SAGE
COMPUTER TECHNOLOGY

Sage II
Sageutil
Bios
Installation

35 NORTH EDISON WAY, SUITE 4 • RENO, NEVADA 89502 • (702) 322-6868

FEDERAL COMMUNICATIONS COMMISSION

RADIO FREQUENCY INTERFERENCE STATEMENT

WARNING - This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a CLASS A computing device pursuant to Subpart J of part 15 of FCC rules, which are assigned to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to correct the interference.

LIMITED WARRANTY

For a period of 1 full year from the date of delivery, SAGE COMPUTER TECHNOLOGY warrants to the original purchaser that the computer hardware described herein shall be free from defects in material and workmanship under normal use and service.

During the warranty period, if a defect should occur, the product must be returned to SAGE or a SAGE authorized dealer for repair, and proof of purchase must be presented. If this product is delivered by mail, purchaser agrees to insure the product or assume the risk of loss or damage in transit, to prepay shipping charges to the warranty service location and to use the original shipping container or equivalent.

SAGE COMPUTER TECHNOLOGY will, at its option, repair or replace products under warranty at no additional charge except for service to repair damage resulting from accident, disaster, misuse, abuse, or modification not authorized by SAGE. Repair parts and replacement products will be furnished on an exchange basis and will be either reconditioned or new. All replaced parts and Products become the property of SAGE.

No representation or other affirmation of fact, including, but not limited to, statements regarding capacity, suitablity for use, or performance of the equipment, shall be or be deemed to be a warranty or representation by SAGE, for any purpose, nor give rise to any liability or obligation of SAGE whatsoever.

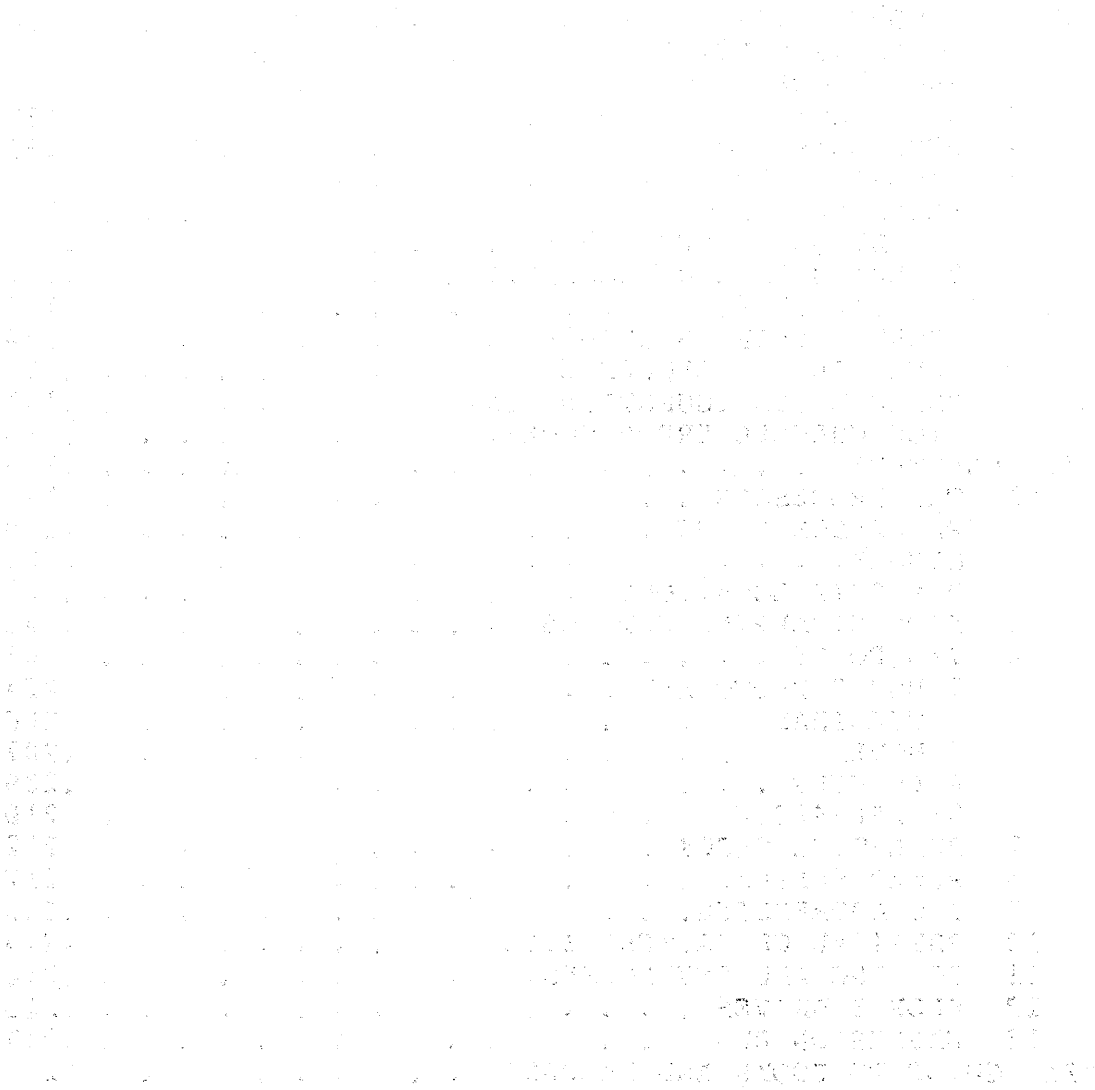
SAGE COMPUTER TECHNOLOGY shall have no liability or responsibility to customer or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by computer equipment or programs sold by SAGE, including but not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer or computer programs.

Some states do not allow certain limitations to warrantees so the above conditions may not apply. Therefore, although this warranty gives specific legal rights, the purchaser may have other rights which vary from state to state.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont...)

TABLE OF CONTENTS (cont...)



INTRODUCTION

Welcome to the SAGE II Owners' Manual. The SAGE II is a high performance microcomputer oriented toward a single user environment. The SAGE II comes with the powerful UCSD p-System software package including the high level languages Pascal, FORTRAN, and BASIC. Much of this manual pertains to the integration of the SAGE II hardware with the UCSD p-System software. Other software environments are available from SAGE Computer Technology or other independent sources. These optional software packages will generally rely on this manual for a system overview and hardware presentation.

This Owners' Manual is not intended to be a tutorial. If you are new to the small system environment, the references given in the GUIDE TO BOOKS AND MANUALS section will be valuable to you. The information presented in this manual is specific to the SAGE II computer. This manual is packaged along with a series of manuals specifically covering the UCSD p-System software.

Chapters II and III of the SAGE II Owners' Manual present necessary information on getting started with the system and installing the hardware. Users with standard cables and peripherals should easily be able to get their system running with this information. Users preparing their own cables or interfaces should refer to the chapter on Hardware for more detailed interface information.

Chapter IV presents many details on the integration of the SAGE II with the p-System. The first few sections explain how to get the p-System software up and running. The concept of using the extra system RAM area as a device (RAM Disk) is explained. The peripheral assignments and memory allocation are described. The processor specific linkage information for combining assembly code with Pascal (or other p-code) programs is described.

Chapter V describes the utility program, SAGEUTIL, which is used for configuring the I/O system as well as for formatting diskettes and copying bootstraps. The earlier chapters on installation will have already guided the user through specific usages of SAGEUTIL in order to bring up the system.

Chapter VI presents the concept of a SAGE Tool Kit, a series of preprogrammed routines (p-System Units) which may be easily used to take advantage of SAGE II features. Initial Tool Kit Units included are a String Utility package and a Time and Date Utility package. Chapter VII continues the SAGE Tool Kit presentation with a Unit for handling the IEEE 488 bus interface.

One of the major benefits of the UCSD p-System is its program portablility to and from other systems. Chapter VIII presents a set of Computer Intercommunication programs to facilitate moving information between a SAGE II and other computers systems.

Chapter IX presents complete information on the permanently resident software located in the hardware PROMS. These routines provide startup initialization and self test as well as a set of primative drivers for some devices. Also located in PROM is a Debugger program for handling processor detected errors and checking out low level assembly code routines.

Chapter X covers the Basic Input Output System (BIOS) which is a set of peripheral devices drivers. The interface to the BIOS will be of interest to anyone wishing to develope a software environment independent of the p-System.

Chapter XI presents information on the SAGE II hardware implementation. Following this information are appendices containing summary information from the preceeding chapters. Also included is a hardware parts location and set of schematics.

The Owners' Manual concludes with a USUS Membership Application and a SAGE Trouble Reporting Form. USUS, the UCSD p-System Users Society, is a non-profit organization of p-System Users who exchange ideas and programs and promote p-System concepts. The SAGE Trouble Reporting Form is provided to document problems or suggestions for feedback to SAGE.

The SAGE II Owners' Manual is distributed in the first of four binders provided with the SAGE II computer. The first binder also contains the p-System Installation Guide written by Softech Microsystems. The only pertinant information in the Installation Guide for SAGE II users is chapter III on Terminal Handling. Note also that some of the Terminal Handling information is amended by the p-System Supplement for Version IV.1 (chapter IV, Installation Guide Supplement) contained in the third binder.

The second and third binders contain the p-System Users' Manual written by Softech Microsystems. The material in the second binder covers the p-System operating system, the Editors, the Pascal compiler, and the Assemblers. The third binder continues with discussion on the Linker, Segments, Units, and Concurrency as well as covering various Utility routines provided by the p-System. The last half of the third binder contains a p-System Supplement for Version IV.1. This manual contains information on enhancements contained in the IV.1 release as well as documentation on features such as Native Code Generators, Print Spooling, XenoFile, and Turtlegraphics.

The fourth binder contains manuals for the p-System programming languages FORTRAN 77 and BASIC. Also this binder contains a p-System Internal Architecture Guide which covers details on internal implementation of the p-System software.

An Interface Design Package is available from SAGE which contains source listings of the resident PROM program, BIOS, SAGEUTIL, and other utilities. Also included in this package is more detailed hardware information with manuals on the 68000 processor and the system's peripheral control components.

I.1 THE SAGE II- FEATURES AT A GLANCE

The SAGE II is a powerful, fast, small computer system with large system capabilities at a low cost. It was designed to implement high-level languages easily. Here, at a glance, are the features that give the SAGE II the most power-per-price of computers available today.

MC68000 16-bit processor 2 million instructions per second

Multi-color status LED

Sage expansion bus: 16-bit data bus, 24-bit address bus

All input and output is interrupt driven, optionally polled

128K to 512K byte dynamic memory

Byte level parity checking

Real-time clock

Task scheduler

Two RS232-C serial ports

Parallel printer port

IEEE-488 GPIB port

Easy to interface BIOS

DEBUGGER for software development

Choice of 48TPI or 96TPI floppy disk drives

Compact sturdy metal case - weighs under 18 lbs

Quiet 20 CFM fan

Low power requirements (70 watts)

Switching power supply

UCSD p-System Software with Pascal, FORTRAN and BASIC

1 year warranty

I.2 CONVENTIONS USED IN THIS BOOK.

The following notation and abbreviations are used throughout this manual:

<CR> = carriage return: The "return" key on most terminals.

I/O = Input/Output: referring to any device or data read or written from the SAGE II processor.

addr = address: a memory location, can be PROM, RAM or I/O.

reg = register: the 68000 data and address registers.

HEX = hexadecimal: A means of expressing a value in base 16 where each digit can vary from 0...9,A,B,C,D,E,F. Numbers given in hex generally have an "H" before or after the value.

xxxK = 1024 times: 256K bytes is pronounced 256 "Kay" bytes.

xxxM = 1,048,576 times: 2M bytes is pronounced 2 "Mega" bytes.

TPI = Tracks per inch: a measure of the storage capacity of a diskette.

SWx = a switch setting on the back of the SAGE II

info = information

specs = specifications

II.1 BEFORE PLUGGING IN.

At this point, you're probably very eager to turn on your computer and see what it can do. However, it is very much worth your while to check out what you have first. This way, if there are any problems, you may be able to spot them now, instead of spending frustrating hours searching for the trouble later.

First, check the contents of your package for completeness. It should contain your computer, one or more manuals, and two or more diskettes. Your invoice should list all of the items you ordered, so make sure it matches what you received.

Now check for any obvious damage that may have occured during shipping. If damage is found, you should contact the shipper.

Keep your receipt in a safe place. It constitutes proof of purchase for your warranty. The statement of warranty is printed at the front of this manual. It is also be a good idea to keep the carton and padding the computer came in. Sage requires that equipment returned for repair or upgrade be sent in the original (or equivalent) container.

Be sure to read the next section on diskettes; know how to protect them. Check out the book list in the back. A good set of references can save you valuable time.

The sections that follow give, in order, the detailed steps needed to bring the system up. Briefly, they are:

1) Connecting power and the terminal. See the INSTALLATION section. See also the section **SAGEUTIL, Terminal configuration.**

2) Loading the p-System operating system. See section **p-System, Loading the system**

3) Backing up the system diskettes. See section **p-System, Backing Up System Files**

Once these are done, turn to your p-System Users' manual.

To load an operating system other than the p-System, refer to the appropriate section in its documentation.

II.2 DISKETTE CARE:

SAGE II uses 5.25 inch Industry Standard soft-sectored diskettes:

320k bytes double-density double-sided format, 40 track
or 640K bytes double-density double-sided format, 80 track

(depending on which model of floppy drive is equipped.)

If you have 80 track (96 TPI) drives be sure to buy diskettes that are **QUALIFIED** for the higher density operation.

To insert a diskette in the drive, first slide it out of its protective jacket. Keep your fingers away from the shiny diskette area. Before inserting it, check the write protect notch located at the side of the diskette. If the notch is covered, you can only read the diskette, you cannot write to it. If you want to write to it, remove the gummed-foil tape covering the notch. It is generally a good idea to write-protect (cover the notch) important system backup diskettes so that they are not accidently overwritten. Your original SAGE system diskettes are shipped already write-protected.

Diskettes are inserted into the drive, label UP. Never remove a diskette while the drive is being accessed. Seat the diskette firmly in the drive then close the drive latch by pressing down until it locks.

To remove the diskette, pull up on the latch. Grasp the diskette firmly between thumb and forefinger and remove it, taking care not to scrape the shiny diskette surface. Put it in its protective jacket.

Leave the disk drive door open when no diskette is in it.

For long life of your diskettes, treat them kindly:

1. Keep the diskette in its jacket whenever it is not in a floppy disk drive.

2. Do not leave a diskette in the floppy drive when powering up or down.

3. Keep diskettes away from devices which generate magnetic fields (transformers, AC motors, magnets, TVs, radios, vacuum cleaners, etc.). Strong magnetic fields will erase data on the diskette.

4. Do not touch the diskette surface. It scratches easily and fingers leave grease prints.

5. Direct sunlight or excessive heat may warp the diskette and make it unusable.

6. Dust, cigarette ashes or other gritty particles will contaminate the diskette.

7. Use a **felt tip pen ONLY** to write on the diskette label. A hard point will press through the label and mar the data surface.

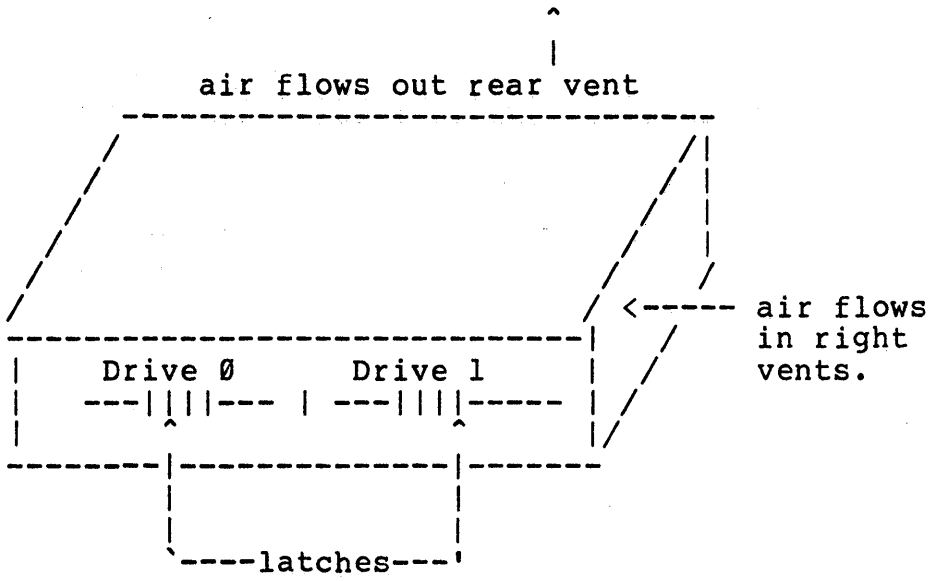
8. Store diskettes so that they are protected from pressure against the sides of the data surface. A vertical file folder is recommended or the original box.

III.1 UNPACKING THE EQUIPMENT

SAGE II is shipped fully assembled and ready to be connected to the terminal and power.

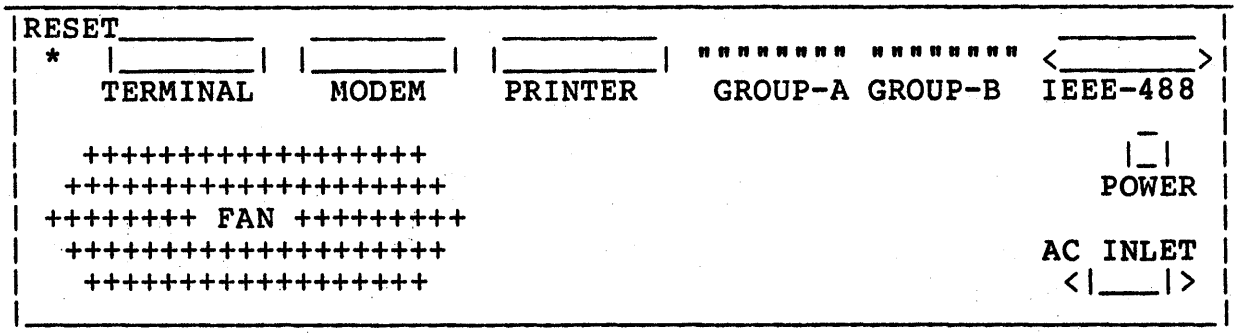
Carefully remove the computer from the shipping box. Remove and SAVE all packing material in case you need to re-ship the unit. Find and check everything on the packing list. Inspect visually for shipping damage.

Place the SAGE II on a flat surface, making sure that the air vents on the right side and the vent in the back are unobstructed for air cooling.



Press up on each floppy drive latch and remove the protective inserts. These inserts should be re-installed in the drives for drive head protection if the sytem is re-shipped.

III.2 POWER CONNECTIONS:



Connect the female plug on the Power cord to the back of the unit at AC INLET.

Connect the other end to a 120VAC 60 hz or 50 hz power source. Note that the power cord has a three-prong safety plug to provide a reliable system ground. Use of a 3-to-2 prong adapter is NOT recommended. If used, the adapter MUST BE GROUNDED.

Note: Systems which have been adapted for 220VAC operation will be indicated on the configuration label located on the bottom of the computer.

III.3 CONNECTING THE TERMINAL:

Plug in the terminal cable to the rear panel connector marked TERMINAL. The terminal connector is a standard RS-232C connector. Sage II terminal cables are available in two lengths:

CA0000 3' RS-232C terminal cable
CA0001 10' RS-232C terminal cable

Refer to the hardware section on the Terminal Port for more detail.

The baud rate is pre-set to 19.2K. This can be changed by configuring the dip switches of GROUP-A:

SW3	**SW2**	**SW1**	**Communications rate**
dwn	dwn	dwn	19.2K baud
dwn	dwn	up	9600 baud
dwn	up	dwn	4800 baud
dwn	up	up	2400 baud
up	dwn	dwn	1200 baud
up	dwn	up	600 baud
up	up	dwn	300 baud
up	up	up	reserved, will default to 19.2K baud.

The terminal port always uses 8 data bits, 1 stop bit, with parity being optional. To change the parity:

SW4	**controls the parity:**
dwn	Even parity is enabled.
up	Parity is disabled.

Note that if some characters are printed correctly but others are not, the problem is probably a mismatch in the number of data bits or parity selection between the terminal and the SAGE computer.

The terminal does not have to be turned on before the SAGE II. The BIOS ignores any illegal characters generated by the terminal so that a program will not be effected if the terminal is turned on/off during execution.

A utility program is available under your operating system to further configure the software for your terminal. For the p-System see the section in this manual on SAGEUTIL.

Caution: For terminals which have a local editing option (such as the Televideo series), make sure that the option is disabled and all editing keys are transmitted to the computer. Failing to do this may be very confusing because the terminal will echo the operations correctly on the screen but the computer will never see the keys to make the corresponding changes to the data.

III.4 INITIAL POWER-UP:

The system is configured by the factory to boot from the left floppy disk drive. The dip switches in GROUP-A determine how the system boots. Sw 7 of GROUP-A is used to configure for either the 40 track (Sw 7 up) or 80 track (Sw 7 down) floppy drives.

After following the previous unpacking and setup instructions turn on all peripherals.

Set the POWER SWITCH to ON. The screen should display:

```
 ----------------------------------------
|                                        |
|  Sage II Startup Test                  |
|  RAM SIZE = xxx K                      |
|                                        |
|  Booting from Floppy                   |
|  Put in BOOT disk and press a key      |
`----------------------------------------'
```

This indicates that your system hardware is operating properly. The ram size should match the amount of ram you ordered.

NOTE: The processor LED indicates the state of the system. Green means the bus is active, Red that it is inactive, any other color that it is in process. The LED should be **GREEN** before booting the operating system or a start-up error has occurred.

If the processor LED is not green or a message indicates a RAM or Memory error, refer to the section on Resident System Software, Power-up Program. which contains a more complete description of the system startup process.

III.5 BOOTING FROM A FLOPPY

To boot from a floppy, Sw5 of GROUP-A should be up, Sw6 should be down.

Put the BOOT floppy for your operating system in the left-hand drive. Once you press a key, the machine will boot the operating system from the BOOT floppy.

If "Not BOOT disk" is displayed, you have the diskette in wrong or it is not the correct diskette. Diskettes go in with the label up.

Now refer to your operating system section to use your SAGE II.

NOTE: THE FIRST THING YOU WANT TO DO IS

BACK UP YOUR SYSTEM SOFTWARE!

III.6 BOOTING TO THE DEBUGGER

To boot to the DEBUGGER program set SW5 and SW6 in GROUP-A down.

Turn the power on.

The terminal screen should display:

```
 ----------------------------------------
|                                        |
|  Sage II Startup Test                  |
|  RAM SIZE = xxx K                      |
|  >                                     |
|                                        |
`----------------------------------------'
```

This indicates that your system hardware is operating properly. The ram size should match the amount of ram you ordered.

Now refer to the section on SDT, the SAGE DEBUGGING TOOL.

III.7 IF YOU HAVE PROBLEMS:

Read the section on the Resident System Software POWER-UP PROGRAM for a complete description of power-up diagnostics. This should explain any error messages given you.

Read the section on Sage II hardware I/O ports to make sure your terminal is setup properly.

FOR A DUAL FLOPPY SYSTEM:

If your power-up check looks ok, but the operating system does not load from the floppy, check how you are inserting the diskette. Are you putting in the LEFT side? Is the label up with the Notch to the left? If this is ok, try changing the switch settings (SW5 and SW6 down) to boot to the debugger. Put the floppy in the right side and type

>IF1

If this works, then your left floppy drive may be bad. If not, then your diskette may be bad.

For repair, service or questions, contact your dealer or our SAGE facility:

SAGE COMPUTER TECHNOLOGY
35 N. EDISON #4
RENO, NV 89502
(702) 322-6868

Once you have determined that your system is working and have booted up sucessfully, the GROUP-A switches should be left configured for your operation.

III.8 CONNECTING THE PRINTER

The SAGE II parallel printer port "PRINTER" is usually used for the system printer. Most Centronics-like printers will run on this port with few software changes. The "MODEM" port can be used for printers with a serial interface. The section on "HARDWARE" gives the pinout of this port for those interfacing their own printer. The section on SAGEUTIL describes what printer options you can adjust under the p-System.

Three printers are currently supported on the SAGE II.

TI OMNI 810
Connect to the second serial port "MODEM"
Use SAGE cable # CA0003 : 10' RS232 cable
or # CA0004 : 25' RS232 cable
Execute SAGEUTIL.
Configure printer option for;
"A(Printer on REMOTE channel"
Configure remote channel OPTION for:
"Data Set Ready check before transmitting
baud rate to match printer
"7 data bits"
"1 stop bit"
"baud rate to match printer"

RADIO SHACK DAISY WHEEL PRINTER II
Connect to the parallel port "PRINTER"
use SAGE cable # CA0005 : 6' flat parallel cable

Execute SAGEUTIL.
Configure printer option for :
"C(printer on parallel port with scheduled polling"
and "L(inefeed after carriage return is inhibited.

NEC SPINWRITER 5530
  Connect to the parallel port "PRINTER"
    use SAGE cable # CA0005 : 6' flat parallel cable
Execute SAGEUTIL.
  Configure printer option for
     "B(Printer on parallel port with interrupts"

Attach the printer to the SAGE II with the proper cable and connect power.

NOTE: Pinouts for the connectors and cables are given in the HARDWARE section under I/O ports.

III.9 RESETTING THE SYSTEM

Pressing the RESET switch causes the computer to abort its current process and re-do the power-up sequence. Diskettes do not need to be removed from the drives for a RESET but should always be removed before powering down.

NOTE: If you are running under the p-System using the SAGE II RAM Disk option and you reset, files in RAM Disk are lost.

IV UCSD 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 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 the 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 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 Intercommunication 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 Users' manual explains all the commands of the Editor, the Filer Linker, Assembler, Compilers and specific information about the UCSD implementation of Pascal.

The following sections of the SAGE II Users' Manual give specific information about the BIOS and special support routines provided by Sage:

a) How to get the operating system loaded.
b) Differences in the SAGE II p-SYSTEM implementation.
c) Special features in the SAGE II p-System implementation.
d) How much memory is available for your program.

IV.1 LOADING THE SYSTEM:

At this point, you should have completed all the instructions given in the installation section. Your terminal screen should say:

```
SAGE II Startup test
Ram Size = xxx K
Booting from Floppy
Put in BOOT disk and press a key.
```

If you have not yet done so, read at least the first three chapters of the p-System Users' Manual to become familiar with the terminology and use of the operating system and the Filer. Insert the disk marked SYSTEM in the left drive. The label should be up. The latch must be closed.

Press a key. The disk drive should begin to read the disk. When finished, the screen should display the p-System command line:

```
Command: E(dit, R(un, F(ile, C(omp......?[IV.1 B4h]

 Welcome SYSTEM, to
 U.C.S.D. p-System IV.1
 Current date is da-mon-yr
```

Each command is the first character of one of the choices shown. If your system does not boot, check that you are using a diskette with a boot area and the system files on it. A later section describes the files you need for booting.

When your system boots up, one of the first things you MUST do is back up the system files you just received.

IV.2 BACKING UP SYSTEM FILES:

Keeping back-up copies of your files is one of the first things a programmer learns, usually the hard way. Losing your only copy of a day's work is painful. Having to re-type in even an hour of changes is valuable time lost.

