relocation information to stand-alone assembly language programs.
COPYDUPDIR.CODE	Program to copy the duplicate directory of a diskette back to the primary directory.
DECODE.CODE	Program to disassemble p-code files and provide other code file information.
DIR.INFO.CODE	Unit used to access directory information.
DISKCHANGE.CODE	Program to access diskettes with different interleaving (documented in Installation Guide, section IV.2).
DISKSIZE.CODE	Program to change the size of a device (see Installation Guide, section IV.2).
ENDBOOT	This file contains no information. Its **position** in the directory indicates to the boot routine to load the files above it into RAM Disk when "Boot to RAM Disk" is enabled.
ERRORHANDL.CODE	Interface section for the operating system's ERRORHANDL Unit.
FILE.INFO.CODE	This Unit is used to access file information.

FINDPARAMS.CODE	Program to find best interleaving for diskettes (see Installation Guide, section IV.2).
FORMATINFO.TEXT	This is a text file containing the Winchester mapping information used by WFORMAT. It does not have to be called FORMATINFO.
FORTLIB2.CODE	Runtime support library for Fortran 77 using 2 word real arithmetic.
FORTLIB4.CODE	Runtime support library for Fortran 77 using 4 word real arithmetic.
FORTRAN2.CODE	Fortran 77 Compiler for generating code using 2 word real arithmetic. (not in standard release)
FORTRAN4.CODE	Fortran 77 Compiler for generating code using 4 word real arithmetic. (not in standard release)
IB.BUS.TEXT	Assembler source for IB.BUS.CODE
IB.BUS.CODE	SAGE provided these 68000 assembly language routines for interfacing to the IEEE 488 Bus.
IB.EX.TEXT	Pascal source for IB.EX.CODE
IB.EX.CODE	Example of Pascal program which uses the IEEE 488 bus Unit.
IB.DEF.TEXT	File of Symbol definitions for IB.BUS routines.

IB.LNK.CODE	Linked form of the IEEE 488 Bus Unit (IB.UNIT & IB.BUS) to allow control from a high level language program.
IB.UNIT.TEXT	Pascal source for IB.UNIT.CODE
IB.UNIT.CODE	The Pascal portion of the IEEE 488 Bus Unit.
INTERP.0.CODE	Interpreter with no real arithmetic.
INTERP.2.CODE	Interpreter with two word real arithmetic.
INTERP.4.CODE	Interpreter with four word real arithmetic.
KERNEL.CODE	Interface section for operating system's KERNEL Unit.
LIBRARY.CODE	Program to maintain code file libraries.
MARKDUPDIR.CODE	Program to install a duplicate directory on a device which previously did not have a duplicate directory.
MC.0.2	Standard Multi-user configuration file for SAGE II, 2 users.
MC.12.4	Standard Multi-user configuration file for SAGE IV, 4 users, 12 Meg disk.

MC.18.5	Standard Multi-user configuration file for SAGE IV, 5 users, 18 Meg disk.
MC.18.6	Standard Multi-user configuration file for SAGE IV, 6 users, 18 Meg disk.
MC.40.5	Standard Multi-user configuration file for SAGE IV, 5 users, 40 Meg disk.
MC.40.6	Standard Multi-user configuration file for SAGE IV, 6 users, 40 Meg disk.
MERGE.CODE	Generates MU.FORMAT by combining the bad tracks section from the FORMATINFO with a standard partial MF.x.x file.
MF.0.2	Standard Multi-user partial FORMATINFO file for SAGE II, 2 users.
MF.12.4	Standard Multi-user partial FORMATINFO file for SAGE IV, 4 users, 12 Meg disk.
MF.18.5	Standard Multi-user partial FORMATINFO file for SAGE IV, 5 users, 18 Meg disk.
MF.18.6	Standard Multi-user partial FORMATINFO file for SAGE IV, 6 users, 18 Meg disk.

MF.40.5	Standard Multi-user partial FORMATINFO file for SAGE IV, 5 users, 40 Meg disk.
MF.40.6	Standard Multi-user partial FORMATINFO file for SAGE IV, 6 users, 40 Meg disk.
MU.BOOTEXT.CODE	The Boot extension file which initializes the RAM Disk under Multi-user.
MU.BOOTEXT.TEXT	Assembler source for MUBOOTEXT.CODE
MULTI.ONE.TEXT	Standard configuration script file to build standard SAGE II Multi-user system.
MULTI.TWO.TEXT	Standard configuration script file to build standard SAGE IV Multi-user system on 12 Meg disk, 4 users.
MULTI.THRE.TEXT	Standard configuration script file to build standard SAGE IV Multi-user system on 18 Meg disk, 5 users.
MULTI.FOUR.TEXT	Standard configuration script file to build standard SAGE IV Multi-user system on 18 Meg disk, 6 users.
MULTI.FIVE.TEXT	Standard configuration script file to build standard SAGE IV Multi-user system on 40 Meg disk, 5 users.
MULTI.SIX.TEXT	Standard configuration script file to build standard SAGE IV Multi-user system on 40 Meg disk, 6 users.

MU4.BIOS	The main Multi-User BIOS program.
MU4.FBOOT.CODE	The SAGE IV Multi-user system Boot program for the floppy.
MU4.FBOOT.TEXT	Assembly source for MU4.FBOOT.CODE
MU4.WBOOT.CODE	The SAGE IV Multi-user system Boot program for the Winchester.
MU4.WBOOT.TEXT	The assembly source for MU4.WBOOT.CODE
MU.CONVERT.CODE	Converts old Multi-User configurations to new release configurations.
MU.CONFIG	This file contains the configuration information maintained by MU.UTIL.CODE. It does not have to be this exact name.
MU.FORMAT.TEXT	File created by MERGE from standard MU files (MF.x.x) and the FORMATINFO.TEXT shipped with system.
MU.INSTAL.TEXT	Source of MU.INSTAL.CODE.
MU.INSTAL.CODE	Main program on BUILD diskette which installs standard multi-user configurations.
MU.PBOOT.CODE	The p-System boot program under Multi-User.
MU.PBOOT.TEXT	Assembly source for MU.PBOOT.CODE

MU.UTIL.CODE	The utility program which controls the many parameters of the physical devices on the Multi-User system and assigns system resources to the User Tasks.
MUTRMSET.CODE	The program used to create new terminal emulation definitions for the Multi-User shared terminal mode.
PATCH.CODE	Program for viewing and altering file information at a primitive (hexadecimal) level.
PEDGEN.CODE	Unit used to perform the functions of QUICKSTART.CODE in a user program.
PRINT.CODE	Simple TEXT file formatter.
QUICKSTART.CODE	Program to speed program startup.
REALCONV.CODE	Program to convert real number constants in code files to a non-portable (but operationally faster) representation.
REALOPS.2.CODE	Real arithmetic library for 2 word real numbers.
REALOPS.4.CODE	Real arithmetic library for 4 word real numbers.
RECEIVE.TEXT	Source for RECEIVE.CODE
RECEIVE.CODE	SAGE provided program to receive the image of a device from another system on the Remote serial channel.

RECOVER.CODE	Program which attempts to re-create the directory of a disk whose directory has accidentally been destroyed.
REMINTEST.TEXT	Source for REMINTEST.TEXT
REMINTEST.CODE	SAGE provided program to test the Remote serial channel input.
REMOUTTEST.TEXT	Source for REMOUTTEST.CODE
REMOUTTEST.CODE	SAGE provided program to test the Remote serial channel output.
REMTALK.TEXT	Source for REMTALK.CODE
REMTALK.CODE	Program to transfer files over the Remote serial channel.
REM.HAYES.CODE	The Remote Unit for the HAYES modem.
REM.VADIC.CODE	The Remote Unit for the RACAL VADIC modem.
SAGEDATE.CODE	SAGE provided program to set the system's date and time.
SAGETOOLS.CODE	SAGE provided Units which are utilities in the SAGE Tool Kit.
SAGE4UTIL.CODE	Program to configure the BIOS, format diskettes, and copy bootstraps.
SAGE.PBOOT.TEXT	Source for SAGE.PBOOT.CODE

SAGE.PBOOT.CODE	Assembly language routine which bootstraps the single-user BIOS and Interpreter into operation.
SAGE.WBOOT.CODE	This is SAGE IV single-user p-System Winchester boot file.
SAGE.WBOOT.TEXT	This is the assembly code source file for SAGE.WBOOT.CODE.
SAMPLEGOTO.TEXT	Sample of a GOTOXY routine for building a custom routine for your terminal.
SCREENOPS.CODE	Interface section for the operating system's SCREENOPS Unit.
SCREENTEST.CODE	Program for testing the SYSTEM.MISCINFO and GOTOXY features
SEND.TEXT	Source for SEND.CODE
SEND.CODE	SAGE provided program to send the image of a device to another system over the Remote serial channel.
SETUP.CODE	Program for altering SYSTEM.MISCINFO for different terminal and system configurations.
SPOOLER.CODE	Program to set up files for printout by the Print Spooler.
SYSTEM.ASSEMBLER	68000 Assembler.
SYSTEM.BIOS	Hardware device drivers.

SYSTEM.COMPILER	Pascal Compiler.
SYSTEM.EDITOR	Screen Oriented Editor.
SYSTEM.FILER	File Utility.
SYSTEM.INTERP	p-System Interpreter for the 68000.
SYSTEM.LIBRARY	Standard library routines (initially contains long integer arithmetic).
SYSTEM.LINKER	Linker for combining p-code with assembly code files.
SYSTEM.MISCINFO	System configuration data file (initially not set up for a special terminal).
SYSTEM.PASCAL	p-System Operating System.
SYSTEM.SYNTAX	Syntax error messages for the Pascal compiler.
SYS.INFO.CODE	Unit to provide access to operating system information.
TELE.HAYES.CODE	is the version of the TELETALKER communications program for the HAYES modem.
TELE.VADIC.CODE	is the version of the TELETALKER communications program for the RACAL VADIC modem.
TEXTIN.TEXT	Source for TEXTIN.CODE.

TEXTIN.CODE	Program to receive ASCII text from another system over the Remote serial channel.
TOPSYS.TEXT	Source for TOPSYS.CODE.
TOPSYS.CODE	Program to read a SAGE CP/M-68K disk and transfer a file to a p-System disk.
TRMDEF.DATA	The library of terminal emulator definitions for the Multi-user shared terminal mode.
WILD.CODE	Unit for performing 'wild card' searches (pattern matching of strings).
WFORMAT.CODE	This program is used for setting up Winchester disk partitions and formatting the disk.
XREF.CODE	Program to produce a procedure cross reference list for Pascal programs.
YALOE.CODE	Line Oriented Editor.
ZAP.CODE	Zeros selected disk partition directories. Used in building Multi-User Systems.
68000.OPCODES	Data file containing op-code information for the 68000 Assembler.
68000.ERRORS	Data file containing error messages for the 68000 Assembler.

Other files may be present on the distribution diskettes. Most of these files will be xxxx.MISCINFO or xxxx.GOTO.CODE which are

SYSTEM.MISCINFO configurations and GOTOXY Units for specific terminals. Any other new files should be covered in a separate document shipped with the system called System Release Notes.

VI SAGE TOOL KIT :

The main SAGE distribution diskette contains most of the SAGE generated UNITs in a file called SAGETOOLS.CODE. The UNITs in SAGETOOLS.CODE are complete with their interface sections. The p-System provided UNITs DIR_INFO, WILD, SYS_INFO, and FILE_INFO are also packaged in the SAGETOOLS.CODE file even though they are a standard part of the p-System distribution.

SAGETOOLS contains these units:

- CONFIG_SAGE provides access to the BIOS configuration.
- MNU_Unit is a menu generator unit.
- SIO_Unit is a string utility unit.
- TAD_Unit handles time and date formats.
- WIN_Unit provides access to Winchester drive information.
- ATT-Unit provides definitions of p-System Event numbers.

VI.01 UNITS :

The UNIT facility of the p-System provides a feature not typically found in small system environments. A UNIT may contain a set of Constants, Types, Variables, and Procedures which are available to other programs. The text which specifies the INTERFACE of the UNIT to the outside world is placed in the compiled code file along with the generated code for the routines. This allows programs which USE the UNIT to be able to easily share data and procedures with little chance of error.

Another valuable feature is that the UNITs need not be linked to the main program until run time. This means that only one copy of the code needs to exist on disk, not a copy linked into every program which USEs the UNIT. In addition to saving space, only one copy of the code need be changed if only a detail in the IMPLEMENTATION section of the UNIT is modified.

UNITs provide an excellent mechanism to hide implementation details and present a common user interface. For instance, a word processing package may want to interface to several types of letter quality printers using different control protocols. The system designer defines a high level interface for the types of activities desired from the printers. Then a UNIT with the same INTERFACE to the main program may be written with a different implementation to control each printer. The final user then only needs to install the UNIT appropriate for his printer into the word processing program.

For SAGE, the UNIT facility provides a method of packaging useful routines which are commonly used by several utility programs. SAGE also uses UNITs to relieve the programmer from learning low level system details in order to interface to various SAGE system functions. The group of UNITs provided by SAGE are collectively called the SAGE Tool Kit. Most of these units are collected together in a file called SAGETOOLS.CODE. Following is a discussion on the placement of UNITs for compile time and run time environments.

A complete discussion of UNITs is contained in the p-System PROGRAM DEVELOPMENT REFERENCE MANUAL under the title Segments, Units, and Libraries. A portion of that information is presented in the following section to explain various options on UNIT file placement. The p-System program LIBRARY.CODE is used to move UNITs from one code file to another. Use of this utility program is covered in p-System OPERATING SYSTEM REFERENCE MANUAL under the Chapter on Utility Programs.

Note that although the Unit concept can allow programs to share common library routines, SAGE has installed the appropriate units into each of its utility program files. This is because new users are typically not aware of the requirements and implications of having separate library files. Experienced users may want to remove the appropriate common units from the utility program files and place them in SYSTEM.LIBRARY or reference SAGETOOLS.CODE in USERLIB.TEXT (see following section for details).

UNIT Code Placement

At run time the operating system will attempt to find a referenced UNIT in one of four places. The UNIT may be found in the code file with the main program, however this does not take advantage of the concept of sharing the same UNIT code with several programs. The UNIT may be found in the operating system file (SYSTEM.PASCAL) but this should be reserved for operating system routines. Shared UNITs are generally placed in the file SYSTEM.LIBRARY or in a separate user library whose name is contained in the file USERLIB.TEXT. Both the files SYSTEM.LIBRARY and USERLIB.TEXT must be present on the system device.

From an ease and speed of use standpoint it is best to put the UNITs including their INTERFACE sections into the file SYSTEM.LIBRARY. Size constraints however may make this approach impractical, especially when using RAM Disk as the system device. Some room may be saved by using the 'T' option of the LIBRARY program to leave out the INTERFACE sections of the UNITs in the SYSTEM.LIBRARY (or user

library) file. When this is done however, any compilation which USEs a UNIT will have to use the (*$U *) option to specify a file which contains the code with the INTERFACE section.

Another reasonable alternative may be to install the most frequently used UNITs in the SYSTEM.LIBRARY and less frequently used UNITs in a user library located on another device. Although the USERLIB.TEXT file must be present on the system device, it can specify one or more user library files on other devices.

Note that the user library files are searched before the SYSTEM.LIBRARY in order to allow a user UNIT to be loaded instead of a normal system UNIT. This type of overriding assignment is often unnecessary so the file *SYSTEM.LIBRARY should be specified as the first file in the USERLIB.TEXT list. This will cause the SYSTEM.LIBRARY to be searched first and prevent having to search the user library files if all the necessary UNITs have already been found in the SYSTEM.LIBRARY. The number of user library files specified in USERLIB.TEXT should be minimized in order to reduce the search time and number disk accesses when loading a program.

VI.02 STRING I/O UTILITY UNIT :

A basic utility of the SAGE Tool Kit is a package of routines for input and output of data into STRINGs. This package of routines is called SIO_Unit and has the following Interface section:

```
INTERFACE

{ String Read Routines }
FUNCTION SIO_IntRd (VAR Cursor:INTEGER; VAR Source:STRING;
                    VAR Result:INTEGER):BOOLEAN;
FUNCTION SIO_HexRd (VAR Cursor:INTEGER; VAR Source:STRING;
                    VAR ResultH,ResultL:INTEGER):BOOLEAN;
FUNCTION SIO_AlphRd(VAR Cursor:INTEGER; VAR Source:STRING;
                    VAR Result:STRING):BOOLEAN;
FUNCTION SIO_AlNuRd(VAR Cursor:INTEGER; VAR Source:STRING;
                    VAR Result:STRING):BOOLEAN;
FUNCTION SIO_CharRD(VAR Cursor:INTEGER; VAR Source:STRING;
                    Check:CHAR):BOOLEAN;
FUNCTION SIO_ByDlim(VAR Cursor:INTEGER; VAR Source:STRING;
                    Check:STRING):BOOLEAN;

{ String Write Routines }
PROCEDURE SIO_CharWt(Value:CHAR; VAR Result:STRING);
PROCEDURE SIO_HexWt (Value:INTEGER; Digits:INTEGER;
                     VAR Result:STRING);
PROCEDURE SIO_IntWt (Value:INTEGER; VAR Result:STRING);
PROCEDURE SIO_Fill  (Count:INTEGER; VAR Result:STRING);
PROCEDURE SIO_Upper (VAR Result:STRING);
PROCEDURE SIO_Suffix(Suffix:STRING; VAR Result:STRING);
```

There are two major types of routines in the SIO_Unit package; ones for reading data from a STRING and ones for writing data to a STRING. Frequently a user will have available a free form command or other data input which has been read into the program from the keyboard or a file. The string read routines are useful in helping to parse the information contained in a string down to basic items of data.

The string read routines are always passed the source data in a string called 'Source'. A variable called 'Cursor' contains the index pointer to the next character in the string to be processed. The 'Cursor' variable is updated by each successful string read to move past the processed data in the string. The 'Cursor' should always be initialized to 1 before starting to parse the string. If 'Cursor' is greater than LENGTH(Source), then the complete string has been processed.

Each of the string read routines is a FUNCTION which returns a TRUE result if the desired type of data is found. Thus the user may attempt to parse an numeric integer value from the string using SIO_IntRd and if the result is FALSE may then try to parse an alphanumeric sequence from the string using SIO_AlNuRd.

The string read processes are described below:

SIO_IntRd This routine returns a 16 bit INTEGER 'Result' from an optionally signed decimal number: (-32768 to +32767).

SIO_HexRd This routine returns a high 16 bit INTEGER 'ResultH' and a low 16 bit INTEGER 'ResultL' from an unsigned hexadecimal number: (0 to FFFFFFFF).

SIO_AlphRd This routine appends to the string 'Result' all the following alphabetic characters until the end of the 'Source' is found or a non-alphabetic character is found.

SIO_AlNuRd This routine appends to the string 'Result' all of the following alphanumeric characters until the end of the 'Source' is found or a non alpha-numeric character is found. Note that the first character of the alphanumeric string must be only alphabetic (typical variable name convention).

SIO_CharRd This routine checks if the single character in 'Check' is found and returns the appropriate FUNCTION value as well as updating the 'Cursor'.

SIO_ByDlim This routine is used to advance past a set of one or more delimiters contained in the string 'Check'. Any characters found in 'Check' are bypassed in the 'Source' string. The FUNCTION result is set TRUE if any of the 'Check' characters are found.

Th following page shows an example of the use of the string read routines to build a table of variable names and their associated values. A simple Pascal assignment syntax (Variable := value) is used and the values may either be decimal or hexadecimal (indicated by a leading 0 digit).

```
PROGRAM SIO_EXAMPLE;
USES SIO_Unit;
CONST
  MaxTable = 20;
VAR
  NameTable:ARRAY[1..MaxTable] OF STRING;
  ValueTable:ARRAY[1..MaxTable] OF INTEGER;
  Source:STRING;
  Cursor,TableIndex,DumInt:INTEGER;
  Legal,Dummy:BOOLEAN;
BEGIN
  TableIndex:=1;
  REPEAT
    Legal:=FALSE;
    WRITE('Assignment? ');
    READLN(Source);
    IF LENGTH(Source) > 0 THEN
      BEGIN
        Cursor:=1;
        NameTable[TableIndex]:='';
        IF SIO_AlNuRd(Cursor,Source,NameTable[TableIndex]) THEN
          BEGIN
            { Bypass any spaces before := }
            Dummy:=SIO_ByDlim(Cursor,Source,' ');
            IF SIO_CharRd(Cursor,Source,':') THEN
              IF SIO_CharRd(Cursor,Source,'=') THEN
                BEGIN
                  { Allow spaces after := too }
                  Dummy:=SIO_ByDlim(Cursor,Source,' ');
                  IF SIO_CharRd(Cursor,Source,'0') THEN
                    BEGIN { indicates hex value, low part only }
                      IF SIO_HexRd(Cursor,Source,DumInt,
                        ValueTable[TableIndex]) THEN Legal:=TRUE;
                    END
                  ELSE
                    IF SIO_IntRd(Cursor,Source,
                      ValueTable[TableIndex]) THEN Legal:=TRUE;
                  IF Legal THEN TableIndex:=TableIndex+1;
                END;
          END;
      END
    ELSE
      Legal:=TRUE;
    IF NOT Legal THEN WRITELN('Error in assignment');
  UNTIL (LENGTH(Source) = 0) OR (TableIndex > MaxTable);
    IF TableIndex > 1 THEN
      FOR Cursor := 1 TO TableIndex-1 DO
        WRITELN(NameTable[Cursor],' := ',ValueTable[Cursor]);
END.
```

The string write routines append their output (except SIO_Upper) to the string 'Result'. Therefore remember to set 'Result' to an initial empty string if only the single output value is desired. The string write procedures are described below:

SIO_CharWt This procedure appends the character from 'Value' to the string 'Result';

SIO_HexWt This procedure appends the unsigned hexadecimal value from the argument 'Value' to the string 'Result'. The argument 'Digits' specifies how many digits to output.

SIO_IntWt This procedure appends the signed decimal value from the argument 'Value' to the string 'Result'.

SIO_Fill This procedure appends the number of spaces indicated by the argument 'Count' to the string 'Result'.

SIO_Upper This procedure changes all the lower case alpha-betic characters in 'Result' to upper case. No other characters are changed or added.

SIO_Suffix This procedure is typically used to append a default extension ('Suffix') to a filename ('Result'). If the string is already terminated by the suffix the string is not changed. If the string is terminated by a period, the period is removed and the suffix is not appended.

The following example uses the string output routines to create a string of ten equally spaced decimal numbers.

```
USES SIO_Unit;
VAR
  I:INTEGER;
  Result,Temp:STRING;
BEGIN
  Result:='';
  FOR I:= 1 TO 10 DO
    BEGIN
      Temp:='';
      SIO_IntWt(Numbers[I],Temp);
      SIO_Fill(8-LENGTH(Temp),Result);
      Result:=CONCAT(Result,Temp);
    END;
  WRITELN(Result);
END;
```

VI.03 TIME AND DATE UNIT :

The Time and Date Utility Unit (TAD_Unit) in the SAGE Tool Kit provides facilities to access and set the SAGE real time clock. The Unit also contains facilities to read and write different time and date formats to strings. This utility allows an application program to easily get the current time and date for display or printout. Although the time and date are normally set by the program SAGEDATE, the TAD_Unit may be used to build in the time and date set function within an application program.

The TAD_Unit relies on having available the SIO_Unit from the SAGE Tool Kit. This Unit must be available in the user program file, SYSTEM.LIBRARY or other user specified library. Following is the Interface section of the TAD_Unit:

```
INTERFACE

TYPE
  TAD_Style = SET OF (TAD_Short,TAD_Long);
  TAD_Ptime = PACKED ARRAY[0..3] OF 0..255;

PROCEDURE TAD_TimO(Hours,Minutes,Seconds:INTEGER;
                   Method:TAD_Style; VAR Result:STRING);
PROCEDURE TAD_DatO(Day,Month,Year:INTEGER; Method:TAD_Style;
                   VAR Result:STRING);
PROCEDURE TAD_DOWO(DayOfWeek:INTEGER; Method:TAD_Style;
                   VAR Result:STRING);

FUNCTION TAD_TimI(VAR Cursor:INTEGER; VAR Source:STRING;
                  Method:TAD_Style; VAR Hours,Minutes,
                  Seconds:INTEGER):BOOLEAN;
FUNCTION TAD_DatI(VAR Cursor:INTEGER; VAR Source:STRING;
                  Method:TAD_Style; VAR Day,Month,
                  Year:INTEGER):BOOLEAN;

PROCEDURE TAD_Pack(Day,Month,Year,Hours,Minutes,Seconds:INTEGER;
                   VAR Result:TAD_Ptime);
PROCEDURE TAD_Unpack(Source:TAD_Ptime; VAR Day,Month,Year,Hours,
                     Minutes,Seconds,DayofWeek,Julian:INTEGER);

PROCEDURE TAD_Fetch(VAR Result:STRING);
PROCEDURE TAD_Set(Day,Month,Year,Hours,Minutes,Seconds:INTEGER);
```

The TAD_Unit Interface defines two Types. The TAD_Style type defines two options (not mutually exclusive) which are used by the time & date string read/write routines to specify formatting variations. The meaning of each option is defined for each specific routine.

The type TAD_Ptime is a 32 bit packed time which is set and read back from the BIOS via UNITREAD & UNITWRITE to device 129. This real time clock access is covered in an earlier section called System Clock Access. The time represents the number of seconds after a base date of January 1, 1970. The data is broken down into bytes for easy manipulation by the routines TAD_Pack and TAD_Unpack. These routines convert between a time and date and the packed 32 bit second count representation.

The TAD_Unit routines are described below:

TAD_TimO This routine converts the time indicated by the arguments 'Hours', 'Minutes', and 'Seconds' into text and appends it to the string 'Result'. The default format (with empty 'Method', []) is a 24 hour time with no seconds (example: 16:30). The TAD_Short option will format the time into a 12 hour time followed by AM or PM (example: 4:30 PM). The TAD_Long option will also output the seconds (example: 16:30:47). The TAD_Short and TAD_Long options may be used together: (example: 4:30:47 PM).

TAD_DatO This routine converts the date indicated by the arguments 'Day', 'Month', 'Year', into text and appends it to the string 'Result'. The default format (with empty 'Method', []) is a date in the form 30-Aug-82. If the TAD_Short option is specified the format is 8/30/82. If the option is TAD_Long the format is August 30, 1982. Note that for the TAD_DatO routine the TAD_Short and TAD_Long options may not be used together.

TAD_DOWO This routine converts the day of week indicated by the argument 'DayOfWeek' into text and appends it to the string 'Result'. The argument numbering is 0 to 6 for Sunday through Saturday.

The default format (with empty 'Method', [ ]) is the day name completely spelled out. The TAD_Short option will abbreviate the output to the first three letters (Example: Fri). The TAD_Long option has no affect.

TAD_TimI This routine converts text from the string 'Source' into the time values 'Hours', 'Minutes', and 'Seconds'. The argument 'Cursor' points to the start of the text to be converted in the 'Source' string (see SIO_Unit for more details on this protocol).

The default format (with empty 'Method', [ ]) is for a 24 hour time with the seconds field optional. The TAD_Short option indicates the input is to be in a 12 hour time format and must be followed by AM or PM. The TAD_Long option is unused. The TAD_TimI routine is a FUNCTION which returns a TRUE result if a legal time is processed.

TAD_DatI This routine converts text from the string 'Source' into the date values 'Day', 'Month', and 'Year'. The argument 'Cursor' points to the start of the text to be converted in the 'Source' string (see SIO_Unit for more details on this protocol).

TAD_DatI can automatically handle three different date formats: (30-Aug-82, 8/30/82, or August 30, 1982).

If the option TAD_Short is specified, a partial date will be allowed. The default

date must be passed in the 'Day', 'Month', and 'Year' arguments and the appropriate values will be modified. If a single number is input, the routine assumes it is the day (30- Aug-82 type format). The option TAD_Long is not used. The TAD_DatI routine is a FUNCTION which returns a TRUE result if a legal date is processed.

TAD_Pack This routine converts the date and time as indicated by the arguments 'Day', 'Month', 'Year', 'Hours', 'Minutes', and 'Seconds' into a packed 32 bit second count in 'Result'.

TAD_Unpack This routine converts a packed 32 bit second count from 'Source' into the date and time arguments 'Day', 'Month', 'Year', 'Hours', 'Minutes', and 'Seconds'. Also provided is a a 'DayOfWeek' number (0 = Sun, 6 = Sat) and a 'Julian' day of year number.

TAD_Fetch This routine appends the current SAGE date and time to the string 'Result'. The string always has 19 characters with the format (30-Aug-82 10:09:47). The time is in the 24 hour format with seconds. If the time has not been set a message is appended to the string ('No time was set. ').

TAD_Set This routine is used to set the SAGE system time to the values contained in the arguments 'Day', 'Month', 'Year', 'Hours', 'Minutes', and 'Seconds'. The routine uses TAD_Pack to form the 32 bit second count and the uses UNITWRITE to device 129 to update the BIOS.

Using TAD_Fetch from an application, it is very simple to printout the current SAGE time and date. Remember that the routines TAD_Fetch, TAD_TimO, TAD_DatO, and TAD_DOWO append to an existing string so the string must be previously set to empty or some text to be output before the data.

```
PROGRAM Application;
USES TAD_Unit;
VAR
  Result:STRING;
BEGIN
  Result:='The current date and time are ';
  TAD_Fetch(Result);
  WRITELN(Result);
END.
```

The next example shows how to use the routines from the TAD_Unit to generate a variation of the date and time format which contains the day of week name.

```
PROGRAM TAD_Demo;
USES SIO_Unit,TAD_Unit;
VAR
  Day,Month,Year,Hours,Minutes,Seconds,
  DayOfWeek,Julian:INTEGER;
  TimeValue:TAD_Ptime;
  Result:STRING;
BEGIN
  Result:='';
  UNITREAD(129,TimeValue,0,0,1);
  TAD_Unpack(TimeValue,Day,Month,Year,Hours,Minutes,Seconds,
             DayOfWeek,Julian);
  TAD_DOWO(DayOfWeek,[],Result);
  Result:=CONCAT(Result,', ');
  TAD_DatO(Day,Month,Year,[],Result);
  SIO_Fill(2,Result);
  TAD_TimO(Hours,Minutes,Seconds,[TAD_Long],Result);
  WRITELN(Result);
END;
```

VI.04 SAGE DATE SETTING UTILITY :

The program SAGEDATE.CODE is provided to allow the user to easily set the system's real time clock. The program also updates the p-System's date in the operating system and in the directories of the on-line devices. These functions are a superset of the D(ate function in the p-System Filer.

Updating on-line devices is important because when booting to RAM Disk, the date set by the Filer only gets written to the System device (RAM Disk). The floppy from which the files (and default date) are copied is never updated by the Filer and if a date set is forgotten the "very old" date from the floppy is used.

The SAGEDATE program contains the Units SIO_Unit and TAD_Unit from the SAGE Tool Kit as well as the p-System Units Sys_info, Wild, and DirInfo. Experienced users may significantly reduce the SAGEDATE file size by removing these units and referencing them in SYSTEM.LIBRARY or SAGETOOLS.CODE.

The program may be executed from the Command line prompt of the p-System. Some users may want to make this routine the SYSTEM.STARTUP file so that they do not forget to set the date after booting. If the user typically uses the p-System Break key or runs into frequent p-System errors the use of SAGEDATE as SYSTEM.STARTUP may be inappropriate as the program is rerun during error recovery.

The initial output of the SAGEDATE program depends on whether the SAGE system's time has been previously set. If the time has been previously set the routine displays:

```
SAGE Time & Date Set:

  SAGE time & date:        Monday, 30-Aug-82  15:22:42

  Last set:                Sunday, 29-Aug-82  12:38:31

New date <CR for same>:
```

If the time was not previously set the routine displays:

```
SAGE Time & Date Set:

  Date on system device:  30-Aug-82

New date <CR for same>:
```

A new date may be typed in using one of three forms.

```
   30-Aug-82
or 8/30/82
or Aug 30, 1982
```

If only a single number is specified as a partial date, the number is used as the day. The remaining portions of the date will remain unchanged. If the complete date is to be unchanged, just enter a carriage return.

Next the program will prompt for the time:

```
New time <CR for none>:
```

If no time is to be set, just enter a carriage return. Otherwise, enter a twenty four hour time such as 15:22:50 for 3:22 PM and 50 seconds (the seconds are optional). If a date but no time value is entered, the SAGE real time clock will not be set. Only the date will be changed in the operating system and in the directories of the on-line devices.

All devices successfully updated will be noted:

```
SAGE updated with 30-Aug-82  15:22:50
Updated device(s) #4, #5, #11 with 30-Aug-82
```

Note that the SAGE real time clock will not automatically update the date normally used by the p- System. If SAGEDATE is run without changing the time or date, the p-System date will be updated if necessary to match the SAGE date from the real time clock. This will also cause an update of the dates in the directories of the on-line devices.

The time is kept using the crystal driving the processor. This crystal does not have the accuracy necessary to maintain exact time over a many day interval. If the time seems to drift objectionably over a several day period, there is a time correction factor which may be entered with the utility SAGE4UTIL. See the writeup on System Configuration under SAGE4UTIL for further details.

The SAGE Tool Kit contains a Time and Date Unit (TAD_Unit) which provides utilities for handling SAGE real time access from user programs. Thus, a user could easily replace the SAGEDATE program with a date set function from within his or her application. This utility also makes it easy to get the current time and date for display or printout by an application program.

VI.05 ATTACH IMPLEMENTATION :

The SAGE II/IV system BIOS supports several types of I/O events which may be ATTACHed to signal a semaphore. The VECTOR number in the ATTACH procedure call represents an Event number. Event numbers for each supported I/O event are generally obtained from a UNIT called ATT_UNIT. This will allow the actual event numbers to be changed in the future when event number standards are developed amoung p-System users. Note that although the current event numbers are presented in this document that they are subject to change in future releases.

The interface section of ATT_Unit is as follows:

```
INTERFACE

TYPE
  ATT_Events = (ATT_Key,ATT_TrmQE,ATT_RemIn,ATT_RotQE,
                ATT_ParQE,ATT_PrtQE,ATT_Break,ATT_Schd1,
                ATT_Schd2,ATT_Schd3,ATT_Schd4,ATT_TaskR,
                ATT_Ex1In,ATT_Ex1QE,ATT_Ex2In,ATT_Ex2QE,
                ATT_Ex3In,ATT_Ex3QE,ATT_Ex4In,ATT_Ex4QE,
                ATT_UKey,ATT_Win,ATT_Floppy,ATT_ComIn,
                ATT_ComDn);
FUNCTION ATT_Lookup(Event:ATT_Events):INTEGER;
```

The function ATT_Lookup is used to translate a symbolic event name into the actual event number which is recognized by the BIOS. The normal range of event numbers is 0 to 63. Although all the stated events are currently supported, the protocol will be to return "-1" for any event which is not supported. The following I/O events are supported by the SAGE II/IV BIOS:

ATT_Key (current value is 19) Attaching to this event will signal a semaphore whenever a key is pressed on the terminal. Note that a user program should use ATT_UKey instead in order to avoid conflicts with the p-System Print Spooler.

ATT_TrmQE (current value is 33) Attaching to this event will signal a semaphore whenever the Terminal Output Queue becomes empty.

ATT_RemIn (current value is 32) Attaching to this event will signal a semaphore whenever a character arrives in the Remote Input Queue.

ATT_RotQE (current value is 34) Attaching to this event will signal a semaphore whenever the Remote Output Queue becomes empty.

ATT_ParQE (current value is 35) Attaching to this event will signal a semaphore whenever the Parallel Printer Port Queue becomes empty. Note that generally the event ATT_PrtQE should be used for the printer because of its possible assignment to the Remote Serial channel.

ATT_PrtQE (current value is 34 or 35) Attaching to this event will signal a semaphore whenever the Printer Output Queue becomes empty. Note that the value returned will vary depending on whether the printer is assigned to the Remote Serial channel or the parallel port.

ATT_Schd1
ATT_Schd2
ATT_Schd3
ATT_Schd4

(current values 36-39) Attaching to one of these events will signal the semaphore whenever the appropriate schedule times out. Schedules for each of the events may be independently set up with a UNITWRITE to device 131 as described below.

ATT_Ex1In
ATT_Ex2In
ATT_Ex3In
ATT_Ex4In

(current values 40,42,44,46) These event numbers are associated with the input side of the four Auxiliary Serial Channels on the SAGE IV Winchester controller board. Attaching to one of these events will signal the semaphore whenever a character arrives in the channel's input queue.

ATT_Ex1QE
ATT_Ex2QE
ATT_Ex3QE
ATT_Ex4QE

(current values 41,43,45,47) These event numbers are associated with the output side of the four Auxiliary Serial Channels on the SAGE IV Winchester controller board. Attaching to one of these events will signal the semaphore whenever the channel's output queue becomes empty.

ATT_Win

(current value is 48) Attaching to this event will signal the semaphore whenever a Winchester UNITREAD or UNITWRITE is complete. This must be used in conjunction with the Asynchronous Transfer Flag (bit 0) in the Control Word of the I/O request.

ATT_Floppy

(current value is 49) Attaching to this event will signal the semaphore whenever a Floppy UNITREAD or UNITWRITE is complete. This must be used in conjunction with the Asynchronous Transfer Flag (bit 0) in the Control Word of the I/O request.

ATT_ComIn (current value is 50) Attaching to this event is only supported under the Multi-User BIOS. The semaphore will be signaled each time that a new message is sent to the user via the User Intercommunication facility.

ATT_ComDn (current value is 51) Attaching to this event is only supported under the Multi-User BIOS. The semaphore will be signaled whenever the transmission of a message is complete under the User Intercommunication facility.

ATT_UKey (current value is 61) Attaching to this event will signal a semaphore whenever a key is pressed on the terminal. This event should be used for general users instead of ATT_Key (event number 19). ATT_Key is used by the Print Spooler and could conflict with the user program.

ATT_Break (current value is 62) Attaching to this event will signal a semaphore whenever the p-System Break key is pressed and the Break is not inhibited (in SYSCOMREC).

Note that when this attachment is active, the Break does not call the normal System Break Routine or terminate the Start/Stop or Flush suspension. Users should be careful to avoid I/O calls which could hang waiting in the BIOS because the Break will not cause them to escape.

ATT_TaskR (current value is 63) The normal operation of p-System Processes is for the current Process to continue executing until it WAIT's on a semaphore or is preempted by a higher priority Process. Processes of equal priority which are ready must wait for the currently running process to WAIT on a semaphore before they get control.

Attaching to ATT_TaskR will cause a rotation of Processes with equal priority which are at the top of the Ready Queue. This rotation will occur on a schedule which is set up with a UNITWRITE to device 131 as described below. Note that if a set of equal priority Processes are being rotated on a periodic schedule and a higher priority Process is running when the scheduler hits, the rotation for that period will be missed. This is because the currently running Process is at the top of the Ready Queue and the rotation only affects Processes of equal priority which are at the top of the Ready Queue.

Note also that even though the attachment requires a semaphore to set up the event, the semaphore is never signaled as the Process rotation is only performed within the interpreter.

Caution: This feature is an extension to the p-System interpreter by SAGE and is generally not available on most other p-Systems.

General Warning

Although concurrent processing is a very powerful tool, two warnings are in order:

1. The ATTACH and EVENT features of the p-System are relatively new and require a more sophisticated BIOS than exists on many implementations. Therefore if the goal of a program is to achieve the greatest portability, these features should be avoided.

2. Concurrent processing is a much more complex and error prone philosophy than standard sequential processing.

Users not experienced with concurrent processing concepts will generally suffer a few painful learning experiences. Because events are generated asynchronously to the program, many problems may not be easy to duplicate. The key to keep in mind is that a higher priority Process or a Process rotation (if enabled) can generally occur at any point in a routine. Any data or resource which is shared must be carefully analyzed and protected against modification by two Processes at the same time.

Scheduled Events

The ATTACH and EVENT implementation on the SAGE II/IV supports four independent scheduled events and one special scheduled event which can cause a Process rotation. The schedules for these events are set up with a UNITWRITE to device 131.

The data buffer must contain a two word time interval for the scheduler. The first word must be a positive number of seconds for the scheduled time interval. The second word is treated as an unsigned integer in the range of 0 to 63999 to represent the number of 1/64000 of a second for the scheduled time interval.

The size field in the UNITWRITE is ignored.

The logical block number should be the event number obtained from ATT_Lookup with an argument of ATT_Schd1 to ATT_Schd4 or ATT_TaskR.

The Control word uses the least significant three bits to control the schedule functions.

Control Word

Bit 0 = 0 set up a new schedule.
= 1 cancel the schedule.

Bit 1 = 0 schedule is for one time only.
= 1 interval is periodic on the specified interval.

Bit 2 = 0 normal scheduled event.
= 1 sleep request (see following section).

A schedule may be canceled even though it was not previously active. The time data for the cancel function is ignored and never modified. A periodic schedule sets up its timeout times based on exact intervals from when the timeout should have happened in order to insure long term time consistency over many timeouts. Thus, if a one second periodic timeout is set up and I/O priorities delay the timeout for .1 second, the next interval will be shortened to .9 second to make the total number of timeouts over a long period be consistent.

Sleep Request

A method for putting a program to sleep for a specified time interval is available. This is performed using a UNITWRITE to device 131 with bit 2 of the Control Word set. The implementation uses one of the five Scheduled Event slots but does not actually cause an Event.

The program gives up control of the CPU until the specified time has elapsed, at which point it will return to the calling program. The logical block number field must contain one of the Event numbers used for Scheduled Events (ATT_Schd1 to ATT_Schd4 or ATT_TaskR). The size field is ignored. The data buffer must contain a two word time interval as defined in the Schedule Event section.

```
Example:

VAR
  Event:INTEGER;
  TimeValue:ARRAY[0..1] OF INTEGER;
BEGIN
  { Put program to sleep for 3 seconds }
  Event:=ATT_Lookup(ATT_Schd4);
  TimeValue[0]:=3;
  TimeValue[1]:=0;
  UNITWRITE(131,TimeValue,0,Event,4);
```

The following page shows an example of using the ATTUNIT.

Example:

```
USES ATTUNIT;
  { Set up a periodic scheduled signal on .5 second intervals }
VAR
  Event:INTEGER;
  Sched1:SEMAPHORE;
  TimeValue:ARRAY[0..1] OF INTEGER;
  Anything:INTEGER; {may be any variable as is not used}
BEGIN
  Event:=ATT_Lookup(ATT_Schd1);
  SEMINIT(Sched1,0);
  ATTACH(Sched1,Event);
  TimeValue[0]:=0;
  TimeValue[1]:=32000;
  UNITWRITE(131,TimeValue,0,Event,2);
  START(Procname,PID,200,129);
  ----
  { Cancel a previous schedule }
  UNITWRITE(131,Anything,0,Event,1);
  ATTACH(NIL,Event);
  ----
  { Set up a one-shot 10 second event }
  TimeValue[0]:=10;
  TimeValue[1]:=0;
  ATTACH(Sched1,Event);
  UNITWRITE(131,TimeValue,0,Event,0);
  ----
END.
{ Set up a Process rotation on 100 millisecond intervals }
VAR
  Event:INTEGER;
  Dummy:SEMAPHORE;
  TimeValue:ARRAY[0..1] OF INTEGER;
BEGIN
  Event:=ATT_Lookup(ATT_TaskR);
  ATTACH(Dummy,Event);
  TimeValue[0]:=0;
  TimeValue[1]:=6400;
  UNITWRITE(131,TimeValue,0,Event,2);
  -----
  { Terminate the rotation }
  ATTACH(NIL,Event);
  UNITWRITE(131,Dummy,0,Event,1);
END.
```

ATTACH/EVENT Numbers

EVENT#	DEVICE	CALL TO ATT_UNIT
19	Keyboard	ATT_Key
32	Remin	ATT_Remin
33	Term Queue empty	ATT_TrmQE
34	Remout Queue empty	ATT_RotQE or ATT_PrtQE
35	Parallel Prt Que empty	ATT_ParQE or ATT_PrtQE
36	Schedule 1	ATT_Schd1
37	Schedule 2	ATT_Schd2
38	Schedule 3	ATT_Schd3
39	Schedule 4	ATT_Schd4
40	Extra Channel 1 in	ATT_Ex1In
41	Extra Channel 1 out	ATT_Ex1QE
42	Extra Channel 2 in	ATT_Ex2In
43	Extra Channel 2 out	ATT_Ex2QE
44	Extra Channel 3 in	ATT_Ex3In
45	Extra Channel 3 out	ATT_Ex3QE
46	Extra Channel 4 in	ATT_Ex4In
47	Extra Channel 4 out	ATT_Ex4QE
48	Winchester	ATT_Win
49	Floppy	ATT_Flpy
50	User Intercommunications	ATT_ComIn
51	User Intercommunications	Att_ComDr
61*	User Keyboard	ATT_Ukey
62	Break Key	ATT_Break
63	Task Totation	ATT_TaskR

* NOTE: Since the printer spooler uses ATT_Key, user programs should use ATT_Ukey to avoid conflicts.

VI.06 MNU_Unit - Menu Unit :

The MNU_Unit was developed to support SAGE4UTIL and other system utility programs. It provides a consistent screen oriented user interface but will also work in a situation where no special terminal features are available. An application using the MNU_Unit first defines the menu information to the Unit. The menu information typically specifies a nested set of menus which display and update application data. The application gets control at each request of MNU_Unit for application data to display and for application data to be updated. Thus the application has total control on how the data is retrieved and stored.

Following is the Interface section of the MNU_Unit.

```
INTERFACE

TYPE

  MNU_Cat = (MNU_Integer,MNU_String,MNU_Hex,MNU_OnorOff,MNU_SubMenu,MNU_Event,
             MNU_Choice);
  MNU_Status= (MNU_Done,MNU_Get,MNU_Put,MNU_Enter,MNU_ReEnter,MNU_Exit);
  MNU_Style = (MNU_1Style,MNU_2Style);

VAR

  MNU_Value:INTEGER;
  MNU_HighValue:INTEGER;
  MNU_Boolean:BOOLEAN;
  MNU_StrValue:STRING;

  MNU_State:MNU_Status;
  MNU_MenuNumber:INTEGER;
  MNU_ItemNumber:INTEGER;
  MNU_General:INTEGER;
  MNU_Category:MNU_Cat;
  MNU_Fancy:BOOLEAN;

  MNU_Aborted:BOOLEAN;
  MNU_Reject:BOOLEAN;

  MNU_File:FILE OF CHAR;
  MNU_Fopen:BOOLEAN;
```

```
{ Procedures to Define a Menu }
PROCEDURE MNU_Menu(MenuName,MenuTitle:STRING; MenuStyle:MNU_Style; MenuNumber,
                   MenuWidth:INTEGER);
PROCEDURE MNU_CopyM(OldName,MenuName,MenuTitle:STRING; MenuNumber:INTEGER);

{ Procedures to Define Items in a Menu }
PROCEDURE MNU_ItemI(ItemName:STRING; General,Number:INTEGER;
                    HighLimit,LowLimit:INTEGER);
PROCEDURE MNU_ItemS(ItemName:STRING; General,Number:INTEGER);
PROCEDURE MNU_ItemO(ItemName:STRING; General,Number:INTEGER);
PROCEDURE MNU_ItemM(ItemName:STRING; General,Number:INTEGER;
                    MenuName:STRING; DispSubItem:BOOLEAN);
PROCEDURE MNU_ItemH(ItemName:STRING; General,Number:INTEGER;
                    Digits:INTEGER);
PROCEDURE MNU_ItemE(ItemName:STRING; General,Number:INTEGER;
                    Redisplay:BOOLEAN);
PROCEDURE MNU_ItemC(ItemName:STRING; General,Number:INTEGER);
PROCEDURE MNU_CopyI(MenuName:STRING; Number:INTEGER);


PROCEDURE MNU_Show(MenuName:STRING);

PROCEDURE MNU_Loop;

{ Procedures for User Screen Output }
PROCEDURE MNU_ClrScreen;
PROCEDURE MNU_Error;
FUNCTION MNU_YesorNo(Prompt:STRING):BOOLEAN;

{ Procedures for Item Name access }
PROCEDURE MNU_GetIname(MenuName:STRING; ItemNumber:INTEGER;
                       VAR ItemName:STRING);
PROCEDURE MNU_PutIname(MenuName:STRING; ItemNumber:INTEGER;
                       ItemName:STRING);

IMPLEMENTATION
```

● Menu Definition

The user first defines a set of menus in the initialization section of his program. Information on the menu entries is saved in Heap memory for the duration of the program. Normally the menus will form a nested tree structure with an outer menu leading down into various subsidiary menus.

Each new menu is initiated by calling the routine MNU_Menu with a set of arguments. The MenuName is a short string containing a unique name which is used to refer to the menu from the program or from other menus. The MenuTitle is a string which is centered at the top of the menu when the menu is displayed. MenuStyle is one of the two defined sytles of menus which will be described later. The MenuNumber is a number which is passed back to the application program when the menu is entered to identify which menu is active. The Menuwidth is the amount of room allocated in the menu to display the menu's data fields.

After defining the head of a menu, a set of menu items may be defined. These items may have values (decimal integer, string, boolean, hexadecimal) or they may indicate a subsidiary menu or an event to be triggered. Most menus fall under this normal first style of menu, referred to as MNU_1Style. A second style of menu, MNU_2Style, allows a mutually exclusive choice of one of the menu items. The item descriptor for this choice may be displayed in the data field of the parent menu.

Each of the menu item definition routines defines an ItemName, a General value, and a Number. The ItemName is a string descriptor that is displayed to identify the menu item. The General field defines a general number which may be used by the application program. The Number is an integer which is returned to the application program to identify which item of the menu is being selected.

A decimal integer item is specified with MNU_ItemI. The high and low limits for the number are specified in the procedure call. A string field is specified with MNU_ItemS. Boolean (on/off) items are specified with MNU_ItemO. MNU_ItemM indicates selection of a subsidiary menu. The subsidiary menu is identified by MenuName. DispSubItem is a boolean which indicates if subsidiary menu information is to be displayed. When the subsidiary menu offers a choice selection (MNU_2Style) the displayed information in the parent menu is the selected item name of the subsidiary menu. If information is displayed with any other style of subsidiary menu, the information must be passed as a string (through MNU_StrValue).

Hexadecimal displayed menu item values are indicated with MNU_ItemH. Digits indicates the number of digits to be displayed. A hexadecimal value may contain up to 8 digits (2 words). An event is indicated by MNU_ItemE. Events indicate to an application that the item was selected but have no other action. The Redisplay value indicates if the menu should be redisplayed after selecting the event (in case the event changes the screen). MNU_ItemC is used in MNU_2Style menus only to indicate a choice item.

Sometimes it is desirable to have a menu item under more than one menu. The MNU_Copy procedure saves memory space by sharing some of the item information. MenuName indicates the source menu and Number indicates the item number to be copied from that menu.

In some cases it is also desirable to have the same menu item selections but with a different title and menu identity. The MNU_Copy routine will set up a new menu header but share a common copy of the menu items. The OldName refers to the original MenuName being copied. MenuName is the name of the new menu. MenuTitle and MenuNumber are typically different for the new menu.

Menu Operation

Once the menus are defined, the application program starts the menu display using MNU_Show with the name of the initial menu. Now the Menu Unit will want to get data from the application program to display in the menu. Also, the Menu Unit may want to pass new data back to the application. The application communicates to the Menu Unit through a set of variables. The application calls MNU_Loop in a main loop which interpretes the Menu Unit's requests.