The first thing you, as a new user of a SAGE II, must do is backup the diskettes that SAGE sent you. These diskettes are write protected and cannot be written to. We suggest that the user make a set of working system masters which are configured for his system, and then use them to generate system diskettes for everyday usage. This means:

1) Format your blank diskettes for your drives. The disk format defines how the data is stored on the diskette. It is slightly different for different disk drives and systems. For example, on the SAGE II, diskettes can be formatted for either 40 track or 80 track drives. Brand new diskettes must always be formatted for your drives.

2) Copy system files to the new formatted diskettes. The diskettes must be given a name and a directory area. A directory is a table of where the files are located on the diskette. ALL files are copied to the diskette.

3) Check the new diskettes to make sure they work. Store the original diskettes in a safe place.

These operations will now be described in detail.

IV.2.1 FORMATTING A DISKETTE:

Your system diskette should be in the left-hand drive (#4:).
Your new diskette should be in the right-hand drive (#5:).

The screen shows:	**You type:**
Command: E(dit, R(un, F(iler, C(omp, L(ink..	X
Execute what file?	SAGEUTIL <CR>
C(onfigure, B(oot Copy, F(ormat, Q(uit	F
Floppy Diskette Formatter Drive to be formatted (4 or 5)?	5
SAGE double side, 80 track (1280 blocks) or SAGE double side, 40 track (640 blocks)	
Is diskette ready for formatting in drive 5?	Y
.....	
.....	
.....	
.....	
Verification	
.....	
.....	
.....	
.....	
Format complete More diskettes to format?	Y if more N if done
C(onfigure, B(oot Copy, F(ormat, Q(uit	Q

Now you should be back at the command line. Refer to the section on SAGEUTIL for more detail on the Formatter.

ZEROING A DISKETTE DIRECTORY:

A diskette should be given a ZEROED directory once it is formatted. To do this:

The screen shows:			**You type:**
Command: E(dit, R(un, F(iler, C(omp, L(ink...?			F
Filer: G(et, S(ave, W(hat, N(ew.....?			Z
Zero dir of what vol?			#4: or #5:<cr>
# of blocks on disk?		(40 track)	640 <cr>
	or	(80 track)	1280 <cr>
New vol name?		(7 char max)	x: <cr>
x: correct?			Y
Filer: G(et, S(ave, W(hat, N(ew.....?			Q

Now you should be back at the Command line:
Command: E(dit, R(un, F(iler, C(omp, L(ink...?

IV.2.2 COPYING SYSTEM FILES

Now that you have a formatted diskette, copy the system files:

The screen shows:	**You type:**
Command: E(dit, R(un, F(iler, C(omp, L(ink...?	F
Filer: G(et, S(ave, W(hat, N(ew.....?	T
Transfer what file?	#4:,#5:<cr>
Transfer 640 blocks? (40 track)	Y
or 1280 (80 track)	
Destroy xx:?	Y
System: ---> #5:	
Filer: G(et, S(ave, W(hat, N(ew.....?	Q

Now you should be back at the Command line:
Command: E(dit, R(un, F(iler, C(omp, L(ink...?

Note that this method makes a complete copy of the image of the left diskette on the right diskette. The bootstrap area will be copied and both diskettes will have the same name. Do not attempt to continue using the system with two diskettes which have the same name. The system may become confused and not use the diskette you expect. Once an image transfer is accomplished, the source diskette should be removed.

SINGLE DRIVE COPY OF FILES:

If you have only one floppy disk drive, copying diskettes can be done using the same method, but it will be very slow. The new diskette must have a zeroed directory, see the previous procedure. In the Filer, specify T to transfer:

```
Transfer what file?                    #4:,#4:
```

This method will be SLOW because you will have to swap the diskettes in and out many times. (20 swaps for an 80 track diskette). If you have RAM Disk, it is much faster to transfer as many files as possible to RAM Disk. Your RAM Disk area (#11:) must be enabled with SAGEUTIL, zeroed and set to its maximum block size. See the section on the RAM Disk for details. In the Filer, specify T to transfer:

```
1. Transfer what file?                 #4:?,#11:$ <cr>
```

The program will ask you if you want to transfer each file, answer "Y" for those that you need. An "ESC" key will stop the process if you run out of room. Remove the master floppy and insert the new (zeroed) one. Transfer the files from RAM DISK to the new floppy:

```
2. Transfer what file?                 #11:?,#4:$ <cr>
```

Now remove all of the files on #11: to make room for the next transfer.

```
3. Remove what file?                   #11:= <cr>
   Update directory?                   Y
```

Put the master floppy back in and repeat steps 1-3 until all files are done.

If you do not have RAM Disk, it will be faster to only transfer those files you need, one at a time, than to transfer all blocks on the diskette.

If you have transferred selected files, you must then use SAGEUTIL to COPY the BOOT area to your new diskette.

CHECKOUT THE NEW SYSTEM DISKETTE:

1. Remove both diskettes.
2. Put the new system diskette in the left side (#4:).
3. RESET the computer.
4. Your system should boot to the command line. (See the Power-up section).

BACKUP THE OTHER DISTRIBUTION DISKETTES:

The **RELEASE NOTES** contain the directories of the distribution diskettes sent with your system. Even though you may not plan to use them right away, you should back all of them up. This is to verify that the diskettes can be read by your system and that you have received all the diskettes with your order.

Store your master distribution diskettes in a safe place. (See the section on DISKETTES for care and handling guidelines.)

IV.3 BUILDING YOUR SYSTEM

There are several configuration options available to customize your system for its peripherals and your application. These options usually require modifying files and file names so it is best to make a 'Working Master' diskette which is configured for your requirements. It is easiest to start creating the 'Working Master' from a copy of the distribution diskette called SYSTEM. Boot up on this 'Working Master' diskette so that it is the system device.

The first configuration that should be done is to adapt the system to your terminal. The p-System software uses some of the special display features available on most terminals. Unfortunately most terminals use different protocalls to control these features. The file SYSTEM.MISCINFO contains the information which tells the system what characters to output to control the terminal display features. The SYSTEM.MISCINFO file shipped with the system does not define any special features so that conflicts with your terminal will not cause an unreadable screen.

Several xxxxxx.MISCINFO files have been provided for certain terminals (see System Release Notes for information on supplied configurations). If an appropriate MISCINFO file is available for your terminal, then use the Filer to T(ransfer the file from the appropriate distribution diskette to the 'Working Master' under the name SYSTEM.MISCINFO. The Filer will ask if the original file is to be deleted and you should answer 'Y'. Try rebooting your system and the Command: prompt line should appear at the top of the screen.

If an appropriate xxxxxx.MISCINFO file is not available for your terminal then you will have to use the program SETUP.CODE to modify the existing SYSTEM.MISCINFO. Information describing SETUP is contained in the p-System Installation Guide, section III.2 (last half of first binder). Note that some new information (generally not involving the terminal) has been added in the Version 4.1 release. These changes to SETUP are documented in the p-System Supplement, section IV.3 (last half of third binder). Remember that the SETUP program creates a file NEW.MISCINFO which must be changed to SYSTEM.MISCINFO and the system rebooted.

The Version 4.1 p-System release can now also handle an ANSI standard terminal such as a DEC VT1ØØ. Handling an ANSI standard terminal is somewhat more complex and involves installing a new SCREENOPS Unit (from file ANSI.CODE) into the operating system. Information on this installation is contained in the p-System Supplement, section IV.1.7.

IV.3.1 SETUP.CODE Notes

The following Setup program information is used by SAGE internally on the TeleVideo 925 terminal. The items marked with a single '*' are appropriate for any SAGE II computer.

	BACKSPACE	BS
*	CODE POOL BASE [FIRST WORD]	1
*	CODE POOL BASE [SECOND WORD]	H3400
*	CODE POOL SIZE	H4DFF
**	EDITOR ACCEPT KEY	ETX = ^C
**	EDITOR ESCAPE KEY	ESC
	EDITOR EXCHANGE - DELETE KEY	W
	EDITOR EXCHANGE - INSERT KEY	Q
	ERASE LINE	NUL
	ERASE SCREEN	NUL
	ERASE TO END OF LINE	T
	ERASE TO END OF SCREEN	Y
**	FIRST SUBSIDIARY VOL NUMBER	13
*	HAS 8510A	FALSE
*	HAS BYTE FLIPPED MACHINE	TRUE
*	HAS CLOCK	TRUE
*	HAS EXTENDED MEMORY	TRUE
	HAS LOWER CASE	TRUE
	HAS RANDOM CURSOR ADDRESSING	TRUE
	HAS SLOW TERMINAL	FALSE
note 1	HAS SPOOLING	FALSE
*	HAS WORD ORIENTED MACHINE	FALSE
**	KEYBOARD INPUT MASK	127
**	KEY FOR BREAK	NUL
**	KEY FOR FLUSH	ACK = ^F
**	KEY FOR STOP	DC3 = ^S
**	KEY TO ALPHA LOCK	DC2 = ^R
**	KEY TO DELETE CHARACTER	BS
**	KEY TO DELETE LINE	DEL
**	KEY TO END FILE	ETX = ^C
	KEY TO MOVE CURSOR DOWN	SYN = ^V
	KEY TO MOVE CURSOR LEFT	BS
	KEY TO MOVE CURSOR RIGHT	FF = ^L
	KEY TO MOVE CURSOR UP	VT = ^K
	LEAD IN FROM KEYBOARD	ESC
	LEAD IN TO SCREEN	ESC

**	MAX NUMBER OF SUBSIDIARY VOLUMES	4
**	MAX NUMBER OF USER SERIAL VOLS	Ø
	MOVE CURSOR HOME	RS = ^^
	MOVE CURSOR RIGHT	FF = ^L
	MOVE CURSOR UP	VT = ^K
**	NON PRINTING CHARACTER	?
	PREFIXED[DELETE CHARACTER]	FALSE
**	PREFIXED[EDITOR ACCEPT KEY]	FALSE
note 2	PREFIXED[EDITOR ESCAPE KEY]	TRUE
	PREFIXED[EDITOR EXCHANGE - DELETE KEY]	TRUE
	PREFIXED[EDITOR EXCHANGE - INSERT KEY]	TRUE
	PREFIXED[ERASE LINE]	FALSE
	PREFIXED[ERASE SCREEN]	FALSE
	PREFIXED[ERASE TO END OF LINE]	TRUE
	PREFIXED[ERASE TO END OF SCREEN]	TRUE
	PREFIXED[KEY TO DELETE CHARACTER]	FALSE
	PREFIXED[KEY TO DELETE LINE]	FALSE
	PREFIXED[KEY TO MOVE CURSOR DOWN]	FALSE
	PREFIXED[KEY TO MOVE CURSOR LEFT]	FALSE
	PREFIXED[KEY TO MOVE CURSOR RIGHT]	FALSE
	PREFIXED[KEY TO MOVE CURSOR UP]	FALSE
	PREFIXED[MOVE CURSOR HOME]	FALSE
	PREFIXED[MOVE CURSOR RIGHT]	FALSE
	PREFIXED[MOVE CURSOR UP]	FALSE
**	PREFIXED[NON PRINTING CHARACTER]	FALSE
**	PRINTABLE CHARACTERS	13,32..126
	SCREEN HEIGHT	24
	SCREEN WIDTH	8Ø
*	SEGMENT ALIGNMENT	Ø
*	STUDENT	FALSE
	VERTICAL MOVE DELAY	6

* indicates proper setting for a SAGE II System
** indicates suggested assignment (but not required)

Note 1 This item should be marked TRUE only if the Printer Spooler is desired.

Note 2 If the LEAD IN FROM KEYBOARD is ESC then the PREFIXED[EDITOR ESCAPE KEY] should be TRUE. This will require that two Escape keys be typed for the EDITOR ESCAPE function.

IV.3.2 GOTOXY ROUTINE

The p-System Editor and some other programs rely on a routine in the operating system called GOTOXY which controls the terminal cursor position. Again, each type of terminal usually has a different protocall for controlling the cursor position. The system is shipped with a routine which uses the single character cursor movements (defined with SETUP) to simulate the GOTOXY function. This method however is generally slower than using the terminal's built-in method.

Several xxxxx.GOTO.CODE files have been provided for certain terminals (see list in the System Release Notes). ANSI standard terminals should use the file ANSIGOTOXY.CODE. If an appropriate GOTO file is available for your terminal, then use the LIBRARY program to install it in the operating system (SYSTEM.PASCAL). If an appropriate GOTO file is not available then the file SAMPLEGOTO.TEXT may be used as a base from which to write a GOTOXY routine for your terminal. Information on writing and/or installing your own GOTOXY is contained in the p-System Installation Guide, section II.3.1. Also important for users writing their own routine is a change to the GOTOXY protocall for Version 4.1 which is documented in the p-System Supplement, section IV.1.6.

If for some reason the standard GOTOXY that is shipped with the system does not work on your terminal, then you may have to use the non-screen oriented editor YALOE.CODE to build a new GOTOXY routine. Note that an initial version of YALOE had a bug which would cause it to fail in an extended memory system (such as the SAGE II). The System Release Notes will indicate if the bug is still present and a method of working around the problem.

IV.3.3 Terminal XON/XOFF

Some terminals cannot keep up with handling certain functions when receiving information at a high baud rate. Many terminals have an XON/XOFF protocall which allow them to tell the computer when to stop sending and when to continue sending. If this option is necessary, the configuration utility routine SAGEUTIL.CODE should be used to select the Terminal Option for XON/XOFF. Note also that the p-System Start/Stop character must be defined as Ctrl S (XOFF). This is the normal assignment but it may have been modified with SETUP. Also the XON/XOFF protocall is normally a terminal option and will have to be enabled in the terminal.

IV.3.4 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 items are dependent on the real arithmetic precision. The Interpreter contains the primative real arithmetic routines. The files INTERP.Ø.CODE, INTERP.2.CODE, and INTERP.4.CODE contain interpreters with no reals, 2 word reals, and 4 word reals. Enough room is allocated in the system memory assignment for either of the interpreters so the memory saving by using no reals (INTERP.Ø.CODE) is not realized. The SYSTEM.INTERP on the distribution diskette is a copy of the file INTERP.4.CODE. If 2 word real arithmetic is desired then use the Filer to change the name of INTERP.2.CODE to SYSTEM.INTERP. The other interpreter files may be removed from the 'Working Master' diskette using the Filer.

The more complex real arithmetic routines, such as transcendental functions, are contained in the files REALOPS.2.CODE and REALOPS.4.CODE for 2 word and 4 word arithmetic. One of these files is usually installed in the SYSTEM.PASCAL operating system then the LIBRARY program should be used to install REALOPS.2.CODE in the operating system. The p-System Supplement, section IV.1.5, contains more detailed information on this installation. Once the REALOPS Unit is installed into the operating system, the REALOPS files may be removed from the 'Working Master' diskette.

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 and FORTLIB4.CODE for 2 word and 4 word arithmetic. The runtime library files for BASIC are BLIB.R2.CODE and BLIB.R4.CODE for 2 word and 4 word arithmetic. The appropriate runtime library for the language and arithmetic being used should be installed into the SYSTEM.LIBRARY file with the LIBRARY program. Pascal programs do not require a special runtime library. Once the appropriate library file is installed in SYSTEM.LIBRARY, the runtime library files may be removed from the 'Working Master' diskette.

The FORTRAN 77 and BASIC compilers each have 2 and 4 word versions. The FORTRAN compiler files are called FORTRAN2.CODE and FORTRAN4.CODE. The BASIC compiler files are called BASIC.R2.CODE and BASIC.R4.CODE. If FORTRAN or BASIC are to be used, the appropriate compiler should be transfered to the 'Working Master' diskette with the name SYSTEM.COMPILER. The distributed SYSTEM.COMPILER file is for Pascal. 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.

IV.3.5 PRINTER CONSIDERATIONS

The printer has several configuration options which are briefly discussed in the previous section on Connecting the Printer. The section on SAGEUTIL pertaining to the Printer Configuration discusses the BIOS configuration options necessary to set up the printer on the Remote Serial port or the Centronics compatible parallel port. An option is available to suppress the automatic Line Feed character normally generated after each Carriage Return character. Also if the printer is connected to the Remote Serial channel, several options are available to set up the channel characteristics (baud rate, number of data bits, number of stop bits and parity) as well as handshaking protocalls (XON/XOFF and Data Set Ready polling).

Printer Spooling

The Print Spooler is a program that allows the user to queue and print files concurrently with the execution of the p-System. The Print Spooling is documented in the p-System Supplement, section III.2. In order to enable the Print Spooling function the option 'HAS SPOOLING' must be set to TRUE in the SYSTEM.MISCINFO file with the program SETUP. In order to take affect, the system must be rebooted. Note that the Spooling option occupies some permanently resident memory and should only be specified if it is definitely going to be used.

IV.3.6 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 transfered 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 SAGEUTIL (MEMORY DISK) is used to enable or disable the RAM Disk feature.

The SAGEUTIL program may also be used to set up for booting to the RAM Disk as the primary System Device. 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 acess to up to another 384K (768 blocks) of RAM using file I/O. The amount of RAM generally available for RAM Disk is given below:

```
with 512K    RAM Disk size = 768 blocks
     384K                  = 512 blocks
     256K                  = 256 blocks
     128K                  = ram disk not available
```

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 booting a diskette which has RAM Disk available but does not boot to RAM Disk. The directory and files will probably still be available on device #11: as long as you do not reset the SAGE II.

IV.3.7 FILES REQUIRED FOR BOOTING:

There are certain files which must exist on a device which is used for Booting up the UCSD p-System on a SAGE II computer. These files are:

SYSTEM.BIOS	Hardware device drivers.
SYSTEM.INTERP	UCSD 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 transfered with a 'device to device transfer' by the Filer or by using the UCSD P-System utility BOOTER.CODE or the SAGE II System utility SAGEUTIL.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 sucessful operation.

Other files which are not required for booting but if used are accessed from the boot device are SYSTEM.LIBRARY, SYSTEM.SYNTAX, and USERLIB.TEXT. The major system files SYSTEM.FILER, SYSTEM.EDITOR, SYSTEM.COMPILER, SYSTEM.ASSMBLER, and SYSTEM.LINKER can be found by the operating system on any on-line file structured device.

IV.3.8 THE WORKING MASTER DISKETTE

After you have performed the previously described configurations you should arrange the order of your files on the 'Working Master' diskette for your convenience. You may do this by copying the files to another diskette in the order you want them stored (don't forget to copy the bootstrap). Also the Filer K(runch option can be told to move the files so that all the empty space is allocated starting at a specified block. Use the Extended directory listing command to determine at what block the hole should be made. Then specify 'N' to the K(runch routine when it asks 'From end of disk, block xxx ?' and give the block number for the hole when it asks 'Starting at block # ?'.

The file SYSTEM.PASCAL should occur first on the booting device. This prevents it from being moved during a K(runch which would require the system to be rebooted. It also insures that if the system is booted to RAM Disk, the operating system will be copied to the RAM Disk device. Other files which must be on the system device should be located after SYSTEM.PASCAL in order to be copied to RAM Disk during booting. These files include SYSTEM.MISCINFO, SYSTEM.LIBRARY, and USERLIB.TEXT. Other files such as various SYSTEM files may be allocated next.

When booting to RAM Disk, enough room should be left on the booted system device to handle the work files from the Editor and Compiler. This may be done without sacrificing space on the booting diskette. The file copy to RAM Disk may be terminated by M(aking a one block file called ENDBOOT[1] with the Filer. Then other utility files may be placed beyond ENDBOOT on the diskette so they won't be copied to the RAM Disk. This makes it convienient to leave the booting diskette in the drive in order to have access to less frequently used programs.

Once you have completely configured the 'Working Master' diskette for your peripherals and application, you should use it to generate a 'Working System' diskette for everyday use. The 'Working Master' probably took a while to generate and should be saved in a safe place as a backup to generate future 'Working System' diskettes.

IV.4.1 DEVICES SUPPORTED:

Device #

1	CONSOLE	(echoing version of terminal)
2	SYSTERM	(non echoing version of terminal)
3	reserved	
4	Left floppy drive	
5	Right floppy drive	
6	PRINTER	
7	REMIN	(Remote serial input channel)
8	REMOUT	(Remote serial output channel)
9	reserved	
10	reserved	
11	RAMDISK	
12	reserved	

User defined devices (by SAGE)

128	Device configuration control
129	Reading system clock
130	General Memory Access
131	Scheduled event control

Note that the IEEE-488 bus is not currently supported as a p_System device. Instead the routines for accessing the 488 interface are contained in a p-System UNIT described in a later section on IEEE-488 support.

IV.4.2 ASYNCHRONOUS I/O

The asynchronous I/O feature for UNITREAD and UNITWRITE is currently not supported. Thus bit Ø of the Control Word in these requests is ignored. The function UNITBUSY always returns false. The procedure UNITWAIT always returns immediately.

It should be noted that the Keyboard, Terminal output, Printer output, and Remote input and output 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, or Remote output channel, UNITSTATUS can be used to determine if characters may be output with UNITWRITE without hanging to wait for the device.

I/O to the floppy drives is always completed before the UNITREAD or UNITWRITE returns to the calling program.

IV.4.3 EXTRA UNITWRITE FEATURES:

The BIOS interface section contains information on additional functions which are accessable via User Control bits in the UNITWRITE procedure. These features control the floppy disk formatting and the REMOTE serial channel signals DTR and RTS.

IV.4.4 EXTRA UNITSTATUS INFORMATION:

The UNITSTATUS procedure returns information about device status as described in the UCSD p-System Internal Architecture Guide. The UNITSTATUS procedure returns information in a 30 word (60 byte) area. The following additional information is returned by the SAGE II BIOS at the end of the data areas to leave room for standard p-System expansion.

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.
```

IV.4.5 BIOS CONFIGURATION FROM p-System

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.

EXAMPLE: Disable the printer in BIOS.

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

IV.4.6 SYSTEM CLOCK ACCESS

The standard UCSD 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:

```
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 II 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 S(ystem Configuration option of SAGEUTIL. 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:

```
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    }
```

IV.4.7 ATTACH IMPLEMENTATION:

The SAGE II 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);

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 protocall will be to return a -1 for any event which is not supported. The following I/O events are supported by the SAGE II BIOS:

ATT_Key (current value is 19)
Attaching to this event will signal a semaphore whenever a key is pressed on the terminal.

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_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_Schd1 (current value is 36)
ATT_Schd2 (current value is 37)
ATT_Schd3 (current value is 38)
ATT_Schd4 (current value is 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_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 prempted 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-System implementations.

Although concurrent processing is a very powerful tool, two warnings are in order. First, 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. Second, 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 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 Ø to 63999 to represent the number of 1/64ØØØ ths 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 two bits to control the schedule functions.

Control Word

Bit Ø = Ø means set up a new schedule.
= 1 means cancel the schedule.

Bit 1 = Ø means schedule is for one time only.
= 1 means schedule is periodic on specified interval.

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 consistancy 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 consistant.

Examples:

```
USES ATTUNIT;
  { Set up a periodic scheduled signal on .5 second intervals }
VAR
  Event:INTEGER;
  Schedl:SEMAPHORE;
  TimeValue:ARRAY[0..1] OF INTEGER;
  Anything:INTEGER; {may be any variable as is not used}
BEGIN
  Event:=ATT_Lookup(ATT_Schdl);
  SEMINIT(Schedl,0);
  ATTACH(Schedl,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(Schedl,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.
```

IV.4.8 GENERAL MEMORY ACCESS

The current UCSD p-System implementation (IV.1) 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 memory from a p-System program.

1)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.

2)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]);
```

IV.4.9 SYSTEM MEMORY ALLOCATION

As shipped the p-System is configured to use memory in the following manner.

000000H to 0000FFH	Interrupt & Exception Vectors
000100H to 0001FFH	Debugger RAM Area
000200H to 0002FFH	BIOS RAM Area
000300H to 0003FFH	Debugger system stack
000400H to 0103FFH	p-System data area
010400H to 0133FFH	p-System Interpreter
013400H to 01CFFFH	p-System code pool
01D000H to 01FFFFH	BIOS code & System Stack (for 128k systems)
01D000H to BIOS	RAM Disk (for > 128k systems) (note: BIOS is placed at top of equipped RAM memory)

This allocation provides a full 64k byte p-System data area. The Interpreter and BIOS code areas are taken from the remaining 64k leaving 39k for the p-System code pool. Although the code pool is not allocated its full possible 64k, most programs are not large and dynamic enough to make additional code pool space useful.

The suggested allocation gives room in the Interpreter and BIOS areas for growth without requiring a configuration change. The two word floating point interpreter currently occupies about 9.5k and the four word interpreter currently occupies about 10.5k. A 12k area has been allocated for the interpreter. The current BIOS & buffers occupy about 10.5k and have been allocated 12k. The BIOS size is expected to continue to grow with as more features are added. Experienced users may want to reconfigure the starting location and size of the p-System code pool using the SETUP.CODE program. This should be done carefully as no cross checks are made for mistakes which cause overlap of areas. Note also that the base of RAM Disk must be changed with SAGEUTIL.CODE if the code pool is expanded beyond 1D000H.

The starting address of the interpreter is hard coded in the p_System bootstrap file SAGE.PBOOT.TEXT. Also hard coded in this file are the base and size of the p-System data area. These values and locations will generally never need to be modified because a full 64k data area is desirable.

IV.5 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 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 Supplement, section II.12.

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 writting 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.

IV.6 SPECIFIC LINK INFORMATION FOR THE 68000

WORKING IN ASSEMBLY LANGUAGE:

Some users may want to generate assembly language routines for linking with Pascal, FORTRAN, or BASIC. Other users may want to generate assembly code for a stand alone environment.

The general documentation on the p-System Assemblers is located in the p-System Users' Manual, Chapter VII (in the second binder).

Specific details on the 68000 **Assembler** are contained in the p-System Supplement, section II.9.2 (in the third binder).

The p-System **Linker** which is used to combine p-code with assembly code, or to combine separately assembled routines, is documented in the p-System Users' Manual, section VIII.4 (in the third binder).

Users who wish to generate a **stand alone** environment will be interested in the utility COMPRESSOR which follows in section X.1 of the p-System Users' Manual. This utility eliminates the p-System code file overhead and applies the relocation information to the code leaving a ready to run memory image. Sophisticated users who may want to write utilities dealing with assembly code files will find information on code file formats in the p-System Internal Architecure Guide, section II.2 (in the fourth binder).

Following is a consolidated example of using assembly code files from a Pascal program. It shows how to create, link and run an assembly code program. Later examples explain some of the operations used here.

EXAMPLE 1:USING THE 68000 ASSEMBLER:

First the Editor is used to create a Pascal program which references two assembly code procedures (denoted by EXTERNAL). This program is S(aved under the name MAINPROG.TEXT.

```
{ This is a sample Pascal program which uses assembly
  language procedures for 32 bit addition and subtraction }

PROGRAM MainExamp;
TYPE
  INT32 = RECORD
            H:INTEGER;
            L:INTEGER;
          END;

VAR
  Val1,Val2,Val3:INT32;
PROCEDURE ADD32(VAR Result,Arg1,Arg2:INT32);EXTERNAL;
PROCEDURE SUB32(VAR Result,Arg1,Arg2:INT32);EXTERNAL;
BEGIN { MainExamp }
  Val1.H:=0; Val1.L:=-1;    { Set up value of 65535 }
  Val2.H:=0; Val2.L:= 4;    { Set up value of     4 }
  { Val3 := Val1 + Val2 }
  ADD32(Val3,Val1,Val2);
  WRITELN('Addition: High word = ',Val3.H,
          '  Low word = ',Val3.L);

  Val2.H:=0; Val2.L:=-2;    { Set up value of 65534 }
  { Val1 := Val3 - Val2 }
  SUB32(Val1,Val3,Val2);
  WRITELN('Subtraction: High word = ',Val1.H,
          '  Low word = ',Val1.L);
END.
```

Next the two assembly procedures are created with the Editor and S(aved in the file ASMPROGS.TEXT.