After each call to MNU_Loop the MNU_State variable will indicate the current status of the Menu Unit. A status of MNU_Done indicates that the outermost menu has been terminated and there is nothing more to do. The status of MNU_Get indicates that the Menu Unit wants a data entry. MNU_MenuNumber indicates the current menu number and MNU_ItemNumber indicates the item within that menu. MNU_General is the general value stored within the menu item. MNU_Category indicates the type of data expected for the item. Integer values are passed to the Menu Unit through MNU_Value. A hexadecimal value is passed with the low word in MNU_Value and the high word in MNU_HighValue. Boolean values are passed through MNU_Boolean and string values are passed through MNU_StrValue.

If the Menu Unit has data to return to the application, the MNU_State will have a status of MNU_Put. The same variables applicable to MNU_Get are used for MNU_Put, only the application should store the appropriate variable rather than retrieve a value. When a menu is first entered, MNU_State will contain a status of MNU_Enter. When a menu is terminated, the MNU_State contains a MNU_Exit status. On re-entry from a subsidiary menu the MNU_State status will be MNU_ReEnter. The MNU_Enter, MNU_Exit, and MNU_ReEnter will always have the MNU_MenuNumber set to the current menu.

If the menu is aborted with an "!" the variable MNU_Aborted is set to True. The system exits each menu level until the application sets the MNU_Aborted variable back to False. This allows the application to abort a number of levels back to a desired point of recovery.

If the application detects an error with an MNU_Get or MNU_Put request, it should set the MNU_Reject flag to True. The MNU_Error routine should be called to position the cursor for output of a user error message. The message should be output with a WRITE statement and will be terminated by the MNU_Unit with the message

```
, type space to continue
```

Two other routines are available for Event type request which may wish to do their own I/O. MNU_ClrScreen clears the screen and positions the cursor to the upper left corner. The function MNU_YesorNo outputs a prompt question and requires a Y or N answer.

If the Menu Unit has a file open for recording menu information, the file is accessable as MNU_File. A boolean variable MNU_Fopen indicates if the recording file is open. Applications may want to record information within event routines.

The MNU_Unit will work with a terminal setup which does not have any special features defined. MNU_Fancy has a TRUE value if GOTOXY is available to allow non-scrolled operation of the display.

For some applications it is desirable to alter the item names at run time verses the fixed item names set up during menu initialization. The routines MNU_GetIname and MNU_PutIname allow the item name of a menu entry to be accessed and updated after the menu has been defined. Note that the item name defined during the menu initialization defines the length of the area reserved for the item name. If an item name longer than the initial name is specified with MNU_PutIname, it will be truncated. A shorter item

name will be padded with trailing spaces to fill the available area.

The example on the following pages demonstrates the usage of the MNU_Unit with a simple application program. Note that the Initialization routine defines all the menu screens. The MNU_Show procedure call starts the menu system running at a specific menu screen. Then the main REPEAT loop of the routine invokes procedure MNU_Loop followed by processing of the menu system requests via interpretation of MNU_State, MNU_MenuNumber, and MNU_ItemNumber.

```
{ Sample Program using MNU_Unit }

PROGRAM SampleMenu;

USES {$U SAGETOOLS.CODE} MNU_Unit;

VAR
  Done:BOOLEAN;

  { Simulated application data }
  Data1:INTEGER;
  Data2:STRING;
  Data3:BOOLEAN;
  Data4H,Data4L:INTEGER;
  Choice1:INTEGER;
  Choice2:INTEGER;

PROCEDURE Initialization;
BEGIN
  { Outer level menu }
  MNU_Menu('M1','Outer Level Menu',MNU_1Style,1,0);
  MNU_ItemM('Demo of mixed menu entries',0,1,'M2',FALSE);
  MNU_ItemM('Demo of menu events',0,2,'M3',FALSE);
  MNU_ItemM('Demo of style 2 menu',0,3,'M4',FALSE);
  MNU_ItemM('Demo of menu copy',0,4,'M7',FALSE);

  { Mixed memu entries }
  MNU_Menu('M2','Mixed menu entries',MNU_1Style,2,12);
  MNU_ItemI('This is a non-zero integer field',0,1,32767,-32767);
  MNU_ItemS('This is a string field',0,2);
  MNU_ItemO('This is a boolean field',0,3);
  MNU_ItemH('This is a hexadecimal field',0,4,6);

  { Menu of events }
  MNU_Menu('M3','Event driver',MNU_1Style,3,0);
  MNU_ItemE('Trigger application event #1',0,1,TRUE);
  MNU_ItemE('Trigger application event #2',0,2,TRUE);
  MNU_ItemE('Trigger application event #3',0,3,TRUE);

  { Menu to demo display of choice from subsidiary menu }
  MNU_Menu('M4','Demo of displayed subsidiary choice',MNU_1Style,4,20);
  MNU_ItemM('Choice from submenu #1',0,1,'M5',TRUE);
  MNU_ItemM('Choice from submenu #2',0,2,'M6',TRUE);
  MNU_CopyI('M2',4); { copy of hex field entry from menu 2 }
```

```
  { Style 2 menues }
  MNU_Menu('M5','Select one item',MNU_2Style,5,1);
  MNU_ItemC('Selection #1',0,1);
  MNU_ItemC('Selection #2',0,2);
  MNU_ItemC('Selection #3',0,3);

  MNU_Menu('M6','Second Choice Menu',MNU_2Style,6,1);
  MNU_ItemC('Choice #1',0,1);
  MNU_ItemC('Choice #2',0,2);
  MNU_ItemC('Choice #3',0,3);

  MNU_CopyM('M2','M7','This is a copy of the mixed entry menu',7);
END;
  { Initialize application data }
  Data1:=1234;
  Data2:='Information';
  Data3:=FALSE;
  Data4H:=254;
  Data4L:=34;
  Choice1:=1;
  Choice2:=3;

PROCEDURE Event1;
BEGIN
  MNU_ClrScreen;
  WRITELN('This is simulated event #1');
  WRITELN('Type a carriage return to continue');
  READLN;
END;

PROCEDURE Event2;
VAR
  Data:BOOLEAN;
BEGIN
  MNU_ClrScreen;
  Data:=MNU_YesorNo('Are you enjoying this demo?');
  WRITELN('Type a carriage return to continue');
  READLN;
END;

PROCEDURE Event3;
BEGIN
  IF MNU_Fopen THEN WRITELN(MNU_File,'Sample log file output');
  MNU_ClrScreen;
  WRITELN('This event will output to the log file');
  WRITELN('Type a carriage return to continue');
  READLN;
END;
```

```
BEGIN
  Initialization;
  MNU_Show('M1');
  Done:=FALSE;
  REPEAT { Main data handling loop }
    MNU_Loop;
    CASE MNU_State OF
      MNU_Get:{ MNU_Unit wants application data to display }
              CASE MNU_MenuNumber OF
                2,7:CASE MNU_ItemNumber OF
                      1:MNU_Value:=Data1;
                      2:MNU_StrValue:=Data2;
                      3:MNU_Boolean:=Data3;
                      4:BEGIN
                          MNU_Value:=Data4L;
                          MNU_HighValue:=Data4H;
                        END;
                    END;
                4:BEGIN
                    MNU_Value:=Data4L;
                    MNU_HighValue:=Data4H;
                  END;
                5:MNU_Value:=Choice1;
                6:MNU_Value:=Choice2;
              END;
      MNU_Put:{ MNU_Unit is updating application data from user }
              CASE MNU_MenuNumber OF
                2,7:CASE MNU_ItemNumber OF
                      1:BEGIN
                          IF MNU_Value = 0 THEN
                            BEGIN
                              MNU_Error;
                              WRITE('Cannot accept zero value');
                              MNU_Reject:=TRUE;
                            END
                          ELSE
                            Data1:=MNU_Value;
                        END;
                      2:Data2:=MNU_StrValue;
                      3:Data3:=MNU_Boolean;
                      4:BEGIN
                          Data4L:=MNU_Value;
                          Data4H:=MNU_HighValue;
                        END;
                    END;
                4:BEGIN
                    Data4L:=MNU_Value;
                    Data4H:=MNU_HighValue;
                  END;
                3:CASE MNU_ItemNumber OF
                    1:Event1;
                    2:Event2;
                    3:Event3;
                  END;
                5:Choice1:=MNU_Value;
                6:Choice2:=MNU_Value;
              END;
      MNU_ReEnter:{ Cancel ! at main level menu }
                  IF MNU_MenuNumber = 1 THEN MNU_Aborted:=FALSE;
    END;
  UNTIL MNU_State = MNU_Done;
END.
```

VI.07 CONFIG_SAGE -Configuration Control Unit :

The Config_Sage Unit provides the record definitions of the SAGE II/IV BIOS configuration information and routines to read and write these configurations.

Config_Sage Unit INTERFACE TYPES:

```
INTERFACE

TYPE

  Conf_Rates =     (Conf_RSync,Conf_X1,Conf_X16,Conf_X64);
  Conf_DataBits = (Conf_5Data,Conf_6Data,Conf_7Data,Conf_8Data);
  Conf_StopBits = (Conf_Invalid,Conf_1Stop,Conf_1_5Stop,Conf_2Stop);
  Conf_PrtMode =  (Conf_None,Conf_Serial,Conf_ParInterrupt,Conf_ParPoll);

  Conf_Type =      (Conf_CvBaud,Conf_CvStepRate);
```

Config_Sage Unit Terminal Channel Configuration Record

```
{ Terminal Configuration }
Conf_Terminal = PACKED RECORD
                  BaudRate:INTEGER;
                  BreakAllowed:BOOLEAN;
                  XonXoffFlag:BOOLEAN;
                  Reserve1:0..63;
                  RateFactor:Conf_Rates;
                  DataBits:Conf_DataBits;
                  ParityEnabled:BOOLEAN;
                  ParityEven:BOOLEAN;
                  StopBits:Conf_StopBits;
                END;
```

Config_Sage Unit Remote Serial Channel Configuration Rec

```
{ Remote Serial Channel Configuration }

Conf_Remote = PACKED RECORD
                BaudRate:INTEGER;
                XONin:BOOLEAN;
                XONout:BOOLEAN;
                Reserve1:0..63;
                RateFactor:Conf_Rates;
                DataBits:Conf_DataBits;
                ParityEnabled:BOOLEAN;
                ParityEven:BOOLEAN;
                StopBits:Conf_StopBits;
                PollDelay:INTEGER;
                Reserve2:0..255;
                DSR_Polling:BOOLEAN;
                Reserve3:0..127;
              END;
```

Config_Sage Unit Printer Channel Configuration Record

```
{ Printer Channel Configuration }

Conf_Printer = PACKED RECORD
                 Timeout:INTEGER;
                 PollTime:INTEGER;
                 NoLF:BOOLEAN;
                 Reserve1:0..127;
                 Mode:Conf_PrtMode;
                 Reserve2:0..63;
               END;
```

Config_Sage Unit Floppy Configuration Record

```
{ Floppy Configuration }

  Conf_Floppy = PACKED RECORD
                  Cylinders:0..255;
                  Sides:0..255;
                  Gap3:0..255;
                  SectorsPerTrack:0..255;
                  DataLength:0..255;
                  Skew:0..255;
                  BytesPerSector:INTEGER;
                  MotorOnDelay:INTEGER;
                  NoDMAflag:BOOLEAN;
                  HeadLoad:0..127;
                  HeadUnload:0..15;
                  StepRate:0..15;
                  IBMflag:BOOLEAN;
                  NCIflag:BOOLEAN;
                  RAWflag:BOOLEAN;
                  Reserve1:0..31;
                  Reserve2:0..63;
                  MFMflag:BOOLEAN;
                  Reserve3:0..1;
                  IgnoreErrors:BOOLEAN;
                  Reserve4:0..127;
                  Tries:0..255;
                  DoubleStep:BOOLEAN;
                  Reserve5:0..127;
                  SoftErrors:0..255;
                  Gap3Format:0..255;
                  PatternFormat:0..255;
                  LastError:0..255;
                  FirstError:0..255;
                  Reserve6:ARRAY[0..4] OF INTEGER;
                END;
```

Config_Sage Unit RAM Disk Configuration Record

```
{ RAM Disk Configuration }

Conf_RamDisk = PACKED RECORD
                 BaseHigh:INTEGER;
                 BaseLow:INTEGER;
                 TopHigh:INTEGER;
                 TopLow:INTEGER;
                 Reserve1:0..255;
                 BootRamDisk:BOOLEAN;
                 Reserve2:0..127;
               END;
```

Config_Sage Unit System Configuration Record

```
{ System Configuration }

Conf_System = PACKED RECORD
                Seconds:0..255;
                Days:0..255;
              END;

Conf_OpSystem = ARRAY[0..15] OF INTEGER;

Conf_ChanTable = ARRAY[0..31] OF PACKED RECORD
                                   Channel:0..255;
                                   SubChannel:0..255;
                                 END;
```

Config_Sage Unit Winchester Configuration Record

```
Conf_Winch = PACKED RECORD
               Cylinders:INTEGER;
               BytesPerSector:INTEGER;
               StepTime:INTEGER;
               SlewTime:INTEGER;
               StepCtr:INTEGER;
               HeadSettleTime:INTEGER;
               PreCompTrack:INTEGER;
               SpecialType:INTEGER;
               Tests:INTEGER;
               ShipTrack:INTEGER;
               HeaderCount:INTEGER;
               LowReadCounter:INTEGER;
               HighReadCounter:INTEGER;
               SectorsPerTrack:0..255;
               Heads:0..255;
               Tries:0..255;
               SelectBit:0..255;
               LastError:0..255;
               FirstError:0..255;
               SoftErrors:0..255;
               LastHardError:0..255;
               SeekUnderflow:INTEGER;
               CRC:INTEGER;
               SyncBit:INTEGER;
               ExtraHeadSettle:INTEGER;
               WriteTries:0..255;
               RawTries:0..255;
               Reserved:ARRAY[1..8] OF INTEGER;
             END;
```

Config_Sage Unit Device Information Records

```
Conf_DevInfo = ARRAY[1..16] OF INTEGER;

Conf_WDevInfo = ARRAY[0..3,0..15] OF INTEGER;

Conf_File = PACKED RECORD
              Header:ARRAY[0..12] OF INTEGER;
              Name:PACKED ARRAY[0..3] OF CHAR;
              BiosSize:INTEGER;
              BiosBuffers:INTEGER;
              BisoStart:INTEGER;
              BiosVersion:0..255;
              BIOSSubVersion:0..255;
              Floppy0:Conf_Floppy;
              Floppy1:Conf_Floppy;
              Terminal:Conf_Terminal;
              Remote:Conf_Remote;
              RamDisk:Conf_RamDisk;
              Printer:Conf_Printer;
              TimeAdj:Conf_System;
              OpSystem:Conf_OpSystem;
              ChanTable:Conf_ChanTable;
              DevInfo:Conf_DevInfo;
              XSerial1:Conf_Remote;
              XSerial2:Conf_Remote;
              XSerial3:Conf_Remote;
              XSerial4:Conf_Remote;
              Winch1:Conf_Winch;
              Winch2:Conf_Winch;
              Winch3:Conf_Winch;
              Winch4:Conf_Winch;
              WinDevInfo:Conf_WDevInfo;
              DummyBlock:ARRAY[0..255] OF INTEGER;
            END;
```

Config_Sage Unit Configuration Procedures

```
PROCEDURE Conf_RD_Terminal(VAR Data:Conf_Terminal);
PROCEDURE Conf_Wt_Terminal(Data:Conf_Terminal);

PROCEDURE Conf_RD_Remote(VAR Data:Conf_Remote);
PROCEDURE Conf_Wt_Remote(Data:Conf_Remote);

PROCEDURE Conf_RD_Printer(VAR Data:Conf_Printer);
PROCEDURE Conf_WT_Printer(Data:Conf_Printer);

PROCEDURE Conf_RD_Floppy(Drive:INTEGER; VAR Data:Conf_Floppy);
PROCEDURE Conf_WT_Floppy(Drive:INTEGER; Data:Conf_Floppy);

PROCEDURE Conf_RD_RamDisk(VAR Data:Conf_RamDisk);
PROCEDURE Conf_WT_RamDisk(Data:Conf_RamDisk);

PROCEDURE Conf_RD_System(VAR Data:Conf_System);
PROCEDURE Conf_WT_System(Data:Conf_System);

PROCEDURE Conf_RD_OpSystem(VAR Data:Conf_OpSystem);
PROCEDURE Conf_WT_OpSystem(Data:Conf_OpSystem);

PROCEDURE Conf_RD_ChanTable(VAR Data:Conf_ChanTable);
PROCEDURE Conf_WT_ChanTable(Data:Conf_ChanTable);

PROCEDURE Conf_RD_XSerial(Device:INTEGER; VAR Data:Conf_Remote);
PROCEDURE Conf_WT_XSerial(Device:INTEGER; VAR Data:Conf_Remote);

PROCEDURE Conf_RD_Winch(Drive:INTEGER; VAR Data:Conf_Winch);
PROCEDURE Conf_WT_Winch(Drive:INTEGER; Data:Conf_Winch);

PROCEDURE Conf_RD_DevInfo(VAR Data:Conf_DevInfo);
PROCEDURE Conf_WT_DevInfo(Data:Conf_DevInfo);

PROCEDURE Conf_RD_WDevInfo(VAR Data:Conf_WDevInfo);
PROCEDURE Conf_WT_WDevInfo(Data:Conf_WDevInfo);
```

Config_Sage Unit Configuration Procedures (cont.)

```
{ Converts raw data to user familiar form }
FUNCTION Conf_Rd_Conversion(Device:INTEGER; DataType:Conf_Type;
                           Data:INTEGER):INTEGER;

{ Converts user familiar form to raw data }
FUNCTION Conf_Wt_Conversion(Device:INTEGER; DataType:Conf_Type;
                           Data:INTEGER):INTEGER;

{ Change Device Map Assignments }
PROCEDURE Conf_Assign(Channel,Device,SubUnit:INTEGER);

{ Restore Device Map Assignments }
PROCEDURE Conf_Restore;

{ Test for SAGE II/IV }
FUNCTION Conf_SGIV:BOOLEAN;
```

Record Descriptions

The Records describing the BIOS configuration data are exact images of the data that is stored in the BIOS. Some information is stored in a form more usable by hardware than by humans. Two routines, Conf_Rd_Conversion and Conf_Wt_Conversion, are provided to help translate certain popular items (serial channel baud rates and floppy step rates) into an easier to understand form. Record entries of the form 'ReserveX' are used to pad the packed records and force the placement of the meaningful data items.

Terminal Configuration

BaudRate Zero defaults to using the switch settings (when using switch settings, the ParityEven must be set to TRUE). The value is 38400 divided by the desired baud rate. The conversion routines may be used to convert back and forth to normal baud rate numbers.

BreakAllowed TRUE if Terminal BREAK Key (not p-System break key) is to enter the PROM Debugger.

XonXoffFlag TRUE if XON/XOFF protocol is to be used on output to the terminal. Note that the p-System Start/Stop character must be defined as XOFF (CTRL S).

RateFactor This selects the USART baud rate factor and should always be left at Conf_X16 for sixteen times clock rate.

DataBits This factor selects the number of terminal data bits (typically Conf_8Data for eight data bits).

ParityEnabled TRUE to enable sending and receiving an extra bit of parity with the transmission, FALSE if parity is disabled.

ParityEven TRUE if Even parity is to be handled, FALSE if Odd parity is to be handled.

StopBits This factor selects the number of terminal stop bits to be processed (typically Conf_1Stop for one stop bit).

Remote Configuration

BaudRate Zero defaults to using 9600 baud. If the Remote channel (on CPU board) is used this value is 38400 divided by the desired baud rate. If one of the Auxiliary Serial channels from the Winchester board is used, the value is a code from the following table. The conversion routines may be used to convert back and forth to normal baud rate numbers.

BaudRate Code (Aux only)

48	-	50 baud
49	-	75 baud
50	-	110 baud
51	-	134.5 baud (unsupported)
52	-	150 baud
53	-	300 baud
54	-	600 baud
55	-	1200 baud
56	-	1800 baud
57	-	2000 baud
58	-	2400 baud
59	-	3600 baud (unsupported)
60	-	4800 baud
61	-	7200 baud (unsupported)
62	-	9600 baud
63	-	19200 baud

XONin TRUE if using XON/XOFF protocol on Remote input.

XONout TRUE if using XON/XOFF protocol on Remote output.

RateFactor This selects the USART baud rate factor and should always be left at Conf_X16 for sixteen times clock rate.

DataBits This selects the number of Remote channel data bits (typically Conf_7Data or Conf_8Data for seven or eight data bits).

ParityEnabled TRUE to enable sending and receiving an extra bit of parity with the transmission, FALSE if parity is disabled.

ParityEven TRUE if Even parity is to be handled, FALSE if Odd parity is to be handled.

StopBits This factor selects the number of terminal stop bits to be processed (typically Conf_1Stop for one stop bit).

PollDelay This delay time (in 1/64000ths of a second) is scheduled between DSR checks if the DSR check flag is set and the DSR signal is not asserted.

DSR_Polling TRUE indicates that the Data Set Ready signal must be asserted before a character is transmitted. If DSR is not asserted, a delay (see PollDelay) is scheduled for a future retest.

Parallel Printer Channel Config.

Timeout This entry is used in printer mode 'ParPoll" to control the number of 4.25 microsecond cycles that the BIOS will poll the printer before going to sleep (236 counts is 1 millisecond).

PollTime This entry is used in printer mode 'ParPoll' to control the amount of time between attempts to poll the printer Busy line.

NoLF TRUE to inhibit the automatic generation of a line feed after each carriage return in the p-System interpreter. Although this flag is in the parallel printer configuration, it applies to all devices assigned to the printer.

Mode

None	-no printer channel is supported.
Serial	-use the Remote Serial output channel. (Note that this selection is no longer valid as printer assignment is handled via the BIOS Channel Map.)
ParInterrupt	-use parallel port with interrupt operation.
ParPoll	-use parallel port with Busy signal polling.

Floppy Configuration

Cylinders Number of cylinders on diskette.

Sides Number of sides on diskette.

Gap3 Internal parameter for floppy controller which varies depending on the number of sectors per track and number of bytes per sector.

SectorsPerTrack Number of sectors on each diskette track.

DataLength Only used for sectors of less than 256 bytes in length (normally set to 255 by default).

Skew Track to track skew value which is only used by the formatter to skew the sector addresses during formatting.

BytesPerSector Number of bytes per sector on diskette.

MotorOnDelay Delay between turning motor on and attempting to use the floppy (in 1/64000ths second).

NoDMAflag This flag must always be TRUE for correct floppy operation.

HeadLoad This is the head load time factor (head load time divided by 4 milliseconds) for the 765 floppy controller.

HeadUnLoad This is the head unload time factor (head unload time divided by 32 milliseconds) for the 765 floppy controller.

StepRate This code determines the seek step rate. The code may be translated to and from millisecond step rate values (2 to 32 milliseconds) by Conf_Rd_Conversion and Conf_Wt_Conversion.

IBMflag TRUE for IBM track compatibility.

NCIflag TRUE for Network consulting's special sector numbering scheme for 10 sector per track diskettes.

RAWflag TRUE for Read after Write data verification.

MFMflag TRUE for double density operation, FALSE for single density operation.

IgnoreErrors TRUE if errors are ignored (typically only used during alignment procedure).

Tries This location (previously called Retries) contains the number of attempts that the driver will try to access the diskette. The value is one greater than the actual retry count since it counts the original attempt.

DoubleStop TRUE if using the cylinder number times two to read 48 TPI diskettes on a 96 TPI drive.

SoftErrors This location is only informational. The value is the number of floppy transfer errors which were retried.

Gap3Format This value is the internal GAP 3 parameter for the floppy controller while formatting. The value depends on the number of sectors per track and bytes per sector.

PatternFormat This is the value which is placed in every byte of the diskette during formatting.

LastError This location is only informational. It contains the code of the last error occuring on the floppy drive.

FirstError This location is only informational. It contains the code of the first error which occured on the floppy drive.

RamDisk Configuration

BaseHigh High word of the base of the RamDisk area.

BaseLow Low word of the base of the RamDisk area. When BaseHigh and BaseLow are both zero, the Ram Disk is disabled.

TopHigh High word of the top of the RamDisk area.

TopLow Low word of the top of the RamDisk area.

BootRamDisk TRUE to copy floppy to RamDisk and then boot to RamDisk device.

System Configuration

Only a time adjustment option is supported under this selection at the current time.

Seconds Time adjustment factor in X days.

Days Time adjustment factor number of seconds. Seconds and Days define a correction for the Real Time clock drift as a number of seconds (+ or - 127) in a number of days (up to 10). Zero days means no correction factor is defined.

Operating System Data

Conf_OpSystem is an area which may contain information to configure an operating system environment when bootstrapping. It is presented here as an array of 16 integer values. This area is currently unused by the p-System.

BIOS Channel Map

Conf_ChanTable is an array of 32 entries indexed by logical device number and containing the physical device (Channel) and physical subdevice (SubChannel) numbers.

Winchester Configuration

Cylinders Number of cylinders on the Winchester drive.

BytesPerSector Number of bytes per logical sector on the Winchester drive.

StepTime Single step time used for estimated seek completion time calculation.

SlewTime Head slew time used for estimated seek completion time calculation.

StepCtr Value applied to 8253 counter to time interval between step pulses.

HeadSettleTime Head settle time used for estimated seek completion time calculation.

PreCompTrack Cylinder number where precompensation is to start on the Winchester drive.

SpecialType This field is reserved for flags to indicate special types of drive operation.

Tests This field contains bits which specify special test modes of operation for the drive and controller.

Bit 0: Read/Write from controller memory to disk.
Bit 1: Generates number of step pulses from Logical Block Number.
Bit 2: Read / Write only to the controller memory buffer.
Bit 3: Inhibit VCO initialization.
Bit 4: Apply first byte of buffer to controller D/A converter.
Bit 5: Do direct seek to cylinder contained in Logical Block Number without reading or writing.
Bit 6: On read, fetch VCO parameters and data. On write, apply calibrate value to VCO.
Bit 7: Read back port A from 8255 in Winchester controller.

ShipTrack Cylinder number at which the head should be positioned for shipping the drive.

HeaderCount Number of zero bytes written to the Winchester drive before the actual system and user data.

LowReadCounter Value used internally by the Winchester driver in the Read initialization sequence.

HighReadCounter Another value used internally by the Winchester driver in the Read initialization sequence.

SectorsPerTrack Number of logical sectors available on each Winchester track.

Heads Number of heads on the Winchester drive.

Tries Number of attempts that the driver will make on normal accesses of the Winchester drive before returning an error code.

SelectBit This bit controls the drive select and should be either 1, 2, 4, or 8.

LastError This field is provided to read back the last error code (possibly soft error) generated by the driver.

FirstError This field is provided to read back the first error code (possibly soft error) generated by the driver.

SoftErrors This field is provided to read back the soft error counter.

LastHardError This field is provided to read back the last hard error code generated by the driver.

SeekUnderFlow This field is provided to read back a count to gauge the efficiency of the seek timing parameters.

CRC This field is provided to read back the last CRC value generated by the Winchester driver.

SyncBit This bit is written to the Winchester disk after the header to provide appropriate byte synchronization.

ExtraHeadSettle This value is used to add an additional head settle time if necessary after the Winchester drive's seek complete indication.

WriteTries This count is the number of times that the Winchester driver will attempt to perform its cycle of Write followed by Read after Write.

RawTries This count is the number of times that the Winchester driver will attempt to do its Read after Write before initiating a re-write.

CP/M Device Information

Conf_DevInfo contains a word of configuration information for each physical device to support the CP/M-68k operating system. Conf_DevInfo is indexed by the physical device number. It handle all devices except for the Winchester drives which have a separate table (Conf_WDevInfo) to provide information on each partition. Conf_WDevInfo for the Winchester is indexed by drive number and partition number.

BIOS File

Conf_File is a record showing the layout of the configuration information which is stored in the SYSTEM.BIOS file. The record starts at the beginning of block 1 in the BIOS file. Header represents the 26 bytes of leading p-System data which precedes the code image. Name should always be the four characters 'BIOS'.

Procedure Definitions

The Conf_Rd_xxxx and Conf_Wt_xxxx procedures are used to read and write the various configuration records to the BIOS. Note that the floppy, Winchester, and Auxiliary Serial channels configuration routines require that the logical channel number also be included to identify the specific device. Note that the Auxiliary Serial Port logical channel numbers are always 28 to 31, not the normal biased numbers used by the p- System.

The Conf_Rd_Conversion routine is used to convert raw values for baud rate or step rate into user understandable numbers. For instance a baud rate value of 4 on the terminal port would be converted to 9600 (for 9600 baud). A step rate value of OFH would be converted into 2 (for 2 milliseconds). The Conf_Wt_Conversion routine is used to convert user understandable numbers back to the raw values for output to the BIOS.

> Note that the device numbers used as arguments for the Conversion routine are Physical device numbers and NOT logical device numbers. See page 35 .

The BIOS Channel Map may be modified using the routine Conf_Assign. Channel represents the logical channel to be assigned. Device and Subunit are the physical device number to be associated with the logical channel. When any program using CONFIGSAGE is started, the original BIOS Channel Map is saved. If a program modifies the Channel Map via Conf_Assign or Conf_Wt_ChanTable, the original contents can be restored by calling the procedure Conf_Restore.

The function Conf_SGIV is used to determine if a Winchester Controller board is equipped. It returns TRUE if the Winchester board exists and FALSE if the Winchester board is missing.

EXAMPLE USAGE

The typical use of the Config_Sage Unit is to set up the Remote Serial Channel for a program to control a special device. In these cases, the Conf_Rd_Remote procedure is generally used to read in the previous Remote configuration information. Then the Conf_Wt_Conversion and Conf_Wt_Remote routines are used to set up a configuration for the special device. When the program terminates, the original Remote configuration is generally restored.

The following page shows a sample routine which uses this technique to set up the Remote channel for the Intex Talker Speech Synthesizer.

```
PROGRAM Speak;

USES {$U SAGETOOLS.CODE} Config_Sage;

VAR
  S:STRING;
  Port:FILE OF CHAR;
  Old_Config,New_Config:Conf_Remote;

BEGIN
  Conf_Rd_Remote(Old_Config); { Save previous Remote }

  { Set up new Remote configuration }
  FILLCHAR(New_Config,SIZEOF(New_Config),0);
  WITH New_Config DO
    BEGIN
      BaudRate:=Conf_Wt_Conversion(Conf_IvBaud,9600);
      XONin:=FALSE;
      XONout:=FALSE;
      RateFactor:=Conf_X16;
      DataBits:=Conf_8Data;
      ParityEnabled:=TRUE;
      ParityEven:=TRUE;
      StopBits:=Conf_1Stop;
      DSR_Polling:=TRUE;
    END;
  Conf_Wt_Remote(New_Config);

  REWRITE(Port,'#8:');

  { Echo input lines to the Remote channel }
  REPEAT
    WRITE('>'); { Prompt }
    READLN(S);
    IF LENGTH(S) > 0 THEN WRITELN(Port,S);
  UNTIL LENGTH(S) = 0; { Blank line terminates }

  { Put back original configuration }
  Conf_Wt_Remote(Old_Config);
END.
```

Configuration Control

Reading and Writing the General System Configuration (Channel 0) has been modified slightly to multiplex in new types of information. The read or write to device 128 with a control word of 0 is still used. However, the block number is now considered to indicate which type of system information is being accessed. A block number of zero is used to maintain the previously described Time Adjustment Factor. A block number of two will read and write the 32 bytes of Operating System Configuration information. A block number of one will read and write the BIOS Channel Map. The Channel Map is arranged as follows:

```
.BYTE   0       ;Ch  0 subdevice - unused
.BYTE   0       ;Ch  0 device    - unused
.BYTE   0       ;Ch  1 subdevice
.BYTE   1       ;Ch  1 device
.BYTE   0       ;Ch  2 subdevice
.BYTE   2       ;Ch  2 device
.
.
.
```

Two routines in the CONFIGSAGE Unit allow easy manipulation of the BIOS Channel Map. The routine Conf_Assign allows the user to assign a logical channel to a physical device (and subdevice). Also, when the program first starts up, the Unit records the current status of the BIOS Channel Map. This initial map may be restored by the program, typically on termination, by envoking the routine Conf_Restore.

The configuration layout of the extra serial channels is the same as for the remote channel. The configuration of a Winchester drive pertains to the drive so the partition portion of the subdevice number is ignored.

VI.08 WIN_UNIT -Winchester Unit :

The following page shows the INTERFACE section on WIN_Unit.

The Winchester driver currently reserves physical track zero on each drive for use only by the driver. This area contains the Device Map and Bad Track Map for the drive. SAGETOOLS contains a Pascal Unit called WIN_Unit which defines records for the layout of the track zero area. Currently only 1024 bytes on track zero are used. The remainder is reserved by SAGE for future expansion.

The record Win_DiskImage is the image of the 1024 bytes stored at physical track zero, offset zero of the disk. The field DevMap defines the first and last track for each of the 16 possible disk partitions. Partition zero is reserved for system use and must always represent the complete disk (including spare tracks).

BadTracks is an array of 64 track numbers which are to be bypassed. PromInfo is a record of information used by the PROM resident Winchester driver after the initial track is read. DevData contains the optional partition boot name. The field System within DevData contains an optional number which may be used to identify the type of system stored in the partition. Currently eight systems are defined.

0 = unspecified system
1 = UCSD p-System
2 = CP/M-68k
3 = Volition Systems' Modula II environment.
4 = HyperForth +
5 = PDOS
6 = MIRAGE
7 = BOS
8 = IDRIS

```
INTERFACE

CONST
  Win_BadMax = 63;

TYPE
  Win_NameType = PACKED ARRAY[0..7] OF CHAR;

  Win_BadTracks = ARRAY[0..Win_BadMax] OF INTEGER;

  Win_DevMap = ARRAY[0..15] OF
                RECORD
                  BaseTrack:INTEGER;
                  TopTrack:INTEGER;
                END;

  Win_DevData = ARRAY[0..15] OF
                 PACKED RECORD
                   Name:Win_NameType;
                   Dummy:0..255;
                   System:0..255;
                   Data:ARRAY[0..10] OF INTEGER;
                 END;

  Win_PromInfo = PACKED RECORD
                   Heads:0..255;
                   SectorsPerTrack:0..255;
                   BytesPerSector:INTEGER;
                   LowReadCounter:INTEGER;
                   HighReadCounter:INTEGER;
                   StepCtr:INTEGER;
                 END;

  Win_DiskImage = RECORD
                    DevMap:Win_DevMap;
                    BadTracks:Win_BadTracks;
                    Dummy:ARRAY[0..154] OF INTEGER;
                    PromInfo:Win_PromInfo;
                    DevData:Win_DevData;
                  END;
```

VII THE IEEE-488 SUPPORT :

The various operating systems available on the SAGE provide support for the IEEE-488 in various ways. This section is specific to support under the p-System. For other operating systems, refer to that manual.

SAGE provides support for the IEEE-488 under the p-System with a UNIT which the user can interface to his main Pascal program. The Unit calls a 68000 assembly code program which accesses the TMS9914. Source of both the unit and assembly code program are provided to the user as all implementations of the IEEE-488 higher level protocol may not be the same. It also gives the user the option of optimizing the code for his own application.

VII.01 GENERAL BUS OPERATION :

MAX DATA RATE 1 Mbyte per second. As all transfers use handshake, devices on the bus do not have to work at the same speed.

MAX DEVICES 15. Max of 1 TALKER and up to 14 listeners at one time. (They can all be SAGE II/IVs) Any of the devices can send an SRQ request for service to the controller at any time.

LINE LENGTH 20 Meters total or 2 meters per device whichever is less.

SIGNAL LINES 8 data lines and 8 interface lines.

PROTOCOL Byte-serial, bit parallel, asynchronous data transfer using interlocking three-wire handshake technique.

HARDWARE TMS9914 controller, open collector or tri-state bus. Tri-state gives the faster speed if the devices support it.

VII.02 IB_UNIT :

The Pascal unit IB.UNIT and the assembly code program IB_BUS are set up to provide a bus configuration where the SAGE II/IV is the bus controller and other devices are simple TALK/LISTEN instruments. Program hooks exist in IB_BUS for transfer of control to another instrument and parallel poll, although these routines are not implemented at this time.

IB_UNIT defines these global variables for the user to access:

```
IB_SAGE,          The bus address of the SAGE II/IV
IB_ERR:INTEGER;   If <>0, an error has occurred
IB_CHK:BOOLEAN;   Check for and write any error messages
IB_X,
IB_Y:INTEGER;     Write the errors at screen postion X,Y
```

The procedures available in IB_UNIT are:

```
FUNCTION  IB_SWITCH:INTEGER;
PROCEDURE IB_INIT(VAR CNTRL,ADDR:INTEGER;CMDWAIT:INTEGER);
PROCEDURE IB_STAT(BDEV,STATUS:INTEGER);
PROCEDURE IB_TALK(VAR TBUF :INTEGER;LNG:INTEGER);
PROCEDURE IB_TALKS(S:STRING);
PROCEDURE IB_HEAR(VAR LBUF ,LNG:INTEGER;MORE:BOOLEAN);
PROCEDURE IB_HEARS(VAR S:STRING);
FUNCTION  IB_CHKSRQ:BOOLEAN;
PROCEDURE IB_SPOLL(VAR PDEV:INTEGER);
PROCEDURE IB_DIR(RDW,REG:INTEGER;VAR VAL:INTEGER);
```

Each procedure will be described in the order that it is normally used.

VII.03 STARTING UP :

There are several simple steps to the initialization process. The following is an example of an initialization procedure using IB_UNIT. Details of how it was created will be described.

```
PROCEDURE DOINIT;
{Assumes SAGE II/IV is controller}
CONST TLK=1;LST=2;TAK=4;CTR=8;SRQ=16;PP=32;IAM=64;
VAR CONTROL,SW:INTEGER;
BEGIN
{YES, DO chk and display err messages}
  IB_CHK:=TRUE;
{wrt them at 0,22}
  IB_X:=0;
  IB_Y:=22;
{Read GROUP-B switch}
  SW:=IB_SWITCH;
{Set SAGE ADRESS}
  IB_SAGE:=ORD(ODD(SW) AND ODD(31));
{INIT:1=CONTROLLER,SAGE ADDR, 46 USEC CMD}
  CONTROL:=1;
  IB_INIT(CONTROL,SW,23);
  IF IB_ERR<>0 THEN EXIT(IB_EX);
{Define devices}
  IB_STAT(IB_SAGE,TLK+LST+TAK+CTR+IAM);
    {talk,listen,control,SELF: =79}
  HP1615:=15;
  IB_STAT(HP1615,TLK+LST+SRQ);
    {HP1615=DEV 15 can talk,listen, send SRQ
    Serial poll by default :  = 19}
END;
```

When first setting up the bus, the user will want to write simple test routines to try out the unit and instruments on the bus. As a convenience during this time, the unit has an option to display any error messages that occurred during a call. Once the user has progressed beyond the start-up stage, he may want to shut off this option and provide error recovery as necessary for his application.

Error codes are always put in **IB_ERR** for the user after every call. A list of error codes exists at the end of this section.

Set **IB_CHK** TRUE for the unit to check for errors and display them on the terminal. **IB_X** and **IB_Y** must be set to tell the unit where to display the error on the screen.

Set **IB_CHK** FALSE to not display any error. **IB_X** and **IB_Y** should be set to 0.

VII.04 IEEE 488 SWITCH OPTIONS: :

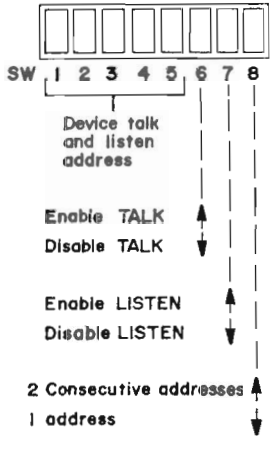
The SAGE II/IV must be assigned a bus address from 0-31. In the program example provided, it is assigned address 7. Generally, the address of a bus device is assigned through use of an 8 bit dip switch. GROUP-B on the SAGE II/IV is used for this purpose. The user does not HAVE to use the switches. The SAGE II/IV address and function bits can be defined directly if the switches are being used for another application.

If used, the standard definition for the switches is:

Sw1-Sw5	A1-A5	Sage II/IV Listen & Talk address.
Sw6	Dat	used to disable TALKER function. set to 0 , enabling TALK.
Sw7	Dal	used to disable Listen function. set to 0 ,enabling LISTEN.
Sw8	edpa	used to assign 2 consecutive address to one device.

As the SAGE II/IV is generally a controller, set Sw6-Sw8 OFF (down). Set Sw1-Sw5 to the address.

This figure shows the factory setting for GROUPB SWITCHES for IEEE-488.



The GROUP-B switches are read by calling FUNCTION IB_SWITCH:

```
SW:=IB_SWITCH;
```

The entire switch setting should be given to the IB_INIT routine. Just the Talk and LISTEN address (the low 5 bits) of the SAGE II/IV should be put in IB_SAGE:

```
IB_SAGE:=ORD(ODD(SW) AND ODD(31)); { mask off address}
```

The switches on the other bus devices are defined as above. No two devices should have the same address.

VII.05 BUS INITIALIZATION :

Initialization of the bus is provided by calling IB_UNIT:

```
PROCEDURE IB_INIT(VAR CNTRL,ADDR:INTEGER;CMDWAIT:INTEGER);
```

Set CNTRL = 1 if the SAGE is the controller, else 0.

Set ADDR = s the Talk & Listen address of the SAGE II/IV. Usually the value read from the switches. It can be set without reading the switch. All 8 bits must be defined as the high bits set / disable talk and listen functions. See the Switch Options.

Set CMDWAIT= x Time required for slowest device to do cmd. Specify in usec, divided by 2.

The CMDWAIT time is necessary for devices (such as the HP1615A logic analyzer) which will handshake on a bus command but not actually finish processing it for a longer period of time. This time is only for a command, not a transfer of data. The time must be set for the SLOWEST device. Each "tick" is 2us. Convert the time to usec then divide by 2 to get CMDWAIT. If the instrument documentation does not specify a command time, set CMDWAIT:=1; If this is too fast, the device may ignore the controller commands especially when changing from one mode of operation to another. Increase CMDWAIT until satisfied with the device response.

During the initialization process these events occur:

1. TMS9914 interrupts are shut off.
2. The 488 bus is cleared with an IFC.
3. The remote enable mode is ON.
4. The SAGE is optionally the IEEE-488 bus controller.
5. All devices are inhibited from talking or listening.
6. The BUSSTATUS array is zeroed.

VII.06 DEVICE DEFINITION :

Now the user program must define the devices on the bus and what they can do. This is done with **IBSTAT** :

```
PROCEDURE BUSSTAT(BDEV,STATUS:INTEGER);
```

Call it once for each device with

BDEV = Bus address of the device. The switch setting on the back of each device determines the device address. These are generally labeled A5-A1 and create an address in the range 0-31. No two devices can have the same address.

STATUS = The value of this is created by adding up the functions that the device can do:

```
TLK=   1: device can talk
LST=   2: device can listen
TAK=   4: device can take control
CTR=   8: device inits with control
SRQ=  16: device can send an SRQ
PP=   32: device expects to be parallel polled
          (default is serial poll)
IAM=  64: self - this device is the SAGE II/IV
```

EXAMPLE:

```
BUSSTAT(DEV15,TLK+LST+SRQ);
```

At this point, the initialization process is now complete. IB_INIT can be recalled if necessary at any time.

VII.07 TALKING :

Sending data from the SAGE II/IV to a device on the bus is done with either :

```
PROCEDURE IB_TALK(LDEV:INTEGER;VAR TBUF:INTEGER;LNG:INTEGER);
PROCEDURE IB_TALKS(LDEV:INTEGER;S:STRING);
```

LDEV is set to the address of the device that will LISTEN.

LNG is the number of byte/chars to send.

S TALKS is a special case of TALK where string S is sent to the listener. Up to 80 chars are sent. If the listener requires a terminator other than CR with EOI (end-of-input flag) it must be added to the string. TALKS is convenient for most character transfers, the user does not have to set up a buffer area.

TBUF is the address of the first byte that the SAGE II/IV will send. See the following explanation on setting up the USER BUFFER.

EXAMPLE:

```
TALKS('MDO;'); {Sets HP1615 to 24-bit mode}
               {The
```

TALK will set up the DEV device to listen, the SAGE II/IV to talk and send the data. An error will be returned via **IB_ERR** if not successful:

4 timeout occurred while talking - no handshake.
7 Listen dev is not capable of listening.

VII.08 LISTENING :

To receive data from another device on the bus use:

```
PROCEDURE IB_HEAR(TDEV:INTEGER; VAR LBUF,LNG:INTEGER;MORE:BOOLEAN);
PROCEDURE IB_HEARS(TDEV:INTEGER;VAR S:STRING);
```

LBUF is the addr/ptr of where to put the received data. HEAR does not set up a buffer but works directly out of the area the user gives it. See explanation following on the USER BUFFER.

LNG is set to the amount of data expected. The amount of data actually received is returned.

MORE is set false to start getting data. If error 2 occurred, indicating that the user area is full but the device has more data to send, save the data and call HEAR again with MORE:=TRUE to finish receiving.

S HEARS is a special case of HEAR where string 'S' is received from the listener. Up to 80 chars can be received. HEARS is convenient for most character transfers, the user does not have to set up a buffer area.

TDEV is set to the device which will talk.

EXAMPLE:

The device DEV is set to talk, the SAGE II/IV to listen. Data receive is terminated when an EOI (end-of-input) is sent with the last byte or LNG number of bytes/chars has been received. If less than LNG has been received and a 5 sec timeout occurs while waiting for the next data, an error exit occurs. The data already received is kept for the user. IB_ERR is returned as non-zero on an error:

FULL	=	2	LNG btyes received but no EOI.
RTMOUT	=	3	timeout occurred while listening.
NOTALK	=	6	Device not capable of talking.

```
HEARS(HPS);
```

VII.09 USER BUFFER :
The TALK and HEAR routines do not set up their own buffer areas. Instead they work directly out of an area defined by the user. This allows the user to create any size buffer needed. It also avoids the overhead of moving data back and forth between the Unit and the user program.

The data sent/received on the IEEE bus is in 8-bit bytes. The user buffer should be defined as a PACKED ARRAY[x..y] OF 0..255. As byte parameters cannot be passed through procedure calls, the buffer definition must include a variant so that the starting address of the buffer can be accessed.

The following example shows how a buffer of 512 bytes is set up. It also shows that the buffer need not be as large as the amount of data expected. Use of the "MORE" option in IB_HEAR allows receiving data in blocks the size of the buffer.

```
PROCEDURE GETDATA;
CONST FULL=2;
VAR STATE:PACKED RECORD CASE INTEGER OF
      1:(B:PACKED ARRAY[0..511] OF BYTE);
      2:(W:INTEGER);
      END;
    MORE:BOOLEAN;
    DLNG:INTEGER;

BEGIN
  MORE:=FALSE;
  DLNG:=512;
  REPEAT
    IB_HEAR(HP1615,STATE.W,DLNG,MORE);
    SAVEIT;                    {Save data}
    MORE:=TRUE;
  UNTIL IB_ERR<>FULL;
END;
```

VII.10 SERVICE REQUESTS :

When a device on the bus wants to talk to the controller, it sends an SRQ (service request). As this implementation has TMS9914 interrupts disabled, the user must check for an SRQ by periodically calling:

```
FUNCTION IB_CHKSRQ;
```

If IB_CHKSRQ is true, then an SRQ has occurred. A serial poll must be done to find out who is requesting service:

```
PROCEDURE IB_SPOLL(VAR PDEV:INTEGER);
```

The address of this device is returned in **PDEV** . The user should then HEAR that device to get the information.

The following procedure is an example of how the controller might wait to receive requests from other devices on the bus:

```
PROCEDURE ACTION;
{Wait for SRQ, Do a serial poll,get action to be done}
CONST NOSRQ=9;
VAR PDEV,SCOUNT:INTEGER;
    ACT:STRING;
BEGIN
  SCOUNT:=0;
  REPEAT
    SCOUNT:=SCOUNT+1;
  UNTIL (IB_CHKSRQ) OR (SCOUNT>100);
  IB_SPOLL(PDEV);
  IF PDEV<>0 THEN
    BEGIN
      HEARS(PDEV,ACT);
    END;
{ACT now contains action requested by PDEV}
END;
```

VII.11 DIRECT REGISTER CONTROL :

The TMS9914 registers can be directly read or written using:

```
PROCEDURE IB_DIR(RDW,REG:INTEGER;VAR VAL:INTEGER);

RDW = 0 to read, 1 to write
REG = a TMS9914 register
VAL = the value read or value written to the register.
```

As there may be user applications that occasionally need more specific control of the bus, this procedure provides a means of accessing the bus directly without changing the assembly code routine. Refer to the

TMS 9914 GPIB ADAPTER DATA MANUAL MP033
Texas Instruments Incorporated

EXAMPLE of using direct access:

```
PROCEDURE SENDIFC;
{use direct access to send an interface clear }
CONST AUX=3;        {Auxillary cmd register}
BEGIN
  SIC:=143;          {Send Interface Clear =8FH}
  IB_DIR(WT,AUX,SIC);
  FOR I:=1 TO 10 DO BEGIN END;  {>10US WAIT}
  SIC:=15;           {Remove interface clear=0FH}
  IB_DIR(WT,AUX,SIC);
END;
```

VII.12 PROGRAM EXAMPLE :

This is an example of a program that initializes the bus, enabling the display of error messages. One other device exists on the bus, an HP1615 analyzer which can talk, listen and send service requests. The HP1615 is told to trace and get ready to send the result of the trace. The controller waits for a service request from the analyzer saying that it has done that. Then the display of error messages is shut off and the SAGE II/IV asks for 512 bytes of data at a time until all data has been sent.

```
PROGRAM IB_EX;
{$U IB.UNIT.CODE}
USES IB_UNIT;
CONST MAX=512;
VAR HP1615:INTEGER;
    ASCII:CHAR;

PROCEDURE DOINIT;
{Assumes SAGE II/IV is controller}
CONST TLK=1;LST=2;TAK=4;CTR=8;SRQ=16;PP=32;IAM=64;
VAR CONTROL,SW:INTEGER;
```

```
BEGIN
{YES, DO chk and display error messages}
IB_CHK:=TRUE;
{wrt them at 0,22}
IB_X:=0;
IB_Y:=22;
{Read GROUP-B switch}
SW:=IB_SWITCH;
{Set SAGE ADRESS}
IB_SAGE:=ORD(ODD(SW) AND ODD(31);
{INIT:1=CONTROLLER,SAGE ADDR, 46 USEC CMD}
CONTROL:=1;
IB_INIT(CONTROL,SW,23);
IF IB_ERR<>0 THEN EXIT(IB_EX);
{Define devices}
IB_STAT(IB_SAGE,TLK+LST+TAK+CTR+IAM);
  {talk,listen,control,SELF: =79}
HP1615:=15;
IB_STAT(HP1615,TLK+LST+SRQ);
  {HP1615=DEV 15 can talk,listen, send SRQ
  Serial poll by default : = 19}
END;
```

```
PROCEDURE GETDATA;
VAR BDATA:PACKED RECORD CASE INTEGER OF
      1:(B:PACKED ARRAY[1..MAX] OF 0..255);
      2:(W:INTEGER);
      END;
    MORE:BOOLEAN;
    TOTAL,DLNG:INTEGER;

BEGIN
  MORE:=FALSE;
  DLNG:=MAX;              {MAX number of bytes for each call}
  TOTAL:=0;
  REPEAT
    IB_HEAR(HP1615,BDATA.W,DLNG,MORE);
    { SAVE DATA HERE}
    MORE:=TRUE;
    TOTAL:=TOTAL+DLNG;
  UNTIL IB_ERR<>2;
  IF IB_ERR<>0 THEN WRITELN('ERR:=',IB_ERR);
  WRITELN('TOTAL=',TOTAL);
END;

PROCEDURE ACTION;
{Wait for SRQ, then Do a serial poll}
CONST NOSRQ=9;
VAR PDEV,SCOUNT:INTEGER;
    ACT:STRING;
BEGIN
  SCOUNT:=0;
  REPEAT
    SCOUNT:=SCOUNT+1;
  UNTIL (IB_CHKSRQ) OR (SCOUNT>100);
  IB_SPOLL(PDEV);
  IF PDEV<>HP1615 THEN WRITELN('other SRQ') ELSE
     WRITELN('FOUND SRQ');
END;
BEGIN
  DOINIT;
  IB_TALKS(HP1615,'RU;');
  IF IB_ERR<>0 THEN EXIT(IB_EX);
  IB_TALKS(HP1615,'DS');
  IF IB_ERR<>0 THEN EXIT(IB_EX);
  ACTION;
  IB_CHK:=FALSE;
  {No errors displayed while getting data}
  GETDATA;
END.
```

VII.13 BUILDING A USER PROGRAM :

IEEE-488 FILES:

These files should be provided on your distribution diskettes.