```
;       Example assembly routines:
;       Procedures for 32 bit arithmetic

        .RELPROC        ADD32,3
        MOVEA.L (SP)+,A0        ;Save return address
        MOVE.W  (SP)+,D7        ;Get address of second argument
        MOVE.W  (SP)+,D6        ;Get address of first argument
        MOVE.L  0(A6,D6.L),D0   ;Get first argument
        ADD.L   0(A6,D7.L),D0   ;Add in second argument
        MOVE.W  (SP)+,D7        ;Get address of result
        MOVE.L  D0,0(A6,D7.L)   ;Save result
        JMP     (A0)            ;Return

        .RELPROC        SUB32,3
        MOVEA.L (SP)+,A0        ;Save return address
        MOVE.W  (SP)+,D7        ;Get address of second argument
        MOVE.W  (SP)+,D6        ;Get address of first argument
        MOVE.L  0(A6,D6.L),D0   ;Get first argument
        SUB.L   0(A6,D7.L),D0   ;Subtract second argument
        MOVE.W  (SP)+,D7        ;Get address of result
        MOVE.L  D0,0(A6,D7.L)   ;Save result
        JMP     (A0)            ;Return
        .END
```

Now **compile** the Pascal program.

SCREEN DISPLAYS:	**YOU TYPE:**
Main prompt line	C - for compile
Compiling...	
Compile what text?	MAINPROG <CR>
To what codefile?	MAINPROG <CR>
Output file for compiled listing?	<CR> - for none
Pascal compiler	
< 0>.....	
ADD32	
< 13>..	
SUB32	
< 15>...	
MAINEXAM	
< 18>.....	
30 lines compiled	
MAINEXAM .	
The code is stored in the file MAINPROG.CODE.	

Next **assemble** the 68000 assembly code routines.

SCREEN DISPLAYS:	**YOU TYPE:**
Main prompt line	A - for assemble
Assembling...	
Assemble what text?	ASMPROGS <CR>
To what codefile?	ASMPROGS <CR>
68000 Assembler [IV a.0]	
Output file for assembled listing	<CR>
< 0>..	
ADD32	
< 2>.....	
SUB32	
< 12>.....	
Assembly complete: 21 lines	
0 errors flagged on this assembly	

Now the assembly routines must be **linked** to the Pascal program. Make sure that the specified output file has the extension '.CODE' or the file will not execute.

SCREEN DISPLAYS:	**YOU TYPE:**
Main prompt line	L - for linking
Linking...	
Host file?	MAINPROG <CR>
Opening RAMDISK:MAINPROG.CODE	
Lib file?	ASMPROGS <CR>
Opening RAMDISK:ASMPROGS.CODE	
Lib file?	<CR>
Map file?	<CR>
Reading MAINEXAM	
Reading ADD32	
Output file?	EXAMPLE.CODE
Linking MAINEXAM #2	
Copying proc ADD32	
Copying proc SUB32	

Now the final linked result in EXAMPLE.CODE is ready to eXecute.

Main prompt line	X - for execute
Execute what file?	EXAMPLE
Addition: High word = 1	Low word = 3
Subtraction: High word = Ø	Low word = 5

STACK AND REGISTER USAGE:

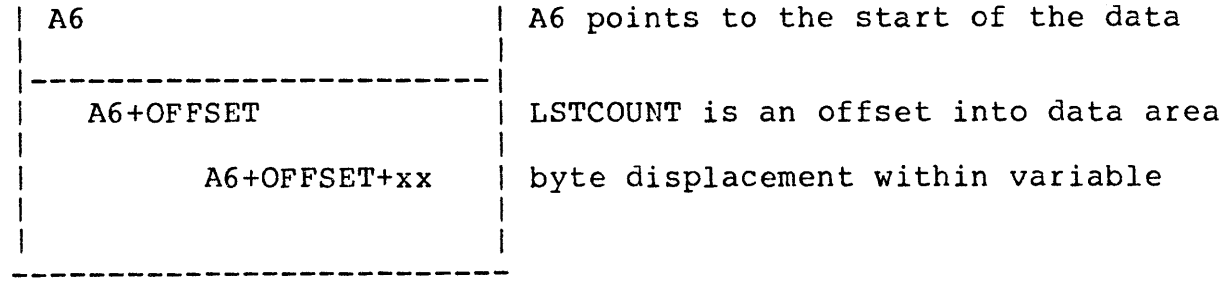
Information from the Pascal calling routine to and from the assembly routine is passed on the User Stack SP=A7. The first 4 bytes are always the return address. Then the arguments, always word values, are passed. The arguments are passed:

```
PROCEDURE EXAMPLE(arg1,arg2.....argN);EXTERNAL;

SP ----->  RET ADDRESS
           argN
           ---
           arg2
           arg1
```

When the argument is specified as VAR, the value passed from PASCAL to the assembly routine is a word OFFSET indicating where the argument is in the p-System data area. Register A6 points to this data area. To calculate the true location use the INDIRECT ADDRESSING WITH DISPLACEMENTS AND INDEXES mode:

```
Effective addr = xx (A6,REG.L)
                  |   |   |
                  |   |   `- index for location of variable
                  |   `- points to the p-System data area
                  `- byte displacement within the variable
```



The OFFSET passed from the PASCAL program must not be sign-extended as it is a positive value from Ø-64K. This means that the high part of the register REG.L used in the effective address must be zero. D6 and D7 are provided with their high 16 bits already zeroed to make it easy to MOVE.W the OFFSET into the register.

Assembly programs may use the data registers DØ-D5 and the low word of D6 and D7 without saving. The high word of D6 and D7 are zero and must stay zero. Only address registers AØ-A2 may be used without saving the previous values. Note also that the p-System Break key routine which is called from an interrupt relies on the register A3 being unmodified. Also if the Attach/Event feature is used, register A6 must also not be modified. If these registers must be used then the Foreground interrupt must be disabled (see BIOS listing) while the registers contain other values. All routines are called in USER mode, not Supervisor mode.

EXAMPLE 2:PASSING THE ADDRESS OF A VARIABLE:

Often when writing an assembly code routine, it becomes necessary to pass the ADDRESS of a PASCAL variable through as an argument, so that the assembly routine and PASCAL routine can both use the variable.

In the PASCAL routine, use the VAR specification for the argument:

```
PROGRAM TRYEXAMPLE;
VAR SHARE:INTEGER;
PROCEDURE EXAMPLE(VAR WESHARE:INTEGER);EXTERNAL;

BEGIN
  EXAMPLE(SHARE);
END.
```

In the assembly code routine calculate the addr:

```
        .PROC   EXAMPLE,1               ;one argument
START   MOVEA.L (SP)+,A0                ;Save the return address
        MOVEQ   #0,D0                   ;High part of reg should
                                        ;be zeroed
        MOVE.W  (SP)+,D0                ;Get the argument.
        LEA     0(A6,D0.L),A1           ;Calculate addr of SHARE
        .........                       ;body of routine
        JMP     (A0)                    ;return to PASCAL
        .END
```

Variables defined as ".PUBLIC" or ".PRIVATE" are also word offsets to the data area just as VAR arguments are.

Register D6 and D7 already have their high 16 bits zeroed. This provides a convenience in that the user does not have to set up a register himself:

```
        .PROC   EXAMPLE,1               ;one argument
START   MOVEA.L (SP)+,A0                ;Save the return address
        MOVE.W  (SP)+,D6                ;Get the argument.
        LEA     0(A6,D6.L),A1           ;Calculate addr of SHARE
        .........                       ;body of routine
        JMP     (A0)                    ;return to PASCAL
        .END
```

EXAMPLE 3:PASSING A STRING TO AN ASSEMBLY ROUTINE.

For a string or byte array parameter which is indicated as variable (VAR), the Pascal calling routine passes the offset of the string address to the assembly code routine. In the example the true address of the string is calculated and the string is changed. Control returns to the Pascal routine which prints out the changed string.

Note: If the address is calculated with a zero displacement (giving the start of the string) that address contains the LENGTH of the string not the first character in it.

```
        PROGRAM TRYS;
        VAR S:STRING;
        PROCEDURE EXAMPLE(VAR S:STRING);EXTERNAL;

        BEGIN
          S:='    ';
          EXAMPLE(S);
          WRITELN(' EXAMPLE SAYS:',S);
        END.
```

```
        .RELPROC   EXAMPLE,1          ;one argument
START   MOVEA.L (SP)+,A0              ;Save the return address
        MOVEQ   #0,D0                 ;Clear the high part D0
        MOVE.W  (SP)+,D0              ;Get the argument.
        LEA     2(A6,D0.L),A1         ;Calculate S[2] addr.
        MOVE.B  #"H",(A1)+
        MOVE.B  #"I",(A1)+
        JMP     (A0)                  ;return to PASCAL

        .END
```

Strings or byte arrays which are passed as Value parameters (without VAR), must be accessed in a special method using a Segment Pointer. This process is necessary because the string may either be in the data area or in the code area (string constants). The Segment Pointer is passed on the stack as two words. The first word (top of stack) contains either 0 (NIL) or a pointer to a segment environment record. If the first word is 0 then the second word is the offset of the string in the data area and may be accessed as described above. If the first word is not 0 then the access is more complicated because the data is a constant in the code area. The user must also insure that the segment with the constant being accessed is resident in the code pool. The following facts are necessary to track down the constant data.

First Word (tos) is a pointer to the EREC in the data area.

Second Word is an offset of the constant into the segment.

The third word of the EREC points to the SIB in the data area.

The first word of the SIB points to the code pool descriptor in the data area.

The second word of the SIB is the offset of the segment in the code pool.

The first two words of the code pool descriptor are a long word pointer to the base of the code pool.

```
;       Example of string value parameter access
        .RELPROC YELLOW_BRICK_ROAD,2 ;Actually one parameter
                                     ;  with 2 words
        MOVEA.L (SP)+,A0          ;Save return address
        MOVE.W  (SP)+,D7          ;Get first word parameter
        BNE.S   $10               ;String is in code area
        MOVE.W  (SP)+,D6          ;Easy access in data area
        LEA     0(A6,D6.L),A1     ;Form address of string
        BRA.S   $20

$10     MOVE.W  4(A6,D7.L),D7     ;Get pointer to SIB
        MOVE.W  0(A6,D7.L),D6     ;Get pointer to pool descriptor
        MOVEA.L 0(A6,D6.L),A1     ;Get base of code pool
        MOVE.W  2(A6,D7.L),D6     ;Get offset of segment in pool
        ADDA.L  D6,A1             ;Form address of segment
        MOVE.W  (SP)+,D6          ;Get offset of string in segment
        ADDA.L  D6,A1             ;Form address of string
$20     ...
```

EXAMPLE 4:A RELOCATABLE ASSEMBLY CODE ROUTINE WITH FIXED PRIVATE AREA.

It is often necessary to setup a data area for an assembly code routine that keeps values between calls to the routine. One way to do this is to make the assembly code a ".PROC" which fixes the entire routine in the heap. This means that there is less room for user data. By making the routine relocatable, and putting only the variables in the data area, the code now resides in the code pool. This is done with ".RELPROC" and ".PRIVATE".

The value of the label specified in the ".PRIVATE" is a word offset to the data area just as VAR arguments are.

```
        PROGRAM TRYIT;                          {simple calling routine}
        PROCEDURE EXAMPLE;EXTERNAL;

        BEGIN
          EXAMPLE;
        END.


        .RELPROC EXAMPLE                ;Relocatable assembly code.
        .PRIVATE LSTCOUNT               ;Setup storage area on heap.


START   MOVE.W  #LSTCOUNT,D6
        ADDQ.L  #4,0(A6,D6.L)           ;Increment count by 4
        RTS                             ;Return addr still on stack
        .END
```

NOTE: Variables defined with ".PUBLIC" are references to data defined in a Pascal program and must also be accessed as an offset within the p-System data area via A6.

IV.7 THE p-SYSTEM BOOTSTRAP:

The bootstrap program is located on blocks Ø 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 4ØØH. 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 SAGEUTIL 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 protocall for all SAGE II bootstrap programs requires that the first four bytes of the code (at 4ØØH) 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 correct the boot program will then be started at location 4Ø4H. The bootstrap is entered in Supervisor mode. The stack contains a return address 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, Ø 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 the proper 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 size of the BIOS code. Offset 6 is the size of the RAM buffer area which must be allocated preceeding 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.

SAGEUTIL -SYSTEM UTILITY
SECTION V

V SAGE II SYSTEM UTILITY PROGRAM:

The program SAGEUTIL.CODE handles configuration of the BIOS, copying and installing of diskette bootstraps, and formatting of diskettes. When eXecuted the file will reply:

SAGE II Computer System Utility Package - Version x

C(onfigure, B(oot copy, F(ormat, Q(uit

Each major section of the Utility Package may be entered by typing the appropriate single character prompt. Most options are selected with single character prompts. However some entries require multiple characters and must be terminated with a carriage return. A carriage return typed in place of a single character prompt will back out a level in the prompting menu.

V.1 BIOS CONFIGURATION MANAGER

Typing a 'C' at the outer level prompt line will cause the utility to enter the BIOS Configuration Manager and ask:

O(n-line, or F(ile change to BIOS?

The O(n-line option will only change the copy of the parameters which is operating in memory. This option should be used for temporary changes that only affect the current computing session.

The F(ile option will ask:
BIOS file name?

The user should specify the device and file name of the BIOS file to be modified (or examined), typically SYSTEM.BIOS. The device driver configuration parameters are located in the block 1 of the BIOS file. This data is read in for examination and/or modification by the configuration manager. The utility will report:

BIOS Version x.x header read successfully
Size including buffers: xxxxx bytes

The main prompt line for the Configuration Manager is:

T(erminal, R(emote, F(loppys, M(emory-disk, P(rinter, S(ystem, Q(uit ?

The desired prompt for the subsection of the Configuration Manager is entered to configure information for the terminal, the remote serial channel, the floppy drives, the RAM Disk storage device, the printer, and the System parameters.

At the end of the session (Q(uit or <CR> at main prompt), if changes have been made to the configuration information the utility asks:

Ready to write changes to your file ?
or Ready to write changes to memory?

The user then should type 'Y' or 'N' resulting in the message:

BIOS changes saved, Type space to continue.
or BIOS changes abandoned, Type space to continue'.

Changes made to memory involving the terminal configuration are questioned by the utility before installing. This is because the changes are applied immediately and may garble the following terminal output until the corresponding changes are made in the terminal.

NOTE: BIOS changes made to a file will **not** take affect until the system is re-booted using the modified BIOS.

V.2 TERMINAL CONFIGURATION

Typing 'T' to the Configuration Manager prompt displays a prompt line for information to configure the terminal serial channel:

```
B(aud-rate, P(arity, S(top-bits, D(ata-bits, O(ptions, Q(uit
```

Selecting B(aud-rate displays:

```
Currently using xxxxx baud rate

A( 19200 baud        H(   600 baud
B(  9600 baud        I(   300 baud
C(  4800 baud        J(   200 baud
D(  2400 baud        K(   150 baud
E(  2000 baud        L(   110 baud
F(  1800 baud        M(    75 baud
G(  1200 baud        N(    50 baud

X(use DIP switch specified baud rate
Q(uit

Select baud rate from above:
```

Selecting P(arity displays:

```
Currently parity is xxxx

D(isable parity
E(ven parity
O(dd parity
Q(uit

Select parity option:
```

Selecting S(top-bits displays:

```
    Currently x stop bit(s) assigned

    A( 1 stop bit
    B( 1.5 stop bits
    C( 2 stop bits
    Q(uit

    Select stop bit option:
```

Selecting D(ata-bits displays:

```
    Currently using x data bits
    A( 5 data bits
    B( 6 data bits
    C( 7 data bits
    D( 8 data bits
    Q(uit

    Select data size option:
```

Selecting O(ptions displays:

```
    Terminal Options

    B(REAK KEY (not p-System Break key) is ignored
 or B(REAK KEY (not p-System Break key) enters PROM Debugger

    X(on/Xoff from terminal is disabled.
 or X(on/Xoff from terminal is enabled.

    Select option to change <CR for none>
```

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 computer. The computer can continue to execute a program without being continuously interrupted by the powered down terminal.

The optional mode for the BREAK key (or framing errors) is to enter the PROM Debugger. This mode 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. The Break key mode may be toggled by selecting the 'B' option.

The 'X' option will toggle the XON/XOFF handling mode for the terminal. If the XON/XOFF mode is enabled, the terminal driver will respond to XON and XOFF characters generated by the terminal to slow down the data going to the terminal. When using this option, 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.

V.3 REMOTE CHANNEL CONFIGURATION

Typing 'R' to the Configuration Manager prompt displays:

B(aud-rate, P(arity, S(top-bits, D(ata-bits, O(ptions, Q(uit?

The baud rate, parity, stop bit, and data bit selections are the same as described for the Terminal Channel. The O(ptions selection displays:

```
I - XON/XOFF for Input is Disabled
O - XON/XOFF for Output is Disabled
D - Data Set Ready is checked before transmitting
P - Polling interval (in 1/64000's second)  16000

Select option to change <CR for none>
```

The XON/XOFF protocall is a method used to keep the receiving system from losing data if the incoming data cannot be processed fast enough and the input buffer fills up. A typical case would be where data is being transfered over the remote serial link to be stored on a diskette. The buffer could fill up and overflow while the program is writing the previously received data to diskette.

Using XON/XOFF protocall, the receiving system will send an XOFF character (13H) when the receive buffer is nearly full (240 out of 255 characters). This leaves some room for the transmitting end to respond to the XOFF by stopping transmission. The receiving program will then process characters in the input buffer. When the amount of room available is 1/2 of the buffer (128 characters) the XON character (11H) is sent to turn the transmitting driver back on. This protocall can only be used where the XON and XOFF characters are unique and are never contained in the data. Thus this method will not work for transmitting binary images of files.

The XON/XOFF protocall may be configured for either direction of transmission or both directions of transmission. When set up in the Input mode, the receiver will transmit XON and XOFF characters to control the transmitting from the opposite system. When set up in the Output mode, the receiver will respond to XON and XOFF characters which are received to start and stop the transmit direction.

The 'I' and 'O' selections will toggle the Input and Output XON/XOFF selections between Enabled and Disabled.

The 'D' selection is provided to control a transmit check on Data Set Ready. When low level signal handshaking is needed on the Remote Serial channel, the Data Set Ready input should be used instead of the Clear to Send input (see writeup on Hardware-Printer Port). When a character is transmitted the DSR bit is checked. If DSR is not active, a delay is scheduled (value controlled by 'P' selection) before the next test. The 'D' selection will toggle the enabling or disabling of the DSR check. When the check is enabled, the 'P' selection will ask:

Polling interval?

This is the delay in 1/64000's second between polling attempts of the Data Set Ready signal.

V.4 FLOPPY PARAMETER MAINTENANCE

Typing 'F' to the Configuration Manager prompt requests:

Select drive number (4 or 5 or just CR quits) ?

When the drive is selected the utility will reply:

Current drive #4 setup:

SAGE double side, 80 track (1280 blocks)

A - SAGE double side, 80 track (1280 blocks)
B - SAGE double side, 40 track (640 blocks)
C - IBM single side, 40 track (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 - Softech Universal Medium, single side, 35 track (280 blocks)
I - No drive equipped
Z - Non standard type

Select floppy option (or CR to quit):

The appropriate letter may be typed to select the option desired. 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 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 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 assending track order on side zero and then back in desending track order on side one. The normal SAGE II method is to store data on side zero and then side one of each track before stepping the head to the next cylinder.

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 provides a compatibility mode to this format but does not attempt to automatically distinguish which format is being used. Note: Attempting to record ten 512 byte sectors of data on a track will generally work but does stretch the drive specifications. This format may not be reliable on all systems.

The 'Univeral Medium' (TM) is a concept proposed by Softech Microsystems for distribution of UCSD p-System application software. This format is thought to be readable by the highest number of p-System implementations. The format may be written by 40 track (48 TPI) drives only but may be read on any SAGE system.

The 'Z' selection for Non standard type 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 protocalls and possibly controller information. The alternate menu is shown below:

Current drive #4 setup:

A - number of Sides:	2	K - bytes per sector:	512
B - cylinders:	80	L - Gap 3 parameter:	42
C - sectors per Track:	8	M - Data length:	255
D - IBM track format:	No	N - Step rate:	4
E - Density:	Dbl	O - Gap 3 for format:	80
F - Retries	3	P - Pattern for format:	229
G - Ignore errors:	No	R - Skew for format:	0
H - Read 48 on 96 TPI:	No	Z - standard options	
I - NCI 10 sects/trk:	No		
J - Read after write:	No		

Select item to change (or CR to quit):

Selecting 'A' will ask for Ø, 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.

Selecting '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.

Selecting '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.

Selecting 'D' toggles the IBM track format compatibility on or off. For double sided diskettes, data is stored in assending track order on side zero and then back in desending track order on side one. The normal SAGE II method is to store data on side zero and then side one of each track before stepping the head to the next cylinder.

Selecting 'E' asks 'S(ingle or D(ouble density. The drives provided on the SAGE II will normally be used in the double density mode. The single density option 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.

Selecting 'F' 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. A carriage return is necessary as a multiple digit number may be typed.

Selecting 'G' will toggle on or off the ignoring of 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.

Selecting 'H' will toggle on or off a feature which allows reading of 48 TPI diskettes on a 96 TPI drive. Note that writing 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.

Selecting 'I' will toggle on or off an 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).

Selecting 'J' will toggle on or off an 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 and should only be used where information is critical and cannot easily be recovered.

Selecting 'K' will ask 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.

Selecting 'L' will ask 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

Selecting 'M' will ask 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.

Selecting 'N' will ask for 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).

Selecting 'O' will ask Gap 3 for formatting. Values for this parameter are used when formatting a diskette and are contained in the Gap 3 table under selection 'L'.

Selecting 'P' will ask for a Pattern for formatting. This value (Ø to 255) is written into each data byte during the formatting of a diskette. The normal default value is 229 (E5 hex).

Selecting 'R' will ask for a Skew factor. This value is normally zero. The 1Ø 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 boundries. 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 1Ø 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.

Selecting 'Z' will toggle the menu back to the standard options.

V.5 MEMORY DISK

Typing 'M' to the Configuration Manager prompt displays:

```
RAM Disk Configuration

RAM Disk is enabled from xxxxx to yyyyy

Boot to RAM Disk is enabled

E(nable/D(isable RAM Disk or B(oot-toggle?
```

This prompt line allows the user to decide how to utilize extra memory in the system which is not being used by the UCSD p-System. Enabling the RAM Disk option allows use of all or a part of the extra memory as device #11:. The B(oot-toggle will enable and disable copying the boot diskette to RAM and then booting the system from device #11:.

When the RAM Disk is enabled the utility will request that the bottom and top addresses for the RAM Disk memory be specified. A value of zero for the Top of RAM Disk memory will default to the first location available below the BIOS & Stack.

```
Bottom of RAM Disk memory?

Top of RAM Disk memory?
```

V.6 PRINTER CONFIGURATION

Typing 'P' to the Configuration Manager prompt displays the current printer configuration along with the menu:

```
Modes:
A( Printer on Remote Channel
   (set up using Remote serial channel parameters)
B( printer on parallel port with interrupts
C( printer on parallel port with scheduled polling
D( printer is disabled

L( inefeed after carriage return is printed
```

The 'A' option is used when the printer is to be driven from the auxiliary (Remote) serial channel. When used with this option the XON/XOFF protocall for Output or Data Set Ready protocall may be specified (under the Remote Configuration section) for printers which will provide this data to control system transmission.

The 'B' option is used for printers which conform to the standard Centronics parallel port protocall with an Acknowledge signal which produces an interrupt to say that the character has been received.

The 'C' option is used for printers which use the Centronics parallel port definition but rely on the system polling the Busy signal from the printer to determine when the next transmission is possible. Using this option, after a character is output the routine will poll the Busy line for a selected number of times. If the printer has an internal buffer it may be able to receive the next character within a reasonably short time. If the printer is still busy after the specified number of polling cycles, a configurable delay is scheduled which releases the processor for other work. After each delay interval the processor returns to check if the printer is busy. In this manner a normally polled printer will not completely tie up the processor. The 'C' option displays:

```
Printing on parallel port with scheduled polling
  P(olling attempts before scheduled delay      xxx
  D(elay in 1/64000's second before repolling   xxx

Select P(olling, D(elay, Q(uit:
```

The 'L' option will toggle on on or off an option to inhibit the automatic printing of a Linefeed after each Carriage Return. Some printers cannot disable their internal Linefeed generation after a Carriage return. The p-System operating system does not have a standard method (via SETUP) of inhibiting automatic Linefeed generation during general output to the PRINTER: device.

V.7 SYSTEM PARAMETER CONFIGURATION

Typing 'S' to the Configuration Manager prompt displays the current System Parameter configuration along with the menu:

System Parameter Maintenance

A - Time adjustment: x second(s) in y days

Select item to change (or CR to quit):

Selecting 'A' will allow changing a time adjustment factor which may be necessary to prevent the Real Time clock from drifting slightly over a period of several days. The Real Time Clock values are derived from the processor clock which does not have the precision required to keep accurate time over long periods. The SAGEDATE utility displays both the current time and the last time that the Real Time Clock was set. Integer values for the number of seconds to adjust over the number of days may be easily determined. If the Real Time gains, the seconds of adjustment value should be negative. If the Real Time loses, the seconds of adjustment value should be positive. The number of days cannot exceed 100 and the number of seconds cannot exceed + or - 60. A zero number of days (or seconds) disables the adjustment. The program will ask:

Seconds of time adjustment:

In number of days:

This time drift factor is only important if the SAGE II is to be left running without reset or power down for a period of over two weeks. Also the factor is only important if an accurate time value is required from the SAGE II by an applications program.

V.8 BOOTSTRAP COPY UTILITY

The Bootstrap Copy Utility is selected from the outer level Utility prompt line with the character 'B'. This utility provides a similar function to the p-System routine BOOTER.CODE except it is specifically for the SAGE II computer system. The routine will copy the bootstrap from blocks Ø & 1 of a device to another device or from a file to a device. The routine insures that the data is a SAGE bootstrap by checking the first four characters of the information for the characters 'BOOT'.

When copying from an assembled code file, the file need not (and must not) be Compressed. The routine accounts for the Version IV header information in the code file which preceeds the code in block 1 of the file.

The Bootstrap Copy Utility will ask:

```
Source file or device <just CR quits> ?
```

Once a file or device number(examples 4, #4, #4:) is specified the utility asks:

```
Ready to load bootstrap from file xxxxxxxx
or                               volume x
```

If the reply is 'Y' the utility will ask:

```
Destination file or device <just CR quits> ?
```

Note that a transfer of a bootstrap to a file is currently not supported.

Once the destination device number is specified the utility asks:

```
Ready to copy bootstrap to volume x?
```

If the reply is 'Y' the utility will perform the transfer.

V.9 FLOPPY DISKETTE FORMATTER

The Floppy Diskette Formatter is selected from the outer level Utility prompt line with the character 'F'. This function is necessary to initialize diskettes with address information so that the hardware floppy controller can determine where to read and write the user's data. Caution: Formatting a diskette will destroy any previous data recorded on the diskette. The Formatter Utility will ask:

Drive to be formatted (4 or 5) ?

Once answered the utility will reply with the current configuration set up in the BIOS for that drive. The formatter will only format diskettes according to the current floppy configuration. If formatting for a different configuration is desired, the SAGEUTIL C(onfiguration option must be used to alter the floppy assignment. The format routine will ask:

Is diskette ready for formatting in drive x?

The user should make sure that the wrong drive (typically containing the system diskette) has not been selected, and that the diskette to be formatted has been placed in the drive before answering 'Y'. The utility will reply with 'Format Started' and print a dot for each track as it is formatted. The 48 TPI diskettes will print two lines of 40 dots (each cylinder has two tracks, one on each side of the diskette). The 96 TPI diskettes will print four lines of 40 dots. After the formatting information has been written the utility displays 'Verification' and reads back all the tracks, again printing a dot for each track processed.

After successfully formatting a diskette the utility will display:

Format Complete

More diskettes to format?

If the user answers 'Y' the utility resumes by asking again if the diskette is reading for formatting.

VI SAGE II TOOL KIT:

The UNIT facility of the UCSD 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 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 protocalls. 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 Users Manual under the title Segments, Units, and Linking. A portion of that information is presented here to explain various options on UNIT file placement in order to run some of the SAGE provided utilities. The p-System program LIBRARY.CODE is used to move UNITs from one code file to another. Use of this utility program is also covered in p-System Users Manual.

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 is generally 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.

The main SAGE distribution diskette (SYSTEM:) contains most of the SAGE generated UNITs in a file called SAGETOOLS.CODE. Also the diskette contains a USERLIB.TEXT file near the beginning which indicates that the files *SYSTEM.LIBRARY and #4:SAGETOOLS.CODE are to be user library files. Thus if the system boots to RAM Disk, the USERLIB.TEXT file is copied to the system device but the SAGETOOLS.CODE file is not. If the diskette remains in the left drive, the SAGE Tool Kit files are available. If the system is typically booted from the right hand drive, the USERLIB.TEXT file should be modified accordingly. 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.

VI.1 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: (Ø 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.

Following is 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 Ø 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:=TRU!
                    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 alphabetic 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 1Ø DO
    BEGIN
      Temp:='';
      SIO_IntWt(Numbers[I],Temp);
      SIO_Fill(8-LENGTH(Temp),Result);
      Result:=CONCAT(Result,Temp);
    END;
  WRITELN(Result);
END;
```

VI.2 TIME AND DATE UTILITY 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 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 abreviate 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 protocall). 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 protocall). 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,Ø,Ø,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.3 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 relies on having available the Units SIO_Unit and TAD_Unit from the SAGE Tool Kit as well as the p-System Units Sys_info, Wild, and DirInfo. These Units must be available in the SYSTEM.LIBRARY or other user specified library.