IB.BUS.TEXT	68000 assembly text file.
IB.BUS.CODE	68000 assembly code file.
IB.DEF.TEXT	Definitions for IB.BUS
IB.UNIT.TEXT	Pascal unit text file.
IB.UNIT.CODE	Pascal unit compiled file.
IB.LNK.CODE	IB.UNIT.CODE linked with IB.BUS.CODE
IB.EX.TEXT	Text example of main program
IB.EX.CODE	Compiled example program

Example User Program

The preceding example program can be compiled and run as follows to show how to set up and run a user program.

COMPILE IB.UNIT.TEXT to IB.UNIT.CODE.

ASSEMBLE IB.BUS.TEXT to IB.BUS.CODE.

LINK host IB.UNIT.CODE with library IB.BUS.CODE to create IB.LNK.CODE.

EDIT your file USERLIB.TEXT. Add a line: IB.LNK

The IB.LNK unit can also be put in a user library. Refer to the chapter on Sage Tool Kit and the p-System manual.

EDIT to create user program IB.EX.TEXT. To use the IEEE-488 unit, it must specify:

```
PROGRAM IB.EX; $$U IB.UNIT.CODE† USES IB_UNIT;
```

Refer to the PASCAL USERS' MANUAL for additional information on how unit calls work.

COMPILE user program IB.EX.TEXT to IB.EX.CODE.

EXECUTE user program IB.EX Execution option L= may also be used. Refer to the p-System OPERATING MANUAL. The code files can be linked together using the LIBRARIAN. Refer to the section on UNITS.

VII.14 IB_BUS DESCRIPTION :

The Pascal Unit IB_UNIT should provide most of the functions needed for an application on the IEEE-488 bus. Should the user need other functions or need to optimize his application, this description defines how the assembly code function IB_BUS called by the Unit works.

The assembly code function IB.BUS is called with several arguments which vary depending on the value of **CD**. It returns a non-zero result if an error occurred.

```
FUNCTION IB_BUS(VAR B,BLNG:INTEGER;CARG:INTEGER;CD:BUSCMD):
        INTEGER;EXTERNAL;
```

ROUTINE	CD	PROCESS
ITALK	1=	SEND data as a talker.
IHEAR	2=	RECEIVE data as a listener.
SESSION	3=	Set up who is to talk and listen.
SETSTAT	4=	Setup status of the devices.
DIRECT	5=	Read/write to TMS9914 register directly.
TRANS	6=	GET/Transfer control. Not yet implemented.
CHKSRQ	7=	Check for SRQ.
SPOLL	8=	Serial poll devices
PPOLL	9=	Parallel poll, not yet implemented
INIT	10=	Initialize TMS9914 if controller
ERREXIT		Error return

Note that some of these functions do not use all of the arguments. When this is the case, the unused argument will be shown as "DUMMY" in the call. The "DUMMY" argument must still be of the type required by IB_BUS.

CD=1..ITALK BLNG bytes of data starting at location B are sent to the listening devices. Call:

```
IB_BUS(B,BLNG,0,ITALK);
```

Any terminators must be put at the end of the data array. ITALK sets the end-of-text flag but does NOT add a special termination character.

CD=2..IHEAR BLNG bytes of data are received starting at location B. The actual number of bytes is returned in BLNG.

```
IB_BUS(B,BLNG,0,IHEAR);
```

IHEAR terminates when:

return=0 NOERR : an EOT flag occurs with the last byte.
2 FULL : BLNG chars (bytes) have been received.
3 RTMOUT: 5 secs have gone by without receiving another character.

CD=3..SESSION Before calling IB_BUS with ITALK or IHEAR, the talker and listener devices must be defined with SESSION. Call:

```
IB_BUS(LDEVs,BLNG,TDEV,SESSION);
```

LDEV byte array of LISTEN device(s). IB_UNIT currently only sends 1 listen device, although IB_BUS supports multiple listeners.
BLNG number of listeners.
TDEV bus address of the TALK device.

CD=4..SETSTAT Each device on the bus must be defined with a status which tells the IB_BUS routine what it is allowed to do: talk, listen, or control the bus. The status values are kept in a 32 byte array which is private to IB_BUS. The status table provides a means of returning a meaningful error to the user when a non-existent or incorrect device is addressed. It also speeds up the serial poll routine as only devices that are defined as capable of sending an SRQ are checked.

IB_UNIT procedure IB_STAT updates one device status in its internal table and sends the 32 byte table to IB_BUS. The pointer to the table is sent in B. Note that no check of the validity of the table entries is made. For example, an entry that specified a device that could send an SRQ, but not talk, would be accepted even though that is invalid.

The status byte is defined:
bit 0 = can talk
bit 1 = can listen
bit 2 = can control
bit 3 = has control
bit 4 = can SRQ
bit 5 = expects parallel poll
bit 6 = self
bit 7 = not used..set to zero

CD=5..DIRECT This routine allows the user to read or write directly to the TMS9914 chip.

```
IB_BUS(B,DUMMY,RDW,DIRECT);
```

B must be set up with the first byte equal to the register# and the next byte= the value to read or write to it.

RDW is equal to 0 to read a register, 1 to write the value to it.

CD=6..TRANS The Get or Transfer control from or to another device is not yet implemented. This command is reserved for use by SAGE. The call will be:

```
IB_BUS(B,DUMMY,XFER,TRANS);
```

where XFER=0 to get control or XFER=1 to transfer control to the device specified by B.

CD=7..CHKSRQ When a device wants to talk to the controller, it sends a Service Request (SRQ). The controller must periodically check for an SRQ by calling:

```
IB_BUS(DUMMY,DUMMY,0,CHKSRQ);
```

It returns HAVESRQ (=10) if an SRQ was found, otherwise a zero. The user must call the serial poll or parallel poll routine to find out who sent the request.

CD=8..SPOLL The CHKSRQ call is used before this call: A check is made to see if any device has requested service from the controller. The SRQ line should be active in this case. If so, the HAVESRQ (=10) is returned. Then the user must initiate a serial poll. Call:

```
IB_BUS(B,DUMMY,0,SPOLL);
```

The status byte (put by SETSTAT into DEVSTAT) for each bus device is checked. If that device can send an SRQ, it is enabled to talk and send it's serial poll status. In this way the device that requested service is found. The device address is returned in **B**.

Note that polling stops at the FIRST SRQ found. Multiple SRQ 's must be handled with multiple calls to CHKSRQ.

With this implementation it is possible for a device with a HIGH bus address to LOCK OUT the SRQ of a device with a lower address. This will only be a problem if the higher device repeatedly generates an SRQ after it's previous SRQ has been processed. This

means that each time the SPOLL is called, the routine will stop at the high device and never see the request of the lower one. To prevent this, assign devices that generate repeated SRQ sequences the lowest bus addresses.

CD=9..PPOLL The Parallel poll routine is not yet implemented. If called, no error will be returned. This call is reserved by SAGE for implementation of the parallel polling process. The call will be:

```
IB_BUS(B,DUMMY,0,PPOLL);
```

The device that generated the SRQ will be returned in B.

CD=10..INIT Initialize TMS9914. If SAGE II/IV is the controller, then the bus is initialized also. Call:

```
IB_BUS(CNTRL,ADDR,CMDWAIT,INIT);
```

CNTRL = 0 if another device is the controller.
1 if the SAGE II/IV is the controller.

ADDR = s the address of the SAGE. Usually the value read from the switches.It can be set without reading the switch. All eight bits (as defined by the address register specification must be set.

CMDWAIT= x count in increments of 2us to wait for devices on the bus to process commands.

CD>10.ERREXIT error return, bad argument (CD)

VII.15 ERROR CODES :

The following error conditions are returned via IB_ERR.

ZERO	=	0	no error, all done.
BADARG	=	1	bad call to GPIB
FULL	=	2	LNG bytes received but no EOI came in.
RTMOUT	=	3	timeout occurred while listening for byte.
XTMOUT	=	4	timeout occurred while sending byte.
NOEOI	=	5	timeout while waiting for talker to finish.
NOTALK	=	6	Talk device is not capable of talking.
NOHEAR	=	7	Listen device is not capable of listening.
NOBODY	=	8	nobody answered the serial poll.
NOSRQ	=	9	Checked for SRQ, was none
HAVESRQ	=	10	Have found an SRQ.

VII.16 IEEE-488 CONNECTOR :

SAGE II/IV implements the IEEE-488 (or GPIB) in hardware with the TMS9914A controller and TI 75160, 75162 buffers.

The connector is labeled IEEE-488. Its pin-outs are:

J5 - 1	DI01	bit 1 of GPIB data byte
2	DI02	2
3	DI03	3
4	DI04	4
13	DI05	5
14	DI06	6
15	DI07	7
16	DI08	8
17	REN	remote enable
9	IFC	interface clear
8	NDAC	not data accepted
7	NRFB	not ready for data byte
6	DAV	data valid
5	EOI	end of identify
11	ATN	attention
10	SRQ	service request

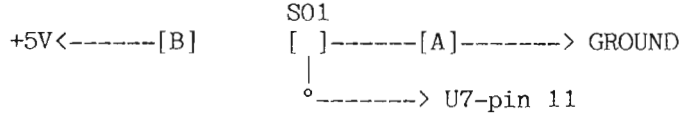
GROUP-B is an 8-bit dip switch which can be read by the computer. Although this port can be used for any application it is normally used to set up the GPIB address for the TMS9914A chip.

The address register set up for the TMS9914 is:

SW1-SW5	A1-A5	Device Listen and Talk address.
SW6	Dat	used to disable TALKER function. set to 0 , enabling TALK.
SW7	Dal	used to disable Listen function. set to 0 ,enabling LISTEN.
SW8	edpa	used to assign the device two consecutive addresses. Normally set to 0.

THE IEEE-488 SUPPORT
IEEE-488 CONNECTOR

The port is normally strapped for open collector I/O. To change to tri-state driver mode, cut the strap trace of S01 on the Sage II/IV board from U7-pin 11 to [A] ground. Reconnect to [B] +5V.



The strap is located at the end of the board at the IEEE-488 connector.

For more information, send for IEEE-488 BUS STANDARDS with ($9.00) to:

IEEE STANDARDS
345 E. 47TH ST
NEW YORK, N.Y. 10017

For more information about the TMS 9914 GPIB chip:

TMS9914 GPIB ADAPTER PERLIMINARY DATA MANUAL MP033
TEXAS INSTRUMENTS, INC
P.O. BOX 5012
DALLAS, TEXAS 75222

Some operating systems on the SAGE provide their own IEEE-488 support. Refer to the appropriate manual.

VIII COMPUTER INTERCOMMUNICATION :

It is expected that a number of SAGE II/IV users will want to port software from a computer system that does not support a compatible storage media. The most universal method of solving this problem is by setting up a communication link over an RS232 channel. Unfortunately setting up this type of communication is usually easier said than done. The number of details which must be coordinated usually require an experienced user. These notes and the accompanying software should help in the task of successfully interconnecting another system to a SAGE II/IV computer.

- REMOUTTEST and REMINTEST are routines which are intended to check out an RS232 remote channel between two p-System computers.

- SEND and RECEIVE are simple programs to communicate the image of a device between p-System computers.

- TEXTIN is used to receive ASCII text and store it in a .TEXT file. TEXTIN is typically used to transfer source programs from a non p-System environment.

- REMTALK is provided by Softech Microsystems (on an as-is basis) and provides individual file transfer capabilities between two p-System environments.

- TELETALKER allows access to electronic mail services over a phone line using a modem.

VIII.01 BACK-TO-BACK COMMUNICATION :

Typically, a user will want to connect 2 computers back-to-back and transfer a file from one to the other. Once a hardware connection is made between the SAGE MODEM port and a serial port of the other computer, a test of the RS232 connection is made. For the p-System, REMOUTTEST and REMINTEST are used.

Then if the foreign (non-SAGE) system supports the p-System, REMTALK may already be available and can be used to transfer files. Also, the routines SEND and RECEIVE are short and may be easily typed into a foreign p-System computer. If information is only to be transferred to the SAGE system, then only SEND (a one page program) is required on the foreign system. If files must be transferred to the foreign system, then RECEIVE may be typed in and used to transfer the more general (and larger) REMTALK program.

Source files from a non p-System environment are usually received by the TEXTIN program.

Text files may be output to a non p-System environment by using the Filer to transfer them to device REMOUT:

VIII.02 HARDWARE INTERCONNECTION :

Four connections are important when interconnecting the Modem port (DB-25 connector) to another computer system. The signal Ground should be connected to pin 7. The Transmit Data output is from pin 2. The Receive Data should go to pin 3. The Clear to Send on pin 5 must be held in the +12V state. Clear to Send should generally not be used for low level handshaking (see writeup under Modem Port). A +12V source is available on pin 9 of the DB-25 connector.

Data Set Ready on pin 6 may be used for low level handshake control of the transmit direction. No signal is currently available from the receive side to control the far end transmission.

VIII.03 CHANNEL CONFIGURATION :

Many variations of low level data format are available on the remote channel. The utility program SAGE4UTIL is used to alter the configuration of the remote channel. The following signal format items must be identical between the two computer systems.

Baud Rate	
Parity	(no parity, even parity, odd parity)
Number of Stop bits	(1, 1.5, or 2)
Number of Data bits	(7 or 8)

The SAGE computer system will ignore characters which generate detectable low level (parity, framing, and overrun) errors by the USART. Discrepancies in the configuration may or may not generate USART detected errors. Thus, some characters may be received (possibly with incorrect values), while others will be totally ignored.

The XON/XOFF and Data Set Ready options must be disabled unless these handshaking options are desired.

VIII.04 COMMUNICATION HANDSHAKING :

Handshaking is a method of two computers exchanging information to insure that they are both working on the same thing. Handshaking may be used to check that a computer is ready, or to provide a delay (typically to write information to a disk).

In order to keep the routines simple enough for the user to easily type them into a system, the programs SEND and RECEIVE do not perform any handshaking. Thus, if RECEIVE detects an error, it just outputs a message and stops rather than asking SEND to retransmit the information. This also simplifies the data link, in that only one direction of transmission is required. The routines do, however, coordinate a pause every 32 blocks (16K bytes) in order to let RECEIVE write the information to disk without losing data. This is especially important if RECEIVE is running on a non-interrupt driven system where all characters which arrived during a disk transfer would otherwise be lost.

On systems slower than the SAGE II/IV, a reduced baud rate may be necessary to allow the receiving computer enough time to process each character. The SAGE II/IV systems will communicate back to back at the maximum baud rate of 19.2K. The constant DELAY at the beginning of the SEND program may also be increased (experimentally) to force a delay between characters. This technique may be used to slow down transmission instead of reducing the baud rate.

Interrupt driven systems may also use a low level BIOS protocol such as XON/XOFF or an RS232 signal line to prevent the transmitter from overflowing the receiver. The SAGE II/IV remote channel supports an XON/XOFF protocol in both transmit and receive directions (see remote channel configuration section). The SAGE II/IV transmitter also can be set up to monitor the Data Set Ready signal as a control line from the receiver in order to slow down transmission. The Clear to Send signal should not be used for this type of handshaking. The BIOS Remote channel receive does not currently generate any RS232 signal line feedback. Note

that these types of handshaking require that both ends be configured to handle the appropriate protocol.

The program TEXTIN also does not do any handshaking and needs only one direction of transmission. The program records data in the file approximately every 512 characters. For channel baud rates of 4800 baud or lower, there is no timing problem recording on floppy diskette. If a 9600 baud rate or greater is necessary, then the file should be recorded in RAM Disk (much faster operation) or the XON/XOFF handshaking protocol must be used.

Examination of the REMTALK program indicates that it does a more complete job in high level handshaking. All characters are received in groups at the low level and processing is performed between transmission of blocks. High level commands are exchanged between transmission of blocks to coordinate the communication.

VIII.05 REMINTEST AND REMOUTTEST :

Two versions of REMINTEST are provided, one for Version IV (or greater) p-Systems (with the UNITSTATUS procedure) and one for pre Version IV systems. Both routines echo characters received from the remote input channel to the terminal. The Version IV variation allows a "Q" to be typed to exit the routine, while the second variation might need to be reset.

Note: for most non-Sage systems, in order to use the p-System Break key, a character must arrive on the remote channel before the BIOS will return and let the system recognize the break.

REMINTEST will echo all characters received on the remote channel to the terminal. The terminal data rate must be the same or higher than the remote channel data rate in order to keep from losing characters. Note that even if the baud rates are the same, if less bits are received on the remote channel (no parity, less stop bits, less data bits, etc.) then the terminal will eventually get behind. Also, the clock rates may be slightly different (even with the same baud rate and number of bits), which could cause missed characters. The constant, DELAY, at the beginning of REMOUTTEST may be increased to slow down transmission.

REMOUTTEST asks 'Output how many lines?'. Each line contains upper and lower case letters and the ten numeric digits followed by a carriage return.

Note: REMINTEST may be used to monitor the output of SEND or any other source that does not require high level handshaking.

VIII.06 SEND AND RECEIVE :

SEND is a routine to download the image of a device to another system. The routine will inquire about the name of the device and the number of blocks to transfer. The user should K(runch the device with the Filer and look at the directory to determine how many blocks need to be sent.

The binary image of the device is converted to ASCII Hexadecimal digits and transferred over the remote channel. Other start and termination characters are sent to synchronize the communication. A short pause is made every 32 blocks for the receiving end to record the data on a device.

The SEND routine prints a dot on the terminal before each block of information is sent. The routine also prints the block number every ten blocks.

RECEIVE is a routine to receive the data from the remote input channel that is output by the routine SEND. RECEIVE will ask for the device name (typically #5:) on which to record the image. Make sure that the device is expendable because the directory will be overwritten. Start the RECEIVE routine before starting the SEND routine.

The RECEIVE routine checks for the character 'S' to start each block of 512 bytes. Between blocks all characters except 'Z' are ignored until an 'S' is detected. A 'Z' causes termination of the RECEIVE program. Once an 'S' is detected, all following characters must be hexadecimal digits or a carriage return or line feed until the block is complete. Each block also contains the block number and a checksum which is verified by RECEIVE to insure data integrity.

The RECEIVE routine prints a dot for each block successfully received, as well as the block number every ten blocks. If the RECEIVE routine cannot keep up with the data it may miss a character and therefore attempt to use the 'S' which starts the next block as a hex character for the current

block. This will cause the message 'Bad character in transmission'. If this problem is suspected, the baud rate on the remote channel for both ends may be lowered or the DELAY constant increased in the SEND routine. Back to back SAGE II/IV systems have been checked out at 19.2K baud with no problems. Problems may occur if RECEIVE is used in a slower system where the incoming data is faster than the receive processing.

VIII.07 TEXTIN :

The program TEXTIN handles text which is typically output by a non p-System computer. The operations of TEXTIN are split into two parts to maximize the possible receive data rate. The main prompt line of the program is:

```
C(onvert, R(eceive, Q(uit ?
```

The R(eceive option is used to receive characters on the remote serial channel input and store them in a file. The C(onvert option should be used on a raw character file to convert the information into the standard p-System .TEXT file format.

The R(eceive option will query:

```
Receive text into what file?
```

Once the file is set up the routine will prompt:

```
A(bort or T(erminate receiving?
```

For each block of characters written to disk, a period is printed on the screen. No End-of-File character is recognized. To indicate to the program that all data has been transferred, the 'T' character should be typed on the keyboard. This indicates that the input should be terminated and the file should be saved. The 'A' character should be typed on the keyboard to immediately abort the program without saving the data.

After termination the data should be C(onverted to the p-System ".TEXT" file format. The routine will query:

```
Source file for conversion?

Destination file for conversion?
```

The source file should be the file specified for the R(eceive process (above). The destination should be the final .TEXT file to be used by the Editor, etc. The conversion process strips out line feeds which are typically output for printers or terminals but are not needed in the .TEXT file format. Null (zero) characters are stripped out. Null characters are typically added in at two block boundaries to prevent lines from crossing a double block boundary. Lines which exceed 255 characters without a carriage return will have a carriage return inserted and generate a message 'Line > 255 characters'.

VIII.08 REMTALK :

No documentation on REMTALK (except the source program) is currently provided by Softech. To use the REMTALK program, one system is set up as a Master and the other as a Slave. All control of the file exchange is done from the Master end. When the program starts, it prompts:

```
M(aster  S(lave  Q(uit
```

The Slave end should be started first. Then when the Master end is started, the program allows selecting between sending a file to the Slave or receiving a file from the Slave.

```
S(end  R(eceive  Q(uit
```

The program then prompts for the appropriate source and destination file names.

VIII.09 TELETALKER :

TELETALKER is a remote communications package developed by Randy Bush for USUS. It is intended for use in talking to an electronic mail service such as TELEMAIL, COMPUSERVE, the SOURCE, etc.

TELETALKER has various names depending on which modem it is used with: TELE.HAYES.CODE, TELE.VADIC.CODE, etc.

A specific example of using TELETALKER with a Hayes modem to connect to TELEMAIL is given in the GETTING STARTED MANUAL.

TELETALKER expects your MODEM to be connected to the SAGE MODEM port. You do not have to change the configuration of the port as TELETALKER will do that for you.

> **IMPORTANT:** Do not, however, escape from TELETALKER by using the software reset key (CTRL/@ on the QUME or TV925, and CTRL/SHIFT/@ on the Freedom 100). If you do this, your port will be left set up for the MODEM, and you will not be able to run a printer or another terminal on that port until you reset or re-configure using SAGE4UTIL.

There are different versions of TELETALKER for each type of MODEM. We have discussed TELETALKER using the HAYES SMARTMODEM 1200 and the RACAL VADIC and given examples of logging on to TELEMAIL and sending messages with these MODEMS. However, the TELETALKER commands are the same for all versions.

● Teletalker Commands

TELETALKER is called by typing an "X" from the command line and then the name of the file for the MODEM being used:

X	TELE.HAYES	for the HAYES MODEM
X	TELE.VADIC	for the RACAL VADIC MODEM

The screen will display the following message:

```
R a n d y ' s   T e l e T a l k e r

Copyright 1979-83, RB&A. All rights reserved.
BaudRate 1(200, 3(00 ?
```

At this point you can type:

```
ESC ESC                 to exit program
1 or CR or Space        selects 1200 baud
3                       selects  300 baud
```

Once the proper baud rate has been selected, the screen shows:

```
Ctrl-A> for option menu
```

You should get a dial tone at this point. If not, the last person to access Telemail did not hang up the phone. Teletalker does not automatically hang up the phone for you when it is done. The user must give instructions to the Modem to do this (ATH 0 to the Hayes) or hang up manually (Racal Vadic). This feature can be very useful. You can use TELETALKER to connect to another computer. Run a program to collect/receive data, and then sign off by running TELETAKER again.

Option Menu:

When a CTRL/A is typed from the on-line mode, TELETALKER will intercept the character and change to an options mode displaying this menu:

```
Options: S(end, R(ecord, G(o, B(reak, T(haw, 7(, 8(, E(xit -
```

In the options mode, nothing is being transmitted or received over the phone line. You may select the option you desire and then return to the on-line mode with either a <cr> or a "G". Illegal characters will sound the terminal bell. Each of these options will now be discussed.

Go Option

To enter the options mode a CTRL/A is used. To LEAVE the options mode and go back on-line, type "G" or a <cr>. When on-line, all characters typed at the keyboard will be sent out the Remote serial channel. Any characters coming in from the Remote serial channel will be output to the terminal.

Send Option

The SEND option informs TELETALKER of a p-System text file that you want to send over the phone. This file MUST be a text file. If you wish to send a code or data file, you must first convert it to a text file.

You should not enter the command mode (with a CTRL/A to SEND) until you are sure that the computer on the other end of the phone line is ready to receive the file. TELETALKER will immediately begin sending the file text as soon as you leave options mode (with a "G").

When you type an "S" from the options line, TELETALKER will display:

```
Send what textfile ?
```

Type in the name of the textfile you want to send. TELETALKER will display:

```
Filename Opened
```

If the file cannot be found, an error will be given and you will be asked to re-enter the textfile name. You do not have to add ".TEXT" to the filename. Typing just a <cr> will exit the SEND option with no file defined.

If a file of that name already exists, a message will be given:

```
Currently Sending Filename  Close it?
```

If you reply "Y", then the file is closed. Any other reply will continue using that file to send. You cannot delete the file at this point.

Once you type a <cr> or a "G" to leave the options mode, Teletalker immediately begins to send the file. Therefore it is important that you have previously given the correct commands to the other computer for it to receive the file.

Any character typed on the terminal while sending will stop transmission and enter the options menu.

When transmission has been completed, the program will output:

```
Filename Finished
```

Record Option

The RECORD option enables you to specify a file for recording information received from the other computer and ALSO keep a log of the exact keys and commands typed. This means that you may have a few extra command lines in the file, but you can always edit them out. If the file was not received properly, you can look at the commands typed and perhaps see what was done incorrectly.

Most users set the RECORD option immediately on entering TELETALKER. This gives them a record of their entire session.

Use CTRL/A to enter the options mode. Type "R" to RECORD:

```
Record as what textfile ?
```

Typing just a <cr> will omit setting up a RECORD file. TELETALKER will add the ".TEXT" suffix to the file name unless you specify a "#" as the first letter of the name. If the file is found, this message will be displayed:

```
Filename Opened
```

If there is already a file open for recording, then you will be asked:

```
Currently Recording Filename  C(lose, P(urge ?
```

If you wish to save the information already received, you should CLOSE the file by typing "C". TELETALKER replies with:

```
Closed
```

To delete the existing record, type "P". TELETALKER replies with:

```
Purged
```

Typing just a <cr> will leave the file open and you will continue recording into it. TELETALKER replies with:

```
Recording continued
```

If the file already exists, then TELETALKER asks:

```
Filename Exists, P(urge ?
```

Typing a "P" (or Y) causes the existing file to be deleted (all previous recording is lost) and this display:

```
Purged
```

Typing anything else saves the existing file:

```
Saved
```

The recording file will be automatically closed when you exit from TELETALKER.

Exit Option

The E(xit option is used to leave the TELETALKER program and return to the p-SYSTEM command line. This is done after the phone line has been disconnected, either by command to the computer (SMARTMODEM 1200) or manually (RACAL VADIC).

To exit, you must first be at the TELETALKER prompt line. To get there, type CTRL/A. When you see the prompt line, type "E" to exit TELETALKER and return to the p-SYSTEM command line.

> If, however, you exit with a p-System Break key you will lose all of the information. You will also leave your REMOTE port configured for the MODEM and will have to reconfigure it in order to use the printer or another terminal on that port.

Break Option

Some computers or Electronic Mail systems use a "BREAK" function for certain actions. For example, you can halt a listing sent by TELEMAIL by typing CTRL-A and then "B" for break, and "G" to return to on-line. The definition of what "BREAK" does is different for each computer.

The "BREAK" signal is 200 millisecond long.

> **IMPORTANT!!** This is NOT the "BREAK" key on your terminal. Typing that key may abort your program (leaving your phone line up) and drop you into the SAGE SDT mode. You should immediately reboot and run TELETALKER to get back to your operation. Although you left your phone line up, the reboot process will probably disconnect the line. In any case, you have lost everything you recorded.

Thawed Option

When communications between the computers become "stuck", this option can be used to try to clear the problem.

After transmitting each line to the remote channel, the program waits for the echo back to be completed. This is normally a line feed. If for some reason the line terminating character does not come back, the Thaw option may be used to unfreeze the remote channel processing.

7 Option

This option clears the remote **input** channel (device number 7). This is usually tried only if something unusual happened during the communications session. Typing "7" will cause the display:

```
Clears remote input channel.
```

8 Option

This option clears the **remote** output channel (device number 8). This is usually used only if something unusual happened during the communications session. Typing "8" will cause the display:

```
Clears remote output channel.
```

Lost Carrier

If your modem has a speaker, you can tell if carrier (the phone line) is present by the high tone that comes on the line once you dial out and reach the other computer. Carrier must be present for communication. Loss of carrier indicates that something happened to cut communications; a bad phone line, the other computer went off-line, etc. The modem tells the SAGE computer that it recognizes carrier by using the signal Clear to Send and by the dial tone heard over the speaker of either the phone or the modem.

If the Clear to Send signal from the modem is lost, the terminal will display:

```
LOST CARRIER
```

If the Clear to Send signal becomes active, the message will be erased. If CTRL/A is typed, the TELETALKER option menu appears on the screen.

Error on Disk Write

Various disk errors may occur:

```
Write error on disk.
 or
ReRead error on disk.
```

Normally, you should not see any of these errors. If one should occur, check for bad floppy media, too many files (77 max), not enough room on disk to write all the data being recorded, and the write-lock tab left on the diskette. Do this BEFORE checking for an actual hardware problem.

● TELETALKER on Other MODEMS

If you have a MODEM, even though it is not a Hayes Smartmodem 1200 or a Racal Vadic, you probably want to use TELETALKER with it. Before examining the technical discussion which follows, try using your modem with the TELE.VADIC version of TELETALKER. This version is very straight-forward and should work with many simple MODEMS.

If you have a MODEM with additional features or special requirements, you must write a special unit (part of a program) for your MODEM. If you are not familiar with either Pascal or the details of MODEM operation, this exercise is not for you. If you do write a unit for a new MODEM, SAGE Computer would appreciate receiving a copy so we can help others.

The following section is very technical and we advise the first-time computer user to skip over it.

● Remote Unit

The files REM.HAYES.CODE and REM.VADIC.CODE contain implementations of the USUS Remote Unit Standard as documented in USUS Newsletter #5 (Fall 1981). REM.HAYES.CODE is for the Hayes Smartmodem 1200, while REM.VADIC.CODE is for a more generic Racal Vadic Modem. The Unit definition was developed primarily by Robert W. Peterson for use in supporting transportable communication software in the p-System environment. The supplied implementation which supports Racal Vadic and Hayes Modems on the SAGE Modem port was contributed to SAGE by Randy Bush. All rights to re-distribute the Remote Unit software on other than SAGE Computers are reserved by Randy Bush.

Following is the Interface section of the Remote Unit.

```
UNIT REMUNIT;
INTERFACE

TYPE
        CrDialResult            = (CrOffHook, CrDialError, CrNoAutoDial );
        CrBaudResult            = (CrBadParameter, CrBadRate, CrSetOk,
                                   CrSelectNotSupported);
        CrWhoAmI                = (CrOrig, CrAns);
        CrState                 = (CrOn, CrOff, CrAuto);
        CrRemPort               = RECORD
            Part1               : INTEGER;
            Part2               : INTEGER
            END;

VAR
        CrAttenChar             : CHAR;
        CrCurrentPort           : CrRemPort;

PROCEDURE CrCommInit (Dir           : CrWhoAmI;
                      Atten         : CHAR;
                  VAR RemExists     : BOOLEAN;
                  VAR DialerExists : BOOLEAN);
PROCEDURE CrSetCommunications (Parity     : BOOLEAN;
                               Even       : BOOLEAN;
                               Rate       : INTEGER;
                               CharBits   : INTEGER;
                               StopBits   : INTEGER;
                               Dir        : CrWhoAmI;
                               Model      : STRING;
                           VAR Rslt       : CrBaudResult );
PROCEDURE CrCommQuit;
PROCEDURE CrPutRem (c : CHAR);
PROCEDURE CrAnswer;
PROCEDURE CrBreak;
PROCEDURE CrDial (Number : STRING; WaitChar : CHAR; VAR Rslt : CrDialResult);
PROCEDURE CrHook (OnHook : BOOLEAN);
PROCEDURE CrSetAddress (Port : CrRemPort);
PROCEDURE CrDelay (Tenths : INTEGER);
PROCEDURE CrSetDTR (Setting : CrState);
PROCEDURE CrSetRTS (Setting : CrState);
FUNCTION  CrRemStat : BOOLEAN;
FUNCTION  CrGetRem : CHAR;
FUNCTION  CrCarrier : BOOLEAN;
FUNCTION  CrClearToSend : BOOLEAN;
FUNCTION  CrRinging : BOOLEAN;
FUNCTION  CrDialTone : BOOLEAN;
FUNCTION  CrKbStat : BOOLEAN;
FUNCTION  CrGetKb : CHAR;

IMPLEMENTATION
```

Following are descriptions of the routines specified above.

CrCommInit This is the first routine called when a communications package wishes to establish communications.

The scalar passed, Dir, indicates the (originate or answer) state of this computer relative to the other. Such knowledge of direction is needed to set a modem to answer or originate mode.

The character passed, Atten, is placed into the global CrAttenChar variable for use by the communications routines. The most frequent use of this character is to allow the user to exit wait loops without aborting the entire program.

RemExists returns TRUE if the current computer configuration supports communications and if initialization was successful; otherwise FALSE is returned.

DialerExists returns TRUE if the communications hardware supports auto-dial, FALSE if auto-dial is not available.

CrCommQuit This routine is called when communications are to cease. In order to resume communications, CrCommInit must be called.

CrKbStat One of the functions needed by most communications packages is the ability to check the keyboard (and remote) for available input, without causing the program to wait for a character. CrKbStat is intended to supply this capability for the keyboard device. That is, if a character is available from the keyboard buffer, CrKbStat returns TRUE. If no character is available, CrKbStat immediately returns FALSE to the caller. CrKbStat never waits for a character.

CrRemStat This routine does exactly the same thing CrKbStat does but returns the status of the SAGE Modem port instead of the keyboard.

CrGetKb Instead of READ or UNITREAD an application should use this routine to receive a character from the keyboard device. It does not echo characters and will not respond to EOF and AlphaLock characters. Note that if a character does not exist in the queue, the routine will hang and wait for a character.

CrGetRem This function is similar to CrGetKb. It returns a character from the SAGE Modem port without handling EOF characters. Note that if a character does not exist in the input queue, the routine will hang and wait for a character.

CrPutRem Instead of WRITE or UNITWRITE, an application should use this routine to transmit a character to the SAGE Modem port. DLE expansion and auto Line Feed after Carriage Return are suppressed.

CrAnswer This procedure takes the modem "off-hook" in answer mode.

CrBreak This procedure causes a Break signal to be transmitted for 200 milliseconds on the SAGE Modem port.

CrCarrier The intent of this function is to indicate when the communication line is ready to handle data. The CrCarrier function is frequently applied in two situations. The first is at startup time when the program may wait for CrCarrier to return TRUE before attempting communications. The second situation is to check CrCarrier during communications to ensure that the connection is still valid and to allow cleanup and recovery to be initiated if the connection has been broken.

CrClearToSend If this function returns TRUE, then RS-232 line Clear to Send is TRUE. CrClearToSend can be used to pace output to the communications line so that the receiver is not sent characters faster than can be processed.

CrDelay

This procedure is used to waste time. The argument indicates how many tenths of a second are to be wasted before CrDelay returns to the caller.

CrDelay is sometimes used to go idle while waiting for a response from the remote device. This is frequently necessary when two programs operating half-duplex attempt to get into synchronization with each other. CrDelay might also be used by a connect routine to wait a specific amount of time between a phone being answered and testing for the presence of a carrier.

CrDial

When the auto-dial support is available, this routine is used to do the dialing.

The number to be dialed is passed as a string of ASCII characters. Where CrDial must wait for a dial tone, WaitChar is inserted into the digit string by the calling routine. WaitChar should not be either the asterisk (*) or number symbol (#). The asterisk and number symbol are valid dialable characters if tone dialing is used, and are ignored if pulse dialing is used. A "P" or "p" embedded in Number indicates that pulse dialing is to be used and a "T" or "t" indicates Touch-Tone dialing. All other nonnumeric characters are ignored.

If auto-dial support is not available, this routine returns CrNoAutoDial in the CrDialResult field.

CrAnswered is returned only if the dialed number has been answered. The carrier may or may not have been detected when CrAnswered is returned. If a connection

cannot be made, CrDialError is returned. The conditions under which Cr_DialError is returned are dependent on the user's environment. An obvious error is the called telephone's failing to answer.

Cr_DialTone After the modem has taken the telephone off-hook and before a number can be dialed, a dial tone must be present. This function returns TRUE when the modem has detected a dial tone.

Cr_Hook This procedure is used to manipulate the status of the phone line. When the argument passed is TRUE, the phone is placed "on-hook", i.e., hung up. When the argument passed is FALSE, the phone is taken "off-hook".

CrRinging This Boolean function returns TRUE if the modem supports auto-answer and the telephone line is ringing. Not that the value returned may return FALSE between rings, as well as when the telephone is not ringing.

CrSetAddress This procedure is used only when more than one communication address is available. CrSetAddress changes the communication port in use and affects the operation of all other procedures. This procedure should not be used in the SAGE implementation.

CrSet Communications

This procedure is used to change the current parameters of the modem including parity, number of bits, and baud rate.

If no parity is to be used the Parity is passed as FALSE. If Parity is passed as TRUE, then the type of parity is set by the Even parameter. If Parity is TRUE and Even is TRUE, even parity will be generated. If the port will not support the parity configuration requested by these two parameters, Result will be returned with a value of CrBadParameter.

CharBits specifies the number of data bits and must be in the range 5..8. StopBits must be in the range 1..2 and indicates the number of stop bits to be used. If the port will not support the configuration requested by these two parameters, Result will be returned with a value of CrBadParameter.

Dir indicates the state of the computer relative to the other device, just as CrCommInit's Dir parameter does.

Model is a string containing user-specific model options. In this implementation of the Remote Unit, three options are available. The keywords specifying these options are "Mask", "FlowIn", and "FlowOut". Any or all of the option keywords may be specified in the Model string separated by commas. If "Mask" is specified, all characters received from the Modem port will be masked with 7FH to remove the eighth bit. If "FlowIn" is specified, the SAGE Computer will transmit XON and XOFF characters to control the senders transmission in order to

prevent overflow of the input buffer. If "FlowOut" is specified, the SAGE Computer will react to incoming XON and XOFF characters on the Modem port by starting and stopping transmission to the other computer.

Rate is the baud rate to be set and must be in the range 0..32767. If the Rate specifies an unsupported baud rate, Result should return Cr_BadRate.

Rate is the baud rate to be set and must be in the range 0..32767. If the Rate specifies an unsupported baud rate, Result should return Cr_BadRate.

CrSetCommunications will be called to initialize the communications port and may also be called after communications have been initiated to change some parameter. This means that CrSetCommunications does not cause an existing link to be broken.

CrSetDTR CrSetDTR allows the application program to manipulate the state of the RS-232 Data Terminal Ready (DTR) line. If CrOn is passed, DTR is set high. If CrOff is passed, DTR is set low. If CrAuto is passed, the line is manipulated by the unit as is appropriate. CrAuto is the default and is set by CrCommInit.

CrSetRTS CrSetRTS allows the application program to manipulate the state of the RS-232 Request to Send (RTS) line. If CrOn is passed, RTS is set high. If CrOff is passed, RTS is set low. If CrAuto is passed, the line is manipulated by the unit or hardware as is appropriate. CrAuto is the default and is set by the CrCommInit.

General Operations

The usual sequence of operation will begin with a call to CrCommInit. This will be followed by a call to CrSetCommunications to initialize the communications hardware. If this is an originate session, a connection will be made.

If auto-dial hardware is available, CrHook will be used to take the phone off-hook; the program will wait until CrDialTone is TRUE and the CrDial to dial the number. Without auto-dial, the application (not the unit) must ask for the user to make the connection, then call CrHook to tell the unit that the phone is off-hook. CrKbStat, CrRemStat, CrGetKb, CrGetRem, and CrPutRem will then be called to do the transmissions. CrCarrier will be monitored to ensure the link is not broken. When communications are completed, CrHook is called to hang up the phone followed by CrCommQuit to terminate.

Note that the SAGE Modem port configuration is saved when the Remote Unit is loaded, and this configuration is restored when the program using the Remote Unit terminates.

IX THE SAGE MULTI-USER SYSTEM :

The SAGE Multi-User Environment is an enhanced version of the Single User I/O system. The Multi-User Environment allows multiple programs to run at the same time. Up to six user terminals are supported on the SAGE IV, and two users are supported on the SAGE II.

The Multi-User system is capable of running multiple copies of traditional single user operating systems. Multiple copies of the p-System operating system, standardly shipped with every SAGE Computer, will run without modification in the SAGE Multi-User Environment. CP/M-68K, HyperFORTH Plus, and the Volition Systems' Modula 2 environment will also run concurrently with the p-System. The SAGE Multi-User Environment allows a user familiar with a specific single user software system to expand the usage to more terminals without a linear increase in hardware cost.

The system supports a Shared Terminal mode which allows a user at one terminal to switch between the screens associated with two or more separately running programs. For instance, a secretary can be set up to do data entry or word processing in one program, and when the phone rings, immediately be able to switch over to another program to enter information about the call.

Switching back and forth between the single-user and Multi-User environments can easily be done. On the SAGE II, this is simply a matter of re-booting the system to the appropriate diskette. On the SAGE IV, the System Manager has the choice of booting to a single-user Winchester partition, a Multi-User Winchester partition or a single user diskette.

The Multi-User Environment provides various facilities to allow users to share information. The system provides Global Semaphores which may be used by application programs to coordinate their accesses to shared data. A User Inter-Communication facility is provided to allow programs to

communicate with each other for sharing of data or control information.

Several standard Multi-User environments are pre-configured for the user. Newcomers to the SAGE should try one of these before trying to do a custom installation. The GETTING STARTED MANUAL describes how to install one of the standard configurations.

IX.01 MULTI-USER OVERVIEW :

The Multi-User BIOS introduces a number of new concepts which may not be familiar to users of a single-user system. The following sections cover a number of topics which will be valuable in understanding and getting the most out of the Multi-User system.

● The System Manager

A Multi-User system is somewhat more complex than a single-user system and therefore will require more management than a single-user system. One person in the using organization should be appointed as the System Manager. The duty of a System Manager is to become as knowledgable about the Multi-User system as possible. With this knowledge, the System Manager should set up how the system is to be used and provide guidelines to the other users about system operations.

There are many factors to take into consideration when implementing a Multi-User system. Because "Multi-User" does mean that more than one person will be using the system at the same time, the needs and habits of these individuals must be carefully meshed with the system in order to achieve a good working environment for all. The SAGE Multi-User system is very flexible, with many different factors to take into consideration. Managing the system can become time-consuming if continual changes are being made .

The most important consideration is to do a good allocation of the system resources to the various users. The current

version requires static association of User Tasks (memory) with the Terminals, disk partitions, RAM Disk, and other devices. This means that any change to the Multi-user system will require re-booting.

A step-by-step example of installing a Multi-User system is included in this manual on page 347. This example attempts to show HOW the System Manager decided on some of the options and may help you understand some of the trade-offs in building a system.

User Tasks

The SAGE Multi-User system is capable of running several programs at the same time. In actuality, the programs do not run at exactly the same time. The computer rotates between them doing a little bit of each user's program so that it appears to all the users that the computer is continuously running their program. This whole process is called Time Sharing and the little bit of time that each user's program gets in the rotation is called a Time Slice.

Each of the separately running programs in the SAGE Multi-User system is called a User Task. The current maximum number of User Tasks in the Multi-User system is 16. One of these Tasks is used internally by the system and is never available for outside (User) work. Several other factors typically limit the number of User Tasks to fewer than the 15 which are available. For instance, the fact that the SAGE IV computer has only 6 serial ports physically limits the number of actual on-line people to six.

User Tasks can be created which do not require a terminal. For instance a program could be created which would provide a special service for all the other Users. This program would talk to the other User programs via the built in Inter-User Communication channels. It would automatically start running (via the p-System SYSTEM.STARTUP facility) when the system booted and wait for service requests from the other Users.

The SAGE Multi-User system also supports a feature by which one person at a terminal can control two or more User Tasks. This is called the Shared Terminal feature in that several Tasks can share one terminal. The User can switch the terminal between the multiple Tasks by just typing a key. For the Tasks that are not currently talking directly to the terminal, the system simulates the terminal and keeps track of what should be on its screen. Then when the User switches the terminal to a new Task, the screen is updated with the information which should be on the new Task's screen.

With the Shared Terminal feature, one would think that there would be no reason to limit the number of User Tasks because even one person could be assigned all 15 available Tasks. Each User Task, however, requires some dedicated RAM memory in which to run, and possibly a storage device on which to store its files. These requirements are the major limiting factors on how many User Tasks are typically configured in a SAGE Multi-User system.

Time Slice and Priority

As previously explained, each User Task is allowed to run for a certain amount of time before the system advances to the next User Task. The amount of time that a User Task runs per rotation thru the Users is called the Time Slice. Each User's Time Slice is independently configurable so that one User may be favored by making his Time Slice larger than the others.

When a User Task is waiting on I/O activities (such as reading a character from the keyboard), the User's Time Slice is skipped until the system completes the operation (ie. gets a key). This means that idle User Tasks do not require any time and all the available time is split up between only those Tasks which are active. It is therefore important that programs be structured so that they are not compute bound (ie. polling the keyboard) when they should be inactive.

In special cases it may become important that one User Task preempt other User Tasks when it is active. This may be accomplished by adjusting the Priority of the User Task. The Priority levels range from 1 to 127 with a normal default in the middle at 64. Only User Tasks which have the highest Priority level among the active Tasks will be processed in the Time Slice rotation. Note that User Tasks configured with a Priority level of zero are disabled, that is they are never allocated any memory and never started.

Note that higher Priority Tasks which become runnable will not actually get processed until the current Task's Time Slice runs out. Therefore time critical Tasks still may not be started for the duration of the longest defined Time Slice of the other Users. Also, it is typical that a high priority task not have a long run time before it hangs again on I/O, or it will dramatically disturb the other Users on the system.

A second possible use of Priority is to set up a User Task which only gets run when nobody else needs the time. This is sometimes done in the *Shared Terminal mode* so that the Background Task (the one not on the screen) only runs when the Foreground Task is idle (ie. waiting on the keyboard). The switching of the screen between the Tasks also exchanges their Priority levels so that the higher priority will follow the Task on the screen.

Multiple Operating Systems

One of the interesting features of the SAGE Multi-User system is its capability of running multiple operating systems at the same time. This is due to its design being based on the evolution of the BIOS rather than the evolution of an operating system. The SAGE Multi-User system was created to run multiple copies of software systems which were originally designed to run in the Single-User environment. By supporting the same BIOS calls and making the Time Slicing invisible to the User Tasks, it is generally not too difficult to handle a normal Single-User operating system.

Many programs, including operating systems, on the 68000 processor are not dependent on where they are run in memory. This is important when it comes to running multiple copies of the operating system under the Multi-User environment. Some operating systems (such as the Volition Modula 2 environment) are dedicated to run at the base of the 68000 memory. In these cases it still may be possible to run one copy of the operating system in the User Task based at the bottom of memory. Of course, only one operating system with this type of restriction can be run at a time.

Operating System implementers should note that the major area of difference that they must consider under the Multi-User BIOS is the setting up of the 68000 Exception Vectors. The Multi-User BIOS provides calls to set up and read back the Exception Vectors so that it can prevent the different User Tasks from directly storing to the same area. Note that Exceptions are indirectly rerouted to each User Task rather than having the system completely change the processor's Exception table at each Task switch.

Static Allocation of Resources

Under the SAGE Multi-User environment, most of the system resources are statically allocated. This means that the assignment of resources (such as RAM memory, disk storage, time, etc.) is configurable, but once the system starts running, the allocations cannot be changed. There are, of course, trade-offs in using this approach rather than a dynamic approach where resources are allocated on the fly.

The dynamic allocation of RAM memory would lead to the problem of having to swap User programs on and off of the disk while the system is running. This puts an extra burden on the system and slows down overall thruput. With the cost of RAM memory dropping, it seems reasonable to install enough RAM so that normal Users may keep their programs resident.

The design decision to support multiple copies of existing single user operating systems dictates that totally separate

file systems be supported. The SAGE disk partitioning scheme provides the logical answer as to how to divide the storage area between systems that do not know that each other exist. The static allocation of disk space also prevents runaway programs (or runaway users) from grabbing all the disk space and forcing the system to halt. Each user is responsible for maintaining order within his own file system to prevent undesired interactions.

All configuration information for the SAGE Multi-User system is maintained by a program called MU.UTIL. The configuration information is stored in a configuration file (typically MU.CONFIG) separate from the Multi-User BIOS (MU4.BIOS). Different configuration files may easily be used to start the Multi-User system for use in completely different modes. For instance, one configuration could be set up for daytime use with a group of secretaries doing word processing. In the evenings, the system may be restarted with a different configuration to provide terminal access to partitions only used for training, or perhaps experimental program development.

Shared Data Access

In the SAGE Multi-user System, User Tasks are generally complete operating system environments and do not know that each other exist. This means that the operating system may not have a built-in mechanism for safely updating a shared device or data base. The Multi-User Environment provides several facilities to help solve the shared data access problem.

1. An exclusive control mechanism is provided to prevent two Users from accessing the same data at the same time. More information on this feature is provided on page 393.

2. Global Semaphores are provided. A Semaphore is an advanced feature available to applications programmers which allows one User Task to coordinate with another User Task. It is commonly used to allow multiple

copies of a data base program to access the same data base files. More information on Global Semaphores is contained on page 395 .

3. Logical communication channels are provided between the users which may be used for a Very Local Area Network (totally within the single computer). This network will require separate device or file server software to coordinate accesses to the mass storage and printer devices. Such software is currently under development for the p-System and other operating environments. Information on the SAGE Inter-communications facilities starts on page 410.

It is important for the System Manager to understand that access to shared data requires some planning. It is not just adequate to make a device area available to multiple users in order to safely share information. Most data handling programs designed for a Single-User environment will not have the necessary implementation to handle data in a shared storage environment. If a Database Program is to be used to share information between users, it must be specifically designed to work in the SAGE Multi-User environment using Global Semaphores, the Inter Communication Channels, or other means of coordinating data access.

IX.02 RUNNING THE MULTI-USER SYSTEM :

An individual user running under the Multi-User Environment has to know little information about the differences between Single-User and Multi-User operation. The System Manager should, however, be aware of some key items, even though a standard configuration is being used. Following is a discussion on some topics which will be useful background material in maintaining the Multi-User Environment.

● Terminal Considerations

Different User Tasks do not have to use the same type of terminal. It is more convenient to maintain the system if all terminals are the same, but there is no software or hardware restrictions in this area.

The environment for each User Task must be configured for the type of terminal used. For the p-System, this means that the Boot partition for each Task must have the correct SYSTEM.MISCINFO file and that SYSTEM.PASCAL has the correct GOTOXY routine installed. Refer to the section in the GETTING STARTED MANUAL for guidelines on how to do this.

The section on "Serial Channel Configuration" in the documentation of MU.UTIL should be read carefully before setting up the terminals.

Terminal Baud Rates

Although the serial ports will support 19.2K baud data rates, it may be desirable to decrease the user terminal baud rates to 9600 or 4800 baud depending on loading. Continuous reception of characters (which may even be created by multi-character escape sequences) could cause a loss of data if all channels are receiving characters at nearly the same time at high baud rates. Decreasing the incoming data rates will insure that there is enough processor time to handle continuous information from all

channels.