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 SAGEUTIL. See the writeup on System Configuration under SAGEUTIL 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.

VII IEEE-488 SUPPORT:

Support for the IEEE-488 is provided 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 protocall may not be the same. It also gives the user the option of optimizing the code for his own application.

GENERAL BUS OPERATION

MAXIMUM DATA RATE	=	1 Mbyte per second. As all transfers use handshake, devices on the bus do not have to work at the same speed.
MAXIMUM DEVICES	=	15. Max of 1 TALKER and up to 14 listeners at one time. (They can all be SAGE IIs!) 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.
PROTOCALL	=	Byte-serial, bit parallel, asynchronous data transfer using interlocking three-wire handshake technique.
SAGE II HARDWARE	=	TMS9914 controller, open collector or tri-state bus. Tri-state gives the faster speed if the devices support it.

VII.1 IB_UNIT:

The Pascal unit IB.UNIT and the assembly code program IB_BUS are setup to provide a bus configuration where the SAGE II 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
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.2 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 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;
```

SET THE ERROR MESSAGE OPTION

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 Ø.

IEEE 488 SWITCH OPTIONS:

The SAGE II must be assigned a bus address from Ø-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 dipswitch. GROUP-B on the SAGE II is used for this purpose. The user does not HAVE to use the switches. The SAGE II 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 Listen & Talk address.
Sw6	Dat	used to disable TALKER function. set to Ø , enabling TALK.
Sw7	Dal	used to disable Listen function. set to Ø ,enabling LISTEN.
Sw8	edpa	used to assign 2 consecutive address to one device.

As the SAGE II is generally a controller, set Sw6-Sw8 OFF (dwn). Set Sw1-Sw5 to the address.

```
EXAMPLE: addr=7 ---> Sw8 Sw7 Sw6 Sw5 Sw4 Sw3 Sw2 Sw1
                     dwn dwn dwn dwn dwn  up  up  up
```

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 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.3 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 Ø.

Set ADDR = s the Talk & Listen address of the SAGE II. 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:

TMS9914 interrupts are shut off.
The 488 bus is cleared with an IFC.
The remote enable mode is ON.
The SAGE is optionally the IEEE-488 bus controller.
All devices are inhibited from talking or listening.
The BUSSTATUS array is zeroed.

VII.4 DEVICE DEFINITION:

Now the user program must define the devices on the bus and what they can do. This is done with **BUSSTAT:**

```
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 Ø-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

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.5 SAGE II AS A TALKER:

Sending data from the SAGE II 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.

TBUF is the address of the first byte that the SAGE II will send. See the following explanation on setting up the USER BUFFER.

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.

```
EXAMPLE:    TALKS('MD0;');    {Sets HP1615 to 24-bit mode}
                              {The ";" is the terminator}
```

TALK will set up the DEV device to listen, the SAGE II to talk and send the data. An error will be returned if not successful in **IB_ERR:**

4	timeout occured while sending byte - no handshake.
7	Listen dev is not capable of listening.

VII.6 SAGE II AS A LISTENER

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);
```

TDEV is set to the device which will talk.

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 8Ø chars can be received. HEARS is convenient for most character transfers, the user does not have to set up a buffer area.

EXAMPLE: HEARS(HPS);

The device DEV is set to talk, the SAGE II 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 came in.
RTMOUT	=	3	timeout occured while listening for byte.
NOTALK	=	6	Talk device is not capable of talking.

VII.7 USER BUFFER:

The TALK and HEAR routines do not setup 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 Ø..255. As byte parameters can not 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[Ø..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.8 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:=Ø;
  REPEAT
    SCOUNT:=SCOUNT+1;
  UNTIL (IB_CHKSRQ) OR (SCOUNT>1ØØ);
  IB_SPOLL(PDEV);
  IF PDEV<>Ø THEN
    BEGIN
      HEARS(PDEV,ACT);
    END;
{ACT now contains action requested by PDEV}
END;
```

VII.9 DIRECT REGISTER CONTROL:

The TMS9914 registers can be directly read or written using:

```
PROCEDURE IB_DIR(RDW,REG:INTEGER;VAR VAL:INTEGER);

RDW = Ø 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 MPØ33
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 1Ø DO BEGIN END;  {>1ØUS WAIT}
  SIC:=15;            {Remove interface clear=ØFH}
  IB_DIR(WT,AUX,SIC);
END;
```

VII.1Ø 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 analzyer saying that it has done that. Then the display of error messages is shut off and the SAGE II 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 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 Ø,22}
  IB_X:=Ø;
  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<>Ø 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;
```

THE IEEE-488 SUPPORT
PROGRAM EXAMPLE:
SECTION VII.10Ø

```
PROCEDURE GETDATA;
VAR BDATA:PACKED RECORD CASE INTEGER OF
      1:(B:PACKED ARRAY[1..MAX] OF Ø..255);
      2:(W:INTEGER);
      END;
    MORE:BOOLEAN;
    TOTAL,DLNG:INTEGER;

BEGIN
  MORE:=FALSE;
  DLNG:=MAX;              {MAX number of bytes for each call}
  TOTAL:=Ø;
  REPEAT
    IB_HEAR(HP1615,BDATA.W,DLNG,MORE);
    { SAVE DATA HERE}
    MORE:=TRUE;
    TOTAL:=TOTAL+DLNG;
  UNTIL IB_ERR<>2;
  IF IB_ERR<>Ø 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:=Ø;
  REPEAT
    SCOUNT:=SCOUNT+1;
  UNTIL (IB_CHKSRQ) OR (SCOUNT>1ØØ);
  IB_SPOLL(PDEV);
  IF PDEV<>HP1615 THEN WRITELN('other SRQ') ELSE
     WRITELN('FOUND SRQ');
END;

BEGIN
  DOINIT;
  IB_TALKS(HP1615,'RU;');
  IF IB_ERR<>Ø THEN EXIT(IB_EX);
  IB_TALKS(HP1615,'DS');
  IF IB_ERR<>Ø THEN EXIT(IB_EX);
  ACTION;
  IB_CHK:=FALSE;
  {No errors displayed while getting data}
  GETDATA;
END.
```

VII.11 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

BUILDING THE USER PROGRAM:

The preceding example program can be compiled and run as follows to show how to setup 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.

Xecute user program IB.EX
Execution option L= may also be used. Refer to the UCSD Pascal Users' Manual section ALTERNATE PREFIXES AND LIBRARIES. The code files can be linked together using the LIBRARIAN. Refer to the section in the UCSD user's manual on SEGMENTS AND UNITS.

VII.12 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 occured.

```
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,Ø,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,Ø,IHEAR);
```

IHEAR terminates when:

```
   return=Ø 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 a byte array of LISTEN device(s). Note that 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-existant 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 Ø = 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 setup with the first byte= the register# and the next byte= the value to read or write to it.
RDW =Ø 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=Ø 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,Ø,CHKSRQ);
```

It returns HAVESRQ (=1Ø) 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 (=1Ø) is returned. Then the user must initiate a serial poll. Call:

```
IB_BUS(B,DUMMY,Ø,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,Ø,PPOLL);
```

The device that generated the SRQ will be returned in **B.**

CD=1Ø.INIT

Initialize TMS9914. If SAGE II is the controller then the bus is initialized also. Call:

```
IB_BUS(CNTRL,ADDR,CMDWAIT,INIT);
```

CNTRL = Ø if another device is the controller.
1 if the SAGE II is the controller.

ADDR = s the address of the SAGE II. Usually the value read from the switches. It can be set without reading the switch. All 8 bits as defined by the address register specification must be set.

CMDWAIT= x the count in increments of 2us to wait for devices on the bus to process commands.

CD>1Ø.ERREXIT

error return, bad argument (CD)

VII.13 ERROR CODES:

ZERO	=	Ø	no error, all done.
BADARG	=	1	bad call to GPIB
FULL	=	2	LNG bytes received but no EOI came in.
RTMOUT	=	3	timeout occured while listening for byte.
XTMOUT	=	4	timeout occured while sending byte.
NOEOI	=	5	had 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	=	1Ø	Have found an SRQ.

VIII COMPUTER SYSTEM INTERCOMMUNICATION

It is expected that a large number of SAGE II 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 computer.

The programs REMOUTTEST and REMINTEST are routines which are intended to check out an RS232 remote channel between two p-System computers. The routines SEND and RECEIVE are simple programs to communicate the image of a device between p-System computers. The program 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. The program REMTALK is provided by Softech Microsystems (on an as-is basis) and provides individual file transfer capabilities between two p-System environments.

The typical scenerio is to use REMOUTTEST and REMINTEST (or equivalent methods of sending and receiving characters) to check out the remote RS232 channel connection. If the foreign (non SAGE) system supports the p-System, then REMTALK may already be available and can be used to transfer files. The routines SEND and RECEIVE are short and may be easily typed into a foreign p-System computer. If information is only to be transfered to the SAGE system then only SEND (a one page program) is required on the foreign system. If files must be transfered 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.1 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.2 CHANNEL CONFIGURATION

Many variations of low level data format are available on the remote channel. The utility program SAGEUTIL 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. Descrepancies 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.3 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 insure 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, a reduced baud rate may be necessary to allow the receiving computer enough time to process each character. The SAGE II 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 protocall such as XON/XOFF or an RS232 signal line to prevent the transmitter from overflowing the receiver. The SAGE II remote channel supports an XON/XOFF protocall in both transmit and receive directions (see remote channel configuration section). The SAGE II 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 protocall.

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 protocall 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.4 Using 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.5 Using 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 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.6 Using 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 UCSD 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 transfered, 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 boundries to prevent lines from crossing a double block boundry. Lines which exceed 255 characters without a carriage return will have a carriage return inserted and generate a message 'Line > 255 characters'.

VIII.7 Using 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.

IX RESIDENT SYSTEM SOFTWARE:

An 8k byte boot ROM, expandable to 64K bytes, provides initialization information. The boot contains simplex non-interrupt drivers for self-test of memory, processor and devices.The programs reside in ROM at locations FE0000 to FE1FFF, although FE2000 TO FEFFFF are reserved for BOOT expansion.

On power-up or when the processor is RESET, the starting address of the BOOT ROMs changes from FE0000 to 000000. The processor reads the initial stack pointer and initial start vector from ROM locations 0 and 4 respectively. The start vector points to an address where the ROMs normally reside (> FE0000). When the address is executed, the hardware switches the ROMs back to their normal address location. Program execution begins. The ROMs remain at their normal location in address block FE0000 to FE1FFF until a power-down or RESET.

IX.1 POWER-UP PROGRAM

On power up, the processor diagnostic is run to check the integrity of the CPU. Registers are set and read and a selected instruction set is run. If the test FAILS, the processor will stop and the CPU light on the front panel will be RED.

The ROM test is run next. It does a simple checksum of the ROM area to insure that the BOOTSTRAP program itself is ok. On an error, the CPU will halt and the CPU light will be RED. Because this error and a CPU error are catastrophic, no further information can be given.

Next, the terminal baud rate is determined by reading GROUP-A DIP switches on the rear panel. Communications always uses 8 data bits, 1 stop bit and even parity.

SW3	**SW2**	**SW1**	**Terminal baud rate**
dwn	dwn	dwn	19.2K baud
dwn	dwn	up	9600
dwn	up	dwn	4800
dwn	up	up	2400
up	dwn	dwn	1200
up	dwn	up	600
up	up	dwn	300
up	up	up	reserved, will default to 19.2K baud.

SW4	**Parity control**
dwn	even parity enabled
up	parity disabled

On startup, the remote serial channel defaults to 9600 baud with 8 data bits, 1 stop bit, and even parity. Use the PS command under the Debugger to change rate for stand alone applications.

The floppy drive option switch is read to determine which drive is installed (always double-density, double-sided format).

SW7	**FLOPPY CONFIGURATION**
up	40 track (48 TPI) drive
dwn	80 track (96 TPI) drive

At this point, the display should read:

" Sage II Startup Test"

IX.2 RAM MEMORY TEST:

Now a test of RAM is done. The first 128k of RAM is checked in the following manner:

Each long word (4 bytes) is set to 00000000 and read back.

The long word is set to FFFFFFFF and read back.

Then it is set to its own address and program goes on to next long word. When all 128K is done, each long word is read to see if it still contains its address.

Then the top word of each 128K bank is read to see if that bank exists. Once the size of the additional memory is determined, it is checked just as the first 128K was.

This test will take a few seconds. Then the message

"RAM SIZE = XXXXXX" will be displayed.

If a bad memory location was found then an error is displayed:

BAD memory @ (addr) is (8 digit value) instead of (8 digit value)

The program stops at the first bad location it finds. Because it re-reads the location to print out the error message, the error value may be the expected value if the RAM is intermittant and reads correctly the second time. The processor will attempt to enter the debugger after a memory error. If the failed memory occurs in the debugger stack area (working down from 400H), the debugger may fail to operate correctly after the memory error.

Next, the bootstrap switches are read to determine what device/program to boot to:

SW6	**SW5**	**BOOT DEVICE**
dwn	dwn	boot to DEBUGGER
dwn	up	boot to Floppy drive 0.
up	dwn	reserved, defaults to DEBUGGER
up	up	reserved, defaults to DEBUGGER

IX.3 BOOTING TO THE FLOPPY DRIVE:

If switches are set to boot from a floppy the screen will display:

"Booting from Floppy"

Then the first 1K of the BOOT diskette (first 2 sectors of 512 bytes each) are read into memory at location 400 hex. The first 4 bytes of this area should be the ASCII string 'BOOT' = 42 4F 4F 54 hex. If this information is not there then an error message is displayed:

"Not BOOT disk"

If a timeout occurs while trying to access the diskette, the program assumes there is no diskette there and outputs:

"Put in BOOT disk and press a key"

Typing a "Q" will display

"Boot aborted on drive 0"

and control will go to the DEBUGGER. Typing any other key will cause a re-try to boot from the Floppy. If a floppy disk error occurs, one of these error messages will be given:

"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

Note: The ROM floppy driver is set for 8 sectors/track, 512 bytes/sector operation. There is no track-to-track skew and no interleaving.

IX.4 PROM ROUTINE ENTRY POINTS:

The SAGE II Computer PROM contains a set of polled I/O routines for the terminal and the floppy disk drives as well as a debugger utility routine. Following is a list of fixed entry points which allow bootstraps and other stand alone (without BIOS) routines access to I/O facilities.

The PROM is located at FE0000 to FE1FFF in the 68000 address space. Routines in PROM may be called using MACRO assembly procedures to create the long address given. The macro is necessary as the assembler only generates addresses with the short direct addressing mode. The listed offset for each routine should be used as the macro argument. The offset + FE0000H is the actual address of the routine.

The routines must be called from the 68000 **SUPERVISOR** mode, not user mode.

THE LONG JSR MACRO:

```
        .MACRO  LJSR
        .WORD   4EB9H
        .WORD   00FEH
        .WORD   %1
        .ENDM
```

THE LONG JMP MACRO:

```
        .MACRO  LJMP
        .WORD   4EF9H
        .WORD   00FEH
        .WORD   %1
        .ENDM
```

1. **KEYBCH** -Get a Keyboard Character. Offset= 8H

This routine waits for and returns a character from the terminal port. Bit 7 is always cleared and lower case alphabetic characters are converted to upper case. The character is returned as a byte in register D0.

On entry: Use LONG JSR MACRO: LJSR KEYBCH

On exit: D0 = typed character
Registers used: D0

2. **KEYCHK** -Check for a Keyboard Character. Offset=ØCH

This routine tests the terminal USART to determine if a character is available for input.

On entry: Use LONG JSR MACRO: LJSR KEYCHK

On exit: condition code NE if char is available
" " EQ " " not available

Registers used: none

3. **TERMCHAR** -Printout a Character to Terminal. Offset=14H

This routine outputs the character from the low byte of DØ to the terminal port.

On entry: Use LONG JSR MACRO: LJSR TERMCHAR
DØ = character for output

On exit: Registers used: none

4. **TERMTEXT** -Printout a Text String. Offset=18H

This routine outputs a string of characters to the terminal. Register AØ is the address of the base of the string and the string must be terminated with a zero byte.

On entry: Use LONG JSR MACRO: LJSR TERMTEXT
AØ = pointer to first character of string

On exit: Registers used: AØ
(AØ advanced to byte beyond zero terminator)

5. **TERMCRLF** -Print a Carriage Return/Line Feed. Offset=1CH

This routine outputs a carriage return and line feed to the terminal. Also five nulls are output after the characters for terminals which need extra time after a vertical positioning change.

On entry: Use LONG JSR MACRO: LJSR TERMCRLF

On exit: Registers used: none

6. **TERMHEXB** Printout a Hexadecimal Byte. Offset=20H

This routine outputs a two character hexadecimal value contained in the low byte of register DØ.

On entry: Use LONG JSR MACRO: LJSR TERMHEXB
DØ = byte value to be output

On exit: Registers used: none

7. **TERMHEXW** -Printout a Hexadecimal Word. Offset=24H

This routine outputs a four character hexadecimal value contained in the low word of register DØ.

On entry: Use LONG JSR MACRO: LJSR TERMHEXW
DØ = word value to be output

On exit: Registers used: none

8. **FDREAD** -Floppy Disk Read. Offset=28H

This routine reads data from a floppy diskette and stores it in memory. The parameters defining the read are passed on the stack. The typical calling sequence is:

```
MOVE.W    BLOCKNUM,-(SP)   ;Logical block number (2 bytes)
MOVE.L    MEMADDR, -(SP)   ;Memory buffer address (4 bytes)
MOVE.L    NUMBYTES,-(SP)   ;Number of bytes (4 bytes)
MOVE.W    DRIVENUM,-(SP)   ;Drive number Ø or 1 (2 bytes)
LJSR      FDREAD
```

On entry:
Use LONG JSR MACRO: LJSR FDREAD
Stack (from top - last in)

4 byte return address (stored by LJSR)
2 byte drive number (Ø - left drive, 1 - right drive)
4 byte size in bytes
4 byte memory address
2 byte logical block number (each block = 512 bytes)

On exit:
DØ = error type (Ø = no error)
condition code NE if transfer failed.
Registers used: DØ, D1, D2, D3, D4, D5, D7, A1, A4

9. **FDWRITE** -Floppy Disk write. Offset=2CH

This routine writes data from memory to a floppy diskette. The parameters defining the write are passed on the stack (see example in Floppy Disk Read above).

On entry:
Use LONG JSR MACRO: LJSR FDWRITE
Stack (from top - last in)

4 byte return address (stored by LJSR)
2 byte drive number (Ø - left drive, 1 - right drive)
4 byte size in bytes
4 byte memory address
2 byte logical block number (each block = 512 bytes)

On exit:
DØ = error type (Ø = no error)
condition code NE if transfer
Registers used: DØ, Dl, D2, D3, D4, D5, D7, Al, A4

10. **DEBUG** -Debugger Entry Point. Offset=3ØH

This is a non returning entry point to the PROM Debugger for use when terminating a user environment or a failure during a bootstrap.

On entry: Use LONG JUMP MACRO: LJMP to Debug
On exit: Never returns!

IX.5 EXCEPTION ERRORS:

When processing an exception error, interrupts are turned off and the BIOS is disabled, unless the user has re-directed the error to his own error handling routine. Non user-intercepted errors have this format:

EXCEPTION: <error type> 'Error at' <8 digit location>

Error types are defined:

Bus Error: The processor tried to read memory and there was no response. Memory may not exist. A hardware strapping option determines what memory is equipped. Additional information is displayed:

"Function:<4 digit word> Access:<8 digit addr> Instr:<4 digit>"

```
Function: Bits Ø-2 ..are the state of the processor
                    function code outputs FCØ,FCl and FC2
          Bit    3 ..is Ø for an instruction, l for not
                    an instruction.
          Bit    4 ..is Ø for write, l for read.

Access    is the address of the attempt.
Instr     is the instruction being exectuted
```

Address error: The processor attempted to access a word or long word on an odd address. Additional information is displayed with this error:

"Function:<4 digit word> Access:<8 digit addr> Instr:<4 digit>"

```
Function: Bits Ø-2 ..are the state of the processor
                    function code outputs FCØ,FCl and FC2
          Bit    3 ..is Ø for an instruction, l for not
                    an instruction.
          Bit    4 ..is Ø for write, l for read.

Access    is the address of the attempt.
Instr     is the instruction being exectuted
```

Illegal Instruction error: There are 2 unused opcodes (Axxx & Fxxx) in the 68000 which are currently undefined and will give this error if an attempt is made to use them. Also, any undefined instruction format or addressing mode will cause this error.

Arithmetic error: An attempt was made to divide by zero or a CHK instruction was executed (user needs to define vector) or a TRAPV instruction was executed (user needs to define vector).

Privilege error: User tried an instruction which requires SUPERVISOR mode.

Reserved TRAP Certain TRAP locations have been reserved by Motorola for future use and should not be used.(This error should never occur.)

Unassigned TRAP error: There are 16 trap locations in the 68000, 0-14 of which are normally unassigned by the Debugger. Trap 15 is used for breakpoints by the Debugger. Traps 8 to 14 are used by the BIOS.

Unassigned Interrupt error: There are 6 auto-interrupt vectors. Normally all of them are unassigned by the debugger.

RAM Parity error: The 7th auto-interrupt vector is non-maskable and is used for RAM parity error reporting. Remember when troubleshooting that the Parity chip itself could be the cause of this error. Note that the location given is where the program was executing and is not necessarity the location with the error.

Unknown error: Either the program entered the TRAP handler illegally or the supervisor stack was not set to point to valid ram.

IX.6 SAGE DEBUGGING TOOL (SDT):

SDT is a powerful tool for analyzing program operation. The DEBUGGER can be entered several ways:

1) on power-up or reset. GROUP-A switches SW5 and SW6 off.

2) errors on power-up.

3) Failure to boot to Floppy or other system device.

4) On all EXCEPTION errors that were not user re-defined. This includes parity errors.

5) Via a breakpoint. This includes a breakpoint not set up by the DEBUGGER itself.

6) Via jump vector in PROM.

7) via BIOS which uses jump vectors.

When the DEBUGGER implements a break-point it replaces the instruction at the specified location with TRAP #15. When this trap occurs and control returns to the DEBUGGER, it restores the instuction so that the location can be properly displayed.

Anytime the instruction which has been replaced by the break-point must be executed, the debugger uses the trace bit in the processor to cause a TRACE trap after the instruction is done. This returns control to the DEBUGGER so that it can restore the break-point.

Once an interrupt-driven BIOS has been loaded, the DEBUGGER does not have to use the polled I/O drivers but can use the BIOS drivers. This resolves conflicts that would occur if both types of drivers were in use on the same device at the same time. To have the Debugger use the BIOS drivers, use TRAP #14, the long call sequence to BIOS with function=3. See the section on BIOS. This must be done BEFORE entering the debugger.

COMMAND STRUCTURE

The general structure of the arguments takes several forms:

Address data appears as

```
   $x+xxxxx      - base register + an offset.
                   EXAMPLE: $1 + 125
   $x            - base register with offset=Ø.
                   EXAMPLE: $2
   xxxxxx        - absolute memory address
   EXAMPLE: 14A7
```

The count appears as

```
   #xxxxxx       - The count replaces the ending address
                   with a loop count. The command starts at the
                   starting address and repeats its function
                   the specified number of times.

   EXAMPLE:
      DM   12Ø,12F dumps 16 bytes of memory
                   from location 12Ø to 12F
                   by specifing an end address.
      DM   12Ø,#1Ø dumps 16 bytes of memory
                   from location 12Ø to 12F
                   by specifing a count.
```

The data argument has two forms:

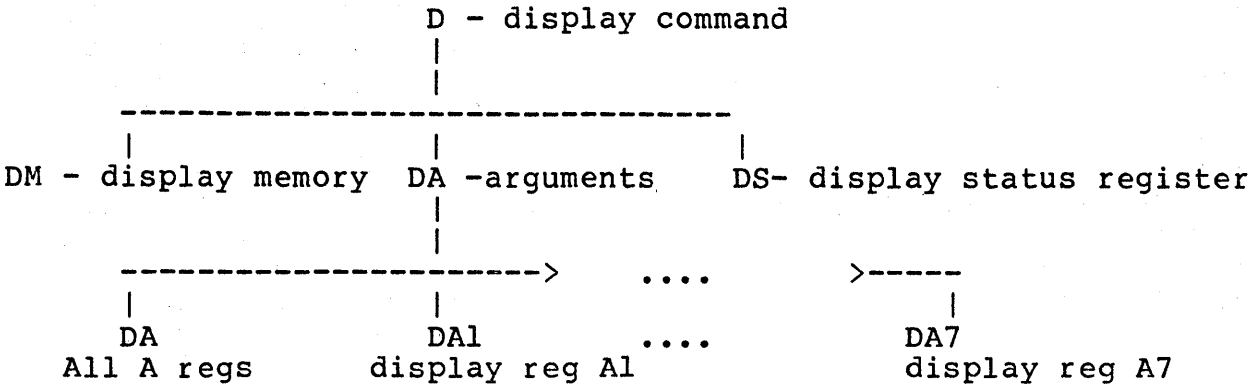
```
    'x'           - Quotes specify an ASCII string.
    xx            - Specifies a Hex number.