A second data rate consideration is that during situations under which several users are compute bound, a high data rate terminal's output may appear jerky. This is because the terminal output is also compute bound and during its time slice, it cannot put out enough characters to keep the terminal busy until the next time slice.

The terminal displays the characters it has been given very fast so the pause until the next time slice is obvious. If the terminal data rate is reduced, the effective throughput under these situations is maintained, but the terminal output appears much smoother. This consideration is only dependent on personal preferences and the configuration decisions are left to the System Manager.

Terminal Cables

Note that there are differences in the cables required to connect terminals to the different types of serial ports. The CPU board Modem port requires a cable with pins 2 and 3 reversed and pin 5 (Clear to Send) asserted (typically tied to +12V on pin 9). The four Aux ports on the Winchester board use pins 2 and 3 straight across (like the main Terminal port) but must have pin 4 (Request to Send) asserted (typically tied to +12V on pin 9) or tied to the terminal's hardware handshaking line. This cable is also usable on the main Terminal port.

For more information, refer to the 'Installing the Terminal' section of the GETTING STARTED MANUAL.

Sharing A Terminal

The system can be configured so that one person can run more than one User Task from the same terminal. The Tasks

"share" the terminal. The user can rotate through the Tasks by typing a special key (typically Control Underline).

In this mode, the BIOS actually emulates the terminal, keeping a representation of what is on the user's screen in memory. When the special control character from the keyboard changes the system to another user environment, the background program's screen is written to the terminal just as if had been outputting all the time.

Emulations for several terminals are currently supported. A facility called MUTRMSET (see page 420) is available to the user to define emulation parameters for other terminals. The new definition can be installed using MU.UTIL.

Multi-User System Files

Both the SAGE II and SAGE IV use nearly the same Multi-User files as the Multi-User BIOS is the same for each. The SAGE II, however, uses the Multi-User floppy boot MU4.FBOOT.CODE while the SAGE IV uses the Winchester boot MU4.WBOOT.CODE.

MU4.BIOS	The main Multi-User BIOS program.
MU4.FBOOT.CODE	The Boot program for the floppy.
MU4.WBOOT.CODE	The Boot program for the Winchester installed in the boot area of the Multi-User startup partition.
MU.CONFIG	Configuration file (name is optional)

Note that the file SYSTEM.BIOS is not used at all under the Multi-User environment.

The following programs are used by the Systems Manager to set up the Multi-User system:

WFORMAT.CODE This program is used for setting up both the single-user environment and the Multi-User environment. The major function used by the Multi-User manager is to specify the Winchester disk partitions. A complete description of this facility is provided on page 62 .

Note WFORMAT can only actually be run under the SINGLE-USER environment.

FORMATINFO.TEXT This is a text file containing the Winchester mapping information used by WFORMAT. It does not have to be called FORMATINFO.

MU.UTIL.CODE This program controls the many parameters of the physical devices on the system and assigns system resources to the User Tasks. It is similar to SAGE4UTIL, which is the configuration program used for the single-user system. Do not attempt to use MU.UTIL to change a single-user environment, or SAGE4UTIL to change a Multi-User environment. A complete description of the menus and options of MU.UTIL.CODE is provided on page 279 .

MU.CONFIG This file contains the configuration information maintained by MU.UTIL.CODE. If a file with this name exists when the Multi-User system is booted, then it is **AUTOMATICALLY** used. If it does not exist, then the bootstrap asks for the name of the configuration file. When first experimenting with the system, the manager is advised to choose a standard configuration (see the section on "Standard Multi-User Configurations) that is close to his needs and copy that file to one with another name to use for

experimenting. Once the configuration is stable, the file should be re-named MU.CONFIG.

Note that unlike the single-user environment which stores the configuration information in the BIOS file, the Multi-User system requires the additional file MU.CONFIG. It is not a TEXT file and must be accessed with MU.UTIL.

MU Booting Specifics

On the Winchester system, one device partition must be the initial bootstrap device. It needs to contain only the bootstrap and the files MU4.BIOS and MU.CONFIG. Any partition can be selected for the initial bootstrap device, as any partition can be booted from the debugger by specifying its number or name.

On the Winchester system, the switches can be set so that the system boots to Device 9, partition 1. Using this partition as the initial bootstrap partition will cause the system always to boot to the Multi- User system on RESET. Alternately, the switches can be set so that the system boots to the debugger.

Remember that the initial BIOS bootstrap is specific for each type of device. The bootstrap code for the floppy based bootstrap is contained in the file MU4.FBOOT.CODE. The bootstrap code for the Winchester based bootstrap is contained in MU4.WBOOT.CODE. These initial bootstraps only load the Multi-User BIOS and are independent of the operating systems which may be loaded by the users. The appropriate bootstrap is installed on the BIOS boot device using the Bootstrap copy option in SAGE4UTIL.

SAGE II Startup Environment

The initial preparation of the Multi-User floppy diskettes must be done using the normal single user environment. The BUILD diskette shipped with every system contains all the files needed to install a Multi-User system. The SAGE II Multi-User files are:

```
MU4.FBOOT.CODE
MU4.BIOS
MU.PBOOT.CODE
MU.UTIL.CODE
MU.CONFIG
```

The configuration file does not have to be called MU.CONFIG but can be any configuration file set up by MU.UTIL.

Three diskettes must be built for the SAGE II Multi-User.

1. Multi-User boot MU4.FBOOT.CODE loaded in the bootstrap area. MU4.BIOS and MU.CONFIG (or equivalent) must be present on the diskette.

2. The operating system boot is loaded into the bootstrap area and standard operating system files. For the p-System, MU.PBOOT.CODE is the bootstrap file, and SYSTEM.PASCAL, SYSTEM.MISCINFO, and SYSTEM.INTERP must be present.

3. This disk is a duplicate of #2.

When the system is booted with the Multi-User disk, each terminal will ask:

```
Insert System Diskette
Type S to start System
```

The Left hand drive (#4:) is normally associated with

the standard terminal port, while the Right hand drive (#5:) is used with the terminal on the Modem port.

When a diskette is inserted, the specific operating system bootstrap is loaded and the main system menu or prompt should be displayed.

SAGE IV Startup Environment

The initial preparation of the user Winchester partitions must be performed using the normal single user environment. The BUILD diskette shipped with your system has all the files needed to install the SAGE IV Multi-User system. The files needed for building the SAGE IV Multi-User are:

MU4.BIOS
MU4.WBOOT.CODE
MU.PBOOT.CODE
MU.UTIL.CODE
MU.CONFIG

The configuration file does not have to be called MU.CONFIG, but can be any configuration file set up by MU.UTIL.

The User boot partitions should each be loaded with the appropriate operating system files and have the operating system bootstrap installed. For the p-System, MU.PBOOT.CODE is the bootstrap file. SYSTEM.PASCAL, SYSTEM.MISCINFO, and SYSTEM.INTERP must be on the partition.

> NOTE: It may be useful to specify one partition as a single-user environment, with all of the files needed to configure the Multi-User system, so that this partition can be used (instead of the floppy) for any future changes. Remember to configure the single-user system with SAGE4UTIL so that it has access to the partition with the initial bootstrap.

The example starting on page 347 gives a step by step description of how to install a SAGE IV Multi-User system.

MU Booting Sequence

Bootstrapping the SAGE Multi-User Environment is a two stage process. The Debugger or automatic booting feature is first used to load in the Multi-User BIOS and initialize the system. A different bootstrap, appropriate for the specific operating system, is then loaded in by the Multi-User system from the device indicated as the Boot Device for each User Task.

The Multi-User bootstrap for the p-System is contained in the file MU.PBOOT.CODE. This bootstrap may be installed on the appropriate user boot partitions using the Bootstrap copy option in SAGE4UTIL.

If the "Boot message" flag is ON for the user, then the user terminal may display the following message after the Multi-User BIOS is loaded:

```
Type S to start System
```

At this point, the specific operating system bootstrap is loaded and the main system menu or prompt should be displayed. Refer to the section giving specifics for your operating system.

IX.03 BOOTSTRAPPING OPTIONS :

There are several available options to control the method in which the Multi-User system starts itself and the individual users. As shipped, the Multi-User BIOS is bootstrapped from one device or Winchester partition and then the users are each expected to boot from a different device or Winchester partition. All users will be bootstrapped at the same time without prompting.

The following is a list of situations which will require setting up additional configuration options.

Initiating a prompt message before booting

Automatic copying of files to RAM Disk

Booting users from RAM Disk

Booting multiple users from one device

Booting the BIOS and users from the same device

BOOT Message

Normally, after the Multi-User BIOS is loaded, all of the users are started at once. In the case of a floppy based system, a user would probably want to bootstrap from the same floppy drive which was used to bootstrap the Multi-User BIOS. Since the user immediately tries to start booting after the BIOS is loaded, an error will result because it cannot find the user's bootstrap on the diskette.

In order to give the user time to change diskettes before the user starts booting, an option is available to put up a boot message. When this option is turned on under the Users Configuration options, the message "Type S to start system" will be displayed and the user will not be bootstrapped until an "S" key is typed.

The "Boot message" option may also be used to manually control the order of the user bootstrapping. When the system normally starts, all the users are bootstrapped at once, and therefore are all competing for disk and CPU time. If the prompt is requested, then unattended user terminals will not slow down the startup of those users who are present.

Initializing RAM Disk

A method has been provided to initialize the directories of the RAM Disks and to copy files to the first RAM Disk. Note that this feature currently is only supported for p-System compatible directories, but could be modified by a knowledgeable user to handle other operating systems.

In order to initiate this process, the file MU.BOOTEXT.CODE must be present on the device from which the Multi-User BIOS is bootstrapped. Also, for each RAM Disk which is to be automatically initialized, the "Initialize Ram Disk" flag in the RAM Disk's configuration must be turned on with MU.UTIL.

Each specified RAM Disk will have its directory zeroed with the number of blocks set according to the available memory.

Only Ram Disk #1 will have files copied from the device containing the Multi-User BIOS.

Note that if the Multi-User system is booted with "IFR" or "IHR" then the RAM Disks will not be initialized. A file named ENDBOOT on the booting device will terminate the file copy. If RAM Disk #1 is to be initialized with a directory, but not have files copied to it, the file ENDBOOT should be placed at the very beginning of the BIOS boot device.

Booting Users from RAM Disk

The RAM Disk initialization is performed before any users are allowed to start. Initializing RAM Disk #1 does not automatically bootstrap any users from the RAM Disk. To accomplish booting a user from RAM Disk, the user must be configured with the RAM Disk visible in the user's channel map and the boot device must be set to the logical channel assigned to the RAM Disk. Also, the bootstrap which is placed on the RAM Disk (at block zero) is taken from the file RAM.BOOT.CODE. The user should put a copy of the appropriate user operating system boot file (such as MU.PBOOT.CODE) onto the BIOS boot device and name the copy RAM.BOOT.CODE.

Note that unlike the single user system, the file SYSTEM.INTERP must be included on the RAM Disk. This is because the user bootstraps are performed later and SYSTEM.INTERP must be present at user boot time. The MU4.BIOS, MU.CONFIG (or other user configurations), and RAM.BOOT.CODE are not necessary on the RAM Disk and should be placed beyond the ENDBOOT file so that they will not be transferred.

The following information is only necessary for users who wish to modify the method for initialization of RAM Disk or to add other functions to the boot extension file. MU.BOOTEXT.CODE contains 68000 code generated by the p-System assembler. It is not linked or compressed.

The first four bytes of the code should contain the ASCII characters, MUBE standing for Multi-User Boot Extension. This is checked by the bootstrap to verify proper file content. The code from MU.BOOTEXT.CODE is loaded into memory by the BIOS bootstrap loader and its address is passed to the BIOS. The code is called after BIOS initialization at offset 4 of the code (just beyond the MUBE password). At completion, the boot extension routine should do a normal return (RTS).

The boot extension is run under the task assignment for User #1. This means that any console interaction (error messages) will go to the console defined in the logical channels 1 and 2 for User #1.

Normally, the Terminal port on the main CPU board should be used for User #1 so that all boot error messages go to the same terminal. The boot extension is started after the BIOS is initialized and therefore it must use the BIOS drivers. Logical device 9 of User #1 is temporarily modified by the BIOS to access the BIOS boot device. This assignment is restored to its original device after the boot extension returns but before the user is started.

Multiple Users on one Device

It is possible to bootstrap multiple users from one device, however, several items should be considered. The p-System currently gets its physical code pool address from the file SYSTEM.MISCINFO. Therefore, the Multi-User p-System bootstrap patches the SYSTEM.MISCINFO file at user boot time with the appropriate memory address and size for the user. At p-System boot time the SYSTEM.MISCINFO file is read by the p-System to get the code pool information.

This information is never used again (even when doing an I command for Initialization). It is therefore necessary to stagger the booting so that one p-System reads its patched SYSTEM.MISCINFO before another p-System bootstrap modifies the file.

Once control is transferred to the p-System interpreter it is difficult for the BIOS to know when the SYSTEM.MISCINFO has been read. Therefore, a "Boot Control Delay" has been provided for each user. If non zero, this delayed value is used as a number of seconds during which other users will not be bootstrapped after the user starts.

For floppy drives, a value of 10 seconds should insure that the SYSTEM.MISCINFO has been read. For Winchester drives and RAM Disks, a value of 5 seconds should be used.

Note that although User #1 always boots first, the order of booting for the other users is indeterminate when using non zero Boot Delays. Setting the Boot Delay for User #1 to a value long enough to perform the complete boot will expedite getting the first user started, even if multiple users are not being bootstrapped from the same device.

When multiple users are booted to a common system device, there is potential for problems with multiple users writing to the system device. For instance, the p-System work files are always written to the system device. Therefore, it is inappropriate to use the standard p-System Editor with work files. In fact it may be desirable to set up the configuration to disallow write access to the system device.

To allow the SYSTEM.MISCINFO patch, the BIOS temporarily allows writing to the system device until the Boot Delay expires.

It is also obvious that if SYSTEM.MISCINFO is being shared by several users, the parameters defined by SYSTEM.MISCINFO will be the same for all users. They must therefore use the same kind of terminal. Different terminals may be used only by setting up different system devices for each type of terminal.

BIOS and User(s) from same Device

Normally the bootstrap for loading the Multi-User BIOS and the bootstraps for loading the Users themselves are located on block zero of the device from which they are bootstrapped. This means that a User normally cannot be booted on the same device (or partition) as the Multi-User BIOS.

To remove this restriction, it is possible to put the User system bootstrap (such as MU.PBOOT.CODE) into a file called USER.BOOT.CODE on the BIOS boot device. The BIOS remembers the device used for the BIOS boot and also the location of the USER.BOOT.CODE file on this device. If a user attempts to boot to the BIOS boot device, it will use the bootstrap code from USER.BOOT.CODE instead of the information at block zero. Note that it is important that the file USER.BOOT.CODE not be moved on the device or the system will fail on the next user boot.

Having a normal user share space on the BIOS boot device is generally not a good idea. The user may inadvertently remove a necessary file. However, it is appropriate to share a device with a User Task which is being used for a dedicated system purpose such as a Login program or file server.

IX.04 RUNNING THE p-SYSTEM :

The p-System operating system has not been modified by SAGE and is still a single-user system internally. Each User Task which runs under the p-System has its own copy of the system programs and is "invisible" to the other users unless disk partitions (or other devices) are shared. Application programs which wish to share information between tasks must use the techniques discussed in the section on "Advance Multi-User Features" and "Inter User Communications".

There are some differences in loading the system and how errors are handled. The system diskettes must not be write-protected because the file SYSTEM.MISCINFO must be patched by the bootstrap. This patching is necessary to insure that the appropriate code pool location and size is indicated for the user's memory partition.

The user's system device must contain all the files necessary for normal booting (and running) except the file SYSTEM.BIOS. The BIOS function is replaced by the Multi-User Environment which has already been loaded. The user's system device must contain the operating system specific bootstrap code from the file MU.PBOOT.CODE installed in the bootstrap area. The same Multi-User p-SYSTEM bootstrap works for both floppy and Winchester boot devices.

Typing H(alt at the p-System command line will return the user to the boot message (currently "Type S to start System") if the user's Boot message flag is on. Otherwise the system will automatically reboot and bring up the p-System command line (or SYSTEM.STARTUP program).

Some system error conditions such as STACK OVERFLOW do not respond to the p-SYSTEM BREAK key. To return to the startup message, use the terminal BREAK key and then select the option to reboot. It is normally NOT necessary to RESET the entire system.

X MU.UTIL - Multi-User UTILITY :

The SAGE Multi-User Environment configuration is somewhat different from configurations of the single user environment. All of the configuration information has been split off into a separate file from the BIOS. This allows the system manager to easily bring up the Multi-User system configured for different purposes.

The Multi-User configuration utility program is MU.UTIL.CODE. It does not have any facility for on-line configuration of resources. It requests that the user enter the file name which contains the configuration information, typically MU.CONFIG. This file is accessed by the Multi-User BIOS bootstrap to set up the BIOS and User Parameters.

The menus for MU.UTIL start on page 283 . The following section describes how to use the menus to setup your options.

X.01 MENU OPERATION :

The MU.UTIL program uses the same menu driver as the SAGE4UTIL configuration program. The units CONFIGSAGE, SIO_UNIT, and MNU_UNIT from the SAGETOOLS file have been installed into MU.UTIL.

Each menu in SAGE4UTIL has the general form:

```
c   item  val
c   item  val
.......

Select menu item <CR exits, ! aborts>
```

Normally the character (c) to the left of the desired menu item is typed to select the item.

A carriage return will exit to the previous parent menu.

An exclamation character (!) will exit a number of levels back to a logical point in the menu tree without making any permanent changes.

A menu item may have a value associated with the item. If present, the value is displayed to the right of the item field. Note that the initial menu that appears in SAGE4UTIL does not have any value fields displayed. Menu items without value fields will either call another "subsidiary" menu or are used to drive other functions of the program.

The On-Line Configuration selection in SAGE4UTIL immediately goes to a subsidiary menu allowing configuration selection for a variety of devices. The BIOS File Configuration selection goes to the same menu after stopping to query the user for the name of the BIOS file to be configured. The Floppy Formatter and Bootstrap Copy selections drive routines in SAGE4UTIL to perform the specified functions.

NUMERIC fields

If a numeric (decimal or hex) value is displayed, selecting the item will display the item field on the prompt line and wait for the user to enter a new value. Typing just a carriage return indicates that the item should not be changed and the selection prompt will be re-displayed.

BOOLEAN fields

If a boolean value (On or Off) is displayed, selecting the item will display the item field on the prompt line and wait for the user to enter either "ON" or "OFF". Typing just a carriage return indicates that the item should not be changed and the selection prompt will be re-displayed.

TEXT fields

Some value fields are text strings which are selected on a mutually exclusive basis in a subsidiary menu. Under the Terminal Configuration Control menu, the Terminal Baud Rate, Terminal Parity, Terminal Stop Bits, and Terminal Data Bits fields are of this category. In the subsidiary menu, an asterisk appears to the right of the current selection. If another item is selected, the asterisk moves to indicate the new item. When a carriage return is typed to return to the previous menu, the new selection is displayed.

PRINTOUT OPTION

A built-in printout option lets the user save the configuration in a file or print it out. Typing a '=' at any main prompt will ask for the file name. Typing a '/' at any main prompt will send the contents of the screen to the output file. Menus without data can also be stored, allowing the user to build a record of a complete configuration session. The output file is closed on exit from SAGE4UTIL or when another '=' is typed.

X.02 MU OPERATION CHECKS :

Various checks are performed by MU.UTIL to insure that the configuration setup by the user is correct. These checks are done at the end of the configuration session when the message 'READY to write changes to MU.CONFIG?' is displayed, and the user responds with a "Y" . If an error occurs at this point, the user has the option of RETURNING to the configuration being built or escaping to the p-System command line.

X.03 MULTI-USER CONFIGURATION :

When first executed, MU.UTIL displays the following information. A row of dots is output during the program initialization.

```
Multi-User Configuration Utility - Version 2.0
Initializing.....................................
```

Then it requests the name of the file containing the configuration to be modified.

```
Multi User configuration File name:
```

After entering the file name, MU.UTIL will reply:

```
Configuration Version 2.0 read sucessfully
Type <space> to continue
```

When a space is typed, the main configuration menu will be displayed:

```
Multi User Configuration Utility

A - User Configuration
B - Serial Channels
C - Left  Floppy
D - Right Floppy
E - RAM Disks
F - Parallel Printer Port
G - Winchester #1 Access
H - Winchester #2 Access
I - Winchester #3 Access
J - Winchester #4 Access
K - Time Adjustment
L - Low Level Configuration
M - Auxitiary Device Info

Select Menu item <CR exits, ! aborts>:
```

Each of these options will now be discussed.

Configuring the User Tasks

The 'A' Selection from the Multi-User menu displays:

```
User Selection

A - User #0 (reserved)
B - User #1
C - User #2
D - User #3
E - User #4
F - User #5
G - User #6
H - User #7
I - User #8
J - User #9
K - User #10
L - User #11
M - User #12
N - User #13
O - User #14
P - User #15

Select Menu item <CR exits, ! aborts>:
```

Once a specific user is selected, the appropriate information is presented in the following menu.

```
          User #1 Configuration

A - Channel Map ( 0 to 15)
B - Channel Map (16 to 31)
C - User Capabilities
D - CP/M Information
E - Operating System Info
F - Boot Device                    9
G - Boot message                 Off
H - Boot control delay             0
I - Shared terminal mode         Off
J - Priority                      64
K - Number of Comm buffers         1
L - Base Memory Address       000400
M - Top Memory Address        023400
N - System Stack Address      023500
O - Time Slice                  1600
P - Capability mask           00000000

Select Menu item <CR exits, ! aborts>:
```

X.04 THE BIOS CHANNEL MAP :

Selections A and B from the User Configuration menu bring up portions of the BIOS Channel Map, also referred to as Channel Map. Each user task may have a different allocation of devices in its Channel Map.

The following two menus show a typical allocation of the Logical to Physical channels for a User Task.

Note that under the p-System, the extra serial channels are assigned only one logical device number (28, 29, 30 or 31). The channels may actually talk to 2 physical devices, an input (keyboard) and an output (terminal). Specifying either the input or output physical device will assign both.

Note that some devices such as Floppy drives, Winchester Drives, and RAM Disks must be enabled under their configuration control menus to be available. For instance, the Right Floppy drive may be disabled because two drive SAGE IV systems are not standard. Simply specifying a logical to physical channel relationship in the BIOS Channel Map does not automatically enable the drive for use.

```
                    User Channel Map (0 to 15)

A - Channel  0 device =    0        Q - Channel  0 subdevice =  0
B - Channel  1 device =    1        R - Channel  1 subdevice =  0
C - Channel  2 device =    2        S - Channel  2 subdevice =  0
D - Channel  3 device =    0        T - Channel  3 subdevice =  0
E - Channel  4 device =    4        U - Channel  4 subdevice =  0
F - Channel  5 device =    5        V - Channel  5 subdevice =  0
G - Channel  6 device =    6        W - Channel  6 subdevice =  0
H - Channel  7 device =    7        X - Channel  7 subdevice =  0
I - Channel  8 device =    8        Y - Channel  8 subdevice =  0
J - Channel  9 device =    9        Z - Channel  9 subdevice =  3
K - Channel 10 device =    0        0 - Channel 10 subdevice =  0
L - Channel 11 device =   11        1 - Channel 11 subdevice =  0
M - Channel 12 device =    0        2 - Channel 12 subdevice =  0
N - Channel 13 device =    0        3 - Channel 13 subdevice =  0
O - Channel 14 device =    0        4 - Channel 14 subdevice =  0
P - Channel 15 device =    0        5 - Channel 15 subdevice =  0

Select Menu item <CR exits, ! aborts>:
```

```
                    User Channel Map (16 to 31)

A - Channel 16 device =    0        Q - Channel 16 subdevice =  0
B - Channel 17 device =    0        R - Channel 17 subdevice =  0
C - Channel 18 device =    0        S - Channel 18 subdevice =  0
D - Channel 19 device =    0        T - Channel 19 subdevice =  0
E - Channel 20 device =    0        U - Channel 20 subdevice =  0
F - Channel 21 device =    0        V - Channel 21 subdevice =  0
G - Channel 22 device =    0        W - Channel 22 subdevice =  0
H - Channel 23 device =    0        X - Channel 23 subdevice =  0
I - Channel 24 device =    0        Y - Channel 24 subdevice =  0
J - Channel 25 device =    0        Z - Channel 25 subdevice =  0
K - Channel 26 device =    0        0 - Channel 26 subdevice =  0
L - Channel 27 device =    0        1 - Channel 27 subdevice =  0
M - Channel 28 device =   19        2 - Channel 28 subdevice =  0
N - Channel 29 device =    0        3 - Channel 29 subdevice =  0
O - Channel 30 device =    0        4 - Channel 30 subdevice =  0
P - Channel 31 device =    0        5 - Channel 31 subdevice =  0

Select Menu item <CR exits, ! aborts>:
```

User Capabilities

The Capability Mask is a two word (32 bit) number represented in hexadecimal. Each of the bits may be set to indicate to the BIOS that a specific BIOS capability is available to the user.

Typing "C" at the User Configuration menu will bring up a menu of the pre-assigned User Capability flags.

```
          User Capabilities

A - System Manager Flag          Off
B - Allow configuration changes   On

Select Menu item <CR exits, ! aborts>:
```

Bit 31 of the long word is assigned to allow the special operations restricted to the System Manager. This is called the **System Manager Flag** . When set **On** , the System Manager flag allows a user program to invoke certain privileged operations. These privileged operations are documented throughout the Multi-User portion of this manual. One of the significant operations is the modification of critical configuration information such as the user's Channel Map.

Bit 30 of the long word is assigned to the **Allow Configuration Changes Flag** . If this flag is set to OFF, no configuration changes may be made under the Multi-User system (even if the System Manager flag is on). If this flag is set to **On** , and the System Manager flag is off, a limited set of configuration changes may be made. These include changing the serial port and floppy characteristics for devices which are visible in the user's Channel Map.

● CP/M Information

This is a menu that allows you to set some low-level information for CP/M-68k. Both the CP/M Auxiliary Device Information Table and Operating System Information can be changed with this menu without knowing how the information is packed in those tables.

```
                    CP/M Information

  A - Number of disk I/O Buffers                  1
  B - Size of buffer in 512-byte blocks           1
  C - Write modified buffers on warm boot         No
  D - Write directory buffers immediately         No
  E - CP/M Disk Drive Configuration

  Select Menu item <CR exits, ! aborts>:
```

Option A controls the number of LRU (least recently used) disk buffers.

Option B controls the Size of each LRU disk buffer.

Option C & D are LRU Disk Buffer Handling Flags. See the SAGE CP/M-68k reference manual.

X.05 CP/M DISK DRIVE CONFIGURATION :

Option E allows control of the CP/M disk characteristics (block size, directory size, etc.) for each disk:

```
CP/M Disk Drive Configuration

A - Disk Drive A
B - Disk Drive B
C - Disk Drive C
D - Disk Drive D
E - Disk Drive E
F - Disk Drive F
G - Disk Drive G
H - Disk Drive H
I - Disk Drive I
J - Disk Drive J
K - Disk Drive K
L - Disk Drive L
M - Disk Drive M
N - Disk Drive N
O - Disk Drive O
P - Disk Drive P

Select Menu item <CR exits, ! aborts>:
```

The Menu for individual drive selection is:

```
        Disk Drive A: Configuration

A - Logical Block Size                  1024
B - Directory Track Offset                 0
C - Number of Directory Entries            0
D - Disk Media                         Fixed

Select Menu item <CR exits, ! aborts>:
```

Operating System Information

32 bytes of information are available for use by the different operating systems. The information is displayed in Hexadecimal. The definition of each byte is different for each operating system.

The p-System does not use any bytes.

```
Operating System Information

A - Word  0     0000
B - Word  1     0000
C - Word  2     0000
D - Word  3     0000
E - Word  4     0000
F - Word  5     0000
G - Word  6     0000
H - Word  7     0000
I - Word  8     0000
J - Word  9     0000
K - Word 10     0000
L - Word 11     0000
M - Word 12     0000
N - Word 13     0000
O - Word 14     0000
P - Word 15     0000

Select Menu item <CR exits, ! aborts>:
```

CP/M Operating System Information.

Under CP/M-68K, the Operating System Information is used to represent some low-level configuration information for CP/M. The first word of the 16-word block (word 1) contains two LRU disk buffer parameters. The first byte is the number of disk buffers minus 1, and the second byte is the size of each disk buffer in 512 byte blocks minus 1.

The second word contains the RAM disk configuration information. See page 322 , the Auxiliary Device Information, for the format of this word.

The next 16 bytes (words 3 through 10) contain the CP/M startup command. This CP/M command is automatically issued when CP/M first boots. The command is a zero-terminated string of ASCII characters. Currently, there is no high-level support for the startup command. You must manually set the ASCII bytes into words 3 through 10 of the Operating System Information.

The next word (word 11) contains two LRU disk buffer flag bytes. If the first byte is non-zero, then all modified disk buffers are written to the disk when a CP/M warm boot occurs. If the byte is zero, modified buffers are not written until they are either required for another use, or the SAGE utility SYNC is executed. The second byte, if non-zero, causes modified directory buffers to always be written immediately after the modification. If the byte is zero, directory buffers are not special.

The Boot Device Parameter

The Boot Device is the Logical Channel in the User's Channel Map which contains the system to be bootstrapped. Note that on Winchester systems, most users will boot to device #9. The BIOS Channel Maps, however, contain different disk partition assignments for each of these User Tasks so the Users actually bootstrap to different areas of the disk.

Boot Message

When the Boot Message flag is set to ON, the message "Type S to start System" will be displayed and the system will wait for the letter S to be typed before it starts the user task. This message is typically used when a user is configured to boot from a floppy disk. In this case, the message "Insert System Diskette" is also displayed.

Boot Control Delay

Certain situations described in the section on Bootstrap Options require that user bootstrapping operations be staggered in time. The Boot Control Delay value specifies a number of seconds after a user bootstrap is started that no other user bootstrap can start.

The Shared Terminal Mode

The Shared Terminal Mode allows running multiple User Tasks from a single terminal. Selecting this option brings up a subsidiary menu as shown.

```
Shared Terminal Mode

A - Off          *
B - Foreground
C - Background

Select Menu item <CR exits, ! aborts>:
```

The normal Shared Terminal Mode setting for separate terminals is Off. When using two or more User Tasks on the same terminal, one User Task should be selected in Foreground mode and the remaining User Tasks for the same terminal should be selected for Background mode. The User Task selected in Foreground mode will be the task which initially comes up on the screen. Typically, the user will want the Foreground task to pre-empt the Background tasks when it is compute bound. Therefore, the Background task Priorities are generally set one lower than the Foreground task (background Priority=63, foreground Priority=64). When the key is pressed to switch the terminal between tasks, the Priority fields of the two tasks are also swapped so that the new Foreground task will get the same Priority as the original Foreground task.

The default key assigned to cause the swapping of tasks is a Control Underline (Control Shift Dash). This key assignment is configurable under the Serial Channel Configuration menu.

Note that each User Task which shares the same terminal must also have its Logical Channels 1 and 2 assigned to the same Physical (Keyboard and Terminal) Channel numbers in their BIOS Channel Maps.

Note also that the "Type of Terminal" selection must be changed in the Serial Channel configuration if the terminal is not a standard SAGE (QUME VT102) terminal.

The Priority Parameter

The Priority factor allows the assignment of higher and lower priority tasks than the normal user priority. It must be noted that only the active tasks of the highest priority will be run in the normal rotation. Thus, a compute bound task running at a higher priority than the normal users will completely lock out the users until it has to wait on I/O. A task running at a lower priority than the normal users will be run at any time that all the normal users are inactive. The highest priority is 127 and the lowest priority is 1. If the priority is set to zero, the task is never configured. The normal user priority is set to the middle of the range at 64.

Number of Comm Buffers

This value determines the number of Inter-User Communication buffers allocated for the user. A value of one is sufficient unless an application wants to use the asynchronous transmission feature to send multiple messages at once. More details may be found in the section on Inter-User Communications.

Base Memory Addresses

The Base Memory Address, Top Memory Address, and System Stack Address have been discussed previously under the section on Memory Allocation. Selection of any of these entries allows the user to type in the desired hexadecimal address. These values seldom require change.

The Time Slice Parameter

The Time Slice parameter allows each user to be allocated a configurable amount of time. All active users of the highest priority are processed on a round robin basis. The amount of time that each user runs before the system advances to the next user is individually configurable. The Time Slice factor is a decimal time value representing the number of 1/64000 of a second allocated. A six user system should be set up with each user getting about 25 milliseconds (Time Slice = 1600) of processing time per rotation. This means the total rotation will take about 150 milliseconds. System Managers may want to experiment with different time allocations to support their needs. For smaller numbers of users, the Time Slice may be increased to reduce the context switching overhead. Normally, the value is never changed from the default.

Capability Mask

The Capability Mask is a two word (32 bit) number represented in hexadecimal. Each of the bits may be set to indicate to the BIOS that a specific BIOS capability is available to the user.

Note that these flags are more easily defined under the option "User Capabilities". A description of the pre-assigned User Capability flags is given in the discussion there.

X.06 SERIAL CHANNEL SELECTION :

Selecting the Serial Channel Configuration option will display a menu allowing the specification of one of the six available serial channels. Note that the four extra serial channels are unavailable on SAGE II computers.

```
        Serial Channel Selection

    A - Sage II Terminal Port
    B - Sage II Modem Port
    C - Sage IV Extra Serial Port #1
    D - Sage IV Extra Serial Port #2
    E - Sage IV Extra Serial Port #3
    F - Sage IV Extra Serial Port #4

    Select Menu item <CR exits, ! aborts>:
```

Once a Serial channel is selected a new menu is displayed.

The Terminal port is the same for both the SAGE II and SAGE IV. Some of its parameters are normally taken from the settings of the DIP switches on the back of the machine at the time the system is booted. (The DIP options is available for all serial ports.) The following table shows the options for a typical terminal.

```
                        Terminal Channel Configuration

    A - Baud Rate          DIP Switch     K - Remote Channel          Off
    B - Parity             DIP Switch     L - Xon/Xoff on output      On
    C - Stop Bits          1 Stop Bit     M - Xon/Xoff on input       Off
    D - Data bits          8 Data Bits    N - Count to send XOFF      240
    E - Xmit Buff Length         256      O - Count to send XON       64
    F - Rec Buff Length          256      P - DSR Polling             Off
    G - Type of Terminal   FREEDOM 100    Q - DSR Poll interval       16384
    H - Char to chng user          _      R - Input event num         0
    I - BREAK to reboot           On      S - Output event num        0
    J - BREAK to debug            Off     T - Access Control

    Select Menu item <CR exits, ! aborts>:
```

The same menu items are presented for each serial channel selection.

Baud Rate

All of the following Baud rates are available under Multi-User, except 200 baud, which will not work on the Auxiliary serial ports. Note that the terminal (or device) on the port must be set (usually by switches) to run at the same baud rate you select for the SAGE serial port.

It is usually convenient to set all devices of the same type to the same baud rate. For example, if two types of printers (a dot matrix and a letter quality) are used on the same port, then switching between them is only a matter of changing cables and no re-configuring has to be done.

```
Baud Rate Selection

A - 19200 baud
B -  9600 baud
C -  4800 baud
D -  2400 baud
E -  2000 baud
F -  1800 baud
G -  1200 baud
H -   600 baud
I -   300 baud
J -   200 baud
K -   150 baud
L -   110 baud
M -    75 baud
N -    50 baud
O - DIP Switch *

Select Menu item <CR exits, ! aborts>:
```

Parity

If parity is not used, then the option "Parity disabled" should be chosen. If Parity is to be used, then a choice of "Even Parity" or "Odd Parity" must be made.

The device connected to the serial port **MUST** have the same parity as the SAGE. The DIP Switch selection is for display only. Control of the DIP Switch option is only available under the Baud Rate menu.

```
Parity Selection

A - Even parity
B - Odd parity
C - Disabled parity
D - DIP Switch      *

Select Menu item <CR exits, ! aborts>:
```

- **Stop bit**

The device connected to the serial port **MUST** have the same number of Stop bits as the SAGE. Note that one stop bit is most commonly used.

```
      Stop Bit Selection

A - 1 Stop Bit    *
B - 1.5 Stop Bits
C - 2 Stop Bits

Select Menu item <CR exits, ! aborts>:
```

- **Data bits**

The device connected to the serial port **MUST** have the same number of Data bits as the SAGE. Note that seven Data bits is most commonly used for printers and eight data bits for terminals.

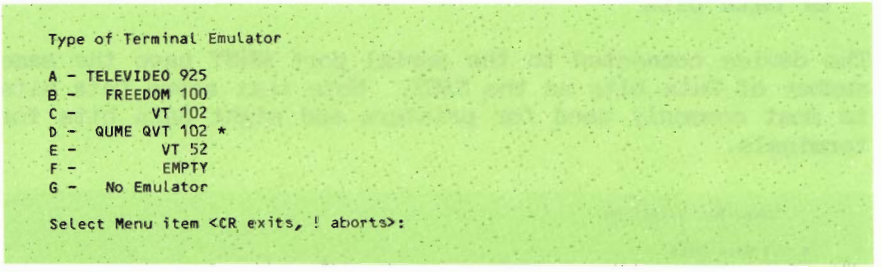
```
      Data Bit Selection

A - 5 Data Bits
B - 6 Data Bits
C - 7 Data Bits
D - 8 Data Bits *

Select Menu item <CR exits, ! aborts>:
```

Xmit and Receive Buffer Length

The Xmit and Receive Buffer lengths are configurable from 2 to 32767 bytes. The default value is the same as single user systems, 256 bytes.

Type of Terminal

This menu selection allows choosing a Terminal Emulator for use with the Shared Terminal option. When entered, this selection will display the following menu of six possible terminals plus the option "No Emulator". If a normal terminal entry is undefined it will have the name "EMPTY" and should not be selected.



Installation of alternate Terminal Emulators into the six choices is done under the "Low Level Configuration" menu with the entry "Load Terminal Emulator". For information on how to create a Terminal Emulator see the section on MUTRMSET on page 420 .

Char to Change User

This menu selection allows the specification of the character to change users when using the Shared Terminal option. Only a single character may be specified. Typically, the default is CTRL underline. The character may be specified in several formats. The ASCII value or name may be used (example US). A control character may be specified with the up arrow notation (example ^_). The hexadecimal value may be specified followed by the character H (example 1FH).

Note that on DEC VT102 terminals (and possibly other DEC terminals) the character generating the value IFH is a CTRL/? (Control question mark) instead of a CTRL underline.

BREAK to reboot

The actual BREAK KEY on normal terminals generates a continuous signal which causes the receiving USART to detect framing errors. The normal option for the BIOS is to ignore framing errors. If a large number of continuous framing errors (>255) is detected, the USART receive channel is turned off and checked only at one second intervals. This allows the terminal to be turned off (often causing continuous framing errors) without turning off the SAGE II/IV computer. The computer can continue to execute a program without being continuously interrupted by the powered down terminal.

The **BREAK to reboot** flag may be turned on to allow the user to exit his environment and restart his task. This is useful in situations such as p-System Stack Overflows which require rebooting the system. Using the terminal's BREAK key (not the p-System Break key) a prompt is brought up as follows:

```
BREAK KEY HIT - C(ontinue, R(eboot?
```

Typing an R will cause the user's operating system to be re-booted. If the Boot Message flag was set ON in the User Configuration menu, the initial user startup message will be displayed.

```
Type S to start system
```

Typing a "C" will put up the message:

```
Continuing after BREAK
```

and go on as though the break had never occurred.

● BREAK to debug

A second type of BREAK option is provided to allow Exiting the Multi-User system and to allow for low level debugging in the Multi-User environment. The flag for this option is called **BREAK to debug.** The "BREAK to reboot" flag must be also turned on with this option.

If enabled, this option displays the following prompt when the BREAK key is typed.

```
BREAK KEY HIT - C(ontinue, R(eboot, D(ebugger, F(reeze, E(xit?
```

Continue returns the user to the program.

Reboot reboots the User Task.

Debugger enters the PROM Debugger without disturbing the other users. Only one user should ever be in the Debugger at any one time as the PROM Debugger is not re- entrant and has only one area for variable storage. It is advisable to only enable one terminal with the "BREAK to debug" feature.

When the 'D' key is typed, the SAGE Debugger will be entered, so this option should only be selected when troubleshooting programs at a low level. All normal program information is preserved and the program may be resumed by typing 'GO' to the Debugger. Refer to the SDT/ASSEMBLER MANUAL for information on the SAGE Debugging Tool.

Freeze disables interrupts, thus freezing the complete system, before entering the PROM Debugger. The Freeze option should be considered non recoverable, requiring the Multi-User Environment to be rebooted. The Freeze also requires that the CPU Terminal port be used for terminal I/O during debugging (even though it can be initiated from another terminal).

Exit allows complete termination of the Multi-User Environment and a non returnable entry into the Debugger.

Remote Channel

The **Remote Channel** flag is used by the BIOS to determine which type of serial driver routine to associate with each channel. It is important that this flag match the system usage. When the Remote Channel flag is off, the keyboard/terminal style driver is used. This driver recognizes the p-System Break and Flush keys as well as the Terminal BREAK key (if enabled).

If the Remote Channel flag is on, the remote/printer style driver is used. This driver supports XON/XOFF protocol in the Input mode as well as the normal Output mode. The Remote Channel driver will also support Data Set Ready polling on the Modem port for use in handshaking. The CPU Terminal port should always be configured as a user terminal and is not supported as a remote device.

The Terminal or Remote drivers will support **XON/XOFF** in the Output mode. The receiver will respond to XON and XOFF characters which are received to start and stop the transmit direction. XON/XOFF in the Input mode is only supported by the Remote driver. This means that the receiver will transmit XON and XOFF characters to control the transmitting from the opposite system.

XON/XOFF

Serial channels configured with the Remote Channel flag on, will support XON/XOFF protocol in both output and input directions. Those channels configured with the Remote Channel flag off will only handle XON/XOFF on output.

When "XON/OFF on output' is turned on, the receiver will respond to XON and XOFF characters which are received to start and stop the transmit direction. This is the normal mode configured to handle XON/XOFF sent by a terminal.

When using the "XON/XOFF on output" for a terminal (Remote Channel flag is off), the p-System start/stop character must be set to XOFF (ASCII DC3, CTRL/S) using the program SETUP. Manual control of the terminal output will also require typing CTRL/S (XOFF) to stop the output and CTRL/Q (XON) to restart the output.

When "XON/XOFF on input" is turned on, the receiver will transmit XON and XOFF characters to control transmitting from the opposite system.

Counts for sending XOFF/XON

The menu selection, Count for Sending XOFF, specifies the threshold for the number of characters in the receive buffer when the XOFF character will be sent.

The menu selection, Count for Sending XON, specifies the threshold for the number of characters in the receive buffer to send the XON character (if XOFF has already been sent).

- **DSR Polling & Interval**

The DSR Polling selection is used for the Modem port only in the Remote driver mode. The Getting Started Manual covers a more detailed description of this option.

Normally, the standard polling interval does not have to be changed.

- **Input & Output event numbers**

Because serial channels may be assigned to different logical device numbers for different users, a consistent event numbering scheme is impossible (a physical device cannot have an arbitrary number of different event number assignments).

Serial channels which are used as user consoles will always have the standard terminal event numbers (19 and 61 for input, 33 for output).

Remote channels will use the event numbers specified in this menu, independent of the logical channel assignment. Note that the shipped default is 32 for input and 34 for output to correspond with the Remin and Remout channels.

- **Access Control**

The Access Control menu allows the System Manager to specify which users can use the device. Refer to page 325 for more information on DEVICE ACCESS CONTROL.

X.07 FLOPPY CONFIGURATION :

The Floppy configuration menus are similar to the menus available under SAGE4UTIL.

```
                    Left Floppy Drive Selection

A - Floppy Configuration          SAGE double side, 80 track (1280 blocks)
B - Access Control

Select Menu item <CR exits, ! aborts>:
```

Selecting "A" displays the following menu:

```
                  Left Floppy Format Selection

A - SAGE double side, 80 track (1280 blocks)                *
B - SAGE double side, 40 track ( 640 blocks)
C - IBM single side, 40 track - Universal Media (320 blocks)
D - IBM double side, 40 track (640 blocks)
E - Network Consulting single side, 40 track (400 blocks)
F - Network Consulting double side, 40 track (800 blocks)
G - Network Consulting double side, 80 track (1600 blocks)
H - IBM double side, 80 track (1280 blocks)
I - Non Standard Drive Configuration
J - No drive equipped
K - SAGE 10 sector per track, 80 track (1600 blocks)

Select Menu item <CR exits, ! aborts>:
```

● 80 vs 40 TRACK DRIVES

The 80 track (96 TPI) diskettes may only be accessed on systems with 80 track drives. The 40 track (48 TPI) diskettes may be read on an 80 track drive but cannot be easily written. Note that only SAGE 80 track or 40 track diskettes may be used to start (bootstrap) the system. The PROM driver used for bootstrapping does not handle all the varied formats which are configurable by the BIOS.

IBM Format

IBM initially provided only single sided drives on their personal computer. Therefore, when they came out with double sided drives, they deviated from the optimum cylinder orientation so that the the same scheme would read both the single sided and double sided drives. With the IBM track format, data is stored in ascending track order on side zero and then back in descending track order on side one. The normal SAGE II/IV method is to store data on side zero and then side one of each track before stepping the head to the next cylinder.

NCI Format

A special sector numbering scheme was developed by Network Consulting Incorporated for their BIOS on the IBM Personal Computer. It allows their software to automatically distinguish between their 10 sector per track diskettes and the normal 8 sector per track IBM standard diskettes. The SAGE II/IV provides a compatibility mode to this format but does not attempt to automatically distinguish which format is being used.

SAGE 10 SECTOR

Note: SAGE also provides a 10 sector per track format allowing 1600 blocks of data. This generally works but does stretch the drive specifications. It is not recommended for a distribution format.

Softech Universal Media

Softech Microsystems has chosen a format to standardize their 5 1/4" diskette distribution. This format is currently compatible with the single side IBM 40 track selection.

X.08 RAM DISK CONFIGURATION :

The RAM Disk Configuration brings up a preliminary menu which allows selection of one of the four possible RAM Disk devices.

A RAM Disk which has a non-zero Top specification greater than the base of the BIOS and system tables will be automatically disabled without any warning message. Therefore, if a RAM Disk access returns a non-existent device error, it may be configured but require more space than is available.

```
RAM Disk Channel Selection

A - RAM Disk #1
B - RAM Disk #2
C - RAM Disk #3
D - RAM Disk #4

Select Menu item <CR exits, ! aborts>:
```

Selecting one of the RAM Disks displays:

```
          RAM Disk #1 Configuration

A - Base of RAM Disk (0 = disabled)   08C800
B - Top  of RAM Disk (0 = to BIOS)    000000
C - Initialize RAM Disk                  Off
D - Access Control

Select Menu item <CR exits, ! aborts>:
```

The "Top of RAM Disk" value for the last enabled RAM Disk may be left at zero to indicate that all remaining memory to the base of the BIOS should be used.

● Initialize RAM Disk Flag

When turned on, this flag is used by the MU.BOOTEXT.CODE program to specify the initialization of the RAM Disk device. See the section on Bootstrap options for more details. Initialization of the first RAM Disk causes files to be copied to the RAM Disk from the BIOS boot device. Other RAM Disks will have their directories zeroed but will not be initialized with files.

X.09 PARALLEL PORT CONFIGURATION :

The Parallel Printer Port configuration is similar to the options under SAGE4UTIL. The buffer size is configurable from 2 to 32767 bytes. The default value is the same as single user systems, 256 bytes. The normal Multi-User Access parameters are available for configuration.

```
                Parallel Printer Configuration Control

A - Printer mode                    Parallel port with interrupts
B - Output Buffer Size                                        256
C - Polling attempts before delay                             500
D - Delay before re-polling                                   236
E - Linefeed after carriage return                             On
F - Access Control

Select Menu item <CR exits, ! aborts>:
```

● Printer mode

The user can select these options for the Parallel Printer port.

```
          Printer Port Assignment

A - Parallel port with interrupts       *
B - Parallel port with scheduled polling
C - Disabled

Select Menu item <CR exits, ! aborts>:
```

X.10 WINCHESTER ACCESS :

Access to the Winchester drives by each partition is controlled by this menu. Each Winchester partition may be configured separately as it is treated as an independent device.

```
Winchester #1 Partition Access

A - Drive partition #0
B - Drive partition #1
C - Drive partition #2
D - Drive partition #3
E - Drive partition #4
F - Drive partition #5
G - Drive partition #6
H - Drive partition #7
I - Drive partition #8
J - Drive partition #9
K - Drive partition #10
L - Drive partition #11
M - Drive partition #12
N - Drive partition #13
O - Drive partition #14
P - Drive partition #15

Select Menu item <CR exits, ! aborts>:
```

X.11 TIME ADJUSTMENT :

The time Adjustment factor is only important if the SAGE is to be left running without reset or power down for a period of over two weeks, and if an accurate time value is required by an applications program.

```
          Time Adjustment

A - Seconds of time adjustment    0
B - In number of days             0

Select Menu item <CR exits, ! aborts>:
```

X.12 LOW LEVEL CONFIGURATION :

The Low Level Configuration parameters for the floppy and Winchester drivers are similar to those available under SAGE4UTIL.

```
      Low Level Configuration

A - Left  Floppy Details
B - Right Floppy Details
C - Shared Floppy Details
D - Winchester #1
E - Winchester #2
F - Winchester #3
G - Winchester #4
H - Number of Semaphores      32
I - Intercept Exceptions      On
J - Load Terminal Emulator

Select Menu item <CR exits, ! aborts>:
```

● Floppy Configuration

The 'A' or 'B' selection for left or Right floppy configuration will bring up an alternate menu which allows changing low level parameters of the floppy disk driver. Note that most selections require knowledge of floppy disk recording protocols and possibly controller information. The Low Level parameters of the Floppy configuration are rarely changed from the factory setup unless the user is attempting to read or write to a non-standard format. The alternate menu is shown below:

```
Low Level Left Floppy Configuration

A - Number of sides              2
B - Number of cylinders         80
C - Sectors per track            8
D - Bytes per Sector           512
E - Retries                      7
F - Motor on delay factor     7D00
G - Data length                255
H - Gap 3 parameter             42
I - Gap 3 for format            80
J - Pattern for format         229
K - Skew for format              0
L - IBM track format           Off
M - NCI 10 sects/trk           Off
N - Read 48 on 96 TPI          Off
O - Read after write            On
P - Ignore errors              Off

Select Menu item <CR exits, ! aborts>:
```

'A' asks for 0, 1 or 2 sides for the diskette. Zero should be specified if the drive is not equipped. This will return an early error without having to go through a timeout process.

'B' asks for the number of cylinders on the diskette. Cylinders and tracks are often used in the same context. A cylinder represents a head position which may access a track on each side of a double sided diskette.

'C' asks for the number of sectors per track. Typical values are 8 for 512 byte sectors or 16 for 256 byte

sectors. Note that the Gap 3 parameter and the Gap 3 for formatting also must be modified for a specific Sectors per Track and Bytes per Sector combination. Also the density selection interacts with all these parameters.

'D' asks for the number of bytes per sector. Typically this is 512 for 8 sectors per track or 256 for 16 sectors per track. Note that the Gap 3 parameter and the Gap 3 for formatting also must be modified for a specific Bytes per Sector, Sectors per Track and density combination. Note also that selections of Bytes per Sector above 512 are not currently supported.

'E' asks for the number of retries. This question should be answered with the number of retries that the diskette driver should make before returning an error. The system is normally shipped with 3 retries specified but this may be increased to attempt to access data on a marginal diskette.

'F' asks for the Motor on delay factor. This should be given in hexadecimal as the number of 1/64000 ths of a second for the delay.

'G' asks for Data Length. This parameter is only used by the controller when the sector size is less than 256 bytes per sector. In these cases the Data Length is the number of bytes per sector (typically 128). For all other cases the Data Length is normally set to 255.

'H' asks for the Gap 3 parameter (in decimal). This parameter is required by the controller for Read and Write commands to avoid the splice point between the data field and the ID field of contiguous sectors. The value depends on the combination of Bytes per Sectors, Sectors per Track, and density selection. Suggested values from the controller documentation are:

Density	BPS	SPT	Gap 3	Gap 3 for format
Single	128	18	7	9
	128	16	16	25
	256	8	24	48
	512	4	70	135
Double	256	18	10	12
	256	16	32	50
	512	8	42	80

'I' asks Gap 3 for formatting. Values for this parameter are used when formatting a diskette and are contained in the Gap 3 table under selection 'H'.

'J' asks for a Pattern for formatting. This value (0 to 255) is written into each data byte during the formatting of a diskette. The normal default value is 229 (E5 hex). Note that certain operating systems (such as CP/M) may require a specific format pattern.

'K' asks for a Skew factor. This value is normally zero. The 10 sector per track formats generally specify a two sector skew to be formatted onto the diskette. This improves the performance when accessing the diskette over track boundaries. The IBM format is normally identical to the standard SAGE 8 sector per track format as the difference is only in where the data is written. On 10 sector per track diskettes with IBM format, the skew is reversed for head one because the tracks are accessed in decreasing order. The special NCI format also requires use of the IBM format selection.

'L' sets the IBM track format compatibility on or off. For double sided diskettes, data is stored in ascending track order on side zero and then back in descending track order on side one. The normal SAGE II/IV method is to store data on side zero and then side one of each track before stepping the head to the next cylinder.

'M' will set the option to use a special sector numbering scheme developed by Network Consulting Incorporated. This scheme was implemented by NCI for their BIOS on the IBM Personal Computer. It allows their software to automatically distinguish between their 10 sector per track diskettes and the normal 8 sector per track IBM standard diskettes. Sectors are numbered from 9 to 18 (except the first sector on the device which is numbered 1).

'N' controls the feature which allows reading of 48 TPI diskettes on a 96 TPI drive. Note that writing from a 96 TPI drive to a 48 TPI diskette is not allowed because the 48 TPI drives cannot read the data. The 96 TPI drive is stepped two physical tracks for every track normally requested by the driver.

'O' controls the option to perform a read of information from sectors which have just been written. This Read after Write feature verifies that the controller can read back the information without detectable errors. This option slows down the writing process but should be left enabled to insure valid operation.

'P' controls the option to ignore errors from the floppy controller. The controller errors should never be ignored in normal operation. This option is only provided to allow a head alignment procedure to be performed using a special alignment diskette which contains unreadable data. The driver must continue to read the diskette so that signals may be observed with test equipment, even though the controller is detecting errors.

Shared Floppy Configuration

These parameters are shared by the two floppies as they are set in the same controller. This means that the change will effect BOTH floppies and care should be taken to verify that this is really what the user wants.

```
Shared Low Level Floppy Configuration

A - Step Time (msec)          6
B - Head load                 3
C - Head unload time          1
D - Double Density           On

Select Menu item <CR exits, ! aborts>:
```

'A' asks for the Step Rate. This is the number of milliseconds allowed between head step pulses by the controller. The value may be varied between 2 to 32 milliseconds (only even number values are supported).

Changing 'B' or 'C' is not advised. These parameters are set by the factory and should not be modified.

'D' will set the density option. The drives provided on the SAGE II/IV will normally be used in the double density mode. The single density option (Off) should only be required to access data from another system which provides only single density drives. The Sectors per Track, Bytes per Sector, and Gap 3 values must all be coordinated with the density selection.

Winchesters Configuration

The low level Winchester configuration allows the format to be changed for different types of drives. Normally this is only done by the factory or service centers.

```
                    Winchester #1 Parameters

A - Number of Cylinders     306    L - High Read Counter      1140
B - Bytes per Sector        512    M - Header Count           400
C - Step Time               250    N - Number of Heads          6
D - Slew Time                20    O - Number of Retries       10
E - Step Counter             16    P - Sectors per Track       19
F - Head Settle Time        840    Q - Select Bit               1
G - Cylinder for Precomp    306    R - Write Sync Bit        4000
H - Special types             0    S - Extra Head Settle        0
I - Test flags                0    T - Rd after Wt Retries      4
J - Track for shipping      329    U - Write Cycle Retries      2
K - Low Read Counter       1100

Select Menu item <CR exits, ! aborts>:
```

● Number of Semaphores

The number of Semaphores used in the system can be set by the System Manager. Refer to the section on Global Semaphores page 395.

● Intercept Exceptions

The "Intercept Exceptions" flag allows the system to be configured to prevent certain processor detected error conditions from stopping the system and entering the Debugger. If the flag is set ON, then the Bus Error, Address Error, Illegal Instruction (including 0AxxxH and 0FxxxH instructions), Zero Divide, CHK, TRAPV, and Privilege Violation TRAPs will put up an appropriate error message and force the individual User to reboot. When an Exception is intercepted there in no method provided for entering the Debugger. If Debugger access is required to look at a problem, the "Intercept Exceptions" flag may be turned off to allow the normal direct entry into the Debugger on an exception TRAP. Continuation from this point is impossible and the whole Multi-User system must be restarted.

● Load Terminal Emulator

The Multi-User configuration file will hold up to six different Terminal Emulators. If a terminal is required which is not present in the Multi-User configuration file, a new Terminal Emulator will have to be loaded into the file. Terminal Emulators are shipped in a file called TRMDEF.DATA.

When the "Load Terminal Emulator" entry is selected, the system will prompt :

```
Terminal Emulator Configuration File:
```

At this point specify the device and file name (example #4:TRMDEF) of the emulator library. After reading the file a menu will be displayed:

```
Emulator Storage Slots

A - TELEVIDEO 925
B -   FREEDOM 100
C -         VT 102
D -  QUME QVT 102
E -         VT 52
F -         EMPTY

Select Menu item <CR exits, ! aborts>:
```

This menu shows the six storage positions in the Multi-User configuration file. Choose one of the positions to be overwritten with a new Terminal Emulator.

Another menu titled "Terminal Emulator Selection" will display up to 26 possible emulators contained in the TRMDEF.DATA file, plus the possibility of "No Emulator".

```
          Terminal Emulator Selection

A - INITIAL                O - EMPTY
B - TELEVIDEO 925          P - EMPTY
C - FREEDOM 100            Q - EMPTY
D - VT 102                 R - EMPTY
E - QUME QVT 102           S - EMPTY
F - EMPTY                  T - EMPTY
G - EMPTY                  U - EMPTY
H - EMPTY                  V - EMPTY
I - EMPTY                  W - EMPTY
J - EMPTY                  X - EMPTY
K - EMPTY                  Y - EMPTY
L - EMPTY                  Z - EMPTY
M - EMPTY                  0 -   No Emulator
N - EMPTY

Select Menu item <CR exits, ! aborts>:
```

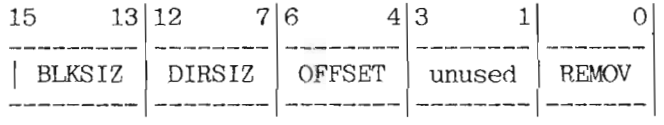
Choose one of the possibilities and then type a carriage return to move the selected Terminal Emulator from the TRMDEF.DATA file to the Multi-User configuration.

NOTE: If a terminal is required for which a Terminal Emulator does not exist in the TRMDEF.DATA file, see the section on the Terminal Emulator Configuration. Using the program MUTRMSET.CODE a new Terminal Emulator may be defined and stored into the TRMDEF.DATA file. Then the new Terminal Emulator may be loaded into the Multi-User configuration file.

X.13 AUXILIARY DEVICE INFORMATION :

The Auxiliary Device Information is not used for the p-System.

Under CP/M-68K, the auxiliary device information is used to represent certain CP/M disk drive characteristics. Only the Floppy and Winchester field are used. For historical reasons, RAM disk drive characteristics are stored in the Operating System Information. This is the format of an auxiliary device information field:



BLKSIZ represents the allocation block size of the CP/M disk. This is encoded as follows:

0 = 1024 bytes
1 = 2048 bytes
2 = 4096 bytes
3 = 8192 bytes
4 = 16384 bytes

DIRSIZ represents the number of directory entries on the CP/M disk. DIRSIZ * 32 gives the number of directory entries.

OFFSET is the track offset of the directory. This is the number of reserved tracks before the CP/M directory starts.

REMOV if non-zero, marks the disk as removable. This should be normally set for floppy drives, but not set for non-removable hard drives.

Menu for Auxiliary Device Information

```
  Auxiliary Device Information

A - Keyboard                0000
B - Terminal                0000
C - Left Floppy             0000
D - Right Floppy            0000
E - Parallel Printer        0000
F - Remote Input            0000
G - Remote Output           0000
H - RAM Disk #1             0000
I - RAM Disk #2             0000
J - RAM Disk #3             0000
K - RAM Disk #4             0000
L - Extra Serial Port #1    0000
M - Extra Serial Port #2    0000
N - Extra Serial Port #3    0000
O - Extra Serial Port #4    0000
P - Winchester #1
Q - Winchester #2
R - Winchester #3
S - Winchester #4

Select Menu item <CR exits, ! aborts>:
```

Winchester Aux Device Info

Selecting "P" - "S" will allow the user to set the Winchester Auxiliary Device information. Each partition on a Winchester drive is considered a different device.

```
Winchester #1 Auxiliary Device Information

A - Partition  0   0000
B - Partition  1   0000
C - Partition  2   0000
D - Partition  3   0000
E - Partition  4   0000
F - Partition  5   0000
G - Partition  6   0000
H - Partition  7   0000
I - Partition  8   0000
J - Partition  9   0000
K - Partition 10   0000
L - Partition 11   0000
M - Partition 12   0000
N - Partition 13   0000
O - Partition 14   0000
P - Partition 15   0000
Select Menu item <CR exits, ! aborts>:
```

XI MULTI-USER SYSTEM CONSIDERATIONS :

This section presents some of the more detailed topics which may be necessary to understand while planning and making a custom Multi-User configuration.

XI.01 USER CHANNEL MAP :

Each User Task has its own BIOS Channel Map which associates Logical user channel numbers with Physical devices. Each user's accessability to the hardware devices and Winchester disk partitions is controlled by the BIOS Channel Map. Page 35 has a table of the physical device numbers. Note that there are differences in some of the physical device numbers between the Single- User and Multi-User BIOS routines. Page 93 has a table of how those numbers relate to p-System volumes.

XI.02 DEVICE ACCESS CONTROL :

User access to a hardware device is also controlled through various masks that are specified for each device. This is a secondary level of control. For the device to be visible to the user at all, it must be configured in the user's BIOS Channel Map. Control of a device may also be obtained via a program without doing an access to the device.

Access Philosophy

The SAGE Multi-User Environment is organized in a different manner than many other Multi-User environments. Most Multi-User systems integrate the operating system and associated file system into the Multi-User scheme. This allows users to access a common file system but typically restricts them to run a dedicated operating system.

The SAGE Multi-User Environment allows users to run separate copies of potentially different operating systems. In general, the operating systems think that they are the only ones accessing their visible devices. This can cause problems if two users are allowed access to the same device. For instance, one user's operating system could read in the device's directory and begin to make a change. At the same time another user could read in the same directory, make a change, and write the directory back before the first user was complete. Then when the first user writes back the directory, the second user's directory changes are over-written.

Without actually changing the operating systems to account for other users, the most secure protection against this problem is to not make any user's devices available to any of the other users. In some applications this is acceptable, but most systems desire some sharing of information between the users. In many environments users would like to share a set of common system files. This can safely be done if the device is allowed read only access by the users. One user in charge of maintaining the device could be allowed write access to make modifications when he is sure that no one will be accessing the device.

If users wish to transfer files to each other, the best method is to establish a common device which has both read and write access for all users. The SAGE Multi-User Environment supports an exclusive control scheme with a timed release. The first user to access the device gets exclusive control of the device until he has not accessed the it for a specified period of time. Typically, simple

file transfers to a device will be completed with only short intervening delays. If a timeout of several seconds is specified, the transfer will always be completed before the exclusive control of the device is automatically released.

Typical data processing environments will want users to access a common set of data files from a group of associated programs. In this environment, read and write access is granted for each user to the device containing the data files. A set of Global Semaphores is available for the data processing programs to coordinate their access to the common data files. The Global Semaphores are more fully described in the section on Advanced Multi-User Features. It should be noted that only programs which contain explicit code to handle the SAGE Global Semaphores will be able to safely share data in this manner.

Access Control Implementation

Each hardware device (including RAM Disks and Winchester disk partitions) provides a method for controlling the access rights to the device for each User Task. Note that for the device to be visible to the User Task at all it must also be in the User's BIOS Channel Map. Access control of a device may also be obtained via a program without doing an access to the device.

The Multi-User Environment maintains four masks and a timeout count for each device. Each mask is a word containing a bit for each of the User Tasks. The mask word's bit numbers correspond to the User Task numbers. The first mask is the 'Read Allowed' mask. Bits which are set in this mask allow the corresponding User Task read access to the device. The second mask is the 'Write Allowed' mask. Bits in this mask allow User Tasks to write to the device.

The third mask is the 'Exclusive Control' mask. When exclusive control for a User Task is indicated with a bit set, the read or write request will not be processed unless no other User Task has control of the device. The first Task that accesses a device automatically gets control of the device. Two methods exist for releasing control of a device. One method is via program requests through the Multi-User BIOS. The second method is through an optional timeout since the last access. This method is typically used for shared serial channels or the parallel printer channel to reduce the chance that two users will attempt to access a channel at the same time.

When a User Task attempts to output to a channel which is already occupied and requires exclusive control, the system may be configured to hang and wait for the channel to become available or to return with an error. The 'Hang and Wait' mask bits control this feature. Generally, the user may break out of the Hang and Wait condition, wait for the timeout from the last access of the other user, or have the other user run a routine to release his exclusive control.

The timeout value may be specified in one second intervals. A value of zero indicates that no timeout exists and the only method for releasing exclusive control is through a special program request.

The access control selection for each device allows viewing and control of the four masks by the complete mask or by the individual bits for each task. Following is the 'Access Control' menu for each device in the MU.UTIL.CODE program.

```
                    Access Control

A - Timeout threshold          5        L - User #6
B - Read allowed mask       FFFF        M - User #7
C - Write allowed mask      FFFF        N - User #8
D - Exclusive control mask  0000        O - User #9
E - Hang & Wait mask        0000        P - User #10
F - User #0                             Q - User #11
G - User #1                             R - User #12
H - User #2                             S - User #13
I - User #3                             T - User #14
J - User #4                             U - User #15
K - User #5

Select Menu item <CR exits, ! aborts>:
line count error, count= 15
```

Selection of a specific User will display a menu which shows the boolean condition of the user's bit in each of the four masks.

```
User #1 Access Control

A - Read allowed          On
B - Write allowed         On
C - Exclusive control     Off
D - Hang & Wait           Off

Select Menu item <CR exits, ! aborts>:
```

● Printer access

A printer can be connected to either the parallel port or one of the serial ports. The Access Control menu for the port in MU.UTIL should be set up so as to make the printer available to the users in the most reasonable manner.

Generally all users are allowed access to the printer with the "Write Allowed" flags set ON. The "Exclusive control" flags must also be set ON to prevent user printouts from becoming intermixed. There are two ways to arbitrate use of the printer.

One way is to set the "Hang and Wait" flags for all users ON and set a "Timeout threshold" of several seconds. Under this scheme, if one user has the printer and another user attempts to send text to it, the second user will "Hang and Wait" until the first user's printout is finished and the timeout threshold is passed. The second user can break out of the "Hang and Wait" state by typing the soft break key. (Generally the CTRL @ under the p-System or CTRL C under CP/M-68K.)

A second way to control printer use is to set all the "Hang and Wait" flags OFF. If the printer is busy when another request occurs, the second user will get an I/O error "Device not equipped". If the user is running an application program, it must be able to handle this error.

Floppy Disk Access

Access to the Floppy disk(s) can be set up much the same as described for the printer. Users will have to show some consideration for each other, such as NOT leaving their disks in the drives and making sure that the disk they are writing to is their own.

Winchester Partition Access

Most of the System Manager's effort will be in the area of partitioning the disk. The section on "Planning Disk Partitions" in the on page 40 offers guidelines for determining disk partition size and how to specify the partition using WFORMAT.

Note that if there is a good chance of having one more user on the system later on, it is best to build a partition for the extra User Task in at the start.

In some configurations, it may be handy to set up a partition as a "Mail drop" where all users have access to the partition and can transfer files back and forth. Because this area will not be well protected, users should be warned not to leave any critical information there. This area should be set up much like the Printer with the "Exclusive Control", a " timeout", and "Hang and Wait" flags set ON.

XI.03 MEMORY ASSIGNMENT :

Allocation of memory in the system is a major consideration in planning. This planning should really take place **before** buying a system to insure that enough memory is available for all users. A general rule of thumb is to allow 140K per user as this is the maximum amount used by the p-System.

In addition, the Multi-User BIOS will use about 36K for the SAGE II and 50K for the SAGE IV, as well as a fixed overhead of about 1K per user task. Using the terminal emulator will take about 4K per user.

Each User Task must be assigned one or two areas of memory in which to run. The first area is used for the user's program and data. The second area is for the System Stack space which must be separate for each user. Note that the registers for each user are left on his private System Stack while he is inactive. Although the p-System never uses the System Stack directly, other environments such as CP/M-68K will use the System Stack. For the p-System, an area must be reserved separately for the System Stack. Other environments may set up the System Stack to be within their normal program/data area.

The p-System single user configurations have 140 Kbytes (23000 Hex) allocated for the p-System interpreter, data area, and code pool. The Multi-User p-System bootstrap automatically gets its memory bounds from the BIOS and allocates the users memory partition between the data area, interpreter, and code pool. The data area is given a full 64 Kbytes. The interpreter is given 12 Kbytes. The remaining space (up to 64K) is allocated to the code pool.

Cutting the user memory allocation back to 120K would leave enough code pool (44K) to run most p-System programs and would allow seven Users in a 1 Meg RAM system.

The System Stack should have at least 256 bytes allocated. This allows for interrupt routine processing (possibly nested levels) as well as for the storage of user registers

while the user is inactive. Because the p-System never modifies the System Stack Pointer, the System Stack area must be allocated outside of the user's p-System memory space. The initial System Stack pointer must be specified in the configuration information. Since the System Stack uses memory below the initial stack pointer, an area of memory may be specified by setting up the initial System Stack Pointer 256 bytes above the User Task's memory space.

The typical six user p-System memory allocation is shown in the following table.

User	Base Address	Top Address	System Stack
0	0	0	0
1	400 hex	23400 hex	23500 hex
2	23500 hex	46500 hex	46600 hex
3	46600 hex	69600 hex	69700 hex
4	69700 hex	8C700 hex	8C800 hex
5	8C800 hex	AF800 hex	AF900 hex
6	AF900 hex	D2900 hex	D2A00 hex

Note that the allocation for User Task #0 is reserved for special system routines and its configuration should not be modified.

No error checking is performed by MU.UTIL to insure that the user areas do not overlap the BIOS and its tables. If a User's Task space overlaps the BIOS system tables, an error will be output when the Multi-User Environment is bootstrapped. This error is of the form:

```
Not enough memory, base of BIOS @ xxxxxxxx
```

where xxxxxxxx is the hexadecimal base of the BIOS and System tables. Note that the allocation of space is dynamic and therefore the highest User number should use the highest memory address for the check to work correctly. If the error is detected before the last User is allocated, the size of the system tables may grow and continue to lower the BIOS base from the value displayed.

If it is necessary to determine the actual base of the BIOS and system tables, the base is stored at location 200H in memory. This may be displayed by allowing BREAK entry into the Debugger and using the DM command to display location 200H.

XI.04 RAM DISK MEMORY :

Typically, one RAM Disk area is configured in a Multi-User Environment to allow fast access to key system programs. Up to four RAM Disk areas may be configured, however this tends to break memory into too many small areas to be useful for most application environments. When specifying RAM Disk areas, the Base and Top addresses of each area must be specified for all but the last assigned area. The last assigned area (not necessarily the fourth RAM Disk) can have its Top memory address set to zero, indicating that the system should set the Top to an address just below the BIOS and System Table area at the top of memory.

Note that the Base of the first assigned RAM Disk area should be allocated at the the top of the last user area, including its System Stack. For the example six user p-System allocation shown previously, this would mean that the first RAM Disk should be assigned at D2A00 hex.

Methods exist to initialize the directories (p-System only) on RAM Disks and to copy files to the first RAM Disk. Users may be bootstrapped from RAM Disk. These options are all covered in the following section on Bootstrapping options.

XI.05 MULTI-USER RECOMMENDATIONS :

The SAGE Multi-User Environment is very flexible. This means that some combinations of configurations may be strange and awkward to use. Before trying a drastically different configuration, review the standard configurations shipped with the system and the guidelines presented in the earlier sections. The System Manager should have a solid understanding of all features in the system and how they interact before experimenting with unusual configurations.

● Preferred Operation

The SAGE II/IV product line does not contain memory management hardware to prevent User Tasks from damaging each other. Therefore some operating system environments or

system usages may be more appropriate than others for running under the Multi-User system.

The p-System and the Volition Modula 2 environments use interpreters and good error catching compilers. Program development under these systems is generally safe unless the user is working with non-standard techniques or assembly language.

The CP/M-68K environment is usable under the Multi-User system to run known working programs and applications but may not be appropriate for testing new programs. The CP/M C compiler does not perform as much error checking as Pascal and Modula 2, and the generated code has total addressing capability to all the CPU's memory if an error occurs.

- **Security**

The SAGE Multi-User Environment has been designed as a friendly environment where users are not expecting to hide information from each other. Although information is normally secure from non-programmers, the lack of memory management and the capability to load or develop programs will allow a knowledgeable user to access any data in the system.

It is possible, however, to develop a turnkey type system which never allows a person access to the normal operating system facilities. Under such an environment even a knowledgeable programmer would not be able to load or create a program to access privileged data. Of course the computer itself must be secured to keep someone from completely resetting it and taking over.

XI.06 STANDARD CONFIGURATIONS :

All the configuration information required for starting the SAGE Multi-User Environment is contained in a file. Typically this file is called MU.CONFIG and if it is present on the booting device, the system will automatically be initialized. If the MU.CONFIG file is not found, then the system will prompt for the name of a configuration file to use. This allows the Multi-User Environment to be easily started in a variety of configurations. Multi-User Environment configuration files are maintained by a utility program called MU.UTIL.CODE. This program is completely described in this manual, see page 279 .

Several typical configuration files are provided which can be used as a starting point for developing a custom configuration. All of the files have memory allocations appropriate for use with the p-System. All serial ports are configured for DIP Switch specified baud rate, even parity, one stop bit, and eight data bits.

One RAM Disk is defined which uses any remaining memory not allocated to the Users or the BIOS. The RAM Disk is made visible to all Users as logical device #11. Each User's system device is only visible to that user.

● Standard MU Installations

The BUILD disk loads a SYSTEM.STARTUP file (MU.INSTAL.CODE) which gives the user the option of installing on of the six standard multi-user configurations. Each configuration has a set of 3 files that are necessary to install it: a "script" file, a configuration file, and a "partial" formatinfo file.

Once a selection is made from the MU.INSTAL menu, the correct MULTI.xxxx script file is run. It calls a program called MERGE which takes the Device and Bad Track information from the factory built FORMATINFO.TEXT file, and combines it with the correct partition information from the MF.x.x file. The result is a file called MU.FORMAT.TEXT

which is loaded into the Map area with WFORMAT. The directory areas of all of the partitions are then zeroed with a program called ZAP.

The files are:

Users	Disk Size	Script File	Config File	Formatinfo File
2	Floppy	MULTI.ONE	MC.0.2	MF.0.2
4	12 Meg	MULTI.TWO	MC.12.4	MF.12.4
5	18 Meg	MULTI.THRE	MC.18.5	MF.18.5
6	18 Meg	MULTI.FOUR	MC.18.6	MF.18.6
5	40 Meg	MULTI.FIVE	MC.40.5	MF.40.5
6	40 Meg	MULTI.SIX	MC.40.6	MF.40.6

The following pages detail the size and configuration of each of the six SAGE Multi-User programs on the BUILD diskette.

The partition table for the SAGE IV systems shows how the partitions are assigned to the users and their sizes.

The device table lists the p-System Volumes that each user will be able to access after the BUILD program has been completed. The partition number is shown after the hard disk device number in parenthesis: 9(x).

> NOTE: In the following tables labeled 'Physical Devices Accessible to Users' the p-System Volumes are shown to be equivalent to Logical Device Numbers. For most devices, they are equivalent. However, for #1: and #2: there is a technical difference (see page 35) in that the p-System Volumes for both channels are used for either input or output.

MULTI.ONE

SAGE II - With two 640K floppy drives and 512K of RAM. A parallel printer may be used with this configuration.

This table lists the p-System Volumes that each user will be able to access after the BUILD program has been completed.

Physical Devices Accessible to Users

p-SYSTEM VOLUMES	DEVICE DESCRIPTION	USER 1 device	USER 2 device
# 1:	Terminal Keyboard	1	7
# 2:	Terminal Display	2	8
# 4:	Left Floppy Drive	4	5
# 5:	Right Floppy Drive	5	4
# 6:	Parallel Printer	6	6
#11:	RAM Disk	11	11

MULTI.TWO

SAGE IV - With one or two 640K floppy drives, one fixed 12 megabyte Winchester hard drive, and 1Mb of RAM Disk. This configuration may be used with BOTH a parallel printer and either a serial printer or a modem.

Partition Assignments

PARTITION	BLOCKS	NAME	PURPOSE
1	1292	SINGLE	Single User Boot Area
2	456	MULTI	Multi User Boot Area
3	1292	USER1	User 1 Boot and System Area
4	1292	USER2	User 2 Boot and System Area
5	1292	USER3	User 3 Boot and System Area
6	1292	USER4	User 4 Boot and System Area
7	3667	DATA1	User 1 Data Files
8	3667	DATA2	User 2 Data Files
9	3667	DATA3	User 3 Data Files
10	3667	DATA4	User 4 Data Files
11	1292	SCRATCH	Common Area for Temporary Use

Physical Devices Accessible to Users

P-SYS VOLUMES	DEVICE DESCRIPTION	USER 1 device	USER 2 device	USER 3 device	USER 4 device
# 1:	Terminal Keyboard	1	13	15	17
# 2:	Terminal Display	2	4	16	18
# 4:	Left Floppy Drive	4	4	4	4
# 5:	Right Floppy Drive	5	5	5	5
# 6:	Parallel Printer	6	6	6	6
# 7:	Remote In (Serial)	7	7	7	7
# 8:	Remote Out (Serial)	8	8	8	8
# 9:	Winchester (Partition)	9(3)	9(4)	9(5)	9(6)
#10:	Winchester (Partition)	9(7)	9(8)	9(9)	9(10)
#11:	RAM Disk	11	11	11	11
#12:	Winchester (Partition)	9(11)	9(11)	9(11)	9(11)

MULTI.THREE

SAGE IV - With one or two 640K floppy drives, one fixed 18 megabyte Winchester hard drive, and 1Mb of RAM Disk. This configuration may be used with BOTH a parallel printer and either a serial printer or a modem.

Partition Assignments

PARTITION	BLOCKS	NAME	PURPOSE
1	1292	SINGLE	Single User Boot Area
2	456	MULTI	Multi User Boot Area
3	1292	USER1	User 1 Boot and System Area
4	1292	USER2	User 2 Boot and System Area
5	1292	USER3	User 3 Boot and System Area
6	1292	USER4	User 4 Boot and System Area
7	1292	USER5	User 5 Boot and System Area
8	4959	DATA1	User 1 Data Files
9	4959	DATA2	User 2 Data Files
10	4959	DATA3	User 3 Data Files
11	4959	DATA4	User 4 Data Files
12	4959	DATA5	User 5 Data Files
13	1292	SCRATCH	Common Area for Temporary Use

Physical Devices Accessible to Users

p-SYS VOLUMES	DEVICE DESCRIPTION	USER 1 device	USER 2 device	USER 3 device	USER 4 device	USER 5 device
# 1:	Terminal Keyboard	1	13	15	17	19
# 2:	Terminal Display	2	14	16	18	20
# 4:	Left Floppy Drive	4	4	4	4	4
# 5:	Right Floppy Drive	5	5	5	5	5
# 6:	Parallel Printer	6	6	6	6	6
# 7:	Remote In (Serial)	7	7	7	7	7
# 8:	Remote Out (Serial)	8	8	8	8	8
# 9:	Win. (Partition)	9(3)	9(4)	9(5)	9(6)	9(7)
#10:	Win. (Partition)	9(8)	9(9)	9(10)	9(11)	9(12)
#11:	RAM Disk	11	11	11	11	11
#12:	Win. (Partition)	9(13)	9(13)	9(13)	9(13)	9(13)

MULTI.FOUR

SAGE IV - With one or two 640K floppy drives, one fixed 18 megabyte Winchester hard drive, and 1Mb of RAM Disk. A parallel printer may be used with this configuration.

Partition Assignments

PARTITION	BLOCKS	NAME	PURPOSE
1	1292	SINGLE	Single User Boot Area
2	456	MULTI	Multi User Boot Area
3	1292	USER1	User 1 Boot and System Area
4	1292	USER2	User 2 Boot and System Area
5	1292	USER3	User 3 Boot and System Area
6	1292	USER4	User 4 Boot and System Area
7	1292	USER5	User 5 Boot and System Area
8	1292	USER6	User 6 Boot and System Area
9	3933	DATA1	User 1 Data Files
10	3933	DATA2	User 2 Data Files
11	3933	DATA3	User 3 Data Files
12	3933	DATA4	User 4 Data Files
13	3933	DATA5	User 5 Data Files
14	3933	DATA6	User 6 Data Files
15	1292	SCRATCH	Common Area for Temporary Use

Physical Devices Accessible to Users

p-SYS VOLUMES	DEVICE DESCRIPTION	USER 1 device	USER 2 device	USER 3 device	USER 4 device	USER 5 device	USER 6 device
# 1:	Terminal Keyboard	1	13	15	17	19	7
# 2:	Terminal Display	2	14	16	18	20	8
# 4:	Left Floppy Drive	4	4	4	4	4	4
# 5:	Right Floppy Drive	5	5	5	5	5	5
# 6:	Parallel Printer	6	6	6	6	6	6
# 9:	Win. (Partition)	9(3)	9(4)	9(5)	9(6)	9(7)	9(8)
#10:	Win. (Partition)	9(9)	9(10)	9(11)	9(12)	9(13)	9(14)
#11:	RAM Disk	11	11	11	11	??	??
#12:	Win. (Partition)	9(15)	9(15)	9(15)	9(15)	9(15)	9(15)

MULTI.FIVE

SAGE IV -With one or two 640K floppy drives, one fixed 40 megabyte Winchester hard drive, and 1Mb of RAM Disk. This configuration may be used with BOTH a parallel printer and either a serial printer or a modem.

Partition Assignments

PARTITION	BLOCKS	NAME	PURPOSE
1	1292	SINGLE	Single User Boot Area
2	456	MULTI	Multi User Boot Area
3	1292	USER1	User 1 Boot and System Area
4	1292	USER2	User 2 Boot and System Area
5	1292	USER3	User 3 Boot and System Area
6	1292	USER4	User 4 Boot and System Area
7	1292	USER5	User 5 Boot and System Area
8	11723	DATA1	User 1 Data Files
9	11723	DATA2	User 2 Data Files
10	11723	DATA3	User 3 Data Files
11	11723	DATA4	User 4 Data Files
12	11723	DATA5	User 5 Data Files
13	10394	SCRATCH	Common Area for Temporary Use

Physical Devices Accessible to Users

p-SYS VOLUMES	DEVICE DESCRIPTION	USER 1 device	USER 2 device	USER 3 device	USER 4 device	USER 5 device
# 1:	Terminal Keyboard	1	13	15	17	19
# 2:	Terminal Display	2	14	16	18	20
# 4:	Left Floppy Drive	4	4	4	4	4
# 5:	Right Floppy Drive	5	5	5	5	5
# 6:	Parallel Printer	6	6	6	6	6
# 7:	Remote In (Serial)	7	7	7	7	7
# 8:	Remote Out (Serial)	8	8	8	8	8
# 9:	Win. (Partition)	9(3)	9(4)	9(5)	9(6)	9(7)
#10:	Win. (Partition)	9(8)	9(9)	9(10)	9(11)	9(12)
#11:	RAM Disk	11	11	11	11	11
#12:	Win. (Partition)	9(13)	9(13)	9(13)	9(13)	9(13)

MULTI.SIX

SAGE IV - With one or two 640K floppy drives, one fixed megabyte Winchester hard drive, and 1Mb of RAM Disk. A parallel printer may be used with this configuration.

Paritition Assignments

PARTITION	BLOCKS	NAME	PURPOSE
1	1292	SINGLE	Single User Boot Area
2	456	MULTI	Multi User Boot Area
3	1292	USER1	User 1 Boot and System Area
4	1292	USER2	User 2 Boot and System Area
5	1292	USER3	User 3 Boot and System Area
6	1292	USER4	User 4 Boot and System Area
7	1292	USER5	User 5 Boot and System Area
8	1292	USER6	User 6 Boot and System Area
9	9557	DATA1	User 1 Data Files
10	9557	DATA2	User 2 Data Files
11	9557	DATA3	User 3 Data Files
12	9557	DATA4	User 4 Data Files
13	9557	DATA5	User 5 Data Files
14	9557	DATA6	User 6 Data Files
15	10355	SCRATCH	Common Area for Temporary Use

Physical Devices Accessible to Users

p-SYS VOLUMES	DEVICE DESCRIPTION	USER 1 device	USER 2 device	USER 3 device	USER 4 device	USER 5 device	USER 6 device
# 1:	Terminal Keyboard	1	13	15	17	19	7
# 2:	Terminal Display	2	14	16	18	20	8
# 4:	Left Floppy Drive	4	4	4	4	4	4
# 5:	Right Floppy Drive	5	5	5	5	5	5
# 6:	Parallel Printer	6	6	6	6	6	6
# 9:	Win. (Partition)	9(3)	9(4)	9(5)	9(6)	9(7)	9(8)
#10:	Win. (Partition)	9(9)	9(10)	9(11)	9(12)	9(13)	9(14)
#11:	RAM Disk	11	11	11	11	11	11
#12:	Win. (Partition)	9(15)	9(15)	9(15)	9(15)	9(15)	9(15)

XI.07 DEVICE VISIBILITY DURING BUILD :

When building a Multi-User system, it is desirable to have all the pertinent disk partitions visible to the single-user system doing the initialization. This eliminates the necessity to run SAGE4UTIL to reconfigure access for each Multi-User disk partition. The xxxxxx.MISCINFO files on the BUILD diskette have been modified to raise the entry for First Subsidiary Volume, thus allowing all the physical Volumes from one drive to be on-line.

The standard xxxxxx.MISCINFO files on the MASTER diskette have the First Subsidiary Volume set to 13. The standard files have the lower default because each extra Volume requires some permanent memory from the p-System Data area. Given the standard BIOS Channel Map configuration the following table shows p-System Volume assignment to the partitions on a single Winchester system when the First Subsidiary Volume is increased.

Vol:	Partition	Vol:	Partition
# 9:	1	#18:	9
#10:	2	#19:	10
#12:	3	#20:	11
#13:	4	#21:	12
#14:	5	#22:	13
#15:	6	#23:	14
#16:	7	#24:	15
#17:	8		

XI.08 MULTI-USER INSTALLATION EXAMPLE :

This section gives a step-by-step example of how to install a 4 terminal, 5 task Multi-User system on a SAGE IV with one floppy drive and an 18 Meg Winchester hard disk. Two different types of printers will be connected; one for program listings and the other for letter quality type. A small RAM Disk area will be available.

This is a five task system. Usually, each task is assigned to one user. However, here there will be only 4 people using the system. One of the terminals will be set up as a shared terminal. This means that 2 of the user tasks are accessed from the same terminal, and in this case by the same person.

The Winchester can be divided into 15 different areas, each of which are called partitions. The 5 user tasks will each be assigned at least one partition. The partitions will be used for different purposes:

1. To hold the operating system files and a "boot" area.

2. As a work area where data and application programs can be kept.

3. As a shared "scratch" area which every user can access. This is used for passing files from user-to-user or for copying floppy diskettes.

4. As a shared database area. (Note there are restrictions on this type of use, refer to page 261.)

The major areas of Multi-User installation that will be covered in this example are:

1. Partitioning the disk
2. WFORMAT
3. MU.UTIL
4. Zeroing Partitions
5. Bootstrap Copy
6. Starting the New System

Example Environment

All of the people, equipment, and software of a system interact to form an "environment". This example sets up a Multi-User system tailored to the working environment at a hypothetical company called MSC (Medium Size Company).

Al, Julie, Nancy and Mac all work for MSC. Al, who is a programmer, is assigned the job of installing the Multi-User system on their new SAGE IV. Julie is a receptionist, who types letters for several executives. Nancy is the OEM account manager. Mac is Vice President of Marketing.

After reviewing the standard Multi-User systems provided by SAGE on the BUILD diskette, Al decides that none of these exactly meets their needs. He would like to use the shared terminal feature on his terminal. He needs two large areas for two different Research and Development programs that he is working on. Julie will only use her areas for writing letters and so will not need a great deal of disk space. Nancy and Mac want to share a database of client information. Al will not have access to this area. Julie will be able to read it but not change it. All users like the idea of the "Scratch" area where they can transfer files to each other.

Total Disk Size

First Al must know exactly how much disk space there is on the machine. Disk space is sized in blocks where each block is 512 bytes.

Due to the exacting process of manufacturing large Winchester disks, disk manufacturers cannot guarantee a perfect disk. Normally they guarantee that not more than a certain number of bad areas will be on the disk. The software "skips" around the bad areas using a process called "Bad Track Mapping".

An 18 Meg disk can have up to 4 bad tracks per head. It has 6 heads so the maximum number of bad tracks possible would be 24.

Al is going to be conservative and use for his calculations the recommended size which allows for the maximum number of bad tracks. Al refers to the User Block Column of the table on page 32 of this manual and finds that 34,409 blocks are available on an 18 Meg disk once the allowance for the Bad tracks are made. Al writes down **34,409** as the usable blocks.

> NOTE: The exact usable size for an individual Winchester hard disk and the bad track information can be found in the file called FORMATINFO.TEXT on the BUILD disk. (This total can also be found by executing WFORMAT and viewing the information loaded on the disk by the factory. SAGE does not recommend using this number.)

Example Partitions

Al now decides how many partitions each user needs. Up to 15 partitions can be assigned, as their SAGE has one hard disk. A partition for the Multi-User system itself is needed. Each user will have a system or boot partition. Each user, except for Julie, will also have a work partition. A scratch area and a database area are also needed. This makes a total of 12 partitions that must be assigned sizes.

Note that the first 19 blocks are set aside for the partition map. These have already been accounted for in the table Al is using, so he has **34,409** blocks to divide among 12 partitions.

The size of each partition must be a multiple of 19, the number of blocks on one Winchester track. As the "boot" partition for the Multi-User holds only a few files, Al makes its size 456 blocks.

Since the number of blocks on a SAGE floppy is 1280, the smaller boot partitions are assigned 1292 blocks, which is the closest value to 1280 that is a multiple of 19. About 4000 blocks makes a good sized work area, so AL assigns 3876 blocks to each user for this purpose. SCRATCH should be floppy sized so it is also allowed 1292. DATABSE takes the large amount left over: 8113 blocks.

Al assigns partition names similar to the name of the person who will use the partition to make an easy to use reference.

The final assignments are:

PARTITION	BLOCKS	NAME	PURPOSE
1	456	MULTI	Multi User Boot Area
2	1292	AL1	Al's first User Boot Area
3	3876	A1WORK	Al's first User work Area
4	1292	AL2	Al's second User Boot Area
5	3876	A2WORK	Al's second User work Area
6	3876	JULIE	Julie's Boot and Work Area
7	1292	NANCY	Nancy's User Boot
8	3876	NWORK	Nancy's Work Area
9	1292	MAC	Mac's User Boot
10	3876	MWORK	Mac's Work Area
11	8113	DATABSE	Shared Client Data Base
12	1292	SCRATCH	Common Area for Temporary Use

Example Assignment of Devices

Now that Al has decided on how to partition the disk, the other devices must be considered. This is a matter of deciding which user can use or "access" each device.

Each person is assigned one of the four SAGE terminals and a port to attach it to. The last serial port (Aux4) will not be used.

Two printers will be connected. Printer #1 will be a serial dot matrix printer connected to the MODEM port. Printer #2 will be a letter quality parallel printer connected to the PRINTER port. All users will be able to access both printers, but Al will mainly use PRINTER #1 while the rest will mainly use PRINTER #2.

Normally, all users are given access to the floppy drive so that they can backup their work and load new programs from the floppy.

Only Al's first user will have access to RAM Disk.

AL writes down the User names and what devices they will want to access:

USER	CONNECTOR	DEVICES	PARTITIONS
AL1	TERMINAL	Terminal #1 (shared)	2 -AL1
	--------	Floppy drive	3 -A1WORK
	MODEM	Printer #1	12 -SCRATCH
	PRINTER	Printer #2	
	--------	RAM Disk #1	
AL2	TERMINAL	Terminal #1 (shared)	4 -AL2
	--------	Floppy drive	5 -A2WORK
	MODEM	Printer #1	12 -SCRATCH
	PRINTER	Printer #2	
JULIE	AUX1	Terminal #2	6 -JULIE
	--------	Floppy drive	12 -SCRATCH
	MODEM	Printer #1	11 -DATABSE
	PRINTER	Printer #2	(Rd only)
NANCY	AUX2	Terminal #3	7 -NANCY
	--------	Floppy drive	8 -NWORK
	MODEM	Printer #1	12 -SCRATCH
	PRINTER	Printer #2	11 -DATABSE
MAC	AUX3	Terminal #4	9 -MAC
	--------	Floppy drive	10 -MWORK
	MODEM	Printer #1	12 -SCRATCH
	PRINTER	Printer #2	11 -DATABSE

Booting the BUILD disk

First Al must boot the system with a copy of the BUILD disk. He checks Switch 6 in GROUP A on the back of the SAGE IV and sets it down so that when he resets the system it will NOT boot to the hard disk but will instead boot to the PROM Debugger. He inserts the BUILD diskette into the left floppy drive (#4:) and resets the system. He answers the ">" prompt by typing "IF", initialize from floppy, to boot to the BUILD diskette. The standard MU selection menu appears, he selects the p-System . These steps are:

```
SCREEN SHOWS:                    YOU TYPE

>................................ IF
Multi-user selection menu........ down arrows <cr>
C(ommand.........................
```

Now Al is at the p-System command line.

Example FORMATINFO file

Al must now build a FORMATINFO file. This file defines the partition information in a specific way for use by the Multi-User system. It is described on page 64 .

The SAGE IV was shipped with a single user system already built. Al will use the FORMATINFO.TEXT file for this system to build the Multi-User system. The FORMATINFO.TEXT file contains the bad track map and the partitions for the Winchester disk that are installed in the current single user system.

AL will now generate NEWINFO.TEXT from FORMATINFO.TEXT defining the new partitions.

The current FORMATINFO.TEXT file is copied to NEWINFO.TEXT with the Filer:

```
SCREEN SHOWS:                    YOU TYPE

C(ommand.......................  F
F(iler.........................  T
Transfer what file?............  FORMATINFO.TEXT <cr>
To where?......................  NEWINFO.TEXT <cr>
F(iler.........................  G
Get what file?.................  NEWINFO <cr>
Text file loaded
F(iler.........................  Q
C(ommand.......................  E
>Edit..........................  (edit commands)
```

The old FORMATINFO.TEXT file will look something like this:

```
Drive 0, Heads 6
  Bad Track Map
    Track = 1381
    Track = 1421
  Device Map
    Partition 0: Cylinder 0, Head 0 to Cylinder 305, Head 5
    Partition 1: Cylinder 0, Head 1, Blocks = 1292
      Name = Pascal
      System = UCSD
    Partition 2: Blocks = 1292
      System = UCSD
    Partition 3: Blocks = 31825
      System = UCSD
```

The first two parts of the file, showing the drive number and bad tracks are left alone. Al changes the partitions using various Editor commands (Refer to page 60 of "Personal Computing with the UCSD p-SYSTEM). Since Al has done his homework and already knows the main Editor commands, it is only a short time before the file is changed as shown on the next page.

Notice that he has not changed the Bad Track Map or the definition of Partition 0, but has changed partitions 1 through 3 and added partitions 4 through 12.

```
Drive 0, Heads 6
  Bad Track Map
    Track = 1381
    Track = 1421
  Device Map
    Partition 0: Cylinder 0, Head 0 to Cylinder 305, Head 5
    Partition  1: Cylinder 0, Head 1, Blocks = 456
                  Name   = MULTI
                  System = UCSD
    Partition  2: Blocks = 1292
                  Name   = AL1
                  System = UCSD
    Partition  3: Blocks = 3876
                  Name   = A1WORK
                  System = UCSD
    Partition  4: Blocks = 1292
                  Name   = AL2
                  System = UCSD
    Partition  5: Blocks = 3876
                  Name   = A2WORK
                  System = UCSD
    Partition  6: Blocks = 3876
                  Name   = JULIE
                  System = UCSD
    Partition  7: Blocks = 1292
                  Name   = NANCY
                  System = UCSD
    Partition  8: Blocks = 3876
                  Name   = NWORK
                  System = UCSD
    Partition  9: Blocks = 1292
                  Name   = MAC
                  System = UCSD
    Partition 10: Blocks = 3876
                  Name   = MWORK
                  System = UCSD
    Partition 11: Blocks = 8113
                  Name   = DATABSE
                  System = UCSD
    Partition 12: Blocks = 1292
                  Name   = SCRATCH
                  System = UCSD
```

Al is satisfied with the partitions he has created, so he types a "Q" to Q(uit the editor. The following menu appears:

```
>Quit:
     U(pdate the workfile and leave
     E(xit without updating
     R(eturn to the editor without updating
     W(rite to a file name and return
```

Al then proceeds to update and save his work file.

SCREEN SHOWS:	YOU TYPE
(Editor Quit Menu).....	U
C(ommand.....	F
F(iler.....	S
Save as Bld:NEWINFO.TEXT?.....	Y
F(iler.....	N
Throw away current workfile?....	Y
Work file cleared	
F(iler.....	Q

Once again he is at the p-System Command-Line.

Loading partitions

The NEWINFO.TEXT file must now be "loaded", that is written to the area on the disk where the Multi-User system expects it. This is done using WFORMAT.

```
     SCREEN SHOWS              YOU TYPE
p-System Command Line.....  X
Execute what file?.......... WFORMAT

Winchester Formatter Version x
File with .. Device Maps:.... NEWINFO


Winchester Drive Selection

   A - Drive 0
   B - Drive 1
   C - Drive 2
   D - Drive 3

Select Menu item.............. A
```

WFORMAT will check the file. If there were mistakes, it will mark the line and go back to the p-System command line. Al had no mistakes so this menu displays:

```
      Function Selections

A - Format complete disk
B - Format selected tracks
C - Verify complete disk
D - Verify selected tracks
E - Update Device Maps
F - Display partition map
G - Display bad track map
H - Translate block to track
I - Format parameters

Select Menu item..... E
```

Now he types "E" for the "Update device maps" option to store the data from NEWINFO.TEXT on the map area. WFORMAT displays

```
Configuration stored successfully
Type <space> to continue
```

Al types a **<space>** to return to the Function Selections Menu. Then he types **"F"** to view the Device Map.

```
Drive 0:  34827 usable blocks, 2 bad tracks

                                                Logical track range
Partition  Name        Blocks    Category     Cyl Hd  to  Cyl Hd

    0                  34884                  0  0      305  5
    1      MULTI         456        UCSD      0  1        4  0
    2      AL1          1292        UCSD      4  1       15  2
    3      A1WORK       3876        UCSD     15  3       49  2
    4      AL2          1292        UCSD     49  3       60  4
    5      A2WORK       3876        UCSD     60  5       94  4
    6      JULIE        3876        UCSD     94  5      128  4
    7      NANCY        1292        UCSD    128  5      140  0
    8      NWORK        3876        UCSD    140  1      174  0
    9      MAC          1292        UCSD    174  1      185  2
   10      MWORK        3876        UCSD    185  3      219  2
   11      DATABSE      8113        UCSD    219  3      290  3
   12      SCRATCH      1292        UCSD    290  4      301  5
Type <space> to continue
```

Then Al quits WFORMAT by typing a **<space>** and then **two carriage returns** . He is now back at the p-System command line.

The NEWINFO.TEXT file should be backed up on floppy diskette in case future changes are required in the partition assignments or in case the Winchester should ever require re-formatting.

Example CONFIGURATION file

Al must now set up a configuration file. This file defines the worksheet information in a specific way for use by the Multi-User system. The program used to do this is MU.UTIL.

The standard 5 user 18 Meg file is close to what is needed, so a copy of this is transferred to a new file MC.NEW which will be modified using MU.UTIL.

```
SCREEN SHOWS:                    YOU TYPE

COmmand.......................... F
FIler............................ T
Transfer what file?.............. MC.5.18 <cr>
To where?........................ MC.NEW <cr>
FIler............................ Q
Command.......................... X
Execute what file?............... MU.UTIL

Multi User Configuration version 2.0
File name.......................  MC.NEW
MC.NEW read successfully
Type <space> to continue.......  <space>
```

The main menu of MU.UTIL appears as follows:

```
Multi User Configuration Utility

A - User Configurations
B - Serial Channels
C - Left  Floppy
D - Right Floppy
E - RAM Disks
F - Parallel Printer Port
G - Winchester #1 Access
H - Winchester #2 Access
I - Winchester #3 Access
J - Winchester #4 Access
K - Time Adjustment
L - Low Level Configuration
M - Auxiliary Device Info

Select Menu item <CR exits, ! aborts>:
```

First Al will set up the User Configurations so he presses "A". The User menu appears:

```
      User Selection

A - User #0 (reserved)
B - User #1
C - User #2
D - User #3
E - User #4
F - User #5
G - User #6
H - User #7
I - User #8
J - User #9
K - User #10
L - User #11
M - User #12
N - User #13
O - User #14
P - User #15

Select Menu item <CR exits, ! aborts>:
```

Al ignores user 0. Like partition 0, User 0 is used by the system only. Al presses "B" and the menu to configure User #1 appears.

```
          User #1 Configuration

A - Channel Map ( 0 to 15)
B - Channel Map (16 to 31)
C - User Capabilities
D - CP/M Information
E - Operating System Info
x - Boot Device                          9
G - Boot message                       Off
H - Boot control delay                   0
I - Shared terminal mode               Off
J - Priority                            64
K - Number of Comm buffers               1
L - Base Memory Address             000400
M - Top Memory Address              023400
N - System Stack Address            023500
O - Time Slice                        1600
P - Capability mask                 00000000

Select Menu item <CR exits, ! aborts>:
```

This shows the current setup for User #1. The channel map assigns the devices to the user, so Al presses "A" to see how the map is set up.

```
                    User Channel Map (0 to 15)

A - Channel  0 device =    0        Q - Channel  0 subdevice =  0
B - Channel  1 device =    1        R - Channel  1 subdevice =  0
C - Channel  2 device =    2        S - Channel  2 subdevice =  0
D - Channel  3 device =    0        T - Channel  3 subdevice =  0
E - Channel  4 device =    4        U - Channel  4 subdevice =  0
F - Channel  5 device =    5        V - Channel  5 subdevice =  0
G - Channel  6 device =    6        W - Channel  6 subdevice =  0
H - Channel  7 device =    7        X - Channel  7 subdevice =  0
I - Channel  8 device =    8        Y - Channel  8 subdevice =  0
J - Channel  9 device =    9        Z - Channel  9 subdevice =  3
K - Channel 10 device =    9        0 - Channel 10 subdevice =  8
L - Channel 11 device =   11        1 - Channel 11 subdevice =  0
M - Channel 12 device =    9        2 - Channel 12 subdevice = 13
N - Channel 13 device =    0        3 - Channel 13 subdevice =  0
O - Channel 14 device =    0        4 - Channel 14 subdevice =  0
P - Channel 15 device =    0        5 - Channel 15 subdevice =  0

Select menu item <CR exits, ! aborts>:
```

Al is aware that the key to a good Multi-User system is setting up this map correctly. He reviews the sections on pages 35 and 93 for an explanation of Channels and device numbers and finds that:

1. The first column matches Channel numbers (on the left) used by the operating system with actual physical device numbers on the right.