    EXAMPLE: assume $1 =1ØØØ (contents of base register 1=1ØØØ):

    FB $1+12Ø,$1+12F  'A'   -Fills locations 112Ø to 112F
                             with the character 'A'.
    FB $1+12Ø,#1Ø     41    -Also fills locations 112Ø to 112F
                             with the character 'A' (=41H) by
                             repeating 1ØH times.
```

The commands are structured by character. The first character defines the command group. The remaining characters divide the command into more subgroups.

For example, D is the display command group.



Command lines are formatted as follows:

```
<command line> ::=<command> <argument>,<argument>,<argument>....
<command>      ::=<command group char>,<subgroupl>...<subgroupN>
<argument>     ::=<argument type> <digits>
<argument type>::=<null>  <$>  <#>
<digits>       ::=<Ø..F> ... <Ø..F>
```

The debugger prompts the user for command as follows:

`>` - If no standard base register is defined.

`$x>` - The base register is displayed before the prompt ">". This register is added to all address arguments not specifing a base register.

`T>` - Trace is enabled and no standard base register defined.

`T$x>` - Trace is enable and x is the standard base regiser.

IX.6.1 DEBUG COMMANDS - Quick description

>DA[x]	Display A registers or Ax
>DB[x]	Display breakpoint info
>DD[x]	Display D registers or Dx
>DM [$X+]xxxxxx,[#,$x+]xxxxxx	Display memory
>DP	Display program counter
>DR	Display all registers.
>DS	Display status register
>DU	Display user stack
>D$[x]	Display base regs or $x
>FB [$x+]xxxxxx, [#,$x+]xxxxxx, xx	Fill memory / byte data
>FL [$x+]xxxxxx, [#,$x+]xxxxxx, xxxxxx	Fill memory / long data
>FW [$x+]xxxxxx, [#,$x+]xxxxxx, xxxx	Fill memory / word data
>GC [[$X+]XXXXXX]	Execute program
>GO [[$X+]XXXXXX]	Execute prog / bkpt reset
>IFx	Boot floppy from drive x (Ø=left,1=right) drive
>IS	Initialize SYSTEM
>LA	Load from a remote device
>LF	Load from a floppy
>LT	Load from the terminal
>M[$X+]XXXXXX,[#,$X+]XXXXXX,[$X+]XXXXXX	Move data in memory
>POB [$X+]XXXXXX, DD	Output data byte to port
>POW [$X+]XXXXXX, DDDD	Output data word to port
>PIB [$X+]XXXXXX	Input data byte from port
>PIW [$X+]XXXXXX	Input data word from port
>PS x	Set remote baud rate
>SA[x] [XXXXXXXX]	Modify A registers or Ax
>SB[x] [XXXXXXXX]	Set breakpoint regs or Bx
>SD[x] [XXXXXXXX]	Modify D registers or Dx
>SM [$X+]XXXXXX	Modify memory
>SP [XXXXXXXX]	Modify Program counter
>SR	Modify all registers
>SS [XXXXXXXX]	Modify Status Register
>SU [XXXX]	Modify User Stack
>S$[x] [XXXXXXXX]	Modify base regs or $x
>TB [[$X+]XXXXXX]	Trace without reg print
>TE	Terminates trace mode
>TR [[$x+]XXXXXX]	Trace with register print
>WF	Write to floppy
>	<CR> trace next inst
>$x	Sets standard base reg
>$	Clears standard base reg

Debugger usage notes:

All substitute commands except substitute breakpoints (SBx) have the following input formats:

```
          <CR>      - skip update
          .<CR>     - terminate substitute (SM only)
          xxxx<CR>  - data to be put in location
        'xxxx'<CR>  - character string to be put in location.
```

The substitute breakpoint (SBx) input format is as follows:

```
          <CR>      - skip update
          .<CR>     - inactivates breakpoint
   [$X+]XXXXXX<CR>  - data to be put in location
```

Data can be entered only into single registers from the command line:

```
        >SA5 10000 - Puts 10000 into A5
        >SA 10000  - Input for all registers will be requested
                     The 10000 will be ignored
```

IX.6.2 DEBUGGER COMMANDS: detailed description

This section describes the debugger commands in detail, giving examples.

Assumptions used in examples:

1) $1 = 1000
2) $2 = 2000
3) $3 = 10000
4) Comments are inclosed in parenthesis: (comment).
5) A <CR> appearing for an entry implies a blank line with only a carriage return entered.
6) All non-blank lines are terminated with a carriage return although no <CR> is shown.

<D> - Display command - Displays registers and memory

<DM> - Displays memory. The command takes four forms.

1) **DM $X+XXXXXX,$X+XXXXXX** - displays memory from the starting address (arg 1) to the ending address (arg 2):

```
>DM $1+10,$1+20

 00001010 00000010: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
```

2) **DM $X+XXXXXX,#XXXXXX** - displays memory from the starting address (argument 1) for a specified number of bytes (arg 2)

```
>DM $1+10,#10

 00001010 00000010: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
```

4) **DM $X+XXXXXX** - displays 256 bytes of memory from the starting address (argument 1):

```
>DM $1+10

 00001010 00000010: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001020 00000020: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001030 00000030: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001040 00000040: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001050 00000050: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001060 00000060: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001070 00000070: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001080 00000080: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001090 00000090: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010A0 000000A0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010B0 000000B0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010C0 000000C0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010D0 000000D0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010E0 000000E0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010F0 000000F0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00002000 00000100: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO

                  `—address relative to base register
        `—actual memory address
```

4) **DM** - displays 256 bytes of memory starting from the last memory location displayed:

(assume last location displayed was 0000100F)

```
>DM

 00001010 00000010: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001020 00000020: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001030 00000030: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001040 00000040: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001050 00000050: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001060 00000060: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001070 00000070: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001080 00000080: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00001090 00000090: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010A0 000000A0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010B0 000000B0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010C0 000000C0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010D0 000000D0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010E0 000000E0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 000010F0 000000F0: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO
 00002000 00000100: 4142 4344 4546 4748 494A 4B4C 4D4E 4F00 ABCDEFGHIJKLMNO

          `--address relative to base register
    `--actual memory address
```

<DR> - Displays all registers.

Register A7 represents the supervisor stack.

```
>DR
D0:0000005A 0001000A 00000000 0000100A 00010234 00011201 00001234 12356700
A0:00003445 12347777 00003344 00001111 23400011 12399990 34436748 12347889
PC:00001000  US: 0007FFF0  SR: 2704
(Prg counter)  (User stack)  (Status register)
```

<DD> - displays the data registers (DØ - D7).

This command comes in two forms

1) **DD** displays all data registers DØ through D7:

```
>DD
    D0:0000005A 0001000A 00000000 0000100A 00010234 00011201 00001234 12356700
```

2) **DDX** displays register specified by the X

```
>DD2
D2: 00000000

>DD5
D5: 00011201
```

<DA> - displays the data registers (AØ - A7).

This command comes in two forms:

1) **DA** displays all data registers AØ through A7:

```
>DA
    A0:00003445 12347777 00003344 00001111 23400011 12399990 34436748 12347889
    (A7=supervisor  stack pointer)
```

2) **DAX** displays register specified by the X

```
    >DA2
    A2: 00003344

    >DA5
    A5: 12399990
```

<DS> - displays the status register

```
    >DS
    SR: 2704                    (Status register)
```

<DP> - displays the program counter

```
    >DP
    PC: 00001000                (Program counter)
```

<DU> - displays the program user stack pointer

```
>DU
US: 0007FFF0                (User stack pointer)
```

<D$> - displays the three base registers.

It has two forms.

1) **D$** - displays the three base registers

```
>D$
$1: 00001000 00002000 00010000
```

2) **D$X** - displays base register x

```
>D$1
$1: 00001000
```

<DB> - displays the three breakpoint registers.

1) **DB** - displays the three breakpoint registers.

```
>DB
Breakpoint 2: 00005000 (0000,0000)
Breakpoint 1: Inactive      (breakpoint register not in use)
Breakpoint 0: 00004500 (0010,0003)
                  ^        ^    ^
                  |        |    |- number of passes thru
                  |        |       breakpoint
                  |        |- max passes thru breakpoint
                  |           before break
                  |- breakpoint address
```

2) **DBX** - display single breakpoint register.

```
>DB1
Breakpoint 1:Inactive
```

<S> - SUBSTITUTE COMMAND - ALTER USER REGISTERS AND MEMORY

<SM> Alters memory. When the new value is requested a carriage return specifies the data is not to be modified. Data typed in replaces the original contents of the memory or registered being modified.

SM [$X+]XXXXXX - alters memory starting at the given address (argument 1). The data entered can be either a hexadecimal constant (between 0-FFFF) or an upper case character string enclosed in quotes. Quotes cannot be entered as a character.

```
>SM $1+10
 00001010 0000010: 4142  AB: 4748
 00001012 0000012: 4344  CD: 'TU'
 00001014 0000014: 4546  EF: .        (period terminates)


 (if the memory is dumped it appears as follows:)

 >DM 1010,#6

 000001010 00001010: 4748 5455 4546  GHTUEF
```

<SR> - alters all user registers.

```
>SR
D0: 0000005A: <CR>          (register is unchanged)
D1: 0001000A: <CR>          (register is unchanged)
D2: 00000000: <CR>          (register is unchanged)
D3: 0000100A: <CR>          (register is unchanged)
D4: 00010234: 00001234      (alters D4 to 00001234)
D5: 00011201: <CR>          (register is unchanged)
D6: 00001234: <CR>          (register is unchanged)
D7: 12356700: <CR>          (register is unchanged)
A0: 00003445: <CR>          (register is unchanged)
A1: 12347777: <CR>          (register is unchanged)
A2: 00003344: <CR>          (register is unchanged)
A3: 00001111: 0             (alters A3 to 00000000)
A4: 23400011: <CR>          (register is unchanged)
A5: 12399990: <CR>          (register is unchanged)
A6: 34436748: <CR>          (register is unchanged)
A7: 12347889: <CR>          (register is unchanged)
                            (system stack)
PC: 00001000: <CR>          (register is unchanged)
                            (Program counter)
US: 0007FFF0: <CR>          (register is unchanged)
                            (User stack)
SR: 2704: <CR>              (register is unchanged)
                            (Status register)
```

<SD> - alters the data registers (DØ - D7).

This command comes in two forms:

1) **SD** alters all data registers DØ through D7:

```
>SD

DØ: 0000005A: <CR>          (register is unchanged)
D1: 0001000A: <CR>          (register is unchanged)
D2: 00000000: <CR>          (register is unchanged)
D3: 0000100A: <CR>          (register is unchanged)
D4: 00010234: 00001234      (alters D4 to 00001234)
D5: 00011201: <CR>          (register is unchanged)
D6: 00001234: <CR>          (register is unchanged)
D7: 12356700: <CR>          (register is unchanged)
```

2) **SDX** alters register specified by the X

```
>SD2
D2: 00000000: <CR>          (register is unchanged)


>SD4
D4: 00010234: 00001234      (alters D4 to 00001234)


>SD5  12345678              (alters D5 to 12345678)
```

<SA> - alters the data registers (A0 - A7). This

command comes in two forms:

1) **SA** displays all data registers A0 through A7:

```
>SA

A0: 00003445: <CR>          (register is unchanged)
A1: 12347777: <CR>          (register is unchanged)
A2: 00003344: <CR>          (register is unchanged)
A3: 00001111: 0             (alters A3 to 00000000)
A4: 23400011: <CR>          (register is unchanged)
A5: 12399990: <CR>          (register is unchanged)
A6: 34436748: <CR>          (register is unchanged)
A7: 12347889: <CR>          (register is unchanged)
                                (system stack)
```

2) **SAX** alters register specified by the X

```
>SA2
A2: 00003344: <CR>          (register is unchanged)


>SA3
A3: 00001111: 0             (alters A3 to 00000000)


>SA4 140                    (alters A4 to 00000140)
```

<SS> - alters the status register

```
>SS
SR: 2704: <CR>              (register is unchanged)


>SS 2700                    (alters register to 2700)
```

<SP> - alters the program counter

```
>SP
PC: 00001000: <CR>          (register is unchanged)


>SP 5000                    (alters register to 5000)
```

<SU> - alters the program user stack

```
>SU
US: 0007FFF0: <CR>          (register is unchanged)

>SU 4000                    (alters register to 4000)
```

<S$> - alters the three base registers.

It has two forms.

1) **S$** - alters all three base registers

```
>S$
$1: 00001000: $1+700        (alters $1 to 1700)
$2: 00002000: <CR>          (no modification made)
$3: 00010000: 1500          (alters $3 to 1500)
```

2) **S$X** - alters base register x

```
>S$1
$1: 00001000: $1+700        (alters $1 to 1700)

>S$1 1500                   (alters $1 to 1500)
```

<SB> - alters the three breakpoint registers.

```
>SB
Breakpoint 2: 00005000: $1+900,10   (alters BP 2 to 1900)
                                    (max pass count to 10)
Breakpoint 1: Inactive; <CR>        (does not alter BP 1)
Breakpoint 0: 00004500: .           (inactivates BP 0)

>SB1
Breakpoint 1: Inactive: <CR>        (does not alter BP 1)

>SB0 .                              (Removes BP0)
```

FILL COMMAND - FILLS MEMORY WITH A CONSTANT

The Fill command fills memory with a given constant. The constant can be either a byte (8 bits), word (16 bits), or long word (32 bits).

<FB> - fill memory byte by byte with a constant

1) **FB [$X+]XXXXXX,[$X]XXXXXX,DD** - Fills memory starting at the starting address (argumentl) through the specified ending address with data specified in (argument 3).

```
>FB $1+20,$1+2F,0   (Fills memory from 1020-102F with zeroes)

>FB 1020,102F,' '   (Fills memory from 1020-102F with spaces)
```

2) **FB [$X+]XXXXXX,#XXXXXX,DD** - Fills memory starting at the starting address (argument 1) for (argument 2) number of bytes with the data specified in (argument 3).

```
>FB $1+20,#10,0     (Fills memory from 1020 - 102F with zeroes)

>FB 1020,#10,' '    (Fills memory from 1020 - 102F with spaces)
```

<FW> - fill memory word by word with a constant

1) **FW [$X+]XXXXXX,[$X]XXXXXX,DDDD** - Fills memory starting at the starting address (argumentl) through the specified ending address with data specified in (argument 3).

```
>FW $1+20,$1+2F,0   (Fills memory from 1020 - 102F with zeroes)

>FW 1020,102F,' A'  (Fills memory from 1020 - 102F with ' A')
```

2) **FW [$X+]XXXXXX,#XXXXXX,DDDD** - Fills memory starting at the starting address (argument 1) for (argument 2) number of bytes with the data specified in (argument 3).

```
>FW $1+20,#10,0     (Fills memory from 1020 - 102F with zeroes)

>FW 1020,#10,' A'   (Fills memory from 1020 - 102F with ' A')
```

<FL> - fill memory long word by long word with a constant

5) **FL [$X+]XXXXXX,[$X]XXXXXX,DDDDDD** - Fills memory starting at the starting address (argumentl) through the specified ending address with data specified in (argument 3).

```
>FL $1+20,$1+2F,0  (Fills memory from 1020-102F with zeroes)

>FL 1020,102F,' ABC' (Fills memory from 1020-102F with ' ABC')
```

6) **FL [$X+]XXXXXX,#XXXXXX,DDDDDD** - Fills memory starting at address (argument 1) for (argument 2) number of bytes with the data specified in (argument 3).

```
>FL $1+20,#10,0    (Fills memory from 1020-102F with zeroes)

>FL 1020,#10,' ABC' (Fills memory from 1020-102F with ' ABC')
```

PORT I/O COMMANDS - READS FROM AND WRITES TO PORTS.

Be careful when using these commands as they do not verify the existance of a port before it is accessed. The commands work on memory as well as ports. These commands allow a port to be read or written in one access.

Note that the substitute memory commands read from the address (twice) before writing which may be undesirable when accessing an I/O device, thus the need for special Port commands.

<PO> - outputs data to a port

1) **POB** - outputs a byte of data (argument 2) to port specified (argument 1).

```
>POB FFFFC021 'A'      (Outputs A to port FFFFC021)

>POB FFFFC022 10       (Outputs 10H to port FFFFC022)
```

2) **POW** - outputs a word of data (argument 2) to port specified (argument 1). The port address must be on a word boundary.

```
>POW FFFFC022 'AB'     (outputs 'AB' to port FFFFC022)

>POW FFFFC022 1007     (outputs 1007H to port FFFFC022)
```

<PI> - inputs data from a port and displays it

1) **PIB** - inputs a byte of data from a port (arg 1). and displays it.

```
>PIB FFFFC021          (inputs a data byte from port FFFFC021)

FFFC021: 21
```

2) **PIW** - inputs a word of data from a port (arg 1). The port address must be on a word boundary

```
>PIW FFFFC022          (inputs a word of data from  port FFFFC022)

FFC022: 33
```

<PS> - SETS BAUD RATE FOR REMOTE SERIAL CHANNEL

1) **PS x** - sets up th baud rate for the remote serial channel according to the following values for x:

x:	Rate:
0	reserved (currently same as 19200 baud)
1	300 baud
2	600 baud
3	1200 baud
4	2400 baud
5	4800 baud
6	9600 baud
7	19200 baud

On startup, the remote serial channel defaults to 9600 baud with 8 data bits, 1 stop bit, and even parity.

MOVE COMMAND - MOVES A BLOCK OF DATA IN MEMORY

The command moves a block of data from one memory address to another. The command assures that overlapping transfers will be handled correctly.

<M> - moves data in memory.

The command has two forms:

1) **M [$X+]XXXXXX,[$X]+XXXXXX,[$X+]XXXXXX** - moves data starting at (argument 1) through (argument 2) to the memory address starting at (argument 3).

```
>M $1+10,$1+20,$1+50  (Moves bytes from 1010 - 1020 to
                      1050 - 1060)

>M $1+10,$1+20,$1+11  (Moves 1010 - 1020 one byte forward in
                      memory)
```

2) **M [$X+]XXXXXX,#XXXXXX,[$X+]XXXXXX** - moves data starting at (argument 1) for (argument 2) number of bytes to (argument 3)

```
>M $1+10,#21,$1+50    (Moves bytes from 1010 - 1020 to
                      1050 - 1060)

>M $1+10,#21,$1+11    (Moves 1010 - 1020 one byte forward in
                      memory)
```

Motorola Object Code Format

The standard Motorola object code format may be used for loading programs and data with the Debugger Load commands into a SAGE II computer from the Terminal or Remote serial channels. This format consists of ASCII characters formed into records (typically printed on one line). Each record starts with the character 'S' and is followed by a record type number, a byte count, an address, the memory data, and a checksum.

record: Stccaaaaddddddddd...ddss
or Stccaaaaaaddddddddd...ddss

S the ASCII character 'S' which always starts a record.

t type of record (single digit):

- Ø - is the header record which generally contains only a program name in the data field. This record is ignored by the loader routine.
- 1 - indicates an object code record with a two byte address field, 'aaaa'.
- 2 - indicates an object code record with a three byte address field, 'aaaaaa'.
- 9 - is a termination record which indicates that the load is complete.

cc Hexadecimal byte count of the remaining characters in the record (address, data, and checksum).

aaaaaa or
aaaa is the hexadecimal memory address where the data which follows is to be loaded. This field is present for all records but ignored for the type Ø (header) and 9 (terminator) records. For type Ø, 1, or 9 records the address is contained in 4 hex characters, while for type 2 records the address is contained in 6 hex characters.

dd represents a two hex character value for each object code byte. Each record may contain up to 252 bytes of object code although 32 is typical in order to allow a paper listing.

ss is the sum of all the ASCII character bytes from the byte count to the end of the data.

Note that at the beginning of each record the loader will ignore all characters except 'Q' which will cause the loader to terminate and 'S' which starts the record. This allows a Carriage Return and Line Feed to terminate each line for printout.

Examples:

```
S00600004844521B
S10710801FFE4E728B
S20A010000323C00035641ED
S9030000FC
```

<L> - Load memory from external device

Load memory from either the auxillary (remote) port or from the terminal port. This command is used to download data or a program from another computer. The data is expected to be sent in Motorola format. The command has two forms:

<LA> - load memory from auxillary port

```
>LA                          (load data from AUX port)
```

<LT> - load memory from terminal port

```
>LT                          (load data from term port)
```

<I> - Initialize system.

Initialize the system.

>IS Initializes the system.

Memory is cleared and retested. All previous registers and breakpoints are lost.

<IF> - Boots UCSD Pascal from the floppy

To initialize the system from the DEBUGGER to a Floppy Diskette, type :

```
>IF          (boots from the left  floppy)
>IFØ         (boots from the left  floppy)
>IF1         (boots from the right floppy)
```

The Debugger reads 1K of data from logical blocks Ø and 1 of the diskette into memory starting at location 4ØØH. The debugger checks if the first four bytes are the ASCII characters 'BOOT' (42H, 4FH, 4FH, and 54H) to verify that the diskette has a bootstrap program installed. The bootstrap routine is then called at location 4Ø4H with the booting drive number (Ø or 1) previously placed on the stack.

If the bootstrap routine returns to the calling program, control reverts to the Debugger.

<$> - Sets the standard base register

The standard base register is added to any address input which does not have a base register specified. To keep the standard register from being added to an address, use $Ø (which is a base register permanently defined as zero). Any system errors are displayed as a physical address and as an offset from the standard register.

The current standard register is displayed with the command line prompt as follows:

```
$1>          ($1 is the current standard base register)
$2>          ($2 is the current standard base register)
$3>          ($3 is the current standard base register)
>            (no current standard base register)
```

The standard base register is set typing $X where X is the standard base register. There are three standard base registers.

```
>$1          (sets the standard base register to $1)

>$2          (sets the standard base register to $2)

>$3          (sets the standard base register to $3)
```

The standard base register is cleared by typing $

```
$3>          ($3 is the current standard base register)

>$           (removes the standard base register)

>            (no current standard base register)
```

<GO> - Go command

The Go command sets any breakpoints requested and starts execution of a program. The command has two forms:

1) **GO** - Start execution from current value of the Program Counter (PC). The breakpoint pass counts are zeroed.

```
>GO                 (start program at address in user PC)
```

2) **GO [$X+]XXXXXX** - Start execution at the specified address (argument 1)

```
>GO $1+5000         <start program at 6000)
```

3) **GC** - Start execution from current value of the Program Counter (PC). The breakpoint pass counts are not cleared

```
>GC                 (start program at address in user PC
```

4) **GC [$X]XXXXXX** - Start execution at the specified address. (argument 1). The pass counts are not cleared.

When a breakpoint (TRAP #15) is reached, the program returns to the debugger displaying the current address of the breakpoint. To restart the program just type GO or GC.

```
>GO                 (restart program from last breakpoint)
>GC                 (restart program from last breakpoint)
```

<T> - Trace command

The trace command will execute a program instruction. There are two forms of the trace command.

<TB> - start the trace.

This subcommand has two forms:

1) **TB** - Starts the trace at the current value in the PC.

```
>TB
```

2) **TB [$X+]XXXXXX** - starts tracing at a specified address.

After trace has been enabled, the command prompt has a T in front of it as follows:

```
>           (trace disabled)
T>          (trace enabled)
T$1>        (trace enabled, $1 standard base register)
$1>         (trace disabled, $1 standard base register)
```

<TR> - start the trace and print registers..

This subcommand has two forms:

1) **TR** - Starts the trace at the current value in the PC and prints registers after each instuction traced.

```
>TR
```

2) **TR [$X+]XXXXXX** - starts tracing at a specified address and prints registers after each instruction traced.

After trace has been enabled, the command prompt has a T in front of it as follows:

```
>           (trace disabled)
T>          (trace enabled)
T$1>        (trace enabled, $1 standard base register)
$1>         (trace disabled, $1 standard base register)
```

All trace information is output BEFORE execution of the displayed instruction. A <CR> executes this instruction and displays the information for the next instruction.

```
(trace next instruction from a TB start trace)
T><CR> Trace: (00005000 $0:00005000): 5280
       (display of next instruction to trace)

T><CR> Trace: (00005002 $0:00005002): 60FC
       (display of next instruction to trace)

(trace next instruction from a TR start trace)
    T><CR> Trace: (00005000 $0:00005000): 5280
    D0:0000005A 0001000A 00000000 0000100A 00010234 00011201 00001234 12356700
    A0:00003445 12347777 00003344 00001111 23400011 12399990 34436748 12347889
    PC:00005000  US: 0007FFF0  SR: 2704

    T><CR> Trace: (00005002 $0:00005002): 60FC
    D0:0000005B 0001000A 00000000 0000100A 00010234 00011201 00001234 12356700
    A0:00003445 12347777 00003344 00001111 23400011 12399990 34436748 12347889
    PC:00005000  US: 0007FFF0  SR: 2704
```

<TE> - Terminates trace mode.

```
T><CR>      (trace next instruction)
(00005000 $0:00005000)      (displays next instruction to trace)

T><CR>      (trace next instruction)
(00005002 $0:00005002)      (displays next instruction to trace)

T>TE                        (terminate trace mode)


>                           (no T implies trace mode disabled)
```

<LF> - Load from floppy

LFX XXXX,[$X+]XXXXXX,XXXXXX Reads floppy into memory.
arg1 =block addr of disk to get data from.
arg2 =memory addr for data.
arg3 =number of bytes to transfer.(HEX)

<WF> - Write to floppy

WFX XXXX,[$X+]XXXXXX,XXXXXX Write contents of memory to floppy.
arg1 =block addr of disk to write data to.
arg2 =memory addr of data to be transferred.
arg3 =number of bytes to transfer. (HEX)

BIOS INTERFACE
SHORT CALLING SEQUENCE
SECTION X

X SAGE II BIOS INTERFACE:

BIOS stands for **B**asic **I**nput/**O**utput **S**ystem. The SAGE BIOS is written in assembly language for the 68000 and is only directly accessable from assembly language routines. Many of the BIOS features are accessable from the p-System via the low level I/O interface routines UNITREAD, UNITWRITE, UNITSTATUS, and UNITCLEAR. This section is only necessary for users who wish to develope their own assembly language programs which deal with I/O.

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 accessable when in SUPERVISOR mode. Access to this area while in USER mode results in an 'EXCEPTION: Bus Error'.

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.

X.1 SHORT CALLING SEQUENCE:

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 DØ. All registers are preserved except for the returned data in DØ 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 DØ. 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,DØ           ;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 DØ. If there are no characters in the queue the routine waits until a character is typed before returning.

TRAP #1Ø. Write Character to Terminal

This TRAP outputs the character from the low byte of DØ 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 AØ.  The sequence is terminated by a
;       zero byte.  Routine modifies DØ and AØ.
TEXTOUT MOVE.B  (AØ)+,DØ
        BEQ.S   $5              ;Found terminator
        TRAP    #1Ø.            ;Output the character to BIOS
        BRA     TEXTOUT         ;Loop back for more characters
$5      RTS                     ;Done with text output


;       Sample usage
        LEA     MESSAGE,AØ
        BSR     TEXTOUT
        ------

MESSAGE .ASCII  "This is a sample message"
        .BYTE   Ø               ;Terminator
```

TRAP #11. Write Character to Printer

This TRAP outputs the character from the low byte of DØ 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. If the printer is not configured in the BIOS the condition code is returned EQ, otherwise the condition code is returned NE.

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 DØ. 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 DØ 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.