2. The second column matches the same Channel numbers with the subdevice (partition) on the right.

By referring to the tables, Al finds that Channel 1 should be the terminal input which is also device 1. Therefore he needs to make no change. Channel 2 should be the terminal output which is device 2. Again no change is needed. Channel 3 is unused and therefore is assigned a zero; no change is needed. Channel 4 is a block-structured device and is commonly mapped to the first floppy which is device 4.

Channel 5 is usually mapped to the second floppy but this system does not have a second floppy. Therefore Al changes the device number to 0 by typing **"F"** to access the Channel 5 device, then **"0 <cr> "** . The screen will show:

```
Channel 5 device =0 <cr>
```

Channel 6 is the printer channel and the PARALLEL printer (Printer #2), which is device number 6, has been assigned to this channel already so it is correct.

Channel 7 is the input for a device on the MODEM port. Since Printer #1 does not "talk back" it is not used and should be disabled. Al changes it to zero by typing **"H"** then **"0 <cr> "** .

Channel 8 is still assigned 8 for the SERIAL printer. No change is needed here.

> **NOTE:** If Al had only a serial printer, he would assign device 8 to channel 6. If the serial prnter was connected to an Auxiliary port (such as AUX4), then he would assign that device number (20) to Channel 6.

Channel 11 remains mapped to device 11, the RAM Disk.

So far this has been easy since the Channel numbers have matched the device numbers. Mapping the Winchester to Channels is the next step.

All channels that have a disk partition are assigned to the Winchester disk, device number 9. User AL1 has 3 partitions so Channel 9, 10, 12 are mapped to device 9.

The Table on page 43 shows that the subdevice numbers for the 16 Winchester partitions are 0 through 15. Each channel that was assigned to the Winchester disk must then be assigned a subdevice number for the partition. (If the Channel number is NOT 9 then the subdevice should be zero.) From Al's list on page 353, he knows he must assign partitions 2, 3 and 12 to User AL1. He types **"Z"** then **"2"** to assign partition 2 to Channel 9. Likewise, partition 12 is assigned to Channel 10 and partition 3 is assigned to Channel 12.

Note that AL assigned partition 12 to Channel 10 because he would like the scratch disk to be the same channel for all users. If he assigned SCRATCH after the other user partitions, it would be on a different channel for each user as the users have a different number of partitions. Under the p-System, this would mean that SCRATCH would have a different VOLUME number for each user. To avoid that confusion, Al set it up so that SCRATCH will turn out to be #10: for all users.

Channels 13-15 are unused so they remain disabled with zeroes.

Now the Map for User #1 (AL1) is complete and looks like this:

```
AL1                User #1 Channel Map (0 to 15)

A - Channel  0 device =     0          Q - Channel  0 subdevice =  0
B - Channel  1 device =     1          R - Channel  1 subdevice =  0
C - Channel  2 device =     2          S - Channel  2 subdevice =  0
D - Channel  3 device =     0          T - Channel  3 subdevice =  0
E - Channel  4 device =     4          U - Channel  4 subdevice =  0
F - Channel  5 device =     0          V - Channel  5 subdevice =  0
G - Channel  6 device =     6          W - Channel  6 subdevice =  0
H - Channel  7 device =     0          X - Channel  7 subdevice =  0
I - Channel  8 device =     8          Y - Channel  8 subdevice =  0
J - Channel  9 device =     9          Z - Channel  9 subdevice =  2
K - Channel 10 device =     9          0 - Channel 10 subdevice =  12
L - Channel 11 device =    11          1 - Channel 11 subdevice =  0
M - Channel 12 device =     9          2 - Channel 12 subdevice =  3
N - Channel 13 device =     0          3 - Channel 13 subdevice =  0
O - Channel 14 device =     0          4 - Channel 14 subdevice =  0
P - Channel 15 device =     0          5 - Channel 15 subdevice =  0

Select Menu item <CR exits, ! aborts>:
```

Al types <cr> to return to the User #1 menu:

```
          User #1 Configuration

A - Channel Map ( 0 to 15)
B - Channel Map (16 to 31)
C - User Capabilities
D - CP/M Information
E - Operating System Info
x - Boot Device                        9
G - Boot message                     Off
H - Boot control delay                 0
I - Shared terminal mode             Off
J - Priority                          64
K - Number of Comm buffers             1
L - Base Memory Address           000400
M - Top Memory Address            023400
N - System Stack Address          023500
O - Time Slice                      1600
P - Capability mask               00000000

Select Menu item <CR exits, ! aborts>:
```

There are no more devices to assign, so the second Channel Map is left alone. A1 types "C" to inspect the User Capabilities.

```
            User Capabilities

    A - System Manager Flag          Off
    B - Allow configuration changes   On

    Select Menu item <CR exits, ! aborts>:
```

A1 is the System Manager and AL1 will be his primary user area so he turns the System Manager Flag on by typing "A" and then "On".

```
            User Capabilities

    A - System Manager Flag           On
    B - Allow configuration changes   On

    Select Menu item <CR exits, ! aborts>:
```

The second options allows him to change some parameters using SAGE4UTIL if he needs to so he leaves that ON. A <cr> takes him back to the user menu.

The CP/M and Operating System Info are not used for the p-System so those are left as is.

The Boot device is AL1 which has been set up as Channel 9. Therefore Al leaves his boot device set to 9. Al decides he does not want the BOOT message. No Control delay is needed for normal operation.

Al must change the Shared terminal mode, to enable him to put two users on his terminal. He wants User1 to come up on the screen first so he must set the selection to "Foreground". (Later he will set User 2 to "background".) To set the Foreground, Al presses "I".

The following menu appears:

```
Shared Terminal Mode

A - Off        *
B - Foreground
C - Background

Select menu item <CR exits, ? ! aborts>:
```

Al selects "B" and the "*" jumps to the Foreground line. A <cr> takes him back to the User menu. Notice that the menu has changed to reflect the changes.

```
          User #1 Configuration

A - Channel Map ( 0 to 15)
B - Channel Map (16 to 31)
C - User Capabilities
D - CP/M Information
E - Operating System Info
x - Boot Device                      9
G - Boot message                   Off
H - Boot control delay               0
I - Shared terminal mode     Foreground
J - Priority                        64
K - Number of Comm buffers           1
L - Base Memory Address         000400
M - Top Memory Address          023400
N - System Stack Address        023500
O - Time Slice                    1600
P - Capability mask           80000000

Select Menu item <CR exits, ! aborts>:
```

Priority, Number of Comm buffers, Base Memory Address, Top Memory Address, System Stack Address, and Time Slice use defaults set up by MU.UTIL and are left alone.

The Capability Mask was changed by the User Capability selection, so there is no need to do more with it.

Al is now done configuring User 1. so he types <cr> to return to the User selection menu.

Users #2 - #5

Al now repeats the user configuration process for Users #2 through #5. Most of the options are the same. The differences are:

1) Only User #1 will have RAM Disk (physical device 11) entered in Channel 11.
2) The Channel 9, 10, 11, and 12 subdevices will have different partition numbers for each user.
3) Only User #1 will have the System Manager's flag on.
4) User #2 will have "Background" for the Shared Terminal option.
5) The Memory addresses will change (automatically) for each user.
6) Terminal assignments will be the same for Channels 1 and 2. User 3 has (13,14), User 4 has (15,16), User 5 (17,18).

When Al is done, the Channel Maps for the users should be set as shown on the next pages. The last half of each Channel Map (Logical Devices 16-31) are all left unchanged.

Al makes a table showing the p-System Volumes and Volume Names he plans to set up for each User Task.

Vol	AL1	AL2	JULIE	NANCY	MAC
# 9:	AL1:	AL2:	JULIE:	NANCY:	MAC:
#10:	SCRATCH:	SCRATCH:	SCRATCH:	SCRATCH:	SCRATCH:
#11:	RAMDISK:	-------	DATABSE:	DATABSE:	DATABSE:
#12:	A1WORK:	A2WORK:	-------	NWORK:	MWORK:

Now Al sets up the Channel Mapes for Users 2-5. The Maps should look like the ones on the following pages.

```
AL2                User #2 Channel Map (0 to 15)

A - Channel  0 device =    0        Q - Channel  0 subdevice =  0
B - Channel  1 device =    1        R - Channel  1 subdevice =  0
C - Channel  2 device =    2        S - Channel  2 subdevice =  0
D - Channel  3 device =    0        T - Channel  3 subdevice =  0
E - Channel  4 device =    4        U - Channel  4 subdevice =  0
F - Channel  5 device =    0        V - Channel  5 subdevice =  0
G - Channel  6 device =    6        W - Channel  6 subdevice =  0
H - Channel  7 device =    0        X - Channel  7 subdevice =  0
I - Channel  8 device =    8        Y - Channel  8 subdevice =  0
J - Channel  9 device =    9        Z - Channel  9 subdevice =  4
K - Channel 10 device =    9        0 - Channel 10 subdevice = 12
L - Channel 11 device =    0        1 - Channel 11 subdevice =  0
M - Channel 12 device =    9        2 - Channel 12 subdevice =  5
N - Channel 13 device =    0        3 - Channel 13 subdevice =  0
O - Channel 14 device =    0        4 - Channel 14 subdevice =  0
P - Channel 15 device =    0        5 - Channel 15 subdevice =  0

Select Menu item <CR exits, ! aborts>:
```

```
JULIE              User #3 Channel Map (0 to 15)

A - Channel  0 device =    0        Q - Channel  0 subdevice =  0
B - Channel  1 device =   13        R - Channel  1 subdevice =  0
C - Channel  2 device =   14        S - Channel  2 subdevice =  0
D - Channel  3 device =    0        T - Channel  3 subdevice =  0
E - Channel  4 device =    4        U - Channel  4 subdevice =  0
F - Channel  5 device =    0        V - Channel  5 subdevice =  0
G - Channel  6 device =    6        W - Channel  6 subdevice =  0
H - Channel  7 device =    0        X - Channel  7 subdevice =  0
I - Channel  8 device =    8        Y - Channel  8 subdevice =  0
J - Channel  9 device =    9        Z - Channel  9 subdevice =  6
K - Channel 10 device =    9        0 - Channel 10 subdevice = 12
L - Channel 11 device =    9        1 - Channel 11 subdevice = 11
M - Channel 12 device =    0        2 - Channel 12 subdevice =  0
N - Channel 13 device =    0        3 - Channel 13 subdevice =  0
O - Channel 14 device =    0        4 - Channel 14 subdevice =  0
P - Channel 15 device =    0        5 - Channel 15 subdevice =  0

Select Menu item <CR exits, ! aborts>:
```

```
NANCY                User #4 Channel Map (0 to 15)

A - Channel  0 device =    0          Q - Channel  0 subdevice =  0
B - Channel  1 device =   15          R - Channel  1 subdevice =  0
C - Channel  2 device =   16          S - Channel  2 subdevice =  0
D - Channel  3 device =    0          T - Channel  3 subdevice =  0
E - Channel  4 device =    4          U - Channel  4 subdevice =  0
F - Channel  5 device =    0          V - Channel  5 subdevice =  0
G - Channel  6 device =    6          W - Channel  6 subdevice =  0
H - Channel  7 device =    0          X - Channel  7 subdevice =  0
I - Channel  8 device =    8          Y - Channel  8 subdevice =  0
J - Channel  9 device =    9          Z - Channel  9 subdevice =  7
K - Channel 10 device =    9          0 - Channel 10 subdevice = 12
L - Channel 11 device =    9          1 - Channel 11 subdevice = 11
M - Channel 12 device =    9          2 - Channel 12 subdevice =  8
N - Channel 13 device =    0          3 - Channel 13 subdevice =  0
O - Channel 14 device =    0          4 - Channel 14 subdevice =  0
P - Channel 15 device =    0          5 - Channel 15 subdevice =  0

Select Menu item <CR exits, ! aborts>:
```

```
MAC                  User #5 Channel Map (0 to 15)

A - Channel  0 device =    0          Q - Channel  0 subdevice =  0
B - Channel  1 device =   17          R - Channel  1 subdevice =  0
C - Channel  2 device =   18          S - Channel  2 subdevice =  0
D - Channel  3 device =    0          T - Channel  3 subdevice =  0
E - Channel  4 device =    4          U - Channel  4 subdevice =  0
F - Channel  5 device =    0          V - Channel  5 subdevice =  0
G - Channel  6 device =    6          W - Channel  6 subdevice =  0
H - Channel  7 device =    0          X - Channel  7 subdevice =  0
I - Channel  8 device =    8          Y - Channel  8 subdevice =  0
J - Channel  9 device =    9          Z - Channel  9 subdevice =  9
K - Channel 10 device =    9          0 - Channel 10 subdevice = 12
L - Channel 11 device =    9          1 - Channel 11 subdevice = 11
M - Channel 12 device =    9          2 - Channel 12 subdevice = 10
N - Channel 13 device =    0          3 - Channel 13 subdevice =  0
O - Channel 14 device =    0          4 - Channel 14 subdevice =  0
P - Channel 15 device =    0          5 - Channel 15 subdevice =  0

Select Menu item <CR exits, ! aborts>:
```

Example of Serial Configuration

Al is now finished configuring the users and is at the User Selection Menu. He types <cr> and is back to the Main Utility Menu.

```
Multi User Configuration Utility

A - User Configurations
B - Serial Channels
C - Left  Floppy
D - Right Floppy
E - RAM Disks
F - Parallel Printer Port
G - Winchester #1 Access
H - Winchester #2 Access
I - Winchester #3 Access
J - Winchester #4 Access
K - Time Adjustment
L - Low Level Configuration
M - Auxiliary Device Info

Select Menu item <CR exits, ! aborts>:
```

Next Al must configure the Serial Channels and chooses **"B"**, for Serial Channels. The six Serial channels are shown in this menu and reference the six ports (connectors) on the back of the SAGE IV.

```
      Serial Channel Selection

A - Sage II Terminal Port
B - Sage II Modem Port
C - Sage IV Extra Serial Port #1
D - Sage IV Extra Serial Port #2
E - Sage IV Extra Serial Port #3
F - Sage IV Extra Serial Port #4

Select Menu item <CR exits, ! aborts>: A
```

Both AL1 and AL2 use the Terminal Port. To inspect the options, AL presses **"A"** and the following menu appears:

```
                    Terminal Channel Configuration

A - Baud Rate              DIP Switch     K - Remote Channel          Off
B - Parity                 DIP Switch     L - Xon/Xoff on output      On
C - Stop Bits              1 Stop Bit     M - Xon/Xoff on input       Off
D - Data bits              8 Data Bits    N - Count to send XOFF      240
E - Xmit Buff Length             256      O - Count to send XON       64
F - Rec Buff Length              256      P - DSR Polling             Off
G - Type of Terminal       QUME QVT 102   Q - DSR Poll interval       16384
H - Char to chng user                     R - Input event num         0
I - BREAK to reboot              On       S - Output event num        0
J - BREAK to debug               Off      T - Access Control

Select Menu item <CR exits, ! aborts>:
```

Most terminal selections are already correct for the SAGE terminal. Since Al is sharing two tasks with a Qume QVT102 terminal, he checks the "Type of Terminal" field. It is currently set to a QVT102 so he continues.

Al also wants to be able to "BREAK to debug" while working with assembly code programs so he selects "J" and then types "On" and <cr>.

He then checks all the other ports, They should all look like the terminal configuration except:

> The MODEM port should have the "Remote Channel" option **ON** as this port is for the serial printer not a terminal.
>
> The MODEM port should be set for 9600 baud for the serial printer.

After checking all the ports. Al returns to the Main menu by typing <cr> at each menu until the Main menu displays.

- **Floppy configuration example**

The company's SAGE IV has only one floppy drive. Al knows that having device 5 disabled (normally the other floppy drive) will speed up the boot process as the system will not spend time searching for a device that is not there.

He reviews what he has done and realizes that he has already taken all of the references to device 5 out of all of the user channel maps. This disables device 5 so he does not have to use the "Floppy configuration" menu to specify that the right floppy is not equipped.

Example of RAM Disk Config

Al would like to use the memory not assigned to a user as a RAM Disk. He selects "E" from the main MU.UTIL menu:

```
Ram Disk Channel Selection

A - Ram Disk #1
B - Ram Disk #2
C - Ram Disk #3
D - Ram Disk #4

Select menu item <CR exists, ! aborts>:
```

Al chooses "A" and sets up one RAM Disk.

```
          RAM Disk #1 Configuration

A - Base of RAM Disk  (0 = disabled) 0AF900
B - Top  of RAM Disk  (0 = to BIOS)  000000
C - Initialize Ram Disk                Off
D - Access Control

Select menu item <CR exits, ! aborts>:
```

The address in "A" should be the same as the Stack address shown in User #5's configuration screen. For a five user system this remains the same.

He also changes "C" to On, and then checks the Access Control to see that all the flags are on. Typing <cr> twice returns him to the main menu.

Example of Printer Configuration

To set up the Parallel Printer, Al types an "F" for "Parallel Printer Port" from the main MU.UTIL menu. Since the parallel port is preset to a default value he checks these values found in the Parallel Printer Configuration Control Menu against those specified in his printer operating manual.

```
                    Parallel Printer Configuration Control

A - Printer mode                    Parallel port with interrupts
B - Output Buffer Size                                        256
C - Polling attempts before delay                             500
D - Delay before re-polling                                   236
E - Linefeed after carriage return                             On
F - Access Control

Select Menu item <CR exits, ! aborts>:
```

This menu is correct so he goes to the next option.

Winchester Access Example

In most systems, some of the partitions are used exclusively by one user while others are shared. For this system, the WORK partitions are "private" partitions while DATABSE is used by MAC and NANCY. SCRATCH is also available to all users.

Normally, the access is set up so that all users have access to all devices. They cannot use those devices, however, unless they are specified in the channel map. Therefore, the access options can all be left enabled and DATABSE: and SCRATCH: will be available to the right people. AL checks to see that all access flags are ON.

Since the system has only one hard disk, Al enters a "G" for Winchester #1 Access.

```
Winchester #1 Partition Access

A - Drive partition #0
B - Drive partition #1
C - Drive partition #2
D - Drive partition #3
E - Drive partition #4
F - Drive partition #5
G - Drive partition #6
H - Drive partition #7
I - Drive partition #8
J - Drive partition #9
K - Drive partition #10
L - Drive partition #11
M - Drive partition #12
N - Drive partition #13
O - Drive partition #14
P - Drive partition #15

Select Menu item <CR exits, ! aborts>:
```

Al selects Partition #1 by typing **"B"** and checks that the flags look like the menu below. Once satisfied that it is ok, Al types **<cr>** to return to the Partition Access Menu and selects the next partition to check. Partitions #1 through #10 should all look like the menu below.

```
                        Access Control

A - Timeout threshold          0        L - User #6
B - Read allowed mask       FFFF        M - User #7
C - Write allowed mask      FFFF        N - User #8
D - Exclusive control mask  0000        O - User #9
E - Hang & Wait mask        0000        P - User #10
F - User #0                             Q - User #11
G - User #1                             R - User #12
H - User #2                             S - User #13
I - User #3                             T - User #14
J - User #4                             U - User #15
K - User #5

Select Menu item <CR exits, ! aborts>:
```

Al skips Partition #11 for the moment and selects **"M"** for Partition #12, SCRATCH. This partition must be accessed by all the users. Al turns Exclusive Control **ON** for all users by typing **"D"** and then **"FFFF <cr> "** . This allows all users access, but only one at a given time.

He also turns Hang & Wait **On** for all users in the same way. A user who is waiting to access the partition will "hang" in

the program until the partition is available instead of getting an error.

Since Hang & Wait is now **On** , Al sets the Timeout Threshold to 2 seconds. This means that 2 seconds after the last access of the user who has exclusive control, control will be passed to the next user who is waiting.

Al types **<cr>** to return to the Partition Access Menu.

Al types **"L"** to reach the Access Control Menu for Partition #11. This is DATABSE. Only Nancy and Mac are to have complete access to it. Julie will only be able to read it. Al types **"I"** for the User #3 Access Control Menu.

```
          User #3 Access Control

   A - Read allowed          On
   B - Write allowed         On
   C - Exclusive Control     Off
   D - Hang & Wait           Off

   Select Menu item <CR exits, ! aborts>:
```

Since JULIE will have Read only access to DATABSE, Al types **"B"** then **"Off <cr> "** to prevent her from writing to that area. Exclusive Control and Hang & Wait are left off as the Database routine will handle arbitration using Global Semaphores.

Typing three **<cr>s** takes Al back to the main menu.

Saving the Configuration'

The other options "Time Adjustment", "Low Level Configuration" and "Auxiliary Device Info" rarely need to be changed. Al skips them.

Al is now finished with the MU.UTIL configuration. He must now save the new information. He types <cr> until the screen prompts:

```
Ready to write changes to MC.NEW:
```

Al enters "Y" for yes. It should be noted that if you respond "N" the the screen will ask:

```
Abandon changes?
```

If a "Y" is typed all changes will be lost. At this point, MU.UTIL checks the configuration. If it finds any errors, it will report them and give AL a chance to go back and fix them.

No errors were found. All the new Multi-User Configuration information has now been saved in the MC.NEW file. The screen shows:

```
Configuration changes saved
Type <space> to continue
```

Typing a <space> takes AL to the p-System command line.

Zeroing Disk Partitions

The BUILD diskette is setup with a special SYSTEM.MISCINFO so that it can access all of the new partitions on the disk. The normal "vanilla" system on the SYSTEM MASTER diskette will only access physical devices up to Volume #12:.

Al builds a table showing how all his device assignments will show up as Volumes under the special SYSTEM.MISCINFO. He references the Volume to partition assignments described on page 346.

The Filer would normally show the disk partitions as in the table (when a "V" for volumes command is done) but their directories must be ZEROED first.

Partition	Vol	Partition	Vol	Partition	Vol
1-MULTI	# 9:	5-A2WORK	#14:	9-MAC	#18:
2-AL1	#10:	6-JULIE	#15:	10-MWORK	#19:
3-A1WORK	#12:	7-NANCY	#16:	11-DATABSE	#20:
4-AL2	#13:	8-NWORK	#17:	12-SCRATCH	#21:

Before Al can load programs to the partitions, he must zero the directories. Al should now refer to his worksheet, page 351 , (or his file NEWINFO.TEXT) showing the block size and name of each volume.

SCREEN SHOWS:	YOU TYPE
C(ommand.....	F
F(iler.....	Z
Zero what vol?.....	#9: <cr>
Duplicate dir?.....	Y
# of blocks on the disk?.....	456
New vol name?.....	MULTI
MULTI correct?.....	Y
MULTI zeroed	
F(iler.....	

Al repeats this process for each volume assigning volume names and sizes. Note that for the partitions sized at 1292 blocks, he only zeroes them to 1280 so that a volume-to-volume copy to a floppy is possible.

Bootstrap Load example

From the Filer A1 types "Q" for quit and "X" from the p-System command level. The prompt asks "Execute what file?" and A1 responds by typing SAGE4UTIL. From the SAGE4UTIL Menu A1 selects "D" and answers the screen prompts:

```
SCREEN SHOWS:                    YOU TYPE

F(iler.............................  Q
C(ommand...........................  X
Execute what file?.................  SAGE4UTIL <CR>
Initializing.......................
SAGE4UTIL.(menu)...................  D
Source file or device?.............  #4:MU4.WBOOT.CODE <CR>
Ready to load bootstrap?...........  Y
Bootstrap read successfully
Destination file or device?........  MULTI: <cr>
Ready to store bootstrap?..........  Y
Bootstrap written  sucessfully

Type <space> to continue...........  <space>
```

A1 has just successfully transfered the Multi-User Bootstrap to the MULTI partition bootstrap area. The Multi-User Bootstrap must be on a partition to initiate the Multi-User environment.

He must now do a similar boot installation process only with boot file MU.PBOOT.CODE for volumes #10:, #13:, #15:, #16: and #18:, Note that the WORK partitions, DATABSE:, and SCRATCH: do not need a boot installed because these areas will only be used for storing data.

Once all of the bootstraps are loaded, A1 leaves SAGE4UTIL by typing a <cr> to exit the menu.

File load example

The partitions are now zeroed and those that need it have a bootstrap for the p-System. It is time for Al to load the programs each user will need into their areas.

First, all of the boot paritions need the p-System files. Al must use the Filer to transfer them onto the volume. The p-System files are available on the MASTER diskettes shipped with every system. Not all of the files are necessary. A good start is to load the files as they are on the BUILD disk itself.

SCREEN SHOWS:	YOU TYPE
C(ommand.....	F
F(iler.....	L
Dir of what vol?.....	#4: <CR>

A complete listing of all the files on the floppy disk is scrolled onto the screen. Now Al must transfer (in order) from the BUILD disk the necessary system files for each user's Systems partition.

The Multi-user partition MULTI needs only these files:

ENDBOOT
MU4.BIOS
MU.CONFIG (MC.NEW)
MU.BOOTEXT.CODE
NEWINFO.TEXT (For Backup)

Note that because RAM Disk will be initialized on boot, the ENDBOOT file is necessary to prevent these files from being copied into RAM Disk. MC.NEW is transferred over as MU.CONFIG. MU.BOOTEXT.CODE is needed to initialize the RAM Disk, that is to zero its directory.

The users' boot partitions need these files:

```
SYSTEM.MISCINFO  (QVT102.MISCINFO)
SYSTEM.PASCAL
SYSTEM.FILER
SYSTEM.EDITOR
SAGE4UTIL.CODE
SYSTEM.LIBRARY
SYSTEM.INTERP
```

To transfer Al types "T" from the Filer and transfers each file in order to each user's partition. NOTE that he does not transfer the BUILD disk's special SYSTEM.MISCINFO, but transfers over the standard QVT102.MISCINFO for his QUME QVT102 terminal.

In addition to System files, Al also transfers the WORD7 files from the Business diskette to each user's System partition except for his AL1 volume. The files needed for WORD7 are:

```
WORD7.CODE
WORD7.PARMS
```

Since Nancy and Mac use a spread sheet extensively, Al loads the TIMBERLINE SPREAD SHEET files into their System area partitions in the same manner to allow more room for the manipulation of data files. The files needed for TIMBERLINE are:

```
TS.FORM
TS.SPEC
SS.CODE
SS.FORM
```

Once the Multi-User system is running, the users can transfer their own files onto their area. For example, Nancy, and Mac must load their database files into the shared database partition.

NOTE: Sharing partitions requires careful thought on how they will be used. Some types of programs are dangerous in this environment. Refer to the guidelines for shared partitions on page 261 .

Multi-User Boot example

The new Multi-User system that Al configured is now ready to go on-line. Al makes sure that all of the terminals and printers are connected and powered up. He now boots to MULTI-USER!

```
SCREEN SHOWS:                YOU TYPE

F(iler.....................  Q
C(ommand.....................  H
>.............................  IH <CR>
```

Each terminal should display the p-System "Welcome" message.

Al checks each user's access by going to each terminal, loading the filer and typing a "V" for volumes. Each user has the correct volumes so Al has done a good job. Each user partition is configured exactly as he planned and the system is ready for business.

After installation

Once the multi-User system has been running awhile, changes are sure to be suggested. While it is very easy to change access to the devices (re-run MU.UTIL and reboot), changing the size of the partitions should not be done casually. This generally requires that the users backup all information currently on the disk (transfer it to floppies). The System Manager must repeat the entire MU build process. Then the files must be loaded into the new areas.

The System Manager should also make backup copies on diskette of the information he used to build the Multi-User system. This includes the modified format information file (NEWINFO.TEXT) and the configuration file (MC.NEW). Also it is generally handy to make an assignment chart showing the users, their Volumes, and their partitions.

XII ADVANCED MU FEATURES :

A number of Advanced Multi-User Environment features are provided using I/O reqeuests via user devices 132 and 133. These features include releasing a User's Time Slice, Getting and Releasing Control of a device, handling Global Semaphores, hooks for implementing Logon programs, and the User Inter-Communication facility. All operations available on devices 132 and 133 will return a bad device number error code (2) if performed on a SAGE single-user environment.

XII.01 RELEASE OF TIME SLICE :

Some programs may wish to poll a device or Global Semaphore and give up control of their time slice if certain conditions are not met. In this manner the polling will have only a small affect on the system throughput when the polled condition is not met. The Release Time Slice request may be placed in a test loop which is run every time the user gets a slice of time. The Release Time Slice request is performed by doing a write to channel 132 with a control word of 2. In assembly language this is function 10 using a TRAP #14. From Pascal in the p-System this is done with a UNITWRITE. The request may be put into a program which is also used in the SAGE single user environments. In the single user case the request will return an IORESULT of 2 and not cause any other action.

```
UNITWRITE(132,x,x,x,2);  { Release User's Time Slice }
                         {The "x"s are dummy values}
```

XII.02 PROGRAMMED DEVICE CONTROL :

All programmed operations for Device Control are done through BIOS read and write requests on device 132 with a control word of 0. In assembly language this is done with read (function 11) and write (function 12) requests using TRAP #14. Under Pascal in the p-System these functions are

directly available as UNITREAD's and UNITWRITE's.

```
UNITREAD(132,Data,Size,Block,0);
UNITWRITE(132,Data,Size,Block,0);
```

The Size and Block fields in the above I/O calls are not used to mean the traditional size and device block number. Instead the Size represents an operation request code plus an optional task number. The Block represents the device number being referenced.

The low byte of the Size is the operation request code. The high byte of the Size is a representation of the Task number plus one. If the high byte is zero then the current partitions Task number is used. Note that most usage pertains to the current task so zero is the normal value. Following is a list of the operation request codes:

0 - Release control of the device

Use a write request (UNITWRITE) to release control of a device. Note that this request does not affect the Data field.

Error Code (IORESULT)
- 0 - successful release of control.
- 1 - did not previously have control.
- 2 - Function not available (single user).

1 - Get control of the device

Use a write request (UNITWRITE) to get control of a device. Note that this request does not affect the Data field.

Error Code (IORESULT)
- 0 - successful granting of control.
- 1 - did not get control.
- 2 - operation not available (single user).

2 - Read/Write the Control Information Block

Use a read request (UNITREAD) to read in the Control Information Block. Use a write request to update a device's Control Information Block. Note that the Current Control Mask and Timeout Counter fields are not updated by the write.

Error Code (IORESULT)
0 - successful read of control info.
2 - operation not available (single user).

The Block number represents either a logical device number if positive or a physical device number if negative. Logical device numbers will be mapped to physical device numbers through the Task's BIOS Channel Map. Physical device numbers may have a subdevice number in their high byte.

● Control Information Block

Function request 2 pertains to the Control Information Block for a device. The block is always a fixed length of 12 bytes and is moved into and out of the Data field of the I/O request.

Offset	
0	Read Allowed Mask (2 bytes)
2	Write Allowed Mask (2 bytes)
4	Exclusive Control Mask (2 bytes)
6	Current Control Mask (2 bytes) **
8	Hang & Wait Mask (2 bytes)
10	Timeout Threshold (1 byte)
11	Timeout Counter (1 byte) **

** These items are not updated during a write of the Control Information Block.

The following example shows the use of the Control Information Block to access and clear control information.

```
Example Device Access Control:

TYPE
  TaskFlags = PACKED ARRAY[0..15] OF BOOLEAN;

VAR
  Dummy:INTEGER;
  CIB:PACKED RECORD
        ReadAllowed:TaskFlags;
        WriteAllowed:TaskFlags;
        Exclusive:TaskFlags;
        Current:TaskFlags;
        HangWait:TaskFlags;
        TimeCtr:0..255;
        TimeThreshold:0..255;
      END;

BEGIN
  { Get Control of physical Winchester Drive 0, Partition 3 }
  UNITWRITE(132,Dummy,1,-((3*256)+9),0);
  IF IORESULT = 0 THEN
    WRITELN('Got Control of Winchester');

  { Find and clear all Tasks with control of my logical device 5}
  UNITREAD(132,CIB,2,5,0);
  FOR Task := 0 TO 15 DO
    WITH CIB DO
      IF Current[Task] THEN
        BEGIN
          WRITELN('Task ',Task,' had control');
          { Release Task's control}
          UNITWRITE(132,Dummy,((Task+1)*256)+0,1,0);
        END;

  { Change logical device 11's timeout threshold }
  REPEAT
    { First get control of device }
    UNITWRITE(132,Dummy,1,11,0);
  UNTIL IORESULT = 0;
  { Update threshold }
  UNITREAD(132,CIB,2,11,0);
  CIB.TimeThreshold:=25;
  UNITWRITE(132,CIB,2,11,0);
  { Release control }
  UNITWRITE(132,CIB,0,11,0);
```

XII.03 GLOBAL SEMAPHORES :

The SAGE Multi-User implementation provides a very generalized Global Semaphore scheme to allow specialized data base programs to share data between users. Typically a data file or set of data files will be stored on a device which is accessible to all users. These data files are only accessed and modified by a program or group of programs which use the SAGE Global Semaphores to coordinate their accesses.

Semaphore Philosophy

The Global Semaphores are a very general abstraction and are not tied to any specific purpose. The program or group of programs sharing a common data base all use the same algorithm to create Global Semaphore names to represent a specific item of data to be protected. The program can then get control of this unique name, therefore also getting control over the specific item of data being represented. While the program has control of the Global Semaphore name no other program may get control of the name and therefore no other program may gain access to the data which is being accessed by the first program. When the program is finished with the data it then releases control of the Global Semaphore name thus allowing other programs to get control of the name and its associated data.

It is important to note that it is up to the program to obey the established protocol in order to protect the data. Nothing actually prevents a program from reading or writing to a file without using a Global Semaphore name.

Getting control of a Global Semaphore name is actually a polling process. Even if control is not granted the request always returns to allow the program to determine an appropriate action. An error reply code is passed back indicating if control was granted or rejected. Typically a program will place the request within a loop which will continue polling until the request is granted. Generally if the request is rejected the user's Time Slice should be

released to allow other users a better chance to complete their work.

A timeout check may be implemented in the loop to prevent a permanent lockup. Of course recovery from the lockup is a complex issue left to the program implementor. The program may determine the internal task number of the task which has control and report this to the user. Booting a user partition will always clear out any semaphores which are owned by the partition.

For many data base transactions several items of data must be locked simultaneously. The SAGE Global Semaphore scheme allows requesting control of several Semaphore names at the same time. Control of the names is only granted if control can be obtained for all the names. This helps avoid the deadlock problem where one program has control of a name for which another program is waiting, and the second program has control of a name for which the first program is waiting.

The data base designer must be sure that under the expected loading of the system, the probability of all the names being available is high enough to insure adequate transaction throughput. Of course control of the names can be obtained one at a time with other methods implemented for avoiding system deadlock.

It is important to note that under the p-System it is inadequate to only lock a record within a file using its relative record number as a base for the Global Semaphore name. The operating system reads and writes complete blocks of data which may contain several records or possibly only a portion of the record. It is therefore necessary to actually gain control over one or more blocks on the device rather than records.

Also writing a record in the p-System does not actually force the data to be written to the device. The data may remain in memory until the file is Closed or a Seek and subsequent I/O operation forces the data to be written.

Semaphore Implementation

All programmed operations for Global Semaphores are done through BIOS read and writes requests to device 132 with a control word of 1. In assembly language this is done with read (function 11) and write (function 12) requests using TRAP #14. Under Pascal in the p-System these functions are directly available as UNITREAD's and UNITWRITE's.

```
UNITREAD(132,Sem,Size,Block,1);

UNITWRITE(132,Sem,Size,Block,1);
```

The "Sem" field is the data buffer which can contain the specification of one or more Global Semaphores. The "Size" field means the number of Semaphores rather than the traditional number of bytes in the data buffer. The "Block" field represents the Global Semaphore operation to be performed.

Each Global Semaphore is an 18 byte area of which the first 2 bytes are reserved by the system for the owner's task number and the last 16 bytes are the user created Semaphore identifier (name). Multiple Semaphores are simply stored in memory sequence in the data buffer. Following is a typical Semaphore layout under the p-System.

```
TYPE
  Sema     = RECORD
               TaskNum:INTEGER;
               BlockNum:INTEGER;
               FileID:PACKED ARRAY[0..9] OF CHAR;
               DeviceNum:INTEGER;
               Project:INTEGER;
             END;

  SemGroup= ARRAY[1..10] OF Sema;
```

TaskNum is the first word and need not be specified by the user. It is filled in by the system when the Semaphore is entered into the system's Semaphore table. In this user's representation of the Semaphore name, the "BlockNum" field represents the relative block number in the file to be protected.

FileID is the first ten characters of the file name (leaving off the .DATA suffix).

DeviceNum represents a specific device in an environment where shared files are stored on multiple devices.

Project field is a fixed number, selected at random by the programmer, to reduce the possibility of clashes with semaphore names used by other developers.

SemGroup The record format defining the Semaphore is totally up to the user, with the exception of the first word. This example is only a sample of a possible Semaphore naming convention. Note that since the internal comparison of Semaphore names is from beginning to end, performance may be enhanced by making the portion most likely to mismatch be first in the Semaphore name definition.

● Semaphore Functions

The available Global Semaphore functions are:

0. Release Control of Semaphores
1. Get Control of Semaphores
2. Check if Semaphores exist
3. Clear out one Task's Semaphores
4. Read back all the system Semaphores
5. Read back Semaphore Status information.
6. Clear out all the Semaphores

Each Semaphore operation will return an error code (IORESULT). Following is a breakdown of the operation request codes.

0 - Release Control of Semaphore(s)

Use a write request with a Block number of 0 to release control of one or more Semaphores. Note that it is possible to release control of a Semaphore by a user for which control was not originally obtained.

Error Code (IORESULT)

0 - Control of all requested Semaphores was released.
2 - Feature not implemented (single user).

1 - Get Control of Semaphore(s)

Use a write request with a Block number of 1 to obtain control of one or more Semaphores. If control cannot be obtained for all the Semaphores, control will not be granted for any of the Semaphores and the request must be repeated.

Error Code (IORESULT)

0 - Control of all requested Semaphores was granted.
1 - Control of the Semaphores was rejected.
2 - Feature is not equipped (single user).
3 - Not enough room in Semaphore table.

2 - Check Semaphores(s)

Use a write request with a Block number of 2 to check on the existence of one or more Semaphores.

Error Code (IORESULT)

0 - The Semaphores exist and are all under the calling user's control.
1 - At least one of the Semaphores does not ex in the Semaphore table.
2 - Feature does not exist (single user).
3 - The Semaphores exist but not all are under the calling user's control.

3 - Clear Task's Semaphores

Use a write request with a Block number of 3 to clear ONLY those Semaphores associated with this user. It should be used during initialization of any program which uses Semaphores to insure that any previous problem has been cleared.

4 - Read back Semaphores

Use a read request with a Block number of 4 to read back all the active Semaphores in the system's Semaphore table. The Size field should be specified as the maximum number of Semaphores that can be held in the caller's data buffer.

Note that a Size should be specified which will hold at least one more than the maximum number of Semaphores in the system.

The first word of each returned Semaphore (TaskNum) will contain the internal task number (one based) of the user with control of the Semaphore. The list will be terminated with a Semaphore whose task number is zero and whose name is undefined.

Error Code (IORESULT)
0 - Semaphore read back is complete.
2 - Feature does not exist (single user).

5 - Get Semaphore Statistics

Use a read request with a Block number of 5 to read in the following parameters pertaining to the Semaphore utility. The Size field is not used.

Data Buffer Offset

0 - Maximum number of system Semaphores.
2 - Number of active Semaphores.
4 - Calling user's task number (one based).

Typical p-System Pascal record:

```
SemaStats = RECORD
              MaxSemas:INTEGER;
              Active:INTEGER;
              TaskNum:INTEGER;
            END;
```

Error Code (IORESULT)

0 - Semaphore statistics were returned.
2 - Feature does not exist (single user).

6 - Clear All Semaphores

Use a write request with a Block number of 3 to clear the complete system Semaphore table. Note that this should only be used by programs running in a captive environment where no other programs running Semaphores would be present.

Error Code (IORESULT)

0 - Clearing the Semaphores was completed.
2 - Feature does not exist (single user).

Semaphore Lockup Protection

Certain situations can occur which will leave a Global Semaphore name in the system's table permanently. If a p-System Break key forces the program to terminate while it has control of a Semaphore, the Semaphore will not get released. A program could also have complex paths under which an error in logic could leave a Semaphore under control of the User. To help solve these problems, a request to clear all of a single User's Semaphores is provided.

Programs using Semaphores should be structured to release all the User's Semaphores at initialization and other key times when the program should not have any outstanding semaphores. The system automatically clears a User's Semaphores when a User is completely re-booted. Reinitialization under the p-System, using I at the command line or after a p-System runtime error does not clear out the User's Semaphores.

Semaphore Example

This example shows the use of Semaphores to access a record in a file called SALES on the Winchester.

```
VAR
  Control:BOOLEAN;
  Dummy:INTEGER;
  SemGroup = ARRAY[1..10] OF Sema;
BEGIN
  UNITWRITE(132,Dummy,0,3,1); { Release User's Semaphores }

  { Create a Semaphore Names }
  WITH SemGroup[1] DO
    BEGIN
      BlockNum:=10; { Secure access to block 10 in file INDEX
                      on RAMDisk }
      FileID:='INDEX      '; { must pad out to ten characters  }
      DeviceNum:=11;
      Project:=8419;  { a random number}
    END;
  WITH SemGroup[2] DO
    BEGIN
      BlockNum:=354; { Secure access to block 354 in file SALES
                       on the Winchester data base }
      FileID:='SALES      ';
      DeviceNum:=10;
      Project:=8419;
    END;
  Control:=FALSE;
  REPEAT
    UNITWRITE(132,SemGroup,2,1,1); { Get control of Semaphores }
    IF IORESULT <> 0 THEN
      UNITWRITE(132,Dummy,0,0,2); { Release time slice }
    ELSE
      Control:=TRUE;
  UNTIL Control ;

  { Do I/O processing on protected blocks }

  UNITWRITE(132,SemGroup,2,0,1); { Release control of Semaphores }
```

XII.04 LOGON HOOKS :

Several BIOS calls have been created to support a Logon facility although a Logon program is currently not provided. The concept would be to have all the user tasks boot to a common partition which contains a Logon program as a SYSTEM.STARTUP file.

The logon program would then do the appropriate name and password checking as well as allowing the user to select a user environment. Precautions should be made to avoid letting two users use the same environment (boot device) unless this is known to be practical. This may be

accomplished using Gobal Semaphores to provide exclusive control of the active environments.

The Logon program should set up the user's BIOS Channel Map according to the requirements of the user requested environment. The maintenance of the environment possibilities, user names, and passwords is up to the Logon inplementor. Note that in order to write the Channel Map the user task must be initially configured to be a System Manager.

The Logon program may then also set up a new boot device (logical channel) number as well as a new capability mask (usually more restrictive). Finally the Logon program issues a BIOS call which will reboot the user task to the newly specified boot device, bringing up the user's requested environment.

At initialization the BIOS remembers its initial boot device (logical channel) and its associated physical device number. Also remembered is its initial capability mask. When a user environment halts or is rebooted via the Break option, the original boot device and capability mask are restored and the user task is rebooted to the Logon partition therefore restarting the Logon program.

The reading and writing of the Channel Map is typically done by the CONFIGSAGE Unit (described elsewhere in this manual). Following is a description of the BIOS calls necessary to implement the described Logon scenerio.

Read/Write Boot Device Number

Read/Write Capability Mask

Reboot User Task

Read/Write of Boot Device Number

Access to the user's boot device number is available with reads (UNITREAD) and writes (UNITWRITE) of device 132 with a control word of 3. Writing a boot device number requires that the user's Capability Mask be configured to be a System Manager. The buffer argument contains the boot device number. The size and block number arguments of the I/O call are ignored.

A newly written boot device number is set up as a temporary override of the original boot device number. If the user terminates the system with an "Exit back to Login" (TRAP #14, function 0), the temporary boot device override is cleared and the original boot device number is used to reboot the system. The temporary boot device override is also cleared by the Reboot option in the Break Key menu. When reading a boot device number, the user gets the temporary override boot device number if it is active, otherwise the original boot device number is returned.

Following is an example of accessing the boot device number:

```
VAR
  OrigBoot:INTEGER;
  NewBoot:INTEGER;

BEGIN
  UNITREAD(132,OrigBoot,0,0,3);
  WRITELN('Original boot device was ',OrigBoot);
  NewBoot:=9; {Now boot to logical device 9}
  UNITWRITE(132,NewBoot,0,0,3);
```

Read/Write Capability Mask

Access to the Capability Mask is provided through reads (UNITREAD) and writes (UNITWRITE) to device 132 with a control word of 4. Writes to the Capability Mask may only be done if the previous mask indicates that the user is a System Manager. Therefore a user with the System Manager priviledge may remove the priviledge but a user without the priviledge cannot make himself a System Manager. The buffer

argument is the two word Capability Mask. The size and block number arguments are ignored.

The second word of the Capability Mask is reserved for user defined capabilities and will not be used by SAGE. The Capability Mask at Multi-User system boot time will be remembered and restored into the user's Capability Mask if the user terminates the task with an "Exit back to Login" (TRAP #14, function 0) or a Break and Reboot.

The Capability Mask has two words with currently only two high order bits defined in the first word. The second word is available for user defined capabilities.

Capability Word 1

Bit 15:	System Manager flag
Bit 14:	Inhibit all configuration changes flag
Bits 13-0:	reserved for future use

Capability Word 2

Bits 15-0:	available for user defined purposes

Following is an example of the Capability Mask access:

```
TYPE
  CMword = PACKED ARRAY[0..15] OF BOOLEAN;

VAR
  Capability:RECORD
               Word1:CMword;
               Word2:CMword;
             END;

BEGIN
  UNITREAD(132,Capability,0,0,4);  {Get previous mask}
  Capability.Word1[15]:=FALSE;     {Turn off System Manager flag}
  UNITWRITE(132,Capability,0,0,4); {Write new mask}
```

Reboot User Task

Any user can cause his task to be rebootstrapped to the current boot device number by doing a read (UNITREAD) or write (UNITWRITE) to device 132 with a block number of 0 and a control word of 5. This boot will use the override boot device number if it has been set, thus allowing a Login program to start up another environment. The buffer and size arguments are ignored.

If the supplied block number is non-zero, the user may cause another user to be rebooted. In this case the user causing the reboot must be configured to be a System Manager.

This feature is useful in the case where a user has mistakenly modified the keyboard mask and cannot type in characters to cause a reboot. A program running in a System Manager's task could then be written to revive the user without restarting the whole Multi-User system. The block number is the User Task number plus 1 of the user to be rebooted. If the first task is rebooted (block = 1), then the internal system management task will be restarted with indeterminate results.

```
Example:

VAR
  Dummy:INTEGER;
  Task:INTEGER;

BEGIN
  Task:=1;
  UNITWRITE(132,Dummy,0,Task+1,5); {Reboot first active user}
  UNITWRITE(132,Dummy,0,0,5);      {Reboot myself}
```

XII.05 USER INTERCOMMUNICATION :

A facility is provided under the SAGE Multi-User environment to send and receive data from one user to another. The implementation uses I/O requests to device 133. Under the p-System these may be accessed from a Pascal program via UNITWRITE, UNITREAD, UNITCLEAR, and UNITSTATUS.

Each user normally has a task number from 0 to 15. The target user's task number (plus one) is passed as the low byte of the Control parameter for writes, reads, and status. A task number of zero refers to one's own task number.

Both synchronous and asynchronous transmissions are provided. A configurable number of Transmit Queue Entries allows multiple asynchronous transmissions to be outstanding. Synchronous transmissions require that the data be received (or aborted) before the program may continue. Asynchronous transmissions allow the program to continue before the receiver actually reads the data. The program may then poll for completion or use the Data Transmitted Event (ATT_ComDn = event 51) to determine that the data has been transferred.

Each Transmit Queue Entry maintained by the BIOS contains information about the transmission. The actual data is not buffered, but is copied from the source user's UNITWRITE buffer address to the destination user's UNITREAD buffer address when the receiver does a UNITREAD. The UNITREAD may do partial transfers to allow flexible user buffer sizes or to set up a message protocol with a header.

A UNITSTATUS command allows the receiver to know how many messages are waiting as well as the source task number, size, and block number of the top message. The receiving task's Data Received Event (ATT_ComIn = event 50) is signaled when a transmission is initiated.

A one word Service Availability Mask is defined to help in connecting a user requiring a specific service with the users providing that service. The sixteen bits of the

Service Availability Mask may be used to represent any services provided by a user program. The most significant four bits may be used by future SAGE software utility programs and should be avoided by user systems if possible.

A special UNITWRITE is provided to specify the services available from the current user. A special UNITREAD is provided to find all the user tasks that can provide a specific service.

● Transmit Queue Detail

Each user task contains four linked lists for intercommunication queues. The first queue is a Free Queue. It maintains Transmit Queue Entries which are available for new transmissions. The second queue is the Sent Queue. When a UNITWRITE occurs to send information to another user, an entry is obtained from the Free Queue. Then information about the transmission is placed in the queue entry. The entry is then placed in the source user's Sent Queue and in the receiving user's Receive Queue (the third queue). Finally, when the receiving user reads all the data, the entry is placed in the sender's Done Queue (fourth queue).

If the transmission was synchronous the queue entry will be immediately moved to the Free Queue and the error status returned to the caller when the receiver reads all of the data. For asynchronous transmissions, the Data Transmitted Event (ATT_ComDn = event 51) is signaled if it was Attached. Otherwise the asynchronous transmitter will have to poll for transmit completion with a UNITSTATUS request described below.

Each user task has a configurable number of transmit queue entries which is maintained for each user task by MU.UTIL.CODE. The standard default is one queue entry per user.

UNITWRITE Detail

The form of the UNITWRITE to send data to a user is:

```
UNITWRITE(133,Buffer,Size,Block,Control);
```

The Buffer and Size fields are the normal information used in all UNITWRITE commands. The Block field has no meaning to the low level transmission but is available to the receiver for any application dependent purpose. The Control word is split into several fields as follows:

Control Word:

Bits 0-4: Destination task number plus one.

Zero refers to one's own task.

Bits 5-7: Reserved for future system address expansion.

Bit 8: Asynchronous flag.

Setting this bit allows the program to continue and not hang waiting for the receiver to fetch the data.

Bit 9: Abort Transmission flag.

This flag indicates that all transmissions to the specified destination should be removed from the receivers Received Queue and the transmitters Sent Queue and placed back into the transmitters Free Queue. This function should only be used after a long timeout when it is expected that the receiver is not functioning correctly. Note that this function is not necessary if using the automatic timeout described below.

Bit 10: Specify Service Availability Mask.

When this control bit is set, one word of data from the Block Number is saved in the task's Service Availability Mask. The Buffer and Size fields are unused.

Bit 11: Reserved for future use.

Bits 12-15: Timeout Value

This field contains an optional timeout value in seconds. If these bits are non-zero, a timeout will be initiated to insure that the message does not get hung. If the receiver fails to fetch all of the data before the timeout, the transmission will be aborted and an error status of 4 will be returned.

Note that if a timeout occurs or the transmission is aborted by the sender after the receiver has seen the message via a UNITSTATUS or a UNITREAD, a subsequent UNITREAD by the receiver will return a Transmission Aborted error code of 2.

IORESULT's for UNITWRITE

0	Normal (no error)
1	Could not get a Free Queue entry.
2	Destination task did not exist.
3	Receive was aborted by a UNITCLEAR.
4	Transmit timeout.
other	Application dependent error reply from receiver (passed back through block number).

UNITREAD Detail

The form of the UNITREAD to receive data from a task is:

```
UNITREAD(133,Buffer,Size,Block,Control);
```

This UNITREAD will never hang. If no data is available an IORESULT of 1 will be returned and no data will be transferred. The Buffer and Size fields have their normal meaning.

If Size is less than the amount of data available, Size bytes will be transferred and the Queue entry size and address will be modified appropriately and left at the head of the Receive Queue.

If Size is more than the data available, the trailing portion of the Buffer will be zero filled and the UNITREAD will complete. The Block number of the final read is returned to the sender as an Error reply code (in IORESULT).

If a non-zero Block number is specified, the reception is aborted and the error reply is returned to the sender. The Control word for reading user data is zero.

IORESULT's for UNITREAD

0	Normal (no error)
1	No data available
2	Transmission was aborted.

NOTE: IORESULT=2 only occurs if the receiver has had a chance to detect an awaiting message and the message is subsequently aborted by a timeout, a transmitter abort flag, or a transmitter's UNITCLEAR. In order for the receiver to have detected the message, it must have been at the top of the Receive Queue during

a UNITSTATUS for Read Information or a portion of the transmitted data must have already been read.

A special read for finding users with a specific service capability is initiated with bit 0 of the Control Word set to one. The requested Service Mask should be passed in the Block number. Note that if more than one bit is set, all requested bits must be present to find a match.

Sixteen words of data are always returned. Each word will contain the task number (plus one) of a user that can provide the requested service. Zero words indicate no additional users can provide the requested service.

If multiple users are found, they are not returned in any specific order. If no users are found, the first Buffer word will be returned zero. The size field is unused.

UNITSTATUS Detail

The UNITSTATUS procedure is used to get three types of status depending on the Control Word. This information is General Queue Information, Read Information, and Write Completion Information. The form of the UNITSTATUS request is:

```
UNITSTATUS(133,Status,Control);
```

● General Queue Information

To request this information the Control Word must have the high byte clear and the low byte must contain the user task number plus one (zero for own task).

Status Reply (byte offset)

0	Number of entries in Receive Queue (2 bytes)
2	Number of entries in Free Queue (2 bytes)
4	Number of entries in Sent Queue (2 bytes)
6	Number of entries in Done Queue (2 bytes)

● Read Information:

To request Read Information, the Control Word must have bit 8 set on.

Status Reply (byte offset)

0	Source Task Number (2 bytes) This value is the source task number plus one of the top entry in the Receive Queue. Zero means no Receive Queue entries exist.
2	Size of data transmission (4 bytes) Note that under the p-System UNITWRITE only the two least significant bytes may be used.
6	Block Number (4 bytes) Note that under the p-System UNITWRITE only the two least significant bytes are passed.
10.	I/O flag (2 bytes) This field is reserved for future use. Leave it zero for normal user to user communications.

● Write Information:

Checking the Write Information need only be done when asynchronous transmissions are outstanding. To request Write Information, the Control Word must have bit 9 set on. After the status is read, the Queue entry is moved from the Done Queue to the Free Queue.

Status Reply (byte offset)

0 Destination Task Number (2 bytes)
This value is the destination task number plus one of the top entry in the Done Queue. Zero means no Done Queue entries exist.

2 Error reply code (2 bytes)
The error reply codes are the same as the IORESULT codes for UNITWRITE operations.

4 Transmitted Block Number (4 bytes)
This is the value of the block number which was transmitted. It may be used as a sequence number to identify messages when multiple transmissions are outstanding. It is not the block number returned by the receiver. This is passed back as the Error Reply Code (offset 2). Note that the p-System only handles the least significant two bytes of the block number.

IORESULT's for UNITSTATUS

0 Normal (no error)
2 User task did not exist

UNITCLEAR Detail

A UNITCLEAR(133) aborts all a user's transmissions and returns all queue entries to their Free Queue. All entries in the user's Receive Queue are aborted and returned to their transmitting Task's Done Queue with an Error Reply Code of 3.

IORESULT's for UNITCLEAR

0	Normal (no error)
2	Non existent facility (not Multi-User system)

User Intercommunication Example

The following Sample User Intercommunication Program is only for demonstration of user to user communciations and is not generally useful in a real world environment. It reads characters from the keyboard, echos them locally and and sends them to another user. It also will receive characters from another user and display them on the terminal.

```
PROGRAM UserToUser;

CONST
Timeout = 5; { Number of seconds}
  Asynch  = 256; {Asynchronous transmission flag}

VAR
  TargetUser:INTEGER;
  DataToSend:BOOLEAN;
  CH:PACKED ARRAY[0..1] OF CHAR;
  CommStatus:ARRAY[0..29] OF INTEGER;
  TermStatus:ARRAY[0..29] OF INTEGER;

BEGIN
  WRITELN;
  WRITE('Target user number <0 to exit, -1 to wait for receive>: ');
  READLN(TargetUser);
  IF TargetUser = 0 THEN EXIT(PROGRAM);
  IF (TargetUser <= 15) AND (TargetUser > 0) THEN
    TargetUser:=TargetUser+1 {Task plus one}
  ELSE
    TargetUser:=-1; {Default to wait for incoming info}
  REPEAT
    IF NOT DataToSend THEN
      BEGIN
        UNITSTATUS(1,TermStatus,1); {Check for keyboard character}
        IF TermStatus[0] <> 0 THEN
          BEGIN
            UNITREAD(1,CH[0],1); {get one character}
            DataToSend:=TRUE;
          END;
      END
    ELSE
      BEGIN {Have data to send}
        IF TargetUser > 0 THEN
          BEGIN
            UNITWRITE(133,CH[0],1,0,(Timeout*4096)+Asynch+TargetUser);
            IF IORESULT <> 1 THEN DataToSend:=FALSE;
          END
        ELSE {Don't send data if no target is known}
          DataToSend:=FALSE;
      END;
    UNITSTATUS(133,CommStatus,512); {Check for completed transmission}
    IF CommStatus[0] <> 0 THEN
      IF CommStatus[1] <> 0 THEN
        BEGIN
          WRITELN;
          WRITELN('Transmission error ',CommStatus[1]);
        END;
    UNITSTATUS(133,CommStatus,256); {Check for receive}
    IF CommStatus[0] <> 0 THEN
      BEGIN
        { Check if waiting for a user to talk to }
        IF TargetUser < 0 THEN TargetUser:=CommStatus[0];
        UNITREAD(133,CH[1],1,0,0); {Read data from other user}
        IF IORESULT = 0 THEN
          UNITWRITE(1,CH[1],1); {Output to terminal}
      END;
  UNTIL FALSE;
END.
```

XII.06 TERMINAL EMULATOR CONFIGURATION :

MUTRMSET (Multi-User Terminal Set-up) is a configuration program which allows the user to build or modify a terminal definition file (typically named "TRMDEF.DATA") which is used by the SAGE Terminal Emulator.

The terminal definitions for several popular terminals have already been prepared and exist in the TRMDEF.DATA file and the standard Multi-User configuration files.

This section is only necessary for users who have an unsupported terminal and want to use the Shared Terminal (Terminal Emulator) feature.

The SAGE Terminal Emulator

The SAGE Terminal Emulator is an assembly language module which has been incorporated into the SAGE Multi-User BIOS, the SAGE Demonstration System, and other programs which need to know exactly what is being displayed on the user's terminal at any given time, or need to restore parts of the screen that have been overwritten by other information (such as windows).

The Terminal Emulator accomplishes these goals by keeping a copy of the terminal's screen in memory. It monitors all output to the terminal and updates its own "screen image" to match what it thinks is appearing on the user's terminal.

To maintain an accurate screen image, the Terminal Emulator must interpret control sequences and escape sequences exactly like the terminal would. It must know, for example, that an escape character (ESC) followed by an asterisk (*) is a "clear screen" sequence, and it must respond by clearing its screen image in memory to reflect the blank screen on the user's terminal.

To further complicate things, the Terminal Emulator must respond differently to various character sequences depending on which terminal it is emulating. The "clear screen"

sequence for a VT-100 terminal is "ESC [2 J", but for TeleVideo, Freedom, and Qume terminals it is "ESC *".

This is where the "Terminal Definition File" comes in. This file, typically named "TRMDEF.DATA", contains information which tells the Terminal Emulator what it should do when it receives certain escape and control sequences. With the proper information installed in this file, the Terminal Emulator can emulate a wide variety of terminals.

● MUTRMSET Overview

MUTRMSET (Multi-User Terminal Set-up) is a program which allows the user to build or modify the terminal definition file to suit his particular terminal. Be forewarned: **This is a very difficult process.** As computer terminals have become smarter, their control sequences have become less and less standardized. You may discover that terminals which claim to be completely compatible with other terminals aren't. In short, the process of setting up a new terminal definition shouldn't be attempted by anyone who isn't familiar with computers and intimately familiar with the nature of their terminal. Fortunately, "TRMDEF.DATA" already contains definitions for several of the more common terminals.

● Running MUTRMSET

To run MUTRMSET, X(ecute MUTRMSET from the p-System prompt. After the program initializes, it asks for the name of the "terminal definition file:"

```
Initializing .............
Name of terminal definition file <CR exits, * creates>:
```

The name of the normal terminal definition file is "TRMDEF.DATA". This file is a structured data file which contains 26 terminal definitions (some of these may be empty). If you wish to create a new terminal definition file with 26 empty records for new terminal definitions,

type an asterisk. This is rarely necessary, however, because the terminal definition file usually contains empty space where new terminal definitions can be placed.

After the program finds or creates the terminal definition file, it will display a directory of the terminal definitions it contains:

```
A - INITIAL              N - EMPTY
B - TELEVIDEO 925        O - EMPTY
C - FREEDOM 100          P - EMPTY
D - VT 100               Q - EMPTY
E - EMPTY                R - EMPTY
F - EMPTY                S - EMPTY
G - EMPTY                T - EMPTY
H - EMPTY                U - EMPTY
I - EMPTY                V - EMPTY
J - EMPTY                W - EMPTY
K - EMPTY                X - EMPTY
L - EMPTY                Y - EMPTY
M - EMPTY                Z - EMPTY

Source record < A to clear> ?
```

In this example, three terminals have been defined (for the TeleVideo 925, the Freedom 100, and the VT-100). Records E through Z have not been defined yet. Record A is a special "initial" record that can be used as a fresh starting point to define new records. It is essentially an empty record, with all its data cleared to default values. Record A cannot be modified, so you can always rely on it as a "clean slate," unlike EMPTY records which might not necessarily be cleared.

MUTRMSET asks you for a letter corresponding to a source record.

Note that this reply and the reply to the following "Destination record" prompt must be terminated with a carriage return.

If you are defining a totally new type of terminal, type "A" to get a fresh start. If, on the other hand, one of the terminals already defined shares some characteristics with your new terminal (Freedom 100 and TeleVideo, or VT-100 and Visual 300, for example), type the letter corresponding to

that terminal record. The computer will load the data for that terminal so that you won't have to re-define all the characteristics that the two terminals have in common. Now pick the letter corresponding to one of the EMPTY records (or an obsolete record) as your "destination" record. This is where your new record will be stored. The computer will ask you to name your new record.

If you simply want to make some adjustments to a record that has already been defined (i.e., you want to "edit" it), type the appropriate letter as both the source and destination records, and type a carriage return instead of a new name (unless, of course, you want to change the name too).

The MUTRMSET main menu will now appear on your screen. The definition process is divided into four activities:

```
        Multi-User Terminal Set-up

A - Define control character functions
B - Define escape sequence functions
C - Define output sequences
D - Define terminal characteristics

Select Menu item <CR exits, ! aborts>:
```

A defines how various control characters affect the terminal (screen-image).

B defines how various escape sequences affect the terminal (screen-image).

C defines character sequences which the Terminal Emulator will need to make your terminal home the cursor, clear the screen, etc.

D defines other terminal characteristics such as line width, lines per screen, etc.

Each of these choices must be chosen at some point to completely define a record. We will examine each submenu individually in a moment.

You will be returned to this main menu after you complete your work in each submenu. When you are finished defining the items in all four submenus, type a carriage return to exit and save your new definition on disk. If you do an abort exit (type "!"), your changes will not be saved.

MUTRMSET Submenus

A - Define Control Character Functions

When you choose this activity, a submenu appears which contains all the control characters (ASCII hex values 01 through 1F). It is now up to you to define all the control characters your terminal recognizes.

```
Define Control Character Functions

A - CTRL-A  01H   IGNORE    Q - CTRL-Q  11H   IGNORE
B - CTRL-B  02H   IGNORE    R - CTRL-R  12H   IGNORE
C - CTRL-C  03H   IGNORE    S - CTRL-S  13H   IGNORE
D - CTRL-D  04H   IGNORE    T - CTRL-T  14H   IGNORE
E - CTRL-E  05H   IGNORE    U - CTRL-U  15H   IGNORE
F - CTRL-F  06H   IGNORE    V - CTRL-V  16H   IGNORE
G - CTRL-G  07H   IGNORE    W - CTRL-W  17H   IGNORE
H - CTRL-H  08H   CLFWRP    X - CTRL-X  18H   IGNORE
I - CTRL-I  09H    CTAB     Y - CTRL-Y  19H   IGNORE
J - CTRL-J  0AH   CDNSCR    Z - CTRL-Z  1AH ERSCRATR
K - CTRL-K  0BH      CUP    0 - CTRL-[  1BH   ESCSEQ
L - CTRL-L  0CH   CRTWRP    1 - CTRL-\  1CH   IGNORE
M - CTRL-M  0DH      CCR    2 - CTRL-]  1DH   IGNORE
N - CTRL-N  0EH   IGNORE    3 - CTRL-^  1EH    CHOME
O - CTRL-O  0FH   IGNORE    4 - CTRL-_  1FH    CNEWL
P - CTRL-P  10H   IGNORE

Select Menu item <CR exits, ! aborts>:
```

The Terminal Emulator will use the information you give it here to mimic the actions of your terminal in memory. Suppose that CTRL-^ (1EH) is the code your terminal recognizes as a "home cursor" command. Type the corresponding menu letter/number ("3" in this case), then type "CHOME" as the name of the cursor home function. Each function has a corresponding name, given in the following table.

Note: Sequences of characters that are not understood or should be ignored by the Terminal Emulator must be specifically ignored using the "IGNORE" functions under the "Miscellaneous" section of this table.

Also: If your terminal uses any escape sequences (multiple character control sequences), escape sequence processing must be enabled by placing the "ESCSEQ" function in the entry for the escape character (usually item 0). If "ESCSEQ" is not placed in this entry, no escape processing will be performed, and you won't need to define anything in the "Define escape sequence functions" menu since it won't be used.

Cursor movement functions

CUP	Moves the cursor up one line.
CUPSCR	Moves the cursor up one line and scrolls the screen down one line if the cursor is on the first line.
CDN	Moves the cursor down one line.
CDNSCR	Moves the cursor down one line. Scrolls the screen up one line if the cursor is on the last line.
CLF	Moves the cursor left one column.
CLFWRP	Moves the cursor left one column. On a " wrapping " terminal (defined in menu D), it wraps around to the previous line if the cursor was at the beginning of the current line. Nothing is done if at "home".
CRT	Moves the cursor right one column
CRTWRP	Move the cursor right one column. On a " wrapping " terminal, it wraps around to next line if the cursor was at the end of the current line. It scrolls the screen up one line if cursor is at end of screen.
CCR	Cursor carriage return. It will position the cursor at the beginning of current line.
CHOME	Move cursor to upper left-hand corner of screen.
CNEWL	Position cursor at start of the following line and scroll if necessary.
CGOTORC	Move cursor to the row & column position indicated by the next two ASCII characters. (Minus the ASCII coordinate conversion value defined in menu D.)
CGOTOR	Move cursor to the row position as indicated by the following ASCII character minus the ASCII coordinate conversion value.
CGOTOC	Move cursor to the column position indicated by the following ASCII character minus the ASCII coordinate conversion value.
CTAB	Advance cursor to the next tab setting. If there is no tab setting following the cursor, then the cursor stays where it is unless an item in menu D specifies movement to the end of the current line.
CBKTAB	Move cursor to the previous tab stop or beginning of current line if no previous tab stop.

Miscellaneous tab functions

SETTAB	Set a tab stop at the current cursor position.
CLRTAB	Clear a tab stop at the current cursor position.
CLRALLT	Clear all tab stops.

Erasure functions

ERBOL	Erase from the cursor to beginning of line with fill character.
EREOL	Erase from cursor to end of line with fill character
EREOLNUL	Erase from cursor to end of line with null character
ERLINE	Erase cursor's line with fill character
ERBOP	Erase from the cursor to beginning of page with fill character.
EREOP	Erase from cursor to end of page with fill character
EREOPNUL	Erase from cursor to end of page with null character
ERSCRATR	Erase entire screen with fill character and clear current character attribute setting also.
ERSCRNUL	Erase screen with null character.

Screen attribute functions

CATR	Set the cursor attributes according to the following ASCII code.
SETATR	Set one or more screen attributes according to the following ASCII code.
SETAT0	Set screen attribute 0
CLRAT0	Clear screen attribute 0
SETAT1	Set screen attribute 1
CLRAT1	Clear screen attribute 1
SETAT2	Set screen attribute 2
CLRAT2	Clear screen attribute 2
SETAT3	Set screen attribute 3
CLRAT3	Clear screen attribute 3
SETAT4	Set screen attribute 4
CLRAT4	Clear screen attribute 4
SETAT5	Set screen attribute 5
CLRAT5	Clear screen attribute 5
SETAT6	Set screen attribute 6
CLRAT6	Clear screen attribute 6

SETAT7 Set screen attribute 7
CLRAT7 Clear screen attribute 7
SETREV Set whole screen to reverse video
CLRREV Set whole screen to normal video

Note: For further information on screen attribute functions, see "Handling Screen Attributes" section of this document.

User/status line functions

LOADUSER Load user line with characters that follow. There will be 79 (TeleVideo) or 80 characters or less ending with a carriage return.
SHOWUSER Show user line
SHOWSTAT Show status line

Editing functions

INSCHR Insert a fill character at the cursor
DELCHR Delete a character at the cursor
INSLIN Insert a line before the cursor's line
DELLIN Delete cursor's line
INSON Turn character insert mode on. The characters are inserted on the screen instead of overwriting the previous characters.
INSOFF Turn the character insert mode off. The characters overwrite previous characters.

Miscellaneous

ESCSEQ This control character is the escape character. It is followed by an escape sequence as defined in the "Define escape sequence functions menu".

DSPCTRL Display the following control character code.

LOADFILL Define the fill character as the following ASCII character.

PARSANSI Parse the ANSI parameter list of the form ESC [99 ; 99 X where X is an ANSI command, 99 is a variable numeric parameter, and ; is a separator. Fifteen parameters maximum.

IGNORE Ignore this character

IGNORE2 Ignore this character and the next

IGNORE3 Ignore this character and two more

IGNORE4 Ignore this character and three more

IGNORE5 Ignore this character and four more

IGNORE6 Ignore this character and five more

IGNORE7 Ignore this character and six more

Note: Sequences of characters that are not understood by the Terminal Emulator must be specifically ignored using the above functions.

B - Define Escape Sequence Functions

There are 96 different escape sequences you can define using this part of MUTRMSET. Because there are so many, they have been split into three sub-menus. When you enter this section, a menu asks you which of these three sub-menus you wish to work on.

```
Define Escape Sequence Functions

A - ESC 20H through ESC 40H
B - ESC 41H through ESC 60H
C - ESC 61H through ESC 7FH

Select Menu item <CR exits, ! aborts>:
```

When you select one of these, a menu will appear on your screen which lists escape sequences like this: A - ESC SP 20H IGNORE. Dissecting this, we see that "A" represents the item's identification letter in this menu, "ESC SP" are the first two characters of this sequence (an escape and a space), "20H" is the ASCII hex code of the second character (the first character is always an escape), and "IGNORE" is the name of the function which would be called to handle this sequence -- in this case the sequence is simply ignored.

The method of defining escape sequences is the same as the method used to define control characters. Type the letter/number which identifies the sequence you wish to define, then type the name of the function which will emulate your terminal's response to the given sequence. The function names are the same as those used to specify control functions -- refer to the previous table.

Note: To ignore a two-character escape sequence, use "IGNORE" to ignore the second character (the escape has already been processed). For example, use "IGNORE7" to ignore the TeleVideo "set time" sequence ESC SP 1 nnnnn (which has eight characters in all).

If you have an ANSI terminal, be sure to place the "PARSANSI" function in the "ESC [" entry of the "Define escape sequence functions" menu. This will enable processing of the standard ANSI sequences.

```
      Define Escape Sequences  ESC 20H - ESC 40H

A - ESC SP  20H   IGNORE          R - ESC 1   31H   SETTAB
B - ESC !   21H   IGNORE          S - ESC 2   32H   CLRTAB
C - ESC "   22H   IGNORE          T - ESC 3   33H   CLRALLT
D - ESC #   23H   IGNORE          U - ESC 4   34H   IGNORE
E - ESC $   24H   SETATR5         V - ESC 5   35H   IGNORE
F - ESC %   25H   CLRATR5         W - ESC 6   36H   IGNORE
G - ESC &   26H   IGNORE          X - ESC 7   37H   IGNORE
H - ESC '   27H   IGNORE          Y - ESC 8   38H   IGNORE
I - ESC (   28H   CLRATR6         Z - ESC 9   39H   IGNORE
J - ESC )   29H   SETATR6         0 - ESC :   3AH ERSCRATR
K - ESC *   2AH ERSCRATR         1 - ESC ;   3BH ERSCRATR
L - ESC +   2BH ERSCRATR         2 - ESC <   3CH   IGNORE
M - ESC ,   2CH ERSCRATR         3 - ESC =   3DH   CGOTORC
N - ESC -   2DH   IGNORE          4 - ESC >   3EH   IGNORE
O - ESC .   2EH   IGNORE          5 - ESC ?   3FH   IGNORE
P - ESC /   2FH   IGNORE          6 - ESC @   40H   IGNORE
Q - ESC 0   30H   IGNORE

Select Menu item <CR exits, ! aborts>:
```

C - Define Output Sequences

This section is also divided into submenus. Select Part A, B, or C.

```
Define Output Sequences

A - Part A (General)
B - Part B (Screen attributes)
C - Part C (Window characters)

Select Menu item <CR exits, ! aborts>:
```

Each of the submenus asks you to input a character sequence which the Terminal Emulator needs to accomplish certain functions. For example, the Terminal Emulator might need to clear the screen of your terminal. Since different terminals have different commands to do this, and since there are often several ways to accomplish this on the same terminal (a clear screen character vs. home up, clear to end page sequence), the "Define Output Sequences" portion of MUTRMSET allows you to tell the Terminal Emulator how each function can be accomplished most efficiently (i.e.,

requiring the fewest characters) on your terminal.

Simply select a menu item by typing its corresponding letter, then type the character sequence (1 to 4 characters separated by spaces). Special characters such as control characters, rubout, or a space can be denoted by their ASCII name (ESC, DEL, SP, CR, etc.), by their hex value (two hex digits followed by an 'H' -- 1BH, 7FH, 20H, 0DH, etc.), or their control representation (an up-arrow followed by an alphabetic character -- ^A,^J,^M).

There is only one way to enter a space in your sequence (since extra spaces are ignored). Use its ASCII name "SP".

If your terminal doesn't have a method of performing a certain function (such as showing the user line), just leave that sequence blank. (If there is already an entry there, you should clear it by typing the item's identifier, a single space, and a carriage return.) Items A through E of the first submenu ("Part A") are the only sequences that are essential to the operation of the Terminal Emulator (they have an asterisk in front of them to remind you); the other sequences are used to implement less essential but nonetheless desirable capabilities.

Part A (further explanation)

```
          Define Output Sequences, Part A

A - * Cursor home
B - * Cursor down                          LF
C - * Cursor right                          L
D - * Cursor new line
E - * Clear screen                         Z
F - Clear all tabs                     ESC 3
G - Set tab at cursor                  ESC 1
H - Normal video
I - Reverse video
J - Show status line                   ESC }
K - Show user line                     ESC {
L - Load user line
M - Change cursor attr. (start of seq.)
N - Invisible cursor (preceded by M)
O - Select fill char. (start of seq.)
P - Normal fill char. (preceded by O)
Q - Screen overwrite mode
R - Screen insert mode

Select Menu item <CR exits, ! aborts>:
```

H -Normal video This sequence sets the whole screen to normal video mode (white or green on black)

I -Reverse video Sets the whole screen to reverse video mode (black on white or green background)

M -Change cursor attr. This is the beginning of the sequence to change the cursor's attributes (blinking block, steady underline, invisible, etc.) The sequence you type should not include character(s) that actually specifies the cursor's attributes. This is simply the start of the sequence.

N -Invisible cursor This is the character(s) that follows the previous sequence to make the cursor invisible.

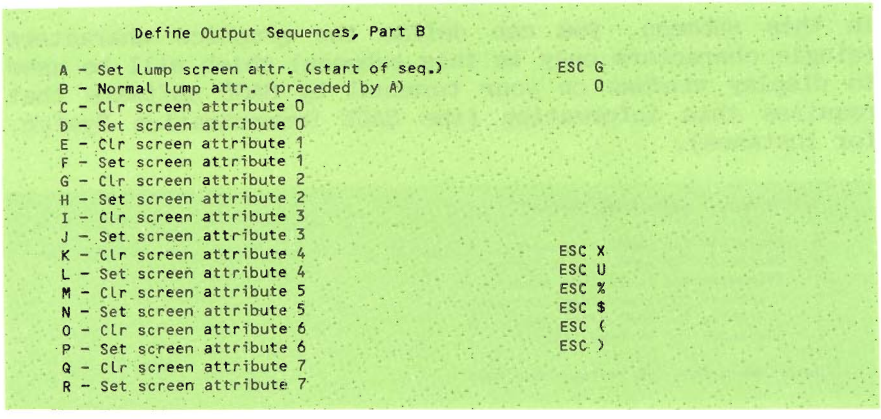
O -Select fill char. This is the beginning of the sequence to specify the fill character (used by some terminals in screen erasures and edits).

P -Normal fill char. This is the character(s) that specifies the normal fill character (usually a space) when preceded by the previous sequence.

Q -Screen insert mode Sequence of characters to turn on a special terminal mode where characters sent to the terminal do not replace (overwrite) previous characters, but are inserted in the current line.

R -Screen overwrite mode Restores normal overwrite operation of the terminal.

Part B (further explanation)



Select Menu item <CR exits, ! aborts>:

See "Handling Screen Attributes" portion of this document.

Part C (further explanation)

In this submenu, you can define the graphics characters (single characters only in this submenu) which will be used to display windows on your terminal by SAGE software that requires this information (the SAGE Demonstration System, for instance).

```
Define Output Sequences, Part C

A - Character for upper left corner  f
B - Character for upper right corner g
C - Character for lower left corner  e
D - Character for lower right corner h
E - Character for horizontal line    k
F - Character for vertical line      j

Select Menu item <CR exits, ! aborts>:
```

If your terminal has line graphics characters, you may want to take advantage of these by specifying the normal characters which will be output as graphics characters when the terminal is in graphics mode. SAGE window-oriented software will put your terminal in graphics mode before attempting to output these characters.

If your terminal does not have line graphics capability, type a vertical bar character (|) for the "Character for vertical line", and dashes (-) for the rest. Window-oriented programs will use this information to display windows like this:

```
-------------------------------
| This is a window.           |
-------------------------------
```

D - Define Terminal Characteristics

This menu allows you to define miscellaneous characteristics of your terminal.

```
                    Define Terminal Characteristics

  A - Terminal Width (char/ln)   80        L - Lump alterable attributes
  B - Terminal Height (ln/scrn)  24        M - ASCII conv. for lump char
  C - ANSI parameter sequences   Off       N - Bit mask for monitor mode
  D - Visible lump char.         On        O - Lump bit affecting attr 0
  E - Cursor wrap-around         On        P - Lump bit affecting attr 1
  F - Home cursor on clr scrn    On        Q - Lump bit affecting attr 2
  G - Move curs. on no tab stop  Off       R - Lump bit affecting attr 3
  H - Coord. conversion code     20        S - Lump bit affecting attr 4
  I - ASCII normal cursor attr.  00        T - Lump bit affecting attr 5
  J - Separately altered attr.   70        U - Lump bit affecting attr 6
  K - Attr. cleared together     00        V - Lump bit affecting attr 7

  Select Menu item <CR exits, ! aborts>:
```

A - Terminal Width — Input the terminal's number of characters per line (decimal)

B - Terminal Height — Input the terminal's number of lines per screen (decimal, not including status/user line)

C - ANSI sequences — If your terminal recognizes ANSI parameter sequences of the form ESC [... X, turn this option on. If you have an ANSI terminal, be sure to place the "PARSANSI" function in the "ESC [" entry of the "Define escape sequence functions" menu. This will enable processing of the standard ANSI sequences.

D - Visible lump char.

Terminals such as the TeleVideo and Qume require a character position on screen whenever the screen attributes are changed. If you have such a terminal, turn this option on. Terminals such as the Freedom 100 have invisible character attribute changes, and require this option to be turned off.

E - Cursor wrap-around

Turn this option on if the cursor automatically proceeds to the next line after reaching the end of the current one.

F - Home curs. on clr scrn

If your terminal's cursor homes automatically when you erase the screen, turn this option on.

G - Move curs. on no tab

If your terminal's cursor moves to the end of the current line when the terminal receives a "tab" code after the last tab stop, turn this option on.

H - Coord. code

This is a hex value which is subtracted from row and column coordinates during a cursor "gotoxy" command to obtain the correct coordinate values (starting at 0,0 in the upper left-hand corner of the screen)

I - ASCII norm curs attr This is the ASCII hex code that is transmitted after a "change cursor attributes" sequence to get a normal cursor (typically a blinking block).

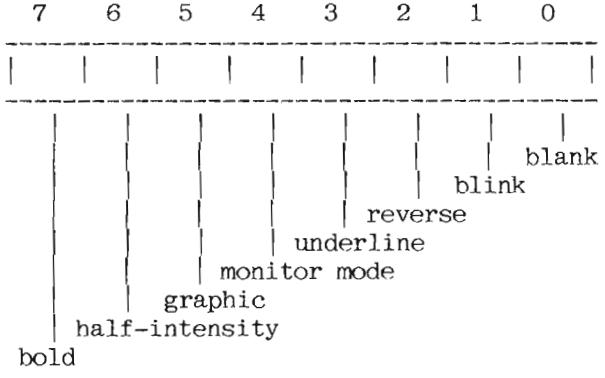
J through V See following section.

● Handling Screen Attributes

The support of various screen attributes is the most complicated part of the Terminal Emulator. Screen attributes are qualities attached to characters on the screen such as "blink", "blank", "reverse", "underline", "graphic", "monitor mode", "half-intensity", "bold", etc. The Terminal Emulator supports up to 8 of these attributes (256 possible combinations).

To take advantage of the attributes available on your terminal, you must first assign each a bit-position (0-7) within a byte. Here's one such assignment (it's pretty arbitrary):

Attribute number:



Let us stress the fact that the attributes of your particular terminal and the bit positions you assign to them can be completely arbitrary.

Now decide which bits (attributes) can be set (turned on) separately (independently of the other attributes). Refer to your terminal manual. Form a bit mask with ones in the bit position of those attributes.

On a TeleVideo, for example, monitor mode and half-intensity can be set independently of all other attributes, so we form a bit-mask with ones in the positions we just chose for those attributes: 01010000 = 50H. Load the hex equivalent of your bit-mask into menu D, item J.

Now that you have decided (based on the terminal manual) which attributes can be turned **on** separately, you must consider how they are turned **off** . On some terminals, such as the VT-100, some attributes which can be turned on separately can only be turned off in a group (not independently of other attributes).

If some of your terminal's attributes fit this description, form a second bit mask with ones in the bit positions of all attributes that can be turned on separately but must be turned off together, and load the corresponding hex code into menu D, item K.

In the case of the VT-100, bold, underline, and reverse have to be turned off together, so the hex code would be 1CH according to our previous bit-attribute assignment.

Next, decide which attributes can be set or cleared in a single lump command. A Freedom 100 can alter its blank, blink, reverse, underline, and half-intensity attributes in a single command. The corresponding bit-mask, in this case, is 4FH, which is loaded into menu D, item L.

A single ASCII character is usually used to set or reset all the "lump" attributes for which we just formed a mask. This character contains two types of bits: bits which are used to set or reset screen attributes, and extraneous ASCII bits whose sole purpose is to make the lump character an acceptable ASCII character. We remove these extraneous bits by subtracting the "ASCII conversion code" in item M.

The remaining bits are used to set or reset screen attributes directly. Since these bits might not be in the same bit-positions as the assignments you made, a mapping mechanism is provided to insure that the bits in the lump

character affect the proper bits (attributes) in your assignment.

If your attribute bit 0 is affected by the lump character, determine which bit in the lump character sets and resets it. Form a bit-mask with a one in the bit position of the proper lump bit, and load the hex code for this mask into item O. If your attribute bit 1 is affected by the lump character, put a bit-mask indicating which lump bit affects it in item P, and so forth. If your attribute bit 7 is affected by the lump character, the mapping for it goes in item V. Only lump-alterable attributes should have mappings; the rest of the entries should be zero.

The Terminal Emulator needs to know which bit in your bit assignment represent "monitor mode". If your terminal has a monitor mode, and if you have included it in your bit assignment, put a mask with a one in the bit position you have assigned for monitor mode into item N. If your terminal does not have monitor mode, or you don't wish to deal with it, type 0 for item N.

Once you have decided on an assignment of attributes to bits (that's our arbitrary scheme, not the mapping), we can go back to menu C, Part B and tell the Terminal Emulator how to set and reset those attributes which can be altered separately. If any of these attributes has to be cleared together, they should all have the same clearing sequence in the output sequences table.

> Caution: Make sure that the attribute functions specified in the control and escape sequence menus agree with your bit attribute assignment.

XIII BIOS INTERFACE :

BIOS stands for Basic Input Output System. The BIOS is the software interface between the hardware in the system and the operating system programs.

The SAGE BIOS is written in assembly language for the 68000 and is only **directly** accessible from assembly language routines. Many of the BIOS features are accessible from the p-System via the low level I/O interface routines UNITREAD, UNITWRITE, UNITSTATUS, and UNITCLEAR.

This section is important for those needing more information on how the hardware is accessed on the SAGE or for those users who wish to develop their own assembly language programs which deal with I/O.

SAGE supports both a single-user and Multi-User BIOS. The single-user BIOS is contained in the file SYSTEM.BIOS while the multi-user BIOS is contained in the file MU4.BIOS. Most of the assembly level BIOS calls will work on either system.

XIII.01 PHYSICAL DEVICE NUMBERS :

Every hardware device is assigned a PHYSICAL DEVICE NUMBER by the BIOS which is used to access that device. They are often "mapped" from another number called the LOGICAL DEVICE NUMBER which is specified by the operating system.

The table on page 35 shows the standard system PHYSICAL DEVICE NUMBERS and their usage for all operating systems.

Note that some physical device numbers (for the auxiliary ports) are different from the single user and multi-user systems.

Also note that the IEEE-488 bus is not currently supported as a BIOS device. Instead an assembly code routine for accessing the 488 interface is distributed (source) on the UTILITY diskette. This routine and a UNIT for use under the p-System are described on page 191.

The relationship between the physical device numbers and the "volume" number by which it is called in the p-System is given in the table on page 93.

XIII.02 BIOS MEMORY LOCATION :

The SAGE BIOS is read into the highest available RAM memory by the system bootstrap. It is accessed using the 68000 TRAP instructions which provide a position independent interface. The user never has to know where the BIOS is located. The TRAP entry puts the BIOS into the processor's Supervisor mode. This allows the BIOS to use a different stack area than the p-System which runs in the processor's USER mode. The Supervisor stack is allocated just below the BIOS. Also the hardware I/O locations (FFCxxx) are only accessible when in SUPERVISOR mode. Access to this area while in USER mode results in an 'EXCEPTION: Bus Error'.

In order to provide memory size optimizations for different equipment configurations the initialization philosophy of the location BIOSBASE has changed (since 8-AUG-83 release). Previously the bootstrap which loaded the BIOS set up BIOSBASE from size information contained in the BIOS header. This occured before the BIOS initialization was called. Now the BIOSBASE location is set up according to memory requirements by the BIOS initialization. An area of 512 bytes is allocated for the System Stack just above BIOSBASE although the system stack pointer is not initialized there. The bootstrap must set up the system stack pointer (typically to the address in BIOSBASE plus 512) after the BIOS initialization call.

Note that the BIOS and buffer size in the BIOS header are still maintained, but are worst case values. The old bootstraps will continue to work but will not have the memory optimization for the Floppy only systems.

XIII.03 SHORT CALLING SEQUENCE :

The SAGE BIOS has two different calling sequences, a short method and a long method. The short method provides an easy interface for simple single character I/O. The long method is used for other operations or where more information is necessary. (See page 449.)

The short calling sequence uses individual TRAP entry points to handle single character I/O to the keyboard, terminal, printer, and remote serial channel. The data is always handled in the low byte of D0. All registers are preserved except for the returned data in D0 for reads.

TRAP #8. Test I/O Queue

This function returns a condition code to indicate if a specified channel has characters for input or has room for a character on output. The channel is specified by a channel number in the low byte of D0. The "equal to" condition is returned if no characters are queued for input channels or if no room exists for characters in output channels. The NE condition is returned if input characters exist or room exists for output characters. If the channel is undefined the routine always returns the "equal to" condition.

The channel numbers are:

1 - Keyboard input queue
2 - Terminal output queue
6 - Printer output queue
7 - Remote serial input queue
8 - Remote serial output queue

Example:

```
MOVEQ   #1,D0           ;Keyboard device number
TRAP    #8.             ;Check for keyboard input
BNE     GOTCHAR         ;Go handle the keyboard character
```

TRAP #9. Read Character from Keyboard

This TRAP pops the top character from the keyboard queue and returns it in the low byte of D0. If there are no characters in the queue the routine waits until a character is typed before returning.

TRAP #10. Write Character to Terminal

This TRAP outputs the character from the low byte of D0 to the terminal output queue. If the terminal output queue is full, the routine waits until there is room for the character in the queue before returning.

Example:

```
;       Output a sequence of characters to the terminal which
;       is pointed to by A0.  The sequence is terminated by a
;       zero byte.  Routine modifies D0 and A0.
TEXTOUT MOVE.B  (A0)+,D0
        BEQ.S   $5              ;Found terminator
        TRAP    #10.            ;Output the character to BIOS
        BRA     TEXTOUT         ;Loop back for more characters
$5      RTS                     ;Done with text output

;       Sample usage
        LEA     MESSAGE,A0
        BSR     TEXTOUT
        ------

MESSAGE .ASCII  "This is a sample message"
        .BYTE   0               ;Terminator
```

TRAP #11. Write Character to Printer

This TRAP outputs the character from the low byte of D0 to the printer output queue. If the printer output queue is full, the routine waits until there is room for the character in the queue before returning.

The BIOS prior to 8-AUG-83 returned the EQ/NE status flag according to the existence of the printer. This seldom used protocol has been eliminated to provide consistency with the other direct TRAP calls (due to possible reassignment of channels.) To determine if the printer exists, examine the Channel Map information for logical channel 6.

TRAP #12. Read Character from Remote Serial Channel

This TRAP pops the top character from the remote channel input queue and returns it in the low byte of D0. If there are no characters in the queue the routine waits until a character is typed before returning.

TRAP #13. Write Character to Remote Serial Channel

This TRAP outputs the character from the low byte of D0 to the remote serial channel output queue. If the output queue is full, the routine waits until there is room for the character in the queue before returning.

XIII.04 LONG CALLING SEQUENCE :

The long calling sequence uses the TRAP #14 instruction for entry into the BIOS. For this sequence, the low word of D0 is a function code. A0 points to an argument block for those functions where additional information is required. Functions which perform I/O to a channel will have the Logical Channel Number in the first word of the argument block. The BIOS uses this number to find the corresponding Physical Device Number in the BIOS Channel Map to determine exactly which device is being accessed.

All registers (including the status register) are preserved. Example:

```
;       Read the left floppy disk directory
        LEA     ARGAREA,A0      ;Set up arguments for BIOS call
        MOVE.W  #4,(A0)         ;Channel number of floppy
        MOVE.L  #2048.,4(A0)    ;Size = 4 blocks (2K bytes)
        PEA     BUFFER          ;Form the address of the buffer
        MOVE.L  (SP)+,8(A0)     ;Put address into arguments
        MOVE.W  #2,12.(A0)      ;Directory is at block 2
        CLR.W   14.(A0)         ;No special control options
        CLR.W   16.(A0)         ;High block number is zero
        MOVEQ   #11.,D0         ;Read is Function 11
        TRAP    #14.            ;BIOS call
        TST.W   2(A0)           ;Check error reply code
        BNE.S   $10             ;Had error
        ------
ARGAREA .BLOCK  18.
```

BIOS Functions

Function=0..... Exit Back to the Debugger

D0=0
A0=Addr of Arguments: none
TRAP #14

This function aborts the currently running program and enters the PROM Debugger. There is never a return from the TRAP with this function. A delay of about one second is allowed for interrupt I/O to complete. The BIOS is aborted and the processor interrupts are disabled. Debugger I/O reverts back to the noninterrupt driven routines.

Function=1..... Reinitialize the BIOS
D0=1
A0=Addr of Arguments: none
TRAP #14

This function completely reinitializes the BIOS routines, losing any I/O data which may have been in progress. All BIOS parameters are set back to their default values which were originally loaded with the BIOS.

Function=2..... Return the Address of the Top of Memory
D0=2
A0=reply value.
TRAP #14

The BIOS and supervisor stack are always loaded at the top of RAM memory. This function returns the address of the base of the BIOS and stack area. The user's software operating system or other environment can only use memory below this location (and above 400H).

Function=3.....Install BIOS Terminal I/O into Debugger
D0=3
A0=Arguments: none
TRAP #14

The Debugger defaults to a built-in set of non-interrupt driven terminal I/O routines. If it is desired to use Debugger services while the BIOS is in operation, it is necessary to have the Debugger use the interrupt driven BIOS routines for its terminal operations. This function sets a flag to the Debugger which will cause it to use BIOS terminal I/O operations.

Function=4.....Disable BIOS Terminal I/O for Debugger
D0=4
A0=Arguments: none
TRAP #14

This function clears the Debugger flag enabled by function 3 described above. The Debugger will revert to using the non-interrupt driven terminal I/O routines.

Function=5.....Read System Clock
D0=5
A0=addr of Arguments:

Offset 0 - long word count in 1/64000ths of a second.
Offset 4 - long word count of seconds.

TRAP #14

This function reads back a time interval since BIOS startup. The interval is presented with one word in increments of 1/64000 of a second and a long word in seconds.

Function=6..... Read Clock in 1/60 second

D0=6
A0=addr of arguments:

Offset 0 - long word count of 1/60 ths of a second.

TRAP #14

An alternate system clock presentation is provided as a long word interval from system startup in increments of 1/60 th of a second.

Function=7..... Schedule an Event
D0=7
A0=a pointer to the schedule entry:
Offset 0: Long word reserved for Scheduler linkage.
Offset 4: Long word address for scheduled routine.
Offset 8: Long word relative timeout time from entry into Scheduler. (High word is in seconds, low word is in 1/64000 ths of a second).
TRAP #14

This function enters a standard 12 byte Scheduler entry into the Scheduler queue. On timeout the scheduled routine is called with registers D0-D2 and address registers A0-A2 are preserved. The routine should perserve any of the other registers that it uses and do an RTS when complete. The routine will be executed in Foreground.

Function=8.....Cancel a Schedule
D0=8
A0=a pointer to the schedule entry.
TRAP #14

This function will remove the scheduled entry from the Sheduler queue.

Function=9.....Load 68000 Object Program

D0=9
A0=addr of arguments:
Offset: 0 - Channel Number (word)
Offset; 2 - Error reply code (word)
Offset: 4 - Size in Bytes (long word)
Offset: 8 - Memory Address (long word)
Offset: 12 - Low word of Logical Block Number (word)
Offset: 14 - Control word (word)
(see description under Read function below)
Offset: 16 - Start Execution Address (long word)
Offset: 20 - High word of Logical Block Number (word)

TRAP #14

This function allows a 68000 assembly program to chain to another program from a storage device. The BIOS TRAP call never returns to the calling program unless an error occurs while reading in the new program. The arguments are stored in the BIOS area and the calling program may be completely overwritten by the new program (caution: do not overwrite the BIOS).

Note:only Channel numbers of mass storage devices are supported.

BIOS INTERFACE
LONG CALLING SEQUENCE

Function=10.....Initialize a Channel

D0=10
A0=addr of arguments:
   Offset: 0 - Channel Number (word)
   Offset: 2 - Error reply code (word)
   Offset: 4 - depends on channel

TRAP #14

Channel Specific Data:

The Keyboard and Terminal channels (1 and 2) are initialized together. When channels 1 or 2 are initialized, three extra arguments must be provided.

Argument offset 4 (for channels 1 & 2 only) must contain the long word address of a routine to run when the Break character is typed. Note that this Break routine is called within an interrupt routine and must save all registers.

Argument offset 8 (for channels 1 & 2 only) must contain the long word address of the special characters for Flush, Start/Stop, and Break as well as the Character Mask. Note that there is an extra byte between the Flush and Start/Stop characters and 7 bytes between the Break and Character Mask. All input keyboard input characters are ANDED with the Character Mask.

	Default character
Flush	Ctrl F
unused (1 byte)	
Start/Stop	Ctrl S
Break	Ctrl @
unused (7 bytes)	
Character Mask	7FH

Argument offset 12. (for channels 1 & 2 only) must contain the long word address of a byte containing the No Break flag. The No Break flag is bit 6 of the byte and when set will cause the Break character to be ignored and not call the Break routine.

Function=11..... Read Data from a Channel

D0=11
A0=addr of arguments:
Offset: 0 - Channel Number (word)
Offset: 2 - Error reply code (word)
Offset: 4 - Size in Bytes (long word)
Offset: 8 - Memory Address (long word)
Offset: 12 - Low Word of Logical Block Number (word)
Offset: 14 - Control word (word)
Offset: 16 - High Word of Logical Block Number (word)

TRAP #14

Note: This BIOS function is accessible to a p-System PASCAL program through the UNITREAD command.

```
UNITREAD(unit,Array,length,[Block],[Control]);
```

Channel Specific Data:

The Logical Block Number is never necessary for the non-mass-storage channels (1 and 7). Only the Winchester driver uses the high word of the logical block number. When using the p-System the high word is always zero because the block address cannot exceed 32K.

Channels 2, 6, and 8 do not allow reading.

For the floppy diskette channels (4 and 5), the Control Word bit 1 indicates that the physical sector mode should be used for access. In this mode the Logical Block Number is the physical sector number, the Size in Bytes is ignored and only one sector is transferred.

Channel 128 is used to read back device configuration information for all devices (see detail in later section).

Channel 129 is used to read Real Time information and is described under the p-System, Time Access section.

Channel 130 is used to read data from any memory location and is described under the p-System, General Memory Access section.

Channel 131 is used to cause the BIOS to stop until the next Break or Event occurs.

Channels 132-133 are used under the Multi-User BIOS only (see the section on Multi-User).

Function=12.....Write Data to a Channel

D0=12
A0=addr of arguments:
Offset: 0 - Channel Number (word)
Offset: 2 - Error reply code (word)
Offset: 4 - Size in Bytes (long word)
Offset: 8 - Memory Address (long word)
Offset: 12 - Low Word of Logical Block Number (word)
Offset: 14 - Control word (word)
Offset: 16 - High Word of Logical Block Number (word)

TRAP #14

Note: This BIOS function is accessible to a p- System PASCAL program through the UNITWRITE command.

```
UNITWRITE(unit,Array,length,[Block],[Control]);
```

Channel Specific Data:

The Logical Block Number is never necessary for the non mass storage channels (2, 6, and 8). The high word of the Logical Block Number is only necessary for the Winchester driver. When using the p-System the high word is always zero because the block address cannot exceed 32K.

Channels 1 and 7 do not allow writing.

For the floppy diskette channels (4 and 5), the Control Word bit 1 indicates that the physical sector mode should be used for access. In this mode the Logical Block Number is the physical sector number. The Size in Bytes is ignored and only one sector is transferred.

For the floppy diskette channels (4 and 5), the Control Word bit 13 indicates that a track on the diskette should be formatted. The high byte of the Logical Block Number is the cylinder address to be formatted. The low byte of the Logical Block Number is the head address to be formatted. The Size in Bytes and Memory Buffer area contain the ID field data to be recorded during formatting (4 bytes per sector - cylinder, head, sector, and bytes per sector code). Note that before formatting, the Gap 3 parameter must be modified using a write to special channel 128 (see later section).

For the Remote Out channel (8), the Control Word bits 12 and 13 are used to specify control of the Data Terminal Ready, Request to Send, and Break signals. When bit 12 is set in the Control Word, the one byte of data from the buffer contains signal bits to be cleared. When bit 13 is set in the control word, the one byte of data from the buffer contains signal bits to be set. The Data Terminal Ready signal bit is 2H, the Break signal bit is 8H, and the Request to Send signal bit is 20H (32 decimal).

Channel 128 is used for controlling device configurations and is fully described in a later section.

Channel 129 is used to set Real Time information and is described under the p-System, Time Access section.

Channel 130 is used to write data to any memory location and is described under the p-System, General Memory Access section.

Channel 131 is used to set up event schedules and is described under the p-System, Scheduled Events section.

Channels 132-133 are used under the Multi-User BIOS only (see the section on Multi-User).

Function=13.....Get Status of a Channel
D0=13

A0=addr of arguments:
Offset 0 -Channel Number (word)
Offset 2 -Error reply code (word)
Offset 4 -Control Word (word)
Offset 6 -Address of 30 word area for status data

TRAP #14

Note: This BIOS function is accessible to a p-System PASCAL program through the UNITSTATUS command.

```
UNITSTATUS(unit,Statusrec,control);
```

Statusrec should be a 30 word area for the Channel Specific data. Control = 0 for output , 1 for input.