X.2 LONG CALLING SEQUENCE:

The long calling sequence uses the TRAP #14 instruction for entry into the BIOS. For this sequence, the low word of DØ is a function code. AØ points to an argument block for those functions where additional information is required. Functions which perform I/O to a channel will have the channel number (from the table below) in the first word of the argument block.

CHANNELS	
1 - Keyboard (for input)	7 - Remote Serial Input Channel
2 - Terminal (for output)	8 - Remote Serial Output Channel
3 - reserved (unused)	9 - Winchester Disk (future)
4 - Floppy Drive #Ø	1Ø - GPIB Channel (future)
5 - Floppy Drive #1	11 - RAM disk
6 - Printer	12 - reserved (unused)
128 - Channel configuration control	
129 - System clock access	
13Ø - General Memory access	

All registers (including the status register) are preserved.

Example:

```
        Read the left floppy disk directory
        LEA     ARGAREA,AØ      ;Set up arguments for BIOS call
        MOVE.W  #4,(AØ)         ;Channel number of floppy
        MOVE.L  #2Ø48.,4(AØ)    ;Size = 4 blocks (2K bytes)
        PEA     BUFFER          ;Form the address of the buffer
        MOVE.L  (SP)+,8(AØ)     ;Put address into arguments
        MOVE.W  #2,12.(AØ)      ;Directory is at block 2
        CLR.W   14.(AØ)         ;No special control options
        MOVEQ   #11.,DØ         ;Read is Function 11
        TRAP    #14.            ;BIOS call
        TST.W   2(AØ)           ;Check error reply code
        BNE.S   $1Ø             ;Had error
        ------
ARGAREA .BLOCK  16.
```

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 non-interrupt 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=4
A0=addr of Arguments:

Offset 0 - word sized count in 1/64000 ths of a second.
Offset 2 - long word sized 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 ths of a second, and a long word in seconds.

Function=6.....Read Clock in 1/60 ths 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 schedule entry into the scheduler queue. On timeout the scheduled routine is called with registers D0-D2 and address registers A0-A2 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 scheduler 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 - Logical Block Number (word)
Offset: 14 - Control word (word)
(see description under Read function below)
Offset: 16 - Start Execution Address (long 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 overwriten by the new program (caution: do not overwrite the BIOS). Note that only Channel numbers of mass storage devices are supported.

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 - Logical Block Number (word)
Offset: 14 - Control word (word)

TRAP #14

Channel Specific Data:

The Logical Block Number is never necessary for the non mass storage channels (1 and 7).

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 transfered.

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.

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 - Logical Block Number (word)
Offset: 14 - Control word (word)

TRAP #14

Channel Specific Data:

The Logical Block Number is never necessary for the non mass storage channels (2, 6, and 8).

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 transfered.

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 and Request to Send 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 while 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.

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

Channel Specific Data (returned in 30 word area):

For the Keyboard channel (1) the following data is returned:

Offset 0 - Number of characters 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 characters 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 -59. Currently unused

For the Printer channel (6) the following data is returned:

```
Offset   Ø - 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  5Ø - 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 Faulup
Offsets 52 -59 - Same as remote input channel 7 if Remote
                 output selected (otherwise unused).
```

For the Remote serial input channel (7) the following data is returned:

```
Offset  Ø - 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)
```

For the Remote serial output channel (8) the following data is returned:

```
Offset  Ø - Number of characters queued for output (word)
Offsets 2 -59. Currently unused
```

For the RAM disk (11.) the following data is returned:

```
Offset  Ø - Number of bytes queued (word)
            (always zero as transfer is immediate)
Offsets 2 -59. Currently unused
```

For the Memory Address Access channel (13Ø.), the p-code data area address in A6 is subtracted from the address of the status data area (offset 6 from AØ). The resulting word value is returned as the Error Reply Code (offset 2 from AØ).

Function=14.....Quiet (disable signaling events)

D0=14
A0=addr of arguments: none.
TRAP #14

This function is used by the UCSD 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 UCSD 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 UCSD 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 UCSD 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 registers A3, A5, and A6 must be unmodified from the values 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 Ø: Offset 2: Printer options flag (byte).
Bit Ø = 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 Ø of the Printer Options flag will prevent the Interpreter from generating any automatic Line Feeds.

Types > Ø are not currently assigned.

Function=19..... Enter Supervisor Mode

DØ=19
AØ=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.,DØ
        TRAP    #14.            ;Enter Supervisor mode
        ------
        ANDI.W  #ØDFFFH,SR      ;Back to User mode
```

FUNCTIONS=2Ø-255..... Currently Unused

These functions are not currently used and result in an immediate return with no action.

X.3 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 (never used - word)
     Offset: 14 - Control word = channel number
                  for access (word).
TRAP #14
```

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, Size in Bytes, and Logical Block Number are unused.

The channel specific configuration information follows for each channel.

Channel Ø (General System Configuration):

To read or write the General System Configuration:

DØ=11 to read
   12 to write
AØ=addr of arguments
   (see previous section for channel 128).
TRAP #14

The data format is:
     Offset: Ø - 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 1Ø). Zero days means no correction factor is defined.

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 0 to 1 - 10 for 16 times clock.
                  Bits 2 to 3 - 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 6 to 7 - 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 protocall 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 independantly configured. A 32 byte configuration area is transfered although only 18 bytes are currently defined. To read or write the 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 - 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).
Offsets 18 to 31 - are transfered but unused.

Channel 6 (Printer):

To read or write the printer configuration:

```
DØ=11 to read
   12 to write
AØ=addr of arguments
   (see previous section for channel 128).
TRAP #14
```

The data format is:

```
     Offset:  Ø - Printer polling Timeout
                  This entry is used in printer mode 3 to control
                  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 1/64ØØØths second)
                  This entry is used in printer mode 3 to control
                  the amount of time between attempts to poll the
                  printer.
     Offset:  4 - Printer mode
                  Ø - no printer channel supported.
                  1 - use Remote Serial output channel.
                  2 - Parallel (interrupt operation).
                  3 - Parallel (polled operation).
     Offset:  5 - Printer Options
                  Bit Ø 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 0 to 1 - 10 for 16 times clock.
                  Bits 2 to 3 - 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 6 to 7 - 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 - transfered 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 - transfered but unused.
```

X.4 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 driver. Codes -14. and -15. may come from any device.

SUMMARY OF BIOS CHANNEL ERROR CODES	
Ø	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 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.
-1Ø	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

XI.1 THE PROCESSOR:

The SAGE II 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.

XI.2 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

This light can also be controlled by software for user applications.

XI.3 THE MEMORY:

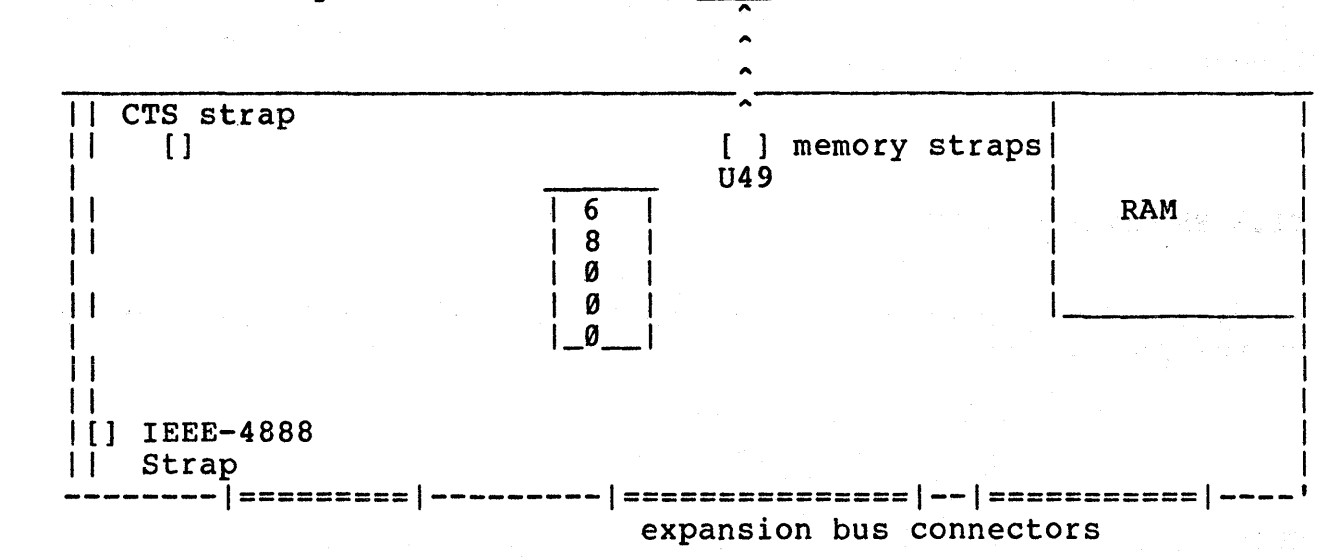
The SAGE II is configurable from 128k to 512k bytes in 128k increments. Memory is implemented in 64k dynamic ram with 150 nsec access time with no wait states. Parity checking is provided on each byte. All 512k bytes reside on the main processor board. 8K (64K reserved) EPROM firmware contains self-test, DEBUGGER, and bootstraps.

Memory is laid out on the board with respect to the chip addressing. There are 4 rows of rams, each row providing 2 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. Refer to the RAM CHIP vs ADDRESS TABLE to find specific ram locations.

A strapping option on the board 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.

The straps are implemented in an 8 pin shunt (U49) on the board.

```
     Remove straps:
     1-8, 2-7, 3-6   = 512K    1 |o----o| 8
          2-7, 3-6   = 384K    2 |o----o| 7  memory straps
               3-6   = 256K    3 |o----o| 6
     All straps in   = 128K    4 |o____o| 5
```



Refer to the parts location drawing to more exactly locate U49.

XI.4 RAM CHIP LOCATIONS:

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

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 BITS                          LOW BITS
 15 14 13 12 11 10  9  8 (parity) 7  6  5  4  3  2  1  0   addr

 || || || || || || || || ||   || || || || || || || || ||  07FFFF
 || || || || || || || || ||   || || || || || || || || ||  060000

 || || || || || || || || ||   || || || || || || || || ||  05FFFF
 || || || || || || || || ||   || || || || || || || || ||  040000

 || || || || || || || || ||   || || || || || || || || ||  03FFFF
 || || || || || || || || ||   || || || || || || || || ||  020000

 || || || || || || || || ||   || || || || || || || || ||  01FFFF
 || || || || || || || || ||   || || || || || || || || ||  000000
 ---------------------edge-of-board-----------------------
```

To find a specific bit, convert the hex digits of the words to binary:

EXAMPLE: Value should be FFFFFFFF but is FFFFFF7F at loc 024021

```
-----F-----  ----F-----          ----7-----  ----F----- = FF7F
 1  1  1  1  1  1  1  1          0  1  1  1  1  1  1  1
|| || || || || || || || ||   || || || || || || || || ||   03FFFF
|| || || || || || || || ||   || || || || || || || || ||   020000
15 14 13 12 11 10  9  8  parity 7  6  5  4  3  2  1  0   bits
                             ^
                             |
```

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

XI.5 PROM STRAPPING OPTIONS:

Various PROMs/ROMs can be used in the SAGE II. The board is currently configured for

8K or 16K byte PROMs

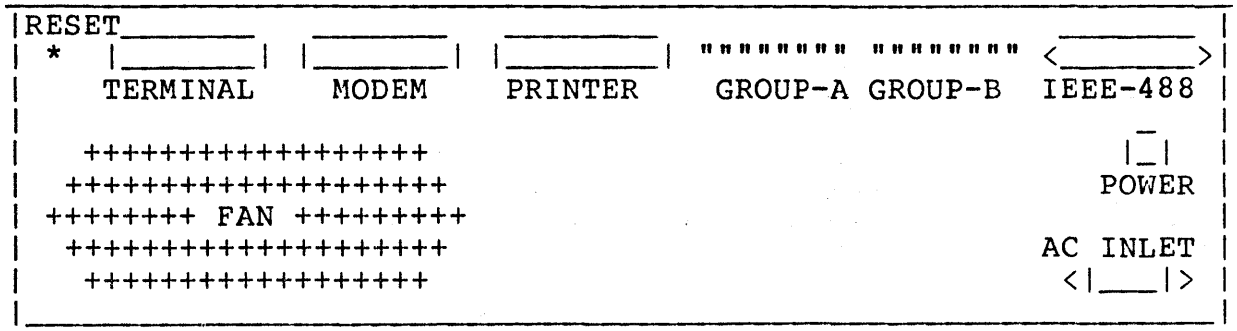
The strapping traces right above the PROMS on the board allow you to configure for larger or smaller PROMS:

64K bits (8K bytes)	as is		
128K bits (16K bytes)	as is		
16K bits (4K bytes)	[W]--[X]	[Y]	cut w-x, add x-y
256K bits (32K bytes)	[P]	[Q]--[R]	cut q-r, add p-q
512K bits (64K bytes)	[S]	[T]--[U]	cut t-u, add s-t

XI.6 I/Ø 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:

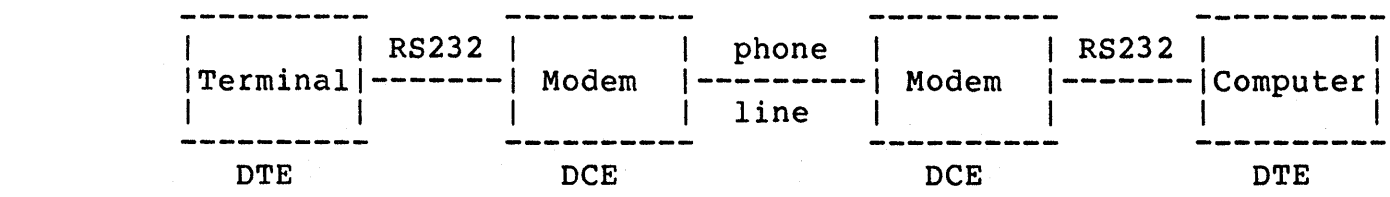
TERMINAL	RS232-C port
MODEM	RS232-C port
PRINTER	parallel port
GROUP-A	dipswitch
GROUP-B	dipswitch
IEEE-488	GPIB bus



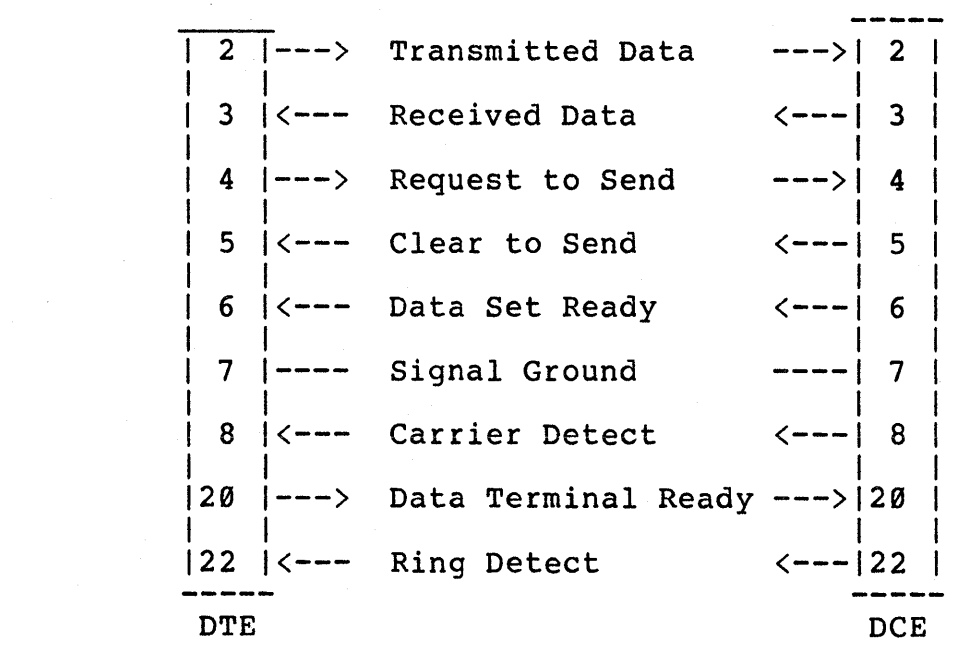
XI.6.1 RS232 PROTOCALL:

There always seems to be a lot of confusion involving the interconnection of serial devices using the RS232 protocall. This section is provided to clarify the protocall and terminology.

The RS232 standard was developed for the typical timesharing senerio where a terminal is connected through a pair of modems to a computer. The terminal and the computer are refered to as Data Terminal Equipment (DTE) and the modems are refered to as Data Communications Equipment (DCE).



A large source of the confusion comes from the fact that the signal names are from the perspective of the Data Terminal Equipment. The following diagram shows the relationship of the typically used RS232 signals.



Notice that for the Data Communications Equipment connector, pin 2 is called Transmitted Data but the signal is actually an input to the device. Therefore in order to know the direction of the signal it is important to know if the device is considered Data Terminal Equipment or Data Communications Equipment. Conversely, if the signal direction is known it will be possible to determine if the device is considered Data Terminal Equipment or Data Communications Equipment.

A second complication arises because in the typical microcomputer application, the terminal is connected directly to the computer without going through a set of modems. If the computer is considered Data Terminal Equipment this requires connecting two DTE devices back to back. To do this the signal lines on pins 2 and 3 must be cross connected with a non standard cable.

The SAGE II terminal port has therefore been set up as Data Communications Equipment in order to allow a standard cable to interconnect the computer to a Terminal. If the SAGE II terminal port is to be connected to a modem, then a special cable cross connecting pins 2 and 3 will be required. Notice that because the SAGE II terminal port is considered Data Communications Equipment the Transmitted Data signal is an input and the Received Data signal is an output.

The SAGE II modem port (Remote serial channel) is set up as Data Terminal Equipment to easily connect to a modem (DCE) with a standard cable. If the modem port is to be connected to a printer, a special cable will probably be required to interconnect two Data Terminal Equipment devices (cross connecting pins 2 & 3).

Another source of confusion is the signal levels on the RS232 lines. For the Transmitted Data and Received Data lines a logic Ø level (called the Space state) is represented by a positive voltage between 3 and 25 Volts. A logic 1 level (called the Mark state) is represented by a negative voltage between -3 and -25 Volts. An idle data circuit will stay in the logic 1 (Mark) state. All of the other RS232 control signals have the opposite signal level. An active Clear to Send (logic 1) is represented by a positive voltage between 3 and 25 Volts.

XI.6.2 "TERMINAL" PORT:

This serial port is provided to interface with the system terminal or other RS-232C device. It operates as Data Communication Equipment at software definable speeds from 50 - 19.2K baud in either interrupt-driven or polled mode. GROUP-A, DIP switches on the back panel defines the initial baud rate. The DB-25 connector is located at the rear of the machine and is marked "TERMINAL". Pin-outs for the signals are:

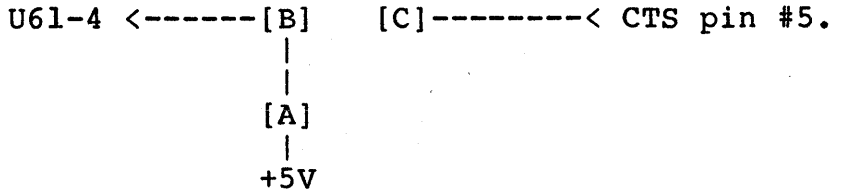
Terminal port is considered Data Communications Equipment (see previous section on RS232 Protocall).

J8 - 1	Protective Ground		
2	Transmitted data	(from terminal)	<---
3	Received data	(to terminal)	--->
5	Clear-to-send	(to terminal)	--->
7	Signal Ground		
9	+12 VDC		
10	-12 VDC		

This port is normally configured for 19.2k baud with even parity, 8 bits of data and 1 stop bit. The parity bit is may be disabled by setting SW4 up.

The CTS function is normally always held high. To change this go inside the computer and cut the strap trace from [B] (U61 pin 4) to [A] (+5V) and re-connect [B] to [C] (pin #5 of the RS-232C connector).

The strap is located at the end of the 8253 chip near the terminal connector at the rear corner of the board.



XI.6.3 "MODEM" PORT:

A second RS-232C serial port is available to connect to a modem or other RS-232C device. 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 (see previous section on RS232 Protocall).

J7 - 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 line (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 SAGEUTIL) 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 previous section on RS232 Protocall). Also verify that the proper baud rate, number of stop bits, and parity are consistant 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.

Using Clear to Send on Remote Channel.

The 8251A USART chip used for the Remote Channel serial interface has the protocall 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 (see writeup on SAGE Configuration Utility). 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.

XI.6.4 "PRINTER" PORT:

A parallel port designed for interface to a printer is also available on the back panel. It can also be used as a general purpose 14-bit software input, output or control port. The connector on the back panel is labeled PRINTER. Pin-outs for the signals are:

J6 -1	STROBE'	- clocks the data from computer to printer on the high to low transition.
3	DATAØ	- Bit Ø (lsb) of output data byte.
5	DATA1	- Bit 1 of output data byte.
7	DATA2	- Bit 2 of output data byte.
9	DATA3	- Bit 3 of output data byte.
11	DATA4	- Bit 4 of output data byte.
13	DATA5	- Bit 5 of output data byte.
15	DATA6	- Bit 6 of output data byte.
17	DATA7	- Bit 7 (msb) of output data byte.
19	ACK'	- acknowledgement from printer to computer that data byte was received.
21	BUSY	- High indicates that printer is busy.
23	PAPER EMPTY	- High indicates that printer is out of paper.
25	SELECT	- High indicates that device was selected.
26	PRIME'	- Computer sends to printer a low to clear buffer and reset printer logic.
28	FAULT'	- Low indicates a printer fault: paper empty, deselect, etc.
2,4,6,8 10,12,14 16,18,20 22,24,27 31,33	GROUND	- common signal ground is every other pin to minimize noise on signals.
29,30,32 34	NC	- These pins are not connected.

(27) 5V ?

XI.6.5 IEEE-488 PORT:

The IEEE-488 General Purpose Interface Bus (GPIB) is available for networking and instrument control. It allows up to 15 instruments to communicate over a common bus. Each device has one or two unique addresses set by software. Information is transmitted in byte serial, bit parallel format. Only one device can send data (TALK) while up to 14 can listen. A wide selection of instruments are available which interface to the GPIB. Maximum data rate can not exceed 1Mbyte/sec. Total transmission path length is 20 meters. SAGE II implements the GPIB in hardware with the TMS9914A controller and TI 75160,75162 buffers.

The connector for the bus is labeled IEEE-488. Its pin-outs are:

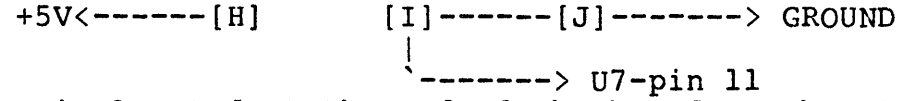
J5 - 1	DI01	bit 1 of GPIB data byte
2	DI02	2
3	DI03	3
4	DI04	4
13	DI05	5
14	DI06	6
15	DI07	7
16	DI08	8
17	REN	remote enable
9	IFC	interface clear
8	NDAC	not data accepted
7	NRFB	not ready for data byte
6	DAV	data valid
5	EOI	end of identify
11	ATN	attention
10	SRQ	service request

GROUP-B is an 8-bit dipswitch which can be read by the computer. Although this port can be used for any application it is normally used to setup the GPIB address for the TMS9914A chip.

The address register setup for the TMS9914 is:

SW1-SW5	A1-A5	Device Listen and Talk address.
SW6	Dat	used to disable TALKER function. set to Ø , enabling TALK.
SW7	Dal	used to disable Listen function. set to Ø ,enabling LISTEN.
SW8	edpa	used to assign the device 2 consecutive addresses. Normally set to Ø.

The port is normally strapped for open collector I/O. To change to tri-state driver mode cut the strap trace on the Sage II board from I U7-pin 11 to ground. Reconnect to H +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.ØØ) to:

IEEE STANDARDS
345 E. 47TH ST
NEW YORK, N.Y. 1ØØ17

For more information about the TMS 9914 GPIB chip:

TMS9914 GPIB ADAPTER PERLIMINARY DATA MANUAL MPØ33
TEXAS INSTRUMENTS, INC
P.O. BOX 5Ø12
DALLAS, TEXAS 75222

Refer also to "IEEE-488 SOFTWARE SUPPORT" in your operating system section. Sage provides a software package for the user to support the IEEE-488 bus. Details of interfacing to the bus are given there.

XI.7 THE REAL-TIME CLOCK:

A 6 byte timer with the least significant bit =1/64000 of a second is kept by SAGE II for access by the user to time events. The clock can be read using a "LONG CALL 5" to 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.

XI.8 POWER SUPPLY:

A switching power supply provides