Channel Specific Data (returned in 30 word area):

For the Keyboard channel (1) the following data is returned:

Offset	0	-Number of chars queued for input (word)
Offsets	2-53	-Currently unused
Offset	54	-Framing error count (word)
Offset	56	-Parity error count (word)
Offset	58	-Overrun error count (word)

For the Terminal channel (2) the following data is returned:

Offset	0	-Number of chars queued for output (word)
Offsets	2-59	-Currently unused

For the floppy diskette channels (4 and 5) the following data is returned:

```
Offset      0 -Number of bytes queued (word)
               (always zero as no asynchronous
               floppy I/O is currently supported)
Offset      2 -Number of bytes per sector (word)
Offset      4 -Number of sectors per track (word)
Offset      6 -Number of tracks per disk (word)
Offsets  8-48 -Currently unused
Offset     50 -Busy flag (word)
Offset     52 -Current error code (word)
Offset     54 -unused
Offset     56 -First error code (word)
Offset     58 -Last error code (word)
Offset     59 -Currently unused
```

For the Printer channel (6) the following data is returned:

```
Offset      0 -Number of bytes buffer for printer (word)
Offsets  2-47 -Currently unused
Offset     48 -Printer mode (word)
               1 -use Remote Serial output channel.
               2 -Parallel (interrupt operation).
               3 -Parallel (polled operation).
Offset     50 -Printer status bits (word)
                  Bit 4 is one if printer is Busy.
                  Bit 5 is one if Out of Paper.
                  Bit 6 is one if printer is Selected.
                  Bit 7 is zero for a printer Fault.
Offsets 52-59 -Same as remote input channel 7 if
                  Remote output is selected (otherwise
                  unused).
```

For the Remote serial input channel (7) the following data is returned:

```
Offset      0 -Number of characters in input queue (word)
Offsets  2-51 -Currently unused
Offset     52 -Ringing & Carrier Detect (byte)
                Bit 2 is zero if ringing detected.
                Bit 3 is zero if carrier detected.
Offset     53 -Data Set Ready (byte)
                Bit 7 is one if DSR is asserted.
Offset     54 -Framing error count (word)
Offset     56 -Parity error count  (word)
Offset     58 -Overrun error count (word)
```

The UNITSTATUS request for any of the four extra serial channels will return the following information:

In output direction:

Offset 0: Number of characters queued for output (word).
Offsets 2 - 59 are unused.

In input direction:

Offset 0: Number of characters in input queue (word).
Offsets 2 - 53 are unused.
Offset 54: Framing error count (word).
Offset 56: Parity error count (word).
Offset 58: Overrun error count (word).

Note: No DSR is connected.

For the Remote serial output channel (8) the following data is returned:

```
Offset      0 -Number of characters queued for output (word)
Offsets  2-59 -Currently unused
```

For the RAM disk (11.) the following data is returned:

```
Offset      0 -Number of bytes queued (word)
               (always zero as transfer is immediate)
Offset      2 -Number of bytes per sector (word, always 512)
            4 -Number of sectors per track (word, always 1)
            6 -Number of tracks per disk (word, always block count)
Offsets  8-59 -Currently unused.
```

The UNITSTUS request for a Winchester device will return the following information:

```
Offset  0: Always zero for number of bytes buffered (word).
Offset  2: Number of bytes per sector (word).
Offset  4: Number of sectors per track (word).
Offset  6: Number of tracks per disk (word).

Offsets 8-49: Currently unused.

Offset 50: Busy flag (word)
Offset 52: Current error code (word)
Offset 54: Last hard error code (word)
Offset 56: First error code (word)
Offset 58: Last error code (word)
```

Function=14.....Quiet (disable signaling events)
D0=14
A0=addr of arguments: none.
TRAP #14

This function is used by the p-System interface to temporarily disable signaling an Attached event.

Function=15.....Enable (signaling events)
D0=15
A0=addr of arguments: none.
TRAP #14

This function is used by the p-System interface to enable signaling an Attached event. If an event is ready, the Event routine will be called immediately.

Function=16.....Attach (indicate attachment to an event)
D0=16
A0=addr of arguments:
OFFSET: 0 - Event number attached (word)
OFFSET: 2 - Control (byte)
(0 = Deattach, 1=attach)
TRAP #14

This function is used by the p-System interface to indicate when an event type has been Attached. Note that this interface is an extension added to the standard interface definition by SAGE. This can reduce the overhead in the BIOS when event types have not been Attached.

Function=17..... Initialize Event Routine Address
D0=17
A0=event routine address (long word)
TRAP #14

This function is used by the p-System interface to indicate the address of the routine to be called by the BIOS in the case of an attached Event. The Event routine receives the event number in D0 and preserves all registers. The Event routine is always called from within an interrupt routine. The address register A5 must be unmodified from the value set up by the interpreter.

Function=18..... Initialize I/O Configuration Routine Address

D0=18
A0=I/O configuration routine address (long word)
TRAP #14

Certain system configuration parameters may apply to the p-System Interpreter rather than the BIOS. This means that the configuration control routines in the BIOS (described later) must pass configuration information back to the Interpreter. Function 18 provides a method for the Interpreter to pass the BIOS a configuration routine address within the Interpreter. This routine may later be called by the BIOS to pass back any necessary control information.

When the Interpreter configuration routine is called, register A0 points to a parameter area. Offset 0 of the parameter area is a word containing the type of information being passed. Offset 2 starts the configuration information and the size is dependent on the type of information being passed.

Configuration Types:
Type 0: Offset 2: Printer options flag (byte).
Bit 0 = 1 means inhibit LF after CR.

> The Interpreter normally generates an automatic Line Feed after each Carriage Return character unless specifically inhibited by a bit in the Control word. The operating system has no universal handling of the LF after CR option but some printers cannot be prevented from generating their own LF after CR. Setting bit 0 of the Printer Options flag will prevent the Interpreter from generating any automatic Line Feeds.

Types > 0 are not currently assigned.

Function=19.....Enter Supervisor Mode

D0=19
A0=addr of arguments: none
TRAP #14

This BIOS call returns to the caller with the processor in Supervisor mode. Note that the processor will be using the Supervisor stack instead of the User stack. The program may return to the User mode by masking out the Supervisor bit in the Status Register. Example:

```
MOVEQ   #19.,D0
TRAP    #14.            ;Enter Supervisor mode
------
ANDI.W  #0DFFFH,SR      ;Back to User mode
```

Function=20.....Restore User Hooks

D0=20
A0=addr of arguments: none
TRAP #14

This function may be used in preparation for loading a stand-alone environment from an operating system. It re-establishes various default I/O parameters which may have been set up to couple the BIOS to the operating system. These parameters are the Break routine address, the Event routine address, the I/O Configuration routine address, the address of the special characters (Flush, Start/Stop, Break, and Character Mask), and the address of the No Break flag.

Function=21.....Exiting the Multi-User Environment

D0=21
A0=addr of arguments: none
TRAP #14

In the Multi-User Environment the normal Long BIOS function 0 call has been modified to exit back to the User Login message instead of turning off the whole system. A new function, D0 = 21, has been added to Exit the Multi-User Environment.

Function=22.....Get Bottom of User's Memory Area

D0=22
A0= returns with base of User's memory area.
TRAP #14

This function is used in the Multi-User Environment to return the address of the base of the User's memory area. This value is typically necessary for bootstrapping an operating system under the Multi-User Environment.

For compatibility, under the Single User BIOS the value 400H will be returned.

Function=23.....Get Top of User's Memory Area

D0=23
A0= returns with top of User's memory area.
TRAP #14

This function is used in the Multi-User Environment to return the the address of the top of the User's memory area. This value is typically necessary for bootstrapping an operating system under the Multi-User Environment.

Under the single-user BIOS the base of the RAM Disk will be returned. If no RAM Disk is defined, the base of the BIOS will be returned.

Function=24.....Get User's Boot device Number

D0=24
A0=addr of arguments: none
TRAP #14

This function is used in the Multi-User Environment to return a word containing the User's boot device number (logical device) in D0.

Function=25.....Put a TRAP Vector

D0=25
A0=new TRAP vector contents
TRAP #14

This function puts the long word address contained in A0 into the Trap Vector address contained in D1. This function is provided primarily for allowing Trap Vectors to be redirected under the Multi-User Environment. For compatibility it is also supported under the Single-user BIOS.

Function=26.....Get a TRAP Vector

D0=26
A0=previous TRAP vector contents
TRAP #14

This function gets the long word address from the Trap Vector location contained in D1 and returns the address in A0. This function is provided primarily for allowing Trap Vectors to be redirected under the Multi-User Environment. For compatibility it is also supported under the single user BIOS.

Function=27..... Translate to Physical Device

D0=27
A0=addr of arguments: none
TRAP #14

This function receives a logical channel number in D1 and returns a physical device number in D2 (low byte is device, high byte is subdevice). Also returned is the CP/M Device Information Table word in register D1.

Function=28..... Get Operating System Data Address

D0=28
A0=returns address of Operating System Data.
TRAP #14

This function returns the base address of the Operating System Data from the BIOS configuration.

FUNCTIONS=31-255..... Currently Unused

Functions 29 & 30 are currently set up to establish a direct Event routine but this protocol may be changed and should not be used. Functions 31 to 255 are not currently used and result in an immediate return with no action.

XIII.05 TECHNICAL NOTES :

Both the Winchester and Extra Serial Ports are accessed via the normal Long BIOS Calls (Initialize, Read, Write, and Status) via TRAP #14. Reads and Writes to the extra serial ports use the physical device numbers (13 to 16) and are not split as in channels 7 and 8.

IMPORTANT NOTE: The Winchester Read and Write calls require an extra word of argument located at offset 16 from the base pointer A0. Since large Winchester disks are available, the word-sized logical block number will not suffice to address the complete disk. Therefore, the current logical block number word at offset 12 will represent the low word of the logical block number. Offset 16 will then represent the high word of the logical block number. This is a little awkward but is upward compatible with the previous BIOS. The Winchester driver is currently the only device which will look at the high block number word. The p-System currently has no facility for handling long word block numbers so the high word will always be zero.

XIII.06 CHANNEL CONFIGURATION CONTROL :

Reading and Writing to channel 128 is used to access and modify device configuration information used by the BIOS. The Control parameter (offset 14) must contain the channel number of the device being accessed.

```
D0=11 to read  a channel configuration
   12 to write a channel configuration
A0=addr of arguments:
   Offset:  0 - Channel Number = 128. (word)
   Offset:  2 - Error reply code (never used - word)
   Offset:  4 - Size in Bytes (never used - long word)
   Offset:  8 - Memory Address (long word)
   Offset: 12 - Logical Block Number (word)
   Offset: 14 - Control word = channel number
                for access (word).
TRAP #14
```

Note: Configuration of a channel can be done by a p-System Pascal program using UNITREAD and UNITWRITE. See the CONFIGSAGE Unit page 163 .

A fixed amount of data, depending on the channel, will be moved in or out of the area specified by Memory Address. The Error reply code and Size in Bytes are unused. The Logical Block Number is only used as a parameter for the General System Configuration (Control word = 0).

The channel specific configuration information follows for each channel.

Channel 0 (General System Configuration):

To read or write the General System Configuration:

```
D0=11 to read
   12 to write
A0=addr of arguments
   (see previous section for channel 128).
TRAP #14
```

The Logical Block number is used to identify which type of system information is being controlled. The following table show the types of information which are accessible or configurable.

LBN	Information
0	Time adjustment factor
1	BIOS Channel Map
2	Operating System Configuration Data
3	SAGE IV Flag
4	BIOS Version Number
5	CP/M General Device Information
6	CP/M Winchester Device Information

Time Adjustment Factor (LBN = 0)

Offset: 0 - Time adjustment factor in X days (byte).
Offset: 1 - Time adjustment factor number of seconds (byte). The above two bytes define a correction for Real Time clock drift as a number of seconds (+ or - 127) in a number of days (up to 10). Zero days means no correction factor is defined.

BIOS Channel Map (LBN = 1)

```
Offset:  0 - Channel 0 subdevice - unused
Offset:  1 - Channel 0 device    - unused
Offset:  2 - Channel 1 subdevice
Offset:  3 - Channel 1 device
Offset:  4 - Channel 2 subdevice
Offset:  5 - Channel 2 device
...
Offset: 62 - Channel 31 subdevice
Offset: 63 - Channel 31 device
```

Operating System Configuration Data (LBN = 2)

The Operating System Configuration Data is an area which may contain information to configure an operating system environment when bootstrapping. It contains 32 bytes which have no meaning to the BIOS.

SAGE IV Flag (LBN = 3)

This flag is a word of data which is returned zero if the Winchester controller board is not equipped and a non-zero value if the Winchester controller board is equipped. The BIOS does not check for the existence of actual drives on line. The flag cannot be modified with a UNITWRITE.

BIOS Version Number (LBN = 4)

The BIOS Version number is returned as a word of data. The Version number cannot be modified with a UNITWRITE.

```
Offset:  0 - Subsidiary version number
Offset:  1 - Primary version number
```

CP/M General Device Information (LBN = 5)

The CP/M-68k system requires an extra word of configuration information for each device to support the operating system. The general table covers all except the Winchester device (physical channel number 9). It contains 16 entries corresponding to 16 physical channel numbers.

Offset: 0 - Physical device #1
Offset: 2 - Physical device #2
Offset: 16 - Physical device #9
(not used, see special
Winchester table)
Offset: 30 - Physical device #16

CP/M Winchester Device Information (LBN = 6)

The CP/M-68k system requires an extra word of configuration information for each device to support the operating system. This table contains four sets of 16 words, one for each partition on each drive.

Offset: 0 - Winchester #0, Partition #0
Offset: 2 - Winchester #0, Partition #1
Offset: 30 - Winchester #0, Partition #15
Offset: 32 - Winchester #1, Partition #0
Offset: 62 - Winchester #1, Partition #15
Offset: 64 - Winchester #2, Partition #0
Offset: 94 - Winchester #2, Partition #15
Offset: 96 - Winchester #3, Partition #0
Offset:126 - Winchester #3, Partition #15

Channels 1 & 2 (Terminal):

The terminal configuration is accessed via either channel 1 or 2. To read or write the terminal configuration:

```
D0=11 to read
   12 to write
A0=addr of arguments
   (see previous section for channel 128).
TRAP #14
```

The data format is:

```
Offset:  0 - Terminal baud rate control (word).
             Zero defaults to using the switch settings
             (when using switch settings, the USART control
              parameter must be set for even parity).
             Value = 38400/baud rate (for rates <= 19200).
Offset:  2 - Terminal USART mode (byte).
             This is the standard 8251A mode byte.
             Bits 1,0 - 10 for 16 times clock.
             Bits 3,2 - 00 for 5 data bits.
                        01 for 6 data bits.
                        10 for 7 data bits.
                        11 for 8 data bits.
             Bit    4 -  0 for disabled parity.
                         1 for enabled parity.
             Bit    5 -  0 for odd parity.
                         1 for even parity.
             Bits 7,6 - 00 invalid.
                        01 one stop bit.
                        10 one and a half stop bits.
                        11 two stop bits.
Offset:  3 - Terminal configuration flags (byte)
             Bit 0 - One if Terminal Break key (framing
                     error) to enter PROM Debugger.
                     Zero if Break key ignored.
             Bit 1 - One if XON/XOFF protocol is to be
                     used on output to the terminal.
                     (Note that the p-System Start/Stop
                     character must be defined as XOFF).
             Bits 2 to 7 - reserved.
```

Channels 4 & 5 (Floppy drives):

Each of the floppy drives is independently configured. A 32 byte configuration area is transferred although only 22 bytes are currently defined. To read or write the configuration:

```
D0=11 to read       A0=addr of arguments      TRAP #14
   12 to write      (see previous section
                     for channel 128).
```

The data format is:

```
Offset:  0 - Number of sides (0 for no drive).
Offset:  1 - Number of cylinders.
Offset:  2 - Number of sectors per track.
Offset:  3 - Gap 3 parameter for controller.
Offset:  4 - Track to track skew flag
             (only used by formatter to skew
              the addresses on formatting).
Offset:  5 - Data Length (only for < 256 bytes
             per sector, otherwise set to 255).
Offset:  6 - Bytes per Sector (word)
Offset:  8 - Motor on Delay (in 1/64000ths second).
Offset: 10 - Step Rate.
             0F0H = 2 Milliseconds
             0E0H = 4 Milliseconds
             0D0H = 6 Milliseconds
             0C0H = 8 Milliseconds
Offset: 11 - Head Load Time.
             Bit 0 must be one to indicate No DMA mode.
             Add in Head Load Time (used for head settle
             time) in milliseconds/2.  Normal value is
             1 + (16/2) for a 16 millisecond head settle
             time.
```

Offset: 12 - Double Density flag (0 = single, 1 = double).
Offset: 13 - Track format flags
Bit 0 is one for IBM track compatibility.
Bit 1 is one for Network Consulting special sector numbering scheme for 10 sector per track diskettes.
Offset: 14 - Retry count.
Offset: 15 - Ignore errors flag (0 = respond to errors, 1 = ignore errors for alignment).
Offset: 16 - Soft error counter (informational only)
Offset: 17 - Double Step flag (0 = normal stepping, 1 = use cylinder number times 2 to read 48 TPI diskettes on 96 TPI drive).
Offset: 18 - Format pattern
Offset: 19 - Gap 3 for formatting
Offset: 20 - First detected BIOS error.
Offset: 21 - Second detected BIOS error.
Offsets 22 to 31 - are transferred but unused.

Channel 6 (Printer):

To read or write the parallel printer configuration:

```
D0=11 to read
   12 to write
A0=addr of arguments
   (see previous section for channel 128).
TRAP #14
```

The data format is:

```
   Offset:  0 - Printer polling Timeout
                This entry is used in printer mode 3 to contro
                the number of 4.25 microsecond cycles that the
                BIOS will poll the printer before going to
                sleep. (236 counts = 1 millisecond)
   Offset:  2 - Printer polling delay (in 1/64000ths second)
                This entry is used in printer mode 3 to contro
                the amount of time between attempts to poll th
                printer.
   Offset:  4 - Printer mode
                0 - Unused.
                1 - Unused.
                2 - Parallel (interrupt operation).
                3 - Parallel (polled operation).
   Offset:  5 - Printer Options
                Bit 0 set to inhibit LF after CR in the
                      interpreter.
```

Channels 7 & 8 (Remote serial channel):

To read or write the remote serial channel configuration:

```
D0=11 to read
   12 to write
A0=addr of arguments
   (see previous section for channel 128).
TRAP #14

The data format is:
Offset:  0 - Remote channel baud rate control (word).
             Zero defaults to 9600 baud.
             Value =38400/baud rate (for rates <= 19200).
Offset:  2 - Remote channel USART mode (byte).
             This is the standard 8251A mode byte.
             Bits 1,0 - 10 for 16 times clock.
             Bits 3,2 - 00 for 5 data bits.
                        01 for 6 data bits.
                        10 for 7 data bits.
                        11 for 8 data bits.
             Bit    4 -  0 for disabled parity.
                         1 for enabled parity.
             Bit    5 -  0 for odd parity.
                         1 for even parity.
             Bits 7,6 - 00 invalid.
                        01 one stop bit.
                        10 one and a half stop bits.
                        11 two stop bits.
Offset:  3 - Remote channel configuration flags (byte).
             Bit 0 - One if using XON/XOFF on input.
             Bit 1 - One if using XON/XOFF on output.
             Bits 2 to 7 are reserved.
```

```
Offset:  4 - Remote channel transmit delay time (word).
             This delay time (in 1/64000ths of a second)
             is scheduled between DSR checks if the
             DSR check flag (see offset 6) is set and the
             DSR signal is not asserted.
Offset:  6 - Data Set Ready check flag (byte).
             Zero indicates no DSR check.
             One indicates that the DSR signal must be
             asserted before a character is transmitted.
             If DSR is not asserted, a delay (see offset 4)
             is scheduled for a future retest.
Offset:  7 - Transferred but unused (byte).
```

Channel 11 (RAM Disk)

```
To read or write the RAM disk configuration:
D0=11 to read
 12 to write
A0=addr of arguments
 (see previous section for channel 128).
TRAP #14

The data format is:
Offset:  0 - Base address of RAM Disk area (long word)
             (0 means no RAM Disk configured).
Offset:  4 - Top address for RAM Disk area (long word)
             (0 means just below BIOS & stack).
Offset:  8 - Boot to RAM Disk flag (byte).
             Zero boots from floppy.
             One copies floppy to RAM Disk and boots.
Offset:  9 - Transferred but unused.
```

XIII.07 BIOS CHANNEL ERROR CODES :

Error codes are returned by the BIOS for the Initialize, Read, Write, and Status check commands. The errors are word values with negative numbers. Zero represents a no error (normal) condition. Most of the error codes pertain to the floppy or Winchester driver. Codes -14 and -15 may come from any device.

	SUMMARY OF FLOPPY ERROR CODES
0	No error.
-1	Floppy controller would not respond.
-2	Floppy controller returned invalid command error
-3	Recalibrate or Seek failure (equipment check).
-4	No diskette (as a result of read/write timeout).
-5	Missing address mark (from floppy controller.)
-6	No Data Found reported by floppy controller.
-7	Overrun reported by floppy controller.
-8	CRC Error reported by floppy controller.
-9	End of Cylinder reported by floppy controller.
-10	Write Protect Violation.
-11	Address out of range.
-12	Wrong Cylinder reported by floppy controller.
-13	Currently unused.
-14	Illegal device number.
-15	Illegal request.

SUMMARY OF WINCHESTER ERROR CODES	
0	No error.
-1	Could not initialize VCO.
-3	Recalibrate or Seek failure (equipment check).
-4	Drive not ready.
-6	Timeout while waiting for data.
-8	CRC Error reported by disk controller.
-9	Verify error.
-10	Write Protect Violation.
-11	Block Address out of range.
-12	Wrong Cylinder reported by disk controller.
-14	Illegal device number.

XIII.08 BIOS NOTES :

BIOS SOURCE FILES:

SAGE.BIOS.TEXT	Includes for BIOS assembly.
SAGE.BIOS1.TEXT	RAM & Port Address assignments.
SAGE.BIOS2.TEXT	Configuration Header & Initialization
SAGE.BIOS3.TEXT	BIOS Entry, Interrupt, and Foreground handler
SAGE.BIOS4.TEXT	Clock Handler & Scheduler
SAGE.BIOS5.TEXT	Terminal Handler
SAGE.BIOS6.TEXT	Remote Serial Channel Handler
SAGE.BIOS7.TEXT	Floppy utility routines.
SAGE.BIOS8.TEXT	Floppy motor and select control.
SAGE.BIOS9.TEXT	Floppy initialization, recalibrate, and seek.
SAGE.BIOSA.TEXT	Floppy main transfer routine.
SAGE.BIOSB.TEXT	RAM Disk Handler.
SAGE.BIOSC.TEXT	Printer Handler.
SAGE.BIOSD.TEXT	Event Handler (for p-System events).
SAGE.BIOSE.TEXT	Winchester driver
SAGE.BIOSF.TEXT	Winchester driver
SAGE.BIOSG.TEXT	Winchester driver

Preparing a BIOS

The BIOS files are assembled together in one assembly. Set the prefix to the device containing the BIOS source files. Get the file SAGE.BIOS and assemble. When the assembly is complete, copy the code file to the target system and rename it to SYSTEM.BIOS. The Bootstrap handles the header information in Version IV code files so the file need not (and must not) be COMPRESSed.

BIOS Configuration

The file SAGE.BIOS2.TEXT generates the first actual object code in the code file. The start of the code contains a configuration area for controlling various aspects of the BIOS. The Configuration Manager in the file SAGE4UTIL.CODE provides a method for changing the configuration area of SAGE's BIOS code file.

BIOS Data Storage

The SAGE Computer system only uses the Autovector interrupts and has no hardware provision for handling interrupt vectors from peripheral controllers. Therefore, the space normally reserved for user interrupt vectors has been allocated to the Debugger and the BIOS. The Debugger uses 100H to 1FFH. The BIOS uses locations from 200H to 300H. The Debugger stack works down from 400H to 300H. Note that this data can be addressed via the short direct addressing mode which is important for interrupt routines to avoid having to load an address into a base register. The 68000 protects against storing into a location addressed relative to the program counter.

Some BIOS data which is normally accessed via an address register is allocated space just before the BIOS code. Also allocated here are the 256 byte buffers for the Keyboard, Terminal output, Remote input & output, and the Printer output.

BIOS Software Interrupt

One of the problems in a priority interrupt system is passing information and control between different levels of priority. A facility has been provided in hardware to allow control of the lowest priority interrupt from a software controlled port. This lowest priority interrupt routine is called the Foreground Handler. The Foreground Handler is started from any level of interrupt routine by setting a foreground flag to indicate which task is to be done and then turning on the software-controlled interrupt. Since the Foreground interrupt is the lowest priority, the handler will not start until the interrupt and all other possible nested interrupts have been processed. When the Foreground Handler gets control, it examines the various foreground task flags and executes the requested tasks.

A background program can gain control of the Foreground by disabling the Foreground interrupt at the 8259 interrupt encoder and enabling the interrupt when complete. Also the background can activate a Foreground task by setting the task flag and turning on the software controlled interrupt.

Currently there are six Foreground tasks defined, the Scheduler Timeout, the Event Handler, the Floppy driver dispatcher, the Remote transmit delay setup, the Keyboard lockout delay setup, and the Remote lockout delay setup. All routines which enter schedules (via ESCHED) or cancel schedules (via CSCHED) must either be in a Foreground task or have gained control of Foreground by disabling the software controlled interrupt (from background only).

The p-System Bootstrap

The single-user bootstrap program is located on blocks 0 and 1 of the floppy diskette. The 'IFx', Initialize from Floppy command in the Debugger will cause the first two blocks of the diskette to be read in to RAM at location 400H. The source file of the bootstrap is called SAGE.PBOOT.TEXT. The file is assembled normally (but not Linked or Compressed). The file SAGE.PBOOT.CODE is installed on a diskette using the Bootstrap Copy facility of the SAGE4UTIL program. Note that the standard p-System utility BOOTER.CODE should not be used for installing the bootstrap (unless the extra steps of Linking and Compressing are performed).

The protocol for all SAGE bootstrap programs requires that the first four bytes of the code (at 400H) be the ASCII characters 'BOOT'. This data is cross-checked by the PROM bootloader and if it is not present the system replies 'Not Boot Diskette'. If the header data is correcti,he boot program will then be started at location 404H. The bootstrap is entered in Supervisor mode. The stack contains a return address which may be used in case of boot failure to return to the debugger. Below the return address on the stack (at 4(A7)) is the drive number, 0 for the left drive or 1 for the right drive.

The routines TERMTEXT, TERMCRLF, and FDREAD in the PROM Debugger are used by the Bootstrap program for terminal and floppy I/O. Note that these routines are accessed via a macro which generates the necessary long absolute addresses. The Bootstrap first reads in the 4 block p-System directory from block 2 of the diskette. The file SYSTEM.BIOS is found, and the first block of the file is read.

The first four bytes of the BIOS code are checked for the four ASCII characters 'BIOS'. If this data is not found, the message 'Not BIOS code in SYSTEM.BIOS' is output and the bootstrap returns to the Debugger. Note that in Version IV code files there is a 26 byte header ahead of the actual code in an unCompressed file.

If the proper BIOS data is found the complete SYSTEM.BIOS file is read in at the top of all RAM memory. Offset 4 from the start of the code in the SYSTEM.BIOS file is the size of the BIOS code. Offset 6 is the size of the RAM buffer area which must be allocated preceding the code. Offset 8 is the offset of the BIOS Initialization routine address from the beginning of the code. Also the RAM Disk boot flag and base address are taken from the configuration area in the BIOS file.

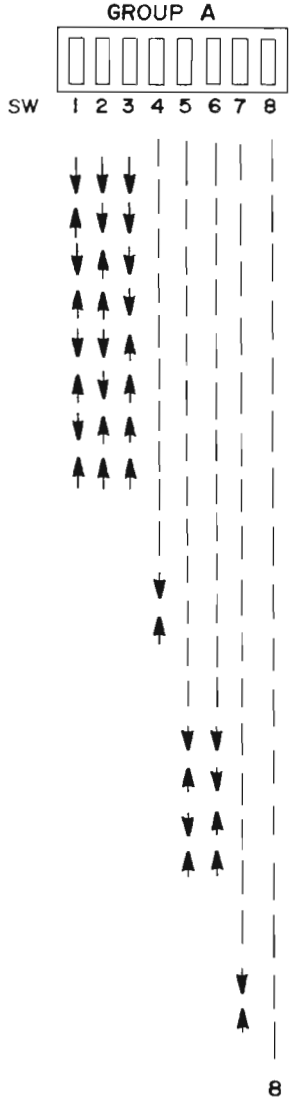
The BIOS Initialization routine is executed which sets up all the hardware and drivers and turns on interrupts. The Debugger is set up to use the BIOS terminal driver.

The file SYSTEM.INTERP is read into its position in memory above the p-System data area and below the p-System code pool area. If the RAM Disk boot flag is set, files from the diskette are copied to the RAM Disk area (size taken from directory). The new directory on the RAM Disk is RAMDISK. A file called ENDBOOT will terminate the copy process.

Finally the processor is set into User mode and several arguments are put on the User stack for initialization of the p-System Interpreter. The routine then transfers to the beginning of the Interpreter.

The single-user Winchester bootstrap is very similar to the floppy bootstrap. It is contained in the source file SAGE.WBOOT.TEXT and the object file SAGE.WBOOT.CODE. This routine accesses the PROM routine WDREAD to read the Winchester disk.

APPENDIX A: SWITCH SETTINGS



TERMINAL BAUD RATE

8 data bits, 1 stop bit and even parity.

19.2 K baud

9600

4800

2400

1200

600

300

reserved, will default to 19.2 K baud

PARITY CONTROL

even parity enabled

disabled

BOOT DEVICE

boot to DEBUGGER

boot to Floppy drive Ø.

boot to first partition of Winchester drive Ø.

reserved, defaults to DEBUGGER

FLOPPY CONFIGURATION

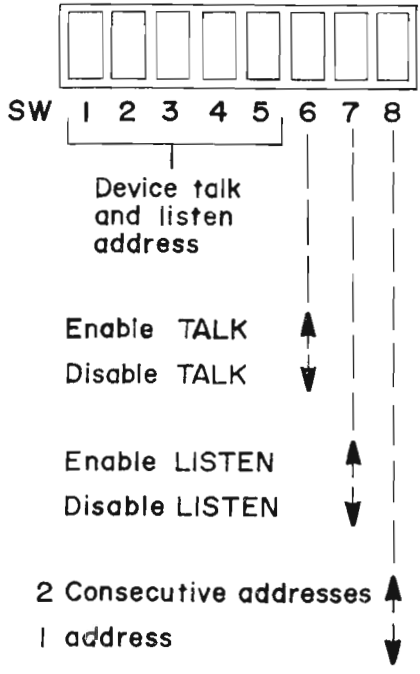
96 TPI drive

48 TPI drive

8 reserved

APPENDIX A: SWITCH SETTINGS (cont)

(GROUP B switches as defined for IEEE-488 use.)



APPENDIX B: FLOPPY DISK BOOT ERRORS

"Not BOOT Disk"
"Boot aborted on drive 0"
"Drive error (code) on drive (0 or 1)" where codes are:

01	- controller failure
02	- invalid command
03	- recalibrate or seek failure
04	- timeout
05	- missing address mark
06	- no data found
07	- overrun
08	- CRC error
09	- end-of-cylinder
0A	- unknown
0B	- address out-of-range

APPENDIX C: WINCHESTER BOOT ERRORS

"Could not find 'bootname' "
"Not BOOT Disk"
"Drive error (code)

01	- could not initialize VCO
03	- recalibrate or seek failure
04	- drive not ready
06	- timeout while waiting for data
08	- CRC error
0B	- block address out-of-range
0C	- wrong cylinder reported by disk controller

(NOTE: Errors not shown are not used.)

APPENDIX D: BIOS PHYSICAL DEVICE NUMBERS

Single User#	Multi User#	Device	Connector (if any)
0	0	no device	
1	1	Keyboard #1	Terminal
2	2	Terminal #1	Terminal
3	3	reserved	
* 4	4	Left floppy	
* 5	5	Right floppy	
6	6	Parallel Printer Port	
7	7	Keyboard #2 (Remote serial input channel)	Modem
8	8	Terminal #2 (Remote serial output channel)	Modem
* 9	9	Winchester drive partitions	
*10	10	Reserved	
*11	11	RAM Disk #1	
*12	12	Reserved	
13	13	Keyboard #3	Aux 1
13	14	Terminal #3	Aux 1
14	15	Keyboard #4	Aux 2
14	16	Terminal #4	Aux 2
15	17	Keyboard #5	Aux 3
15	18	Terminal #5	Aux 3
16	19	Keyboard #6	Aux 4
16	20	Terminal #6	Aux 4
*	21	RAM Disk #2 (MU only)	
*	22	RAM Disk #3 (MU only)	
*	23	RAM Disk #4 (MU only)	
		User defined devices (by SAGE)	
128	128	Device configuration control	
129	129	Reading system clock	
130	130	General Memory Access	
131	131	Scheduled event control	
	132	Multi-User advanced features	
	133	Multi-User Inter Communications	

APPENDIX E: SHORT CALLS TO BIOS

TRAP #8.	Test I/O Queue
TRAP #9.	Read Character from Keyboard
TRAP #10.	Write Character to Terminal
TRAP #11.	Write Character to Printer
TRAP #12.	Read Character from Remote Serial Channel
TRAP #13.	Write Character to Remote Serial Channel

APPENDIX F: LONG CALLS TO BIOS

```
X..BIOS function (in D0)
-------------------------------------------------------------
 0..Exit to Debugger
 1..Reinit BIOS
 2..Return Addr Top of Usable Memory
 3..Install BIOS for debugger
 4..Disable BIOS for debugger
 5..Read System Clock
        0 - word count in 1/64000 ths of a second.
        2 - long word count of seconds.
 6..Read Clock in 1/60 ths second
        0 - word count of 1/60 ths of a second.
 7..Schedule an Event. A0=PTR to schedule entry.
        0:  Long word reserved for scheduler linkage.
        4:  Long word address for scheduled routine.
        8:  Long word relative timeout time.
            High=sec,low=1/64000 ths.
 8..Cancel a Schedule. A0=PTR to schedule entry.
 9..Load 68000 Object Program
        0 - Channel Number (word)
        2 - Error reply code (word)
        4 - Size in Bytes (long word)
        8 - Memory Address (long word)
       12 - Logical Block Number (word)
       14 - Control word (word)
       16 - Start Execution Address (long word)
10..Initialize a Channel
        0 - Channel Number (word)
        2 - Error reply code (word)
            following for channels 1 or 2 only:
        4 - addr of BREAK routine
        8 - addr of chars for Flush, START/STOP, Break
       12 - addr of No Break flag (bit 6)
```

continued on next page...

continued...

APPENDIX F: LONG CALLS TO BIOS

```
11..Read Data from a Channel. (NOT ch 2,6, or 8)
        0 - Channel Number (word)
        2 - Error reply code (word)
        4 - Size in Bytes (long word)
        8 - Memory Address (long word)
       12 - Logical Block Number (word)
       14 - Control word (word)
12..Write Data to a Channel.
        0 - Channel Number (word)
        2 - Error reply code (word)
        4 - Size in Bytes (long word)
        8 - Memory Address (long word)
       12 - Logical Block Number (word)
       14 - Control word (word)
13..Get Status of a Channel.
14..Quiet (disable signaling an event)
15..Enable (signaling an event)
16..Attach (indicate attachment to an event)
17..Initialize Event Routine Address
18..Initialize System Configuration Routine Address
19..Enter 68000 Supervisor mode
20..Restore User Hooks
21..Exiting the Multi-User Environment
22..Get Bottom of User's Memory Area
23..Get Top of User's Memory Area
24..Get User's Boot Device number
25..Put a TRAP Vector
26..Get a TRAP Vector
27..Translate to a Physical Device
28..Get Operating System Data Address
```

	APPENDIX G: FLOPPY ERROR CODES (BIOS)
0	No error.
-1	Floppy controller would not respond.
-2	Floppy controller returned invalid command error.
-3	Recalibrate or Seek failure (equipment check).
-4	No diskette (as a reseult of read/write timeout).
-5	Missing address mark reported by floppy controller
-6	No Data Found reported by floppy controller.
-7	Overrun reported by floppy controller.
-8	CRC Error reported by floppy controller.
-9	End of Cylinder reported by floppy controller.
-10	Write Protect Violation.
-11	Address out of range.
-12	Wrong Cylinder reported by floppy controller.
-13	currently unused
-14	Illegal device number
-15	Illegal request

	APPENDIX H: WINCHESTER ERROR CODES (BIOS)
0	No error.
-1	Could not initialize VCO
-3	Recalibrate or Seek failure (equipment check).
-4	Drive not ready
-6	Timeout while waiting for data
-8	CRC Error reported by disk controller.
-9	Verify error
-10	Write Protect Violation.
-11	Block Address out of range.
-12	Wrong Cylinder reported by disk controller.
-14	Illegal device number

APPENDIX I: SAGE I/O PARTITIONS

BOARD	ADDRESSES (HEX)	BOARD NAME:
# 1	FFC000 - FFC3FE	Main CPU
# 2	FFC400 - FFC7FE	HARD DISK
# 3	FFC800 - FFCBFE	
# 4	FFCC00 - FFCFFE	
# 5	FFD000 - FFD3FE	
# 6	FFD400 - FFD7FE	
# 7	FFD800 - FFDBFE	
# 8	FFDC00 - FFDFFE	
# 9	FFE000 - FFC3FE	
#10	FFE400 - FFE7FE	
#11	FFE800 - FFEBFE	
#12	FFEC00 - FFEFFE	
#13	FFF000 - FFF3FE	Reserved for User
#14	FFF400 - FFF7FE	Reserved for User
#15	FFF800 - FFFBFE	Reserved for User
#16	FFFC00 - FFFFFE	Reserved for User

Refer to the SDT-ASSEMBLER for a complete description of the I/O addresses.

APPENDIX I: ASCII CODES

The charts on the next page are useful for finding the decimal and hexadecimal values of an ASCII character.

To find the value of an ASCII character, first get its HEXADECIMAL value. Locate the ROW (left hand number) and the COLUMN (at top) of the character in TABLE 1. The ROW is the most significant digit, msd; the COLUMN is the least significant digit, lsd, of the HEX number.
EXAMPLE: the value of character "A" = 41 (HEX)

To find the decimal value of the character, convert the hexadecimal value. The same ROW and COLUMN in TABLE 2 contain the decimal number.
EXAMPLE: the value of character "A" = 65 (DECIMAL)

The second half can also be used as a general purpose conversion from HEX to decimal or from decimal to HEX.

** ASCII figure

TABLE I - ASCII CODES (HEX)

msd	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	Ø	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

TABLE 2 - HEX TO DECIMAL CONVERSION

	Ø	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Ø	Ø	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
2	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
3	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
4	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
5	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
6	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
7	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127

INDEX

- A -

Access Control 325
  Floppy Disk 332
  Implementation 328
  Masks 328
  Menu 330
  Partitions 332
  Philosophy 326
  Printer 331
  Winchester 332
Addressing
  I/O ports 17
  Memory space 108
Advanced Features 390
Air flow 12
Altitude 12
ASCII Codes 497
Asynchronous I/O 96
Attach 96, 143, 149, 463
  Table 153
ATT_Unit 143
AUTOVECTOR
  Interrupts 484
Auxiliary Channel Device Numbers 39
Auxiliary Ports 22

- B -

Background 293
Bad Tracks 189
  Mapping 71
BASIC 88
Basic Input/Output System 443
Baud Rate 172, 173, 262, 297
  Configuration 297
  Reduced 226
  Remote 224
BIOS
  Channel errors 481
  Channel Map 34, 48, 285
  Configuration 104, 484

Data Storage 484
Data storage 484
Files 483
Interrupt 485
Interrupts 484
Long calling sequence 449
Long Call Summary 493
Memory base of 335
Short calling sequence 443
Short Call Summary 492
Source 484
Technical Notes 470
BLIB.Rx 88
Block-structured devices 92
Boot
Copy 61
Files 82
Boot Device 292
BOOTER.CODE 82, 486
Booting 271
Bootstrap
Header 486
Multi-User 272, 279
Partition 46
Break 140, 228, 278, 303
Configuration Unit 172
Options 303
To reboot allowed 302
Buffer
Serial Receive 300
Serial Xmit 300
BUS ERROR 17
Bus, Expansion 15, 16
Byte Sex 109

-C-

Capability Mask 287
Carrier Detect 103
C compiler 336
Channel
BIOS 449
Configuration 471

Error codes 481
Map 285, 328
Read 455
Status 459
Write 457
CHK Error 319
Clear to Send 19, 226
Clock 449
Processor 4
Real Time 10
Time access 105
Code Pool 333
Compiler
BASIC 88
FORTRAN 87
PASCAL 86
CONFIG_SAGE 125, 163
Configuration
Control Unit 163
Environment 279
Files 279
Flag 287
Floppy drive 307, 312
Low Level 312
Parallel Port 310
Partitions 311
Printer 310
RAM Disk 91, 309
Serial Channel 296
Standard 338
Terminal 296
Terminal Emulation 420
User Tasks 283, 284
Utility Program 279
Winchester Drive 311, 312
Connector Locations 17
Continue Option 303
Control Information Block 393
Cooling and Environment 12
CP/M 336
Memory Assignment 333

CRC 183
Current Control 393
Cylinders 176, 180

- D -

Database 260, 395
Data Bits 172, 174, 297, 299
  Remote channel 224
Data Memory Space 108
Data Set Ready 19, 101, 226, 304
Data Terminal Ready 224
Date, System 140
Date Unit 135
Days 179
DB-25 Connector 19
Debugger
  Entry 319
  Interrupts 484
  Option 303
Device
  #11: 89
  Auxiliary Device Numbers 39
  Channel Map 325
  Control 325, 390
  Numbers 34
  Physical Numbers 35
  p-System 92
  p-System numbers 36
  User defined 92
Device not Equipped 331
Dipswitches 297, 488
  GROUP-B 196
  IEEE-488 196
Directory
  RAM Disk 89
DIR_INFO 125
Disable Signaling Events 463
Disk
  Size 53
DSR Polling 174
DSR Signal 19

– E –

Emulation 420
ENDBOOT 84, 89, 487
Error
  BIOS channel 481
  Break 302
  Bus 17
  Destroying other users 336
  Device not Equipped 331
  Exception 319
  Executive #17 84
  Floppy 495
  Floppy Disk Boot 490
  Framing 305
  IEEE-488 220
  IEEE-488 option 194
  I/O 331
  Non-existant device 91, 309
  Not enough memory 335
  Printer 331
  RAM Disk 309
  RAM disk 91
  Stack Overflow 278, 302
  Task memory overlap 335
  Winchester 495
  Winchester Boot 490
Event 149
  Table 153
Example
  Control routine 394
  Semaphore usage 404
  Six User system 334
Exception Errors 319
Exclusive Control 260, 326, 328, 393
Executive Err#17 84
Exit Option 303
Expansion Bus 15, 16

- F -

FDREAD 486
FILE_INFO 125
Files
    BIOS 483
    IEEE-488 210
    Multi-User System 264
    p-System 112
Floppy Drive
    Access Control 332
    Channels 476
    Configuration 307, 312, 313
    Connector 31
    Disabled 285, 338
    Operation 31
    Shared Parameters 317
    Specifications 28
Foreground 293
Format
    10 sector 308
    Winchester 69
FORMATINFO.TEXT 54, 55, 64, 265
FORTLIBx 87
FORTRAN 87
Four Word
    BASIC 88
    FORTRAN 87
    PASCAL 86
Framing Errors 305
Freeze option 303

- G -

Gap3 176
Global Semaphores 260, 327, 390, 395
GO command 303
GPIB 191
GROUP-B Switches 196

- H -

Handshaking, Remote 226
Hang and Wait 328, 331, 393
Humidity 12

- I -

IBM format 308
IB_UNIT 193
IEEE-488 444
  Connector 221
  Device set up 200
  Error codes 220
  Error option 194
  Files 210
  IB_UNIT 193
  Initialization 198, 212, 218
  Listeners 202, 213
  Parallel poll 218
  References 221
  Register control 206, 214
  Serial poll 216
  Service requests 205, 216
  Strapping options 221
  Support Programs 191
  Switch options 196
  Talkers 201, 213
  Tri-state drivers 221
  User buffer 204
  User program 210
IFx boot command 47, 486
IHx boot command 46
Initial Bootstrap 267
Intercept Exceptions 319
Intercommunication 223
INTERP.x 85
Interrupts
  AUTOVECTOR 484
  BIOS 484, 485
  Debugger 484
  Driven drivers 96
  8259 Encoder 485

Priority 485
I/O
Addressing 17
Asynchronous 96
Auxiliary Ports 22
Connector locations 17
Device 132 390
Error 331
LED color 5
Modem 19
Partition Summary 496
Queue 446
RAM Disk 89
Remote 19
Serial Ports 22

- K -

Keyboard
Channel 449
Read 447
UNITSTATUS 101
Key, Task swapping 293

- L -

LED 5
Line Filter 11
Listeners, IEEE-488 . . . . . 202, 213
Local Area Network 261
Logical Channels 261
Logical Device Numbers 36
Low Level Configuration 312

- M -

Mail drop 332
Masks, Access Control 328
Memory
Allocation 109
Assignment 333
Base of BIOS 335
Channel 460
Expansion 6
General Access 108
Management 336

Strapping options 6
Menus 280
Menu Unit 154
MNU_UNIT 266, 280
MNU_Unit 125, 154
Mode
Supervisor 465
Modem Connection 19
Modula 2 336
Motorola Object Code 453
MU4.BIOS 264
MU.CONFIG 265, 279
MU4.FBOOT 264
Multi-User 254
Boot Message 273
Bootstrap 271, 272
Configuration 279
Database considerations 260
Example 256, 347
Initial Boot Partition 267
Multiple Operating Systems 258
Recommendations 336
Resource Allocation 259
Shared Data Access 260
Standard Configurations 338
System Files 264
System Manager 255
Terminal Emulation 420
Time Slice 257
User Intercommunication 410
User Tasks 256
MUTRMSET.CODE 420
MU.UTIL 265
(see also Configuration) 279
MU4.WBOOT 264, 267

- N -

NCI format 308
Network 261

-O-

Operating System 68, 258

-P-

Parallel Poll, IEEE-488 218
Parallel Port 101
  Buffer Size 310
  Configuration 310
Parallel Printer
  Configuration 478
Parity 297
  Chip location 9
  Configuration 298
  Enabled 172, 174
  Even 172, 174
  Remote channel 224
  UNITSTATUS 101
Partitions
  Access Control 332
  Bootstrap 46
  Factory Installed 44
  Mail drop 332
  Names 52
  Numbering 42
  Size 52
PASCAL compiler 86
Physical Characteristics 12
Physical Device Numbers 35
Poll Delay 174
Polled
  Printer operation 101
Portability 109, 223
Power Supply 10
Precision, Real 84
Printer 338
  Access Control 331
  Configuration 310
  Data 460
  Error 331
  Parallel Port 310
  UNITSTATUS 101

Write a character 448
Print Spooler 98
Priority
Interrupt 485
User Tasks 294
Privilege Violation 319
Processor
68000 4
LED 5
Programmed Device Control 390
Program Memory Space 108
PROM
Sizes 10
Strapping options 10
p-System 278, 336
Boot files 82
Bootstrap 486
Channel Map 38
Code pool 333
Data area 333
Description 81
Devices 92
File Descriptions 112
Logical Device Numbers 36
Memory Assignment 333
System Stack 333
p-System (See p-System) 336

- R -

RAM 9
RAM Disk 273
Access to 108
Boot files 82
Bootstrap 89
Configuration 91, 309, 480
Directory 89
Memory allocation 336
Multi-User 91
Operation 89
Recovering 89
Size 89
Standard Configurations 338

Read Allowed 328, 393
Real Arithmetic 84
REALCONV 109
REALOPS.x 85
Real Time Clock 10, 105, 142
Reboot option 303
RECEIVE 223, 226, 229
Receive Buffer Length 300
Registers, TMS9914 . . . . . 206, 214
REMINTEST 223, 228
Remote Channel
    Configuration 224, 479
    Connection 19, 224
    Data 460
    Data transfer 223
    Flag 304
    handshaking 226
    Read 448
    Transmit 19
    UNITSTATUS 101
    Write 448
Remote Mode 297
REMOUT 223
REMOUTTEST 223, 228
REMTALK 227, 233
Reset 278
Ringing Detect 103
RS232 223

- S -

SAGEDATE 140
SAGE II MU Environment 268
SAGE IV MU Environment 270
SAGE.PBOOT.TEXT 486
SAGE 10 Sector Format 308
SAGETOOLS 125, 127, 280
SAGE4UTIL 89, 307
Scheduled Events 149, 452
Seconds 179
Sectors 176
Security 337
Semaphores 260, 327, 395

Example 404
Implementation 397
Lockup Protection 403
Philosophy 395
Request codes 400
Size 398
SEND 223, 226, 229
Serial
Devices 92
MU Channel Configuration 296
Poll, IEEE-488 216
Ports 22
Service Requests, IEEE-488 216
Shared
Data Access 260
Terminal 263
Shared Terminal 254, 293
Emulation 420
Shipping Track 181
Sides 176
Signals, Bus 15, 16
SIO_EXAMPLE 132
SIO_Unit 125, 129, 140
SIO_x routines 129
Six Users 295, 334
Size
Partitions 52
PROM 10
RAM Disk 89
Winchester 32
Skew 176
Stack Overflow 278, 302
Standard Configurations 338
Startup Environment 270
Startup Message 302
Stop Bits 172, 174, 297, 299
Remote channel 224
Strapping Options
IEEE-488 221
memory 6
PROMs 10

STRING I/O UNIT 129
Strings 130
Subsidiary Volumes 51
SUPERVISOR Mode 17, 465
Switches
    GROUP-B 196
    IEEE-488 196
SYS_INFO 125
System
    Clock 451
    Clock Access 105
    Configuration 472
    Manager 255
    Manager Flag 287
    MU Files 264
    Stack Address for User Tasks 294
SYSTEM.BIOS 82, 486
SYSTEM.INTERP 82, 487
SYSTEM.MISCINFO 82, 89, 262
SYSTEM.PASCAL 82, 89
System Stack 333
SYSTEM.STARTUP 140

- T -

TAD_Unit 125, 140, 142
TAD_x Routines 135
Talkers, IEEE-488 . . . . . 201, 213
Tasks, (See User Tasks) 284
TELE.HAYES 234
TELE.RACAL 234
TELETALKER 234
TeleVideo 925 . . . . . 264, 293
Temperature 12
TERMCRLF 486
Terminal
    Baud Rate 262, 297
    Cables 263
    Configuration 296, 475
    Data bits 297
    Emulation 420
    Mode 297
    Parity 297

Shared 263, 293
Stop bits 297
Types 262
Write a character 447
XON/XOFF 305
TERMTEXT 486
TEXTIN 223, 227, 231
Time
Access 105
Adjustment 312
Correction 105
Polling 175
Slice 257, 295, 390
TAD_Unit 135
Timeout 175
Timeout Threshold 329, 393
Timer 10
TMS9914
Address register: 221
Register control 206, 214
Support Programs 191
Tool Kit 125
Transmit Queue 410
TRAPs
BIOS 443
TRAPV error 319
Two Word
BASIC 88
FORTRAN 87
PASCAL 86

- U -

UNITBUSY 96
UNITCLEAR 390
UNITREAD 96, 104, 108
Units 126
Special SAGE 125
UNITSTATUS 96, 101
UNITWRITE 96, 104, 108
USART 8251A 19
User Configuration 283
User Tasks 256

Base Memory Address 294
Boot Device 292
Capability Mask 287
Configuration 283, 284
Exclusive control 260
Key to swap 293
Memory Assignment 333
Priority 294
p-System 278
System Stack Address 294
Task #0 335
Time Slice 295
Top Memory Address 294

- V -

Verify Winchester 69
Very Local Area Network 261

- W -

Waiter Task 98
Weight 12
WFORMAT.CODE 54, 55, 62, 73, 264
WILD 125
Winchester 32
Control Unit 189
Drive Configuration 311, 312
WIN_Unit 125, 189
Working Master Diskette 84
Write Allowed 328, 331, 393

- X -

Xmit Buffer Length 300
XON/XOFF 224, 226, 304
Configuration Flag 172
Terminal 305

- Z -

Zero Divide 319