```
+ 5V at 6.0 amps     .....SAGE II uses 3A
+12V at 2.5 amps
-12V at 0.6 amps
```

SAGE II comes with a detachable power cord for easy hookup. SAGE II will operate without modification on either 50 or 60 hz 120VAC.

XI.9 I/O ADDRESSING: To access any of the following I/O locations

for the system devices the processor must be in SUPERVISOR mode or a "BUS ERROR" will occur.

FE0000-FE1FFF		SYSTEM ROM	
FFC001		REAL TIME CLOCK	(A)
C003		SERIAL PORT 1 BAUD RATE	(B)
C005		SERIAL PORT 2 BAUD RATE	(C)
C007		MODE WORD FOR A,B AND C ABOVE	(8253-S)
C009-	C00F	RESERVED *	
FFC011-	C01F	IEEE-488 INTERFACE (TMS9914)	
FFC021		GROUP-A DIP SWITCH	(A)
C023		GROUP-B DIP SWITCH	(B)
C025		FLOPPY CONTROL PORT	(C)
C027		CONTROL FOR A,B AND C ABOVE	(8255A-S)
C029-	C02F	RESERVED *	
FFC031-		SERIAL PORT 2 (REMOTE) DATA	(8251A)
C033-		SERIAL PORT 2 CONTROL/STATUS	
C035-	C03F	RESERVED *	
FFC041-	C044	INTERRUPT ENCODER CONTROL	(8259)
C045-	C04F	RESERVED *	
FFC051-		FLOPPY DISK STATUS	(NEC 765)
C053-		FLOPPY DISK CONTROL	
CO55-	C05F	RESERVED *	
FFC061-		PRINTER INTERFACE PORT A	
C063-		PRINTER INTERFACE PORT B	
C065-		PRINTER INTERFACE PORT C	
C067-		PRINTER INTERFACE CONTROL	(8255A-S)
C069-	C06F	RESERVED *	
FFC071-		SERIAL PORT 1 (TERMINAL) DATA	(8251A)
C073-		SERIAL PORT 1 CONTROL/STATUS	
C075-	C07F	RESERVED *	
FFC081-	C087	REAL TIME CLOCK	(8253-S)
C089-	C08F	RESERVED *	

* Using these addresses may cause unpredictable results.

XI.10 PHYSICAL CHARACTERISTICS:

SAGE II measures 5.75" x 12" x 16.75" in a sturdy .063 aluminum case. It weighs a mere 18 lbs, is compact and easy to service. All chips are mounted in sockets.

XI.11 COOLING AND ENVIRONMENT:

A quiet 20 CFM fan is used for cooling. Air is pulled across the equipment in the case from the right air vents and forced out the rear. Care should be taken that the vents are not blocked.

Temperature:

16 to 44	degrees C operating	(60 to 112 F)
-44 to 71	degrees C non-operating	(-40 to 160 F)

Relative humidity:

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

Altitude: (operating & non-operating)

304.8 m (500 ft)	below sea level
15,240.0 m (50,000 ft)	above sea level

XI.12 FLOPPY DRIVE:

Fast, reliable, 5 1/4" floppy drives access either 320 k 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. SAGE II determines when it should turn off the motor by tracking previous usuage 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.

The floppy drives have been modified to work with the SAGE II controller. Detailed information on this modification and the drive specification is available in the Technical Design Package #DC0027.

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
	9	index
	30	raw data
	1-33	all odd pins grounded.

XI.13 SAGE EXPANSION BUS:

The SAGE II 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 follow which assumes the user is familiar with the general architecture of the 68000.

CONN-PIN	SIGNAL	DESCRIPTION
J1-1-49 odd pins	ground	System and logic ground. All odd pins are grounded to reduce signal noise.
J1-2-46 even pins	A1-A23	These 23 outputs are the cpu address lines buffered with octal drivers.
J1-48	FC2-	This output reflects the state of the supervisor bit (S) 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. It causes a level 7 interrupt on a parity error.

CONN-PIN	SIGNAL	DESCRIPTION
J2-1-33 odd pins	ground	System and logic ground on these odd pins reduces noise on the data bus.
J2-2-32 even pins	D0-D15+	The 68000 data bus is buffered with octal tranceivers which are 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, 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, 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 ouput,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 been completed.
J2-odd pins 35-45	+5V	
J2-47	+12V	
J2-49	-12V	

XII GUIDE TO BOOKS AND MANUALS:

In order for you to take full advantage of your Sage II computer, we want to point out the following literature. This is not intended to be a complete list by any means, but it should get you started. Please note that some of these works are valuable only if you have acquired the software option for which they are written. Sage does not gaurantee completeness or accuracy of any literature described here. Most are available at your local computer book supplier, or contact the publishers. The SOFTECH books are available from Sage directly, as is the Motorola manual.

MC68000 16-BIT MICROPROCESSOR USER'S MANUAL

Available from Motorola Inc, this manual is intended for use by computer designers, software architects, and design engineers. It explains the instruction set of the 68000 chip. Also included are timing information, pin descriptions and hardware interfacing notes for this powerful CPU.

THE 68000: PRINCIPLES AND PROGRAMMING

By Leo J. Scanlon, a SAMS publication. This is essentially a textbook for learning assembly level programming for the 68000. Included is some good discussion of the philosophy of the 68000, and some excellent charts and tables for reference.

UCSD p-SYSTEM APPLICATION CATALOG

Published by Softech Microsystems. This is a catalog and cross-reference of p-system application software available. Although not listing all the software we know of, it is certainly a good starting point if you're looking for good software. Currently some 80 or so companies are listed, and most or all of the software is directly compatable with the p-system on the Sage II.

THE UCSD PASCAL HANDBOOK

Specific to the UCSD implementation of Pascal, this book defines the complete UCSD Pascal Language. As a definition of the language itself is not covered in the standard UCSD p-SYSTEM USER'S MANUAL, this book is a very valuable reference. By Randy Clark and Stephen Koehler, it is published by Prentice-Hall, Inc.

PASCAL USER MANUAL AND REPORT

By Kathleen Jensen and Niklaus Wirth, and published by Springer Verlag. This is a user manual for the Standard definition of Pascal. UCSD Pascal contains many extensions beyond this definition. The second section is a report on the language. This is the original Pascal manual.

MICROCOMPUTER PROBLEM SOLVING USING PASCAL

By Kenneth L. Bowles, and published by Springer Verlag. This is the original textbook for learning UCSD Pascal, and still an excellent guide for the beginner.

PASCAL PRIMER

Published by Howard Sams & Co., written by David Fox and Mitchell Waite. A good beginner's text, it is aimed at those who have some previous contact with BASIC. There are lots of comparisons between the two languages, and it shows many examples of the superiority of Pascal. So if you've been working with BASIC, this may be the book to convert you.

THE PASCAL HANDBOOK

Written by Jacques Tiberghien, published by Sybex. An indispensable text for anyone writing in Pascal, it should have been called the encyclopedia of Pascal. That's really what it is, with 473 pages of terms, words, and phrases explained in alphabetical order. Not only that, but the known differences between the various dialects of Pascal are fully explained. So if you get involved in converting from a non-UCSD Pascal, you will NOT want to be without it.

TMS9914 GPIB ADAPTER PRELIMINARY DATA MANUAL MPØ33

Published by Texas Instuments, Inc., here is a technical manual for those who need to interface with the GPIB (IEEE-488) bus. The Sage II uses the TMS9914 Chip for that purpose. This manual is for hardware designers and engineers.

IEEE-488 BUS STANDARDS

Published by IEEE STANDARDS, 345 E. 47th St., New York, N.Y. 1ØØ17. Another text for those who need to interface with the GPIB (IEEE-488) bus. This is the official IEEE-488 definition.

APPENDIX A: SWITCH SETTINGS

GROUP-A

SW3	SW2	SW1	Terminal baud rate
8 data bits, 1 stop bit and even parity.			
dwn	dwn	dwn	19.2K baud
dwn	dwn	up	9600
dwn	up	dwn	4800
dwn	up	up	2400
up	dwn	dwn	1200
up	dwn	up	600
up	up	dwn	300
up	up	up	reserved, will default to 19.2K baud.

SW4	Parity control
dwn	even parity enabled
up	disabled

SW6	SW5	BOOT DEVICE
dwn	dwn	boot to DEBUGGER
dwn	up	boot to Floppy drive 0.
up	dwn	reserved, defaults to DEBUGGER
up	up	reserved, defaults to DEBUGGER

SW7	FLOPPY CONFIGURATION
dwn	96 TPI drive
up	48 TPI drive

SW8 **Reserved**

GROUP-B - Standard IEEE-488

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 2 consecutive addresses. Normally set to 0.

APPENDIX B: FLOPPY DISK BOOT ERRORS

"Not BOOT disk"
"Boot aborted on drive Ø"
"Drive error (code) on drive (Ø or 1)" where codes are:

Ø1	- controller failure
Ø2	- invalid command
Ø3	- recalibrate or seek failure
Ø4	- timeout
Ø5	- missing address mark
Ø6	- no data found
Ø7	- overrun
Ø8	- CRC error
Ø9	- end-of-cylinder
ØA	- unknown
ØB	- address out-of-range

APPENDIX C: BIOS CHANNELS

1 - Keyboard (for input)	7 - Remote Serial Input Channel
2 - Terminal (for output)	8 - Remote Serial Output Channel
3 - reserved (unused)	9 - Winchester Disk (future)
4 - Floppy Drive #0	10 - GPIB Channel (future)
5 - Floppy Drive #1	11 - RAM disk
6 - Printer	12 - reserved (unused)

128 - Channel configuration control
129 - System clock access
130 - General Memory access
131 - Scheduling Events

APPENDIX D: 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 E: 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)
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)
```

continued on next page...

continued...

APPENDIX E: LONG CALLS TO BIOS

12..Write Data to a Channel.
- Ø - 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 68ØØØ Supervisor mode

APPENDIX F: BIOS CHANNEL ERROR CODES

Ø	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.
-1Ø	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 G: PROM ROUTINE ENTRY POINTS

KEYBCH....	Get a Keyboard Character	loc=FE0008H
KEYCHK....	Check for a Keyboard Character	loc=FE000CH
TERMCHAR..	Printout a Character to Terminal	loc=FE0014H
TERMTEXT..	Printout a Text String	loc=FE0018H
TERMCRLF..	Print a Carriage Return/Line Feed	loc=FE001CH
TERMHEXB..	Printout a Hexadecimal Byte	loc=FE0020H
TERMHEXW..	Printout a Hexadecimal Word	loc=FE0024H
FDREAD....	Floppy Disk Read	loc=FE0028H
FDWRITE...	Floppy Disk write	loc=FE002CH
DEBUG.....	Debugger Entry Point	loc=FE0030H

MACROS USED WITH PROM ROUTINE ENTRY POINTS:

LONG JSR MACRO:

```
.MACRO  LJSR
.WORD   4EB9H
.WORD   00FEH
.WORD   %1
.ENDM
```

LONG JMP MACRO:

```
.MACRO  LJMP
.WORD   4EF9H
.WORD   00FEH
.WORD   %1
.ENDM
```

NOTE: These routines must be called in 68000 SUPERVISOR mode.

APPENDIX H: I/O ADDRESSING

FE0000-FE1FFF		SYSTEM ROM	
FFC001		REAL TIME CLOCK	(A)
C003		SERIAL PORT 1 BAUD RATE	(B)
C005		SERIAL PORT 2 BAUD RATE	(C)
C007		MODE WORD FOR A,B AND C ABOVE	(8253-S)
C009-	C00F	RESERVED *	
FFC011-	C01F	IEEE-488 INTERFACE (TMS9914)	
FFC021		GROUP-A DIP SWITCH	(A)
C023		GROUP-B DIP SWITCH	(B)
C025		FLOPPY CONTROL PORT	(C)
C027		CONTROL FOR A,B AND C ABOVE	(8255A-S)
C029-	C02F	RESERVED *	
FFC031-		SERIAL PORT 2 (REMOTE) DATA	(8251A)
C033-		SERIAL PORT 2 CONTROL/STATUS	
C035-	C03F	RESERVED *	
FFC041-	C044	INTERRUPT ENCODER CONTROL	(8259)
C045-	C04F	RESERVED *	
FFC051-		FLOPPY DISK STATUS	(NEC 765)
C053-		FLOPPY DISK CONTROL	
CO55-	C05F	RESERVED *	
FFC061-		PRINTER INTERFACE PORT A	
C063-		PRINTER INTERFACE PORT B	
C065-		PRINTER INTERFACE PORT C	
C067-		PRINTER INTERFACE CONTROL	(8255A-S)
C069-	C06F	RESERVED *	
FFC071-		SERIAL PORT 1 (TERMINAL) DATA	(8251A)
C073-		SERIAL PORT 1 CONTROL/STATUS	
C075-	C07F	RESERVED *	
FFC081-	C087	REAL TIME CLOCK	(8253-S)
C089-	C08F	RESERVED *	

APPENDIX I: DEBUGGER COMMANDS

>DA[x]	Display A registers or Ax
>DB[x]	Display breakpoint info
>DD[x]	Display D registers or Dx
>DM [$X+]xxxxxx,[#,$x+]xxxxxx	Display memory
>DP	Display program counter
>DR	Display all registers.
>DS	Display status register
>DU	Display user stack
>D$[x]	Display base regs or $x
>FB [$x+]xxxxxx, [#,$x+]xxxxxx, xx	Fill memory / byte data
>FL [$x+]xxxxxx, [#,$x+]xxxxxx, xxxxxx	Fill memory / long data
>FW [$x+]xxxxxx, [#,$x+]xxxxxx, xxxx	Fill memory / word data
>GC [[$X+]XXXXXX]	Execute program
>GO [[$X+]XXXXXX]	Execute prog / bkpt reset
>IFx	Boot floppy from drive x (Ø=left,1=right) drive
>IS	Initialize SYSTEM
>LA	Load from a remote device
>LF	Load from a floppy
>LT	Load from the terminal
>M[$X+]XXXXXX,[#,$X+]XXXXXX,[$X+]XXXXXX	Move data in memory
>POB [$X+]XXXXXX, DD	Output data byte to port
>POW [$X+]XXXXXX, DDDD	Output data word to port
>PIB [$X+]XXXXXX	Input data byte from port
>PIW [$X+]XXXXXX	Input data word from port
>PS x	Set remote baud rate
>SA[x] [XXXXXXXX]	Modify A registers or Ax
>SB[x] [XXXXXXXX]	Set breakpoint regs or Bx
>SD[x] [XXXXXXXX]	Modify D registers or Dx
>SM [$X+]XXXXXX	Modify memory
>SP [XXXXXXXX]	Modify Program counter
>SR	Modify all registers
>SS [XXXXXXXX]	Modify Status Register
>SU [XXXX]	Modify User Stack
>S$[x] [XXXXXXXX]	Modify base regs or $x
>TB [[$X+]XXXXXX]	Trace without reg print
>TE	Terminates trace mode
>TR [[$x+]XXXXXX]	Trace with register print
>WF	Write to floppy
>	<CR> trace next inst
>$x	Sets standard base reg
>$	Clears standard base reg

APPENDIX J: ASCII CODES

TABLE 1- ASCII CODES (HEX)

msd \ lsd	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	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	0	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

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	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

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)

TABLE 2 can also be used as a general purpose conversion from HEX to decimal or from decimal to HEX.

APPENDIX K: 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.
BASIC.R2.CODE	BASIC Compiler for generating code using 2 word real arithmetic.
BASIC.R4.CODE	BASIC Compiler for generating code using 4 word real arithmetic.
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.
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 disassembly 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).
ERRORHANDL.CODE	Interface section for the operating system's ERRORHANDL Unit.

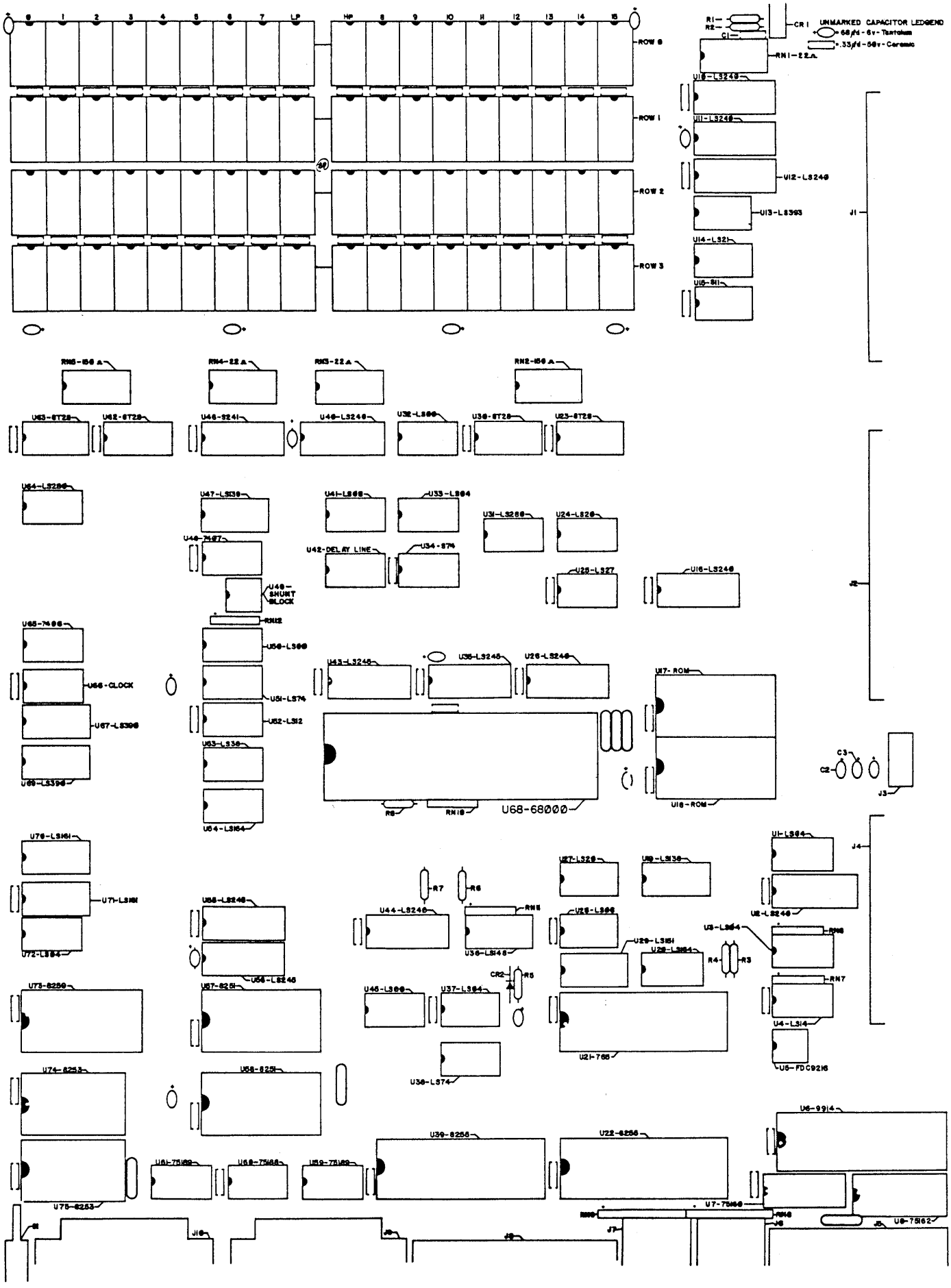
FILE.INFO.CODE	Unit used to access file information.
FINDPARAMS.CODE	Program to find best interleaving for diskettes (see Installation Guide, section IV.2).
FORTLIB2.CODE	Runtime support library for Fortran 77 using 2 word real arithmetic.
FORTLIB4.CODE	Runtime support library for Fortran 77 using 4 word real arithmetic.
FORTRAN2.CODE	Fortran 77 Compiler for generating code using 2 word real arithmetic.
FORTRAN4.CODE	Fortran 77 Compiler for generating code using 4 word real arithmetic.
IB.BUS.CODE IB.BUS.TEXT	SAGE provided 68000 assembly language routines for interfacing to the IEEE 488 Bus.
IB.EX.CODE IB.EX.TEXT	Example of Pascal program which uses the IEEE 488 bus Unit.
IB.DEF.TEXT	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.CODE IB.UNIT.TEXT	The Pascal portion of the IEEE 488 Bus Unit.
INTERP.0.CODE	Interpreter with no real arithmetic.
INTERP.2.CODE	Interpreter with 2 word real arithmetic.
INTERP.4.CODE	Interpreter with 4 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.
PATCH.CODE	Program for viewing and altering file information at a primitive (hexadecimal) level.

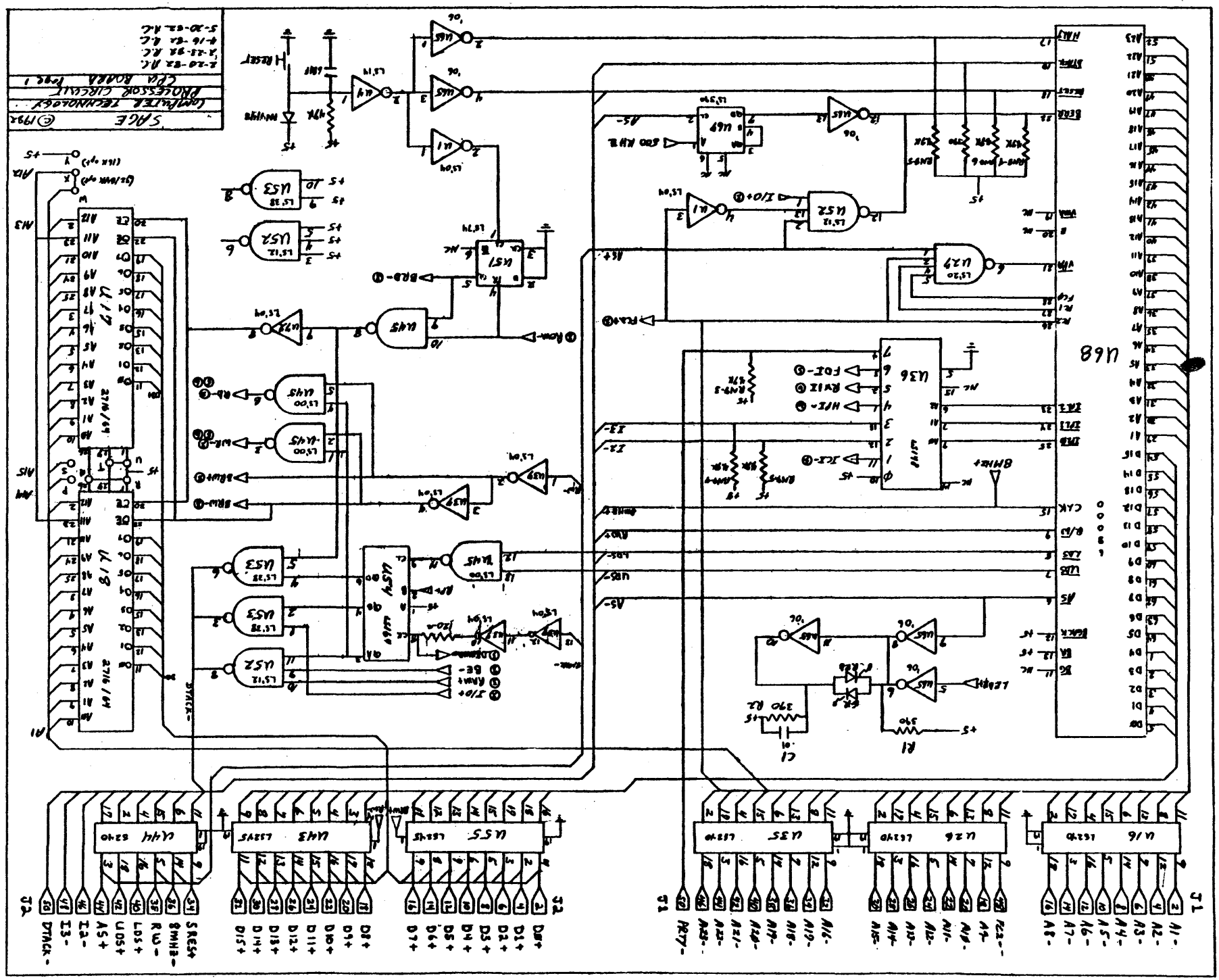
REALCONV.CODE	Program to convert real number constants in code files to a non-portable (but operationally faster) representation.
REALOPS.2.CODE	Real arithmetic library for 2 word real numbers.
REALOPS.4.CODE	Real arithmetic library for 4 word real numbers.
RECEIVE.CODE RECEIVE.TEXT	SAGE provided program to receive the image of a device from another system on the Remote serial channel.
RECOVER.G.CODE	Program which attempts to re-create the directory of a disk whose directory has accidentally been destroyed.
REMINTEST.CODE REMINTEST.TEXT	SAGE provided program to test the Remote serial channel input.
REMOUTTEST.CODE REMOUTTEST.TEXT	SAGE provided program to test the Remote serial channel output.
REMTALK.TEXT REMTALK.CODE	Program to transfer files over the Remote serial channel.
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.
SAGEUTIL.CODE	Program to configure the BIOS, format diskettes, and copy bootstraps.
SAGE.PBOOT.CODE SAGE.PBOOT.TEXT	Assembly language routine which bootstraps the BIOS and Interpreter into operation.
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 (documented in the Internal Architecture Guide, section IV.4).
SCREENTEST.CODE	Program for testing the SYSTEM.MISCINFO and GOTOXY features (documented in the Installation Guide, section III.4).

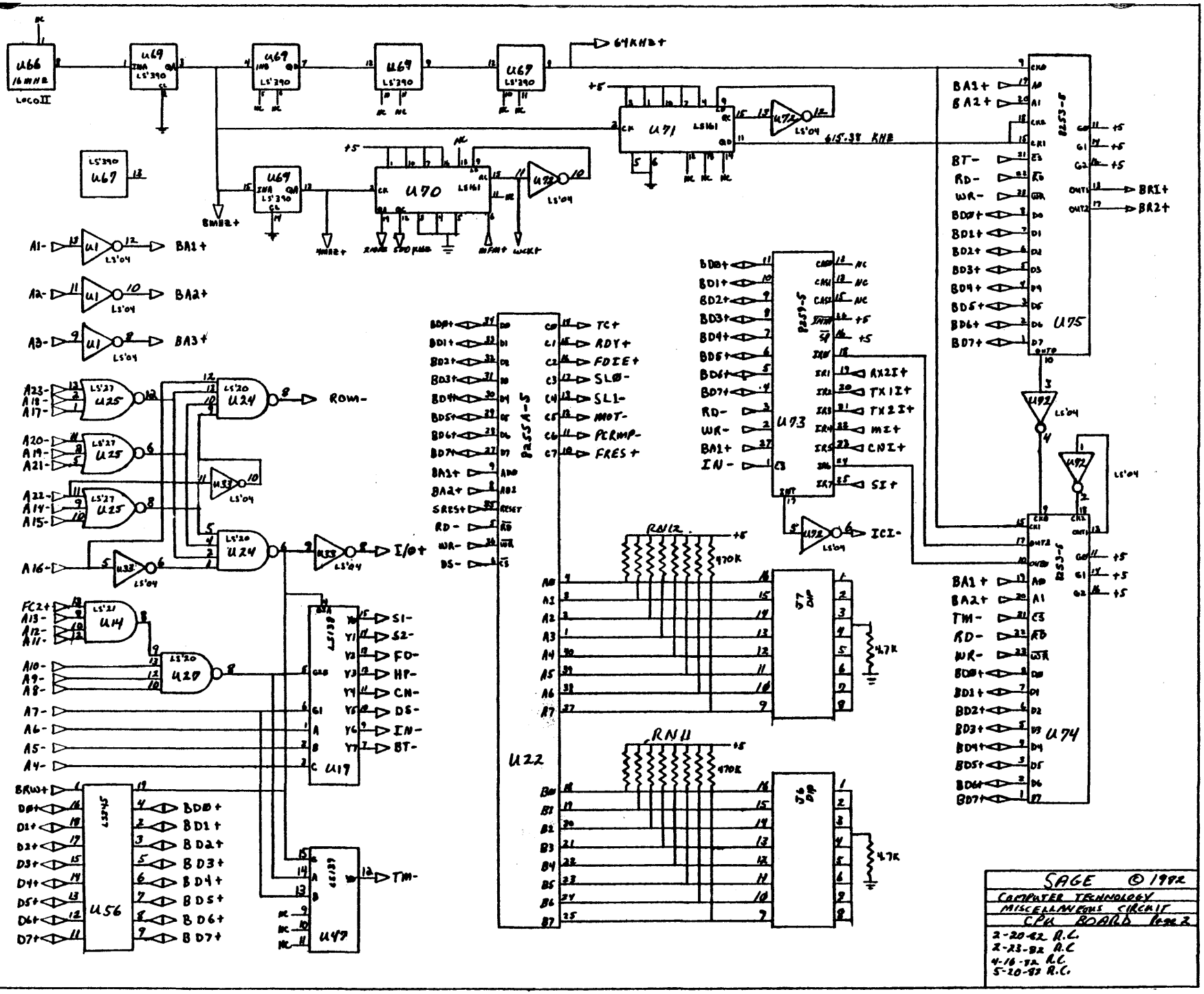
SEND.CODE SEND.TEXT	SAGE provided program to send the image of a device to another system over the Remote serial channel.
SETUP.CODE	Program for altering SYSTEM.MISCINFO for different terminal and system configurations.
SPOOLER.CODE	Program to set up files for printout by the Print Spooler.
SYSTEM.ASSMBLER	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.
TEXTIN.CODE TEXTIN.TEXT	Program to receive ASCII text from another system over the Remote serial channel.

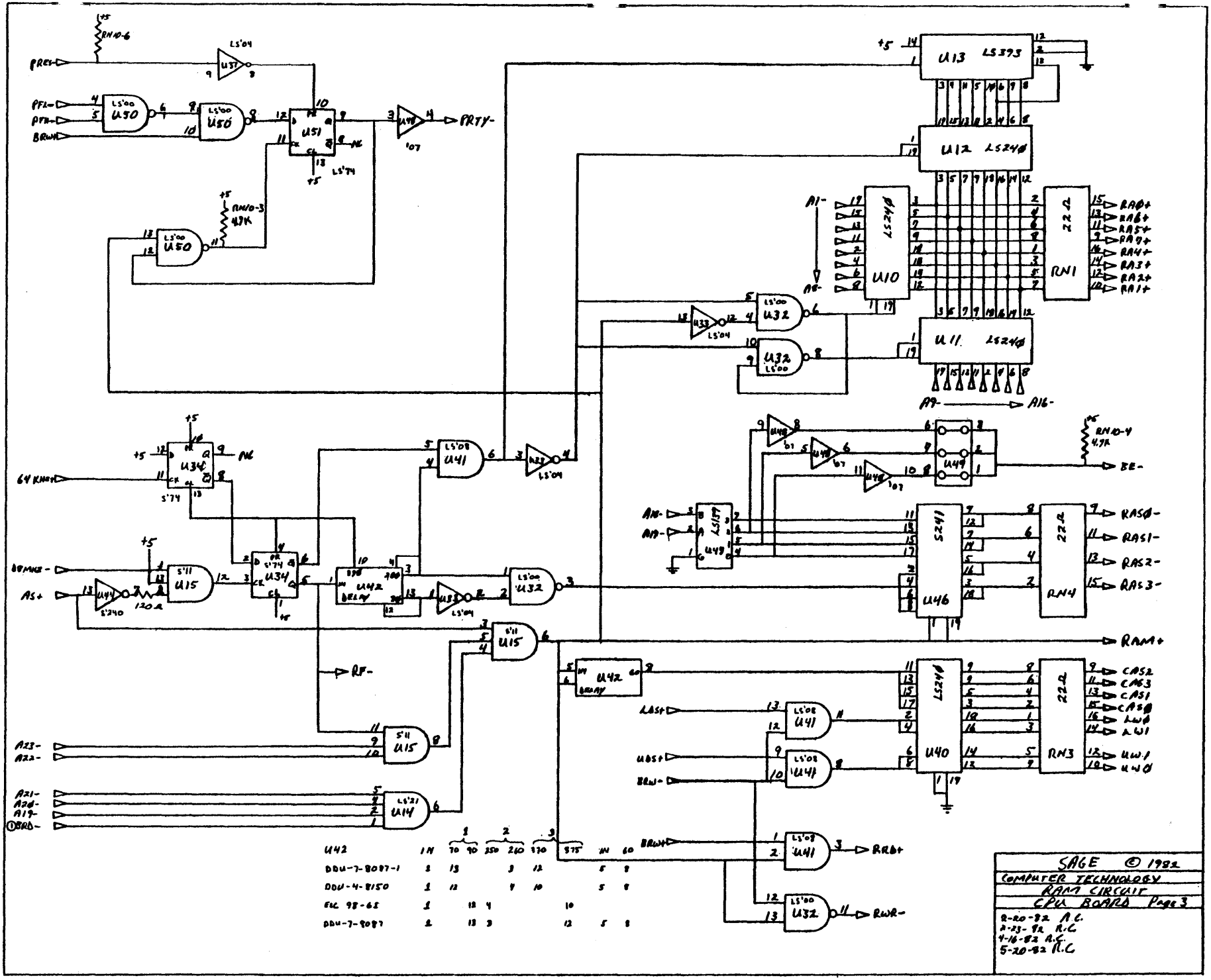
WILD.CODE	Unit for performing 'wild card' searches (pattern matching of strings).
XREF.CODE	Program to produce a procedure cross reference list for Pascal programs.
YALOE.CODE	Line Oriented Editor.
68000.OPCODES	Data file containing op-code information for the 68000 Assembler.
68000.ERRORS	Data file containing error messages for the 68000 Assembler.

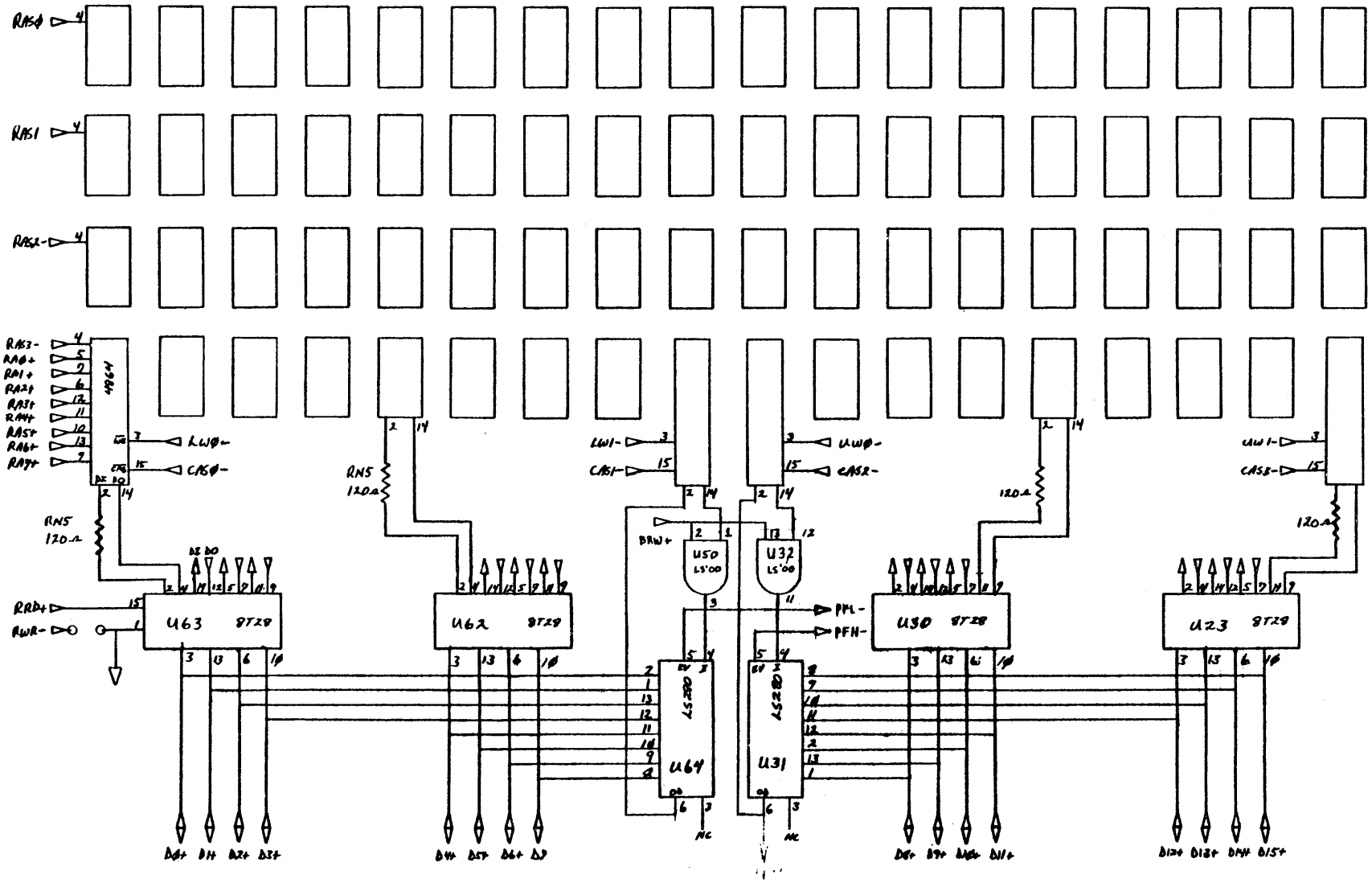
Other files may be present on the distribution diskettes. Most of these files will be xxxxx.MISCINFO or xxxx.GOTO.CODE which are SYSTEM.MISCINFO configurations and GOTOXY Units for specific terminals. These and any other new files should be covered in a separate document shipped with the system called System Release Notes.


UNMARKED CAPACITOR LEDGEND
.68 µfd - 6v - Tantalum
.33 µfd - 50v - Ceramic
ROW 0
ROW 1
ROW 2
ROW 3
R1
R2
C1
CR1
RN1 - 22 A
U10 - LS240
U11 - LS240
U12 - LS240
U13 - LS393
U14 - LS21
U15 - 811
J1
RN5 - 150 A
RN4 - 22 A
RN3 - 22 A
RN2 - 150 A
U63 - 8T28
U62 - 8T28
U46 - S241
U40 - LS240
U32 - LS86
U30 - 8T28
U25 - 8T28
U64 - LS280
U47 - LS138
U41 - LS85
U33 - LS86
U31 - LS280
U24 - LS20
U48 - 7407
U42 - DELAY LINE
U34 - 874
U48 - SHUNT BLOCK
U25 - LS27
U16 - LS240
J2
U65 - 7486
RN12
U56 - LS86
U66 - CLOCK
U51 - LS74
U67 - LS390
U52 - LS32
U69 - LS390
U53 - LS30
U43 - LS245
U35 - LS245
U26 - LS240
U17 - ROM
U54 - LS164
R8
RN19
U68 - 68000
U18 - ROM
C3
C2
J3
U70 - LS161
U71 - LS151
U72 - LS86
U58 - LS240
U56 - LS240
U73 - 8256
U57 - 8251
R7
R6
U44 - LS240
RN11
U36 - LS148
U27 - LS20
U19 - LS138
U28 - LS86
U29 - LS151
U20 - LS164
R4
R3
CR2
R5
U46 - LS86
U37 - LS164
U38 - LS74
U21 - 765
U1 - LS86
J4
U2 - LS240
U3 - LS86
RN6
RN7
U4 - LS14
U5 - FDC9216
U74 - 8253
U75 - 8253
U50 - 8251
U39 - 8256
U22 - 8256
U6 - 9914
U61 - 75189
U60 - 75188
U59 - 75189
U7 - 75188
RN8
RN9
J6
J7
J8
J5
U8 - 75162
J10
J9

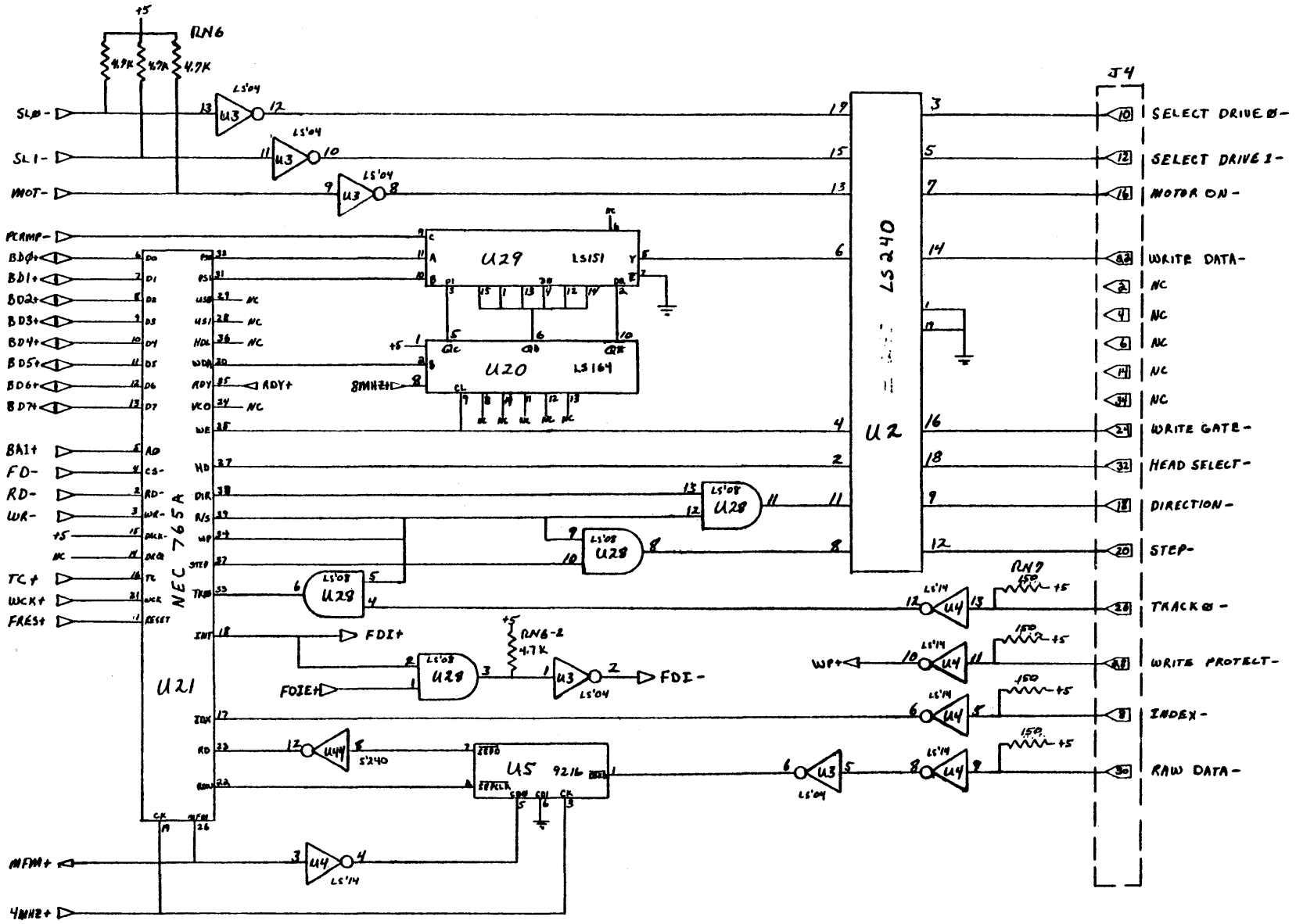



SAGE © 1982
COMPUTER TECHNOLOGY
MISCELLANEOUS CIRCUIT
CPU BOARD Issue 2
2-20-82 R.C.
2-23-82 R.C.
4-16-82 R.C.
5-20-82 R.C.

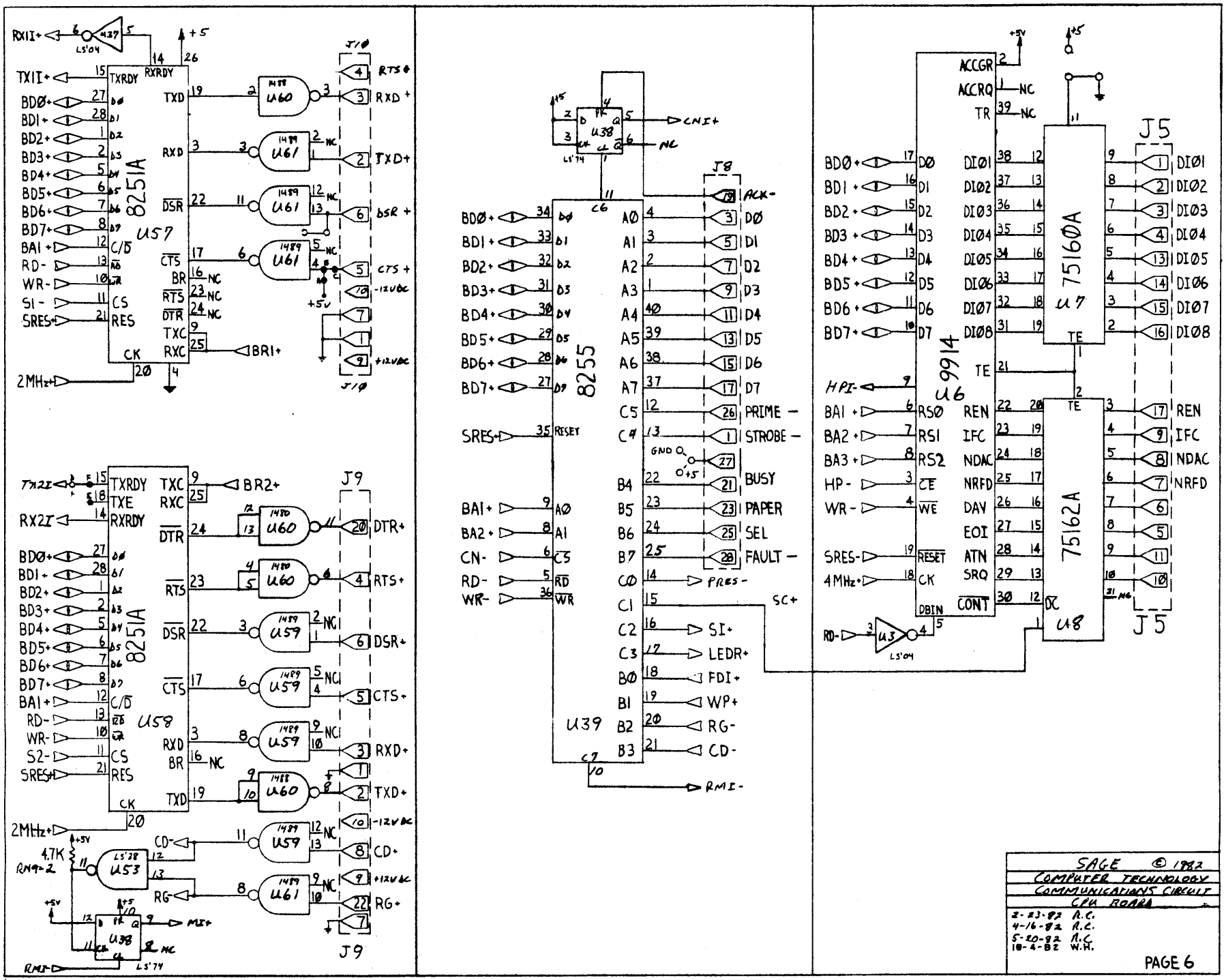

PRTY-
U13 LS373
U12 LS240
U11 LS240
U10 LS257
RN1 220
RN4 220
RN3 220
U46 S241
U40 LS240
U49 LS157
U42 DDU-7-8087-1
RAM+
RF-
BE-
RAS0-
RAS1-
RAS2-
RAS3-
CAS2
CAS3
CAS1
CAS0
LWR
LWI
UWI
UWR
RRD+
RWR-
SAGE © 1982
COMPUTER TECHNOLOGY
RAM CIRCUIT
CPU BOARD Page 3
8-20-82 A.C.
9-23-82 R.C
4-16-82 A.C
5-20-82 R.C

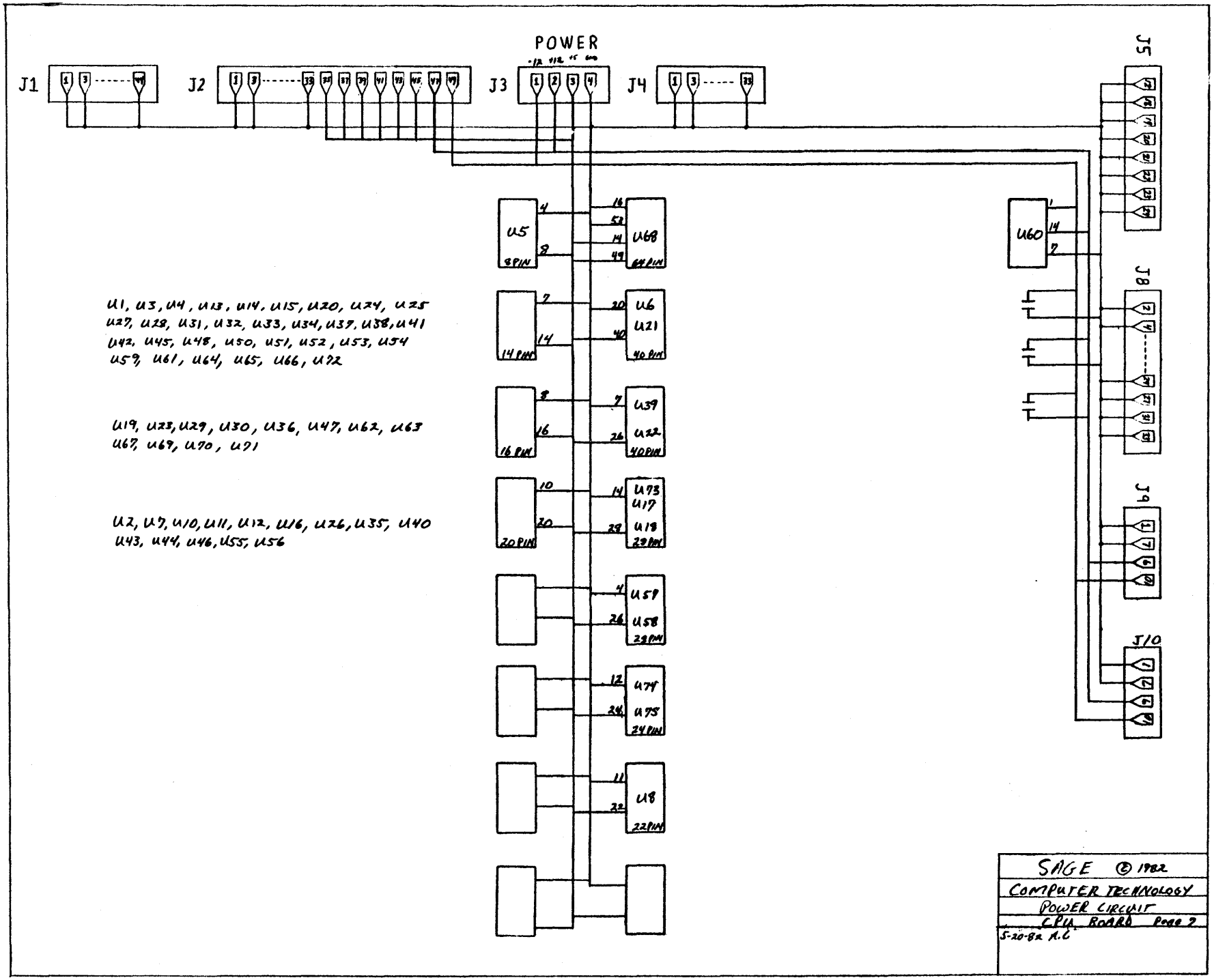


SAGE © 1982
COMPUTER TECHNOLOGY
RAM ARRAY
CPU BOARD Page4
2-20-82 R.C
2-23-82 R.C
4-16-82 R.C.
5-20-82 R.C.



SAGE © 1982
COMPUTER TECHNOLOGY
FLOPPY DISK CONTROLLER
CPU BOARD PAGE 5
5-20-82 R.G


8251A
U57
8251A
U58
8255
U39
9914
U6
75160A
U7
75162A
U8
U37
U38
U53
U60
U61
U59
J10
J9
J8
J5
SAGE © 1982
COMPUTER TECHNOLOGY
COMMUNICATIONS CIRCUIT
CPU BOARD
2-23-82 R.C.
4-16-82 R.C.
5-20-82 R.C.
10-4-82 W.H.
PAGE 6


POWER
J1
J2
J3
J4
J5
J8
J9
J10
U1, U3, U4, U13, U14, U15, U20, U24, U25
U27, U28, U31, U32, U33, U34, U37, U38, U41
U42, U45, U48, U50, U51, U52, U53, U54
U59, U61, U64, U65, U66, U72
U19, U23, U29, U30, U36, U47, U62, U63
U67, U69, U70, U71
U2, U7, U10, U11, U12, U16, U26, U35, U40
U43, U44, U46, U55, U56
U5
8 PIN
U68
64 PIN
U6
U21
14 PIN
40 PIN
U39
U22
16 PIN
40 PIN
U73
U17
U18
20 PIN
28 PIN
U57
U58
28 PIN
U74
U75
24 PIN
U8
22 PIN
U60
SAGE © 1982
COMPUTER TECHNOLOGY
POWER CIRCUIT
CPU BOARD Page 2
5-20-82 A.C

07-19-82

SAGE II MICROCOMPUTER

SAGE #	QTY	U/M	DESCRIPTION
BD0000	1	ea	SAGE 68000 CPU BOARD, assembled
CA0009	1	ea	DETACHABLE AC POWER CORD
CC0006	1	ea	AC/DC WIRING HARNESS
CC0007	1	ea	FLOPPY DRIVE CABLE
CN0011	1	ea	AC POWER RECEPTACLE HOUSING
DC0000	1	ea	SAGE II OWNER'S MANUAL
FN0002	4	ea	#6-32 X .375 SCREW, BLACK
FN0006	4	ea	.625 X .375 RUBBER FOOT
FN0007	80	ea	#6-32 X .250 SCREW, BLACK
MC0004	1	ea	SERIAL NUMBER
MC0005	0.7	ft	.250" X 1" RUBBER STRIP
MC0006	1	ea	TA300S FAN
SH0000	1	bx	CARDBOARD SHIPPING BOX
SH0001	.02	rl	PACKING TAPE
SH0002	1	ea	PACKING LIST HOLDER
SU0000	1	ea	FLOPPY DISK DRIVE, 48 TPI
SU0001	1	ea	COMPOWER POWER SUPPLY
SM0000	1	ea	BASE
SM0001	1	ea	COVER
SM0002	1	ea	DRIVE SHIELD
SM0003	1	ea	POWER SHIELD
SW0004	1	ea	POWER SWITCH

07-12-82

SAGE 68000 CPU BOARD

SAGE #	QTY	U/M	DESCRIPTION
BC0000	1	ea	68000 MICROPROCESSOR --U68
BC0001	1	ea	NEC-765 FLOPPY DISK CONTROLLER --U21
BC0002	1	ea	9914A IEEE-488 CONTROLLER --U6
BC0003	2	ea	8255A-5 PARALLEL PORT --U22,U39
BC0004	2	ea	8251A USART --U57,U58
BC0005	1	ea	8259-5 INTERRUPT CONTROLLER --U73
BC0006	2	ea	8253-5 TIMER --U74,U75
BC0007	2	ea	2732 EPROM --U17,U18
CB0001	1	ea	CIRCUIT BOARD, 11.625" X 15.500"
CP0000	15	ea	TANTALUM, 68UFD, 6.3VDC
CP0001	80	ea	0.33UFD 2-pin DIP
CP0002	1	ea	0.01UFD 2-pin DIP --C1
DI0000	1	ea	LED, RIGHT ANGLE --CR1
DI0001	1	ea	1N4148 SWITCHING DIODE --CR2
FN0000	14	ea	#4-40 X .250 SELF-THREAD SCREW
HD0000	2	ea	RS-232C HEADER --J9,J10
HD0002	1	ea	IEEE-488 HEADER --J5
HD0003	1	ea	IEEE-488 KIT
HD0004	2	ea	34-pin 3M HEADER --J4,J8
HD0005	2	ea	50-pin 3M HEADER --J1,J2
HD0006	1	ea	4-pin BOARD POWER HEADER --J3
RN0001	3	ea	22 OHMS, 16-pin DIP --RN1,RN3,RN4
RN0002	2	ea	470K OHMS, 10-pin SIP --RN8,RN9
RN0003	4	ea	4.7K OHMS, 6-pin SIP --RN6,RN10,RN11,RN12
RN0004	2	ea	120 OHMS, 16-pin DIP --RN2,RN5
RN0005	1	ea	150 OHMS, 6-pin SIP --RN7

SAGE #	QTY	U/M	DESCRIPTION
RS0000	3	ea	390 OHMS, --R1,R2,R8
RS0001	2	ea	4.7K OHMS --R3,R4
RS0002	1	ea	47K OHMS --R5
RS0003	2	ea	120 OHMS --R6,R7
SK0000	2	ea	8-pin SOCKET
SK0001	33	ea	14-pin SOCKET
SK0002	87	ea	16-pin SOCKET
SK0004	14	ea	20-pin SOCKET
SK0005	1	ea	22-pin SOCKET
SK0006	2	ea	24-pin SOCKET
SK0007	5	ea	28-pin SOCKET
SK0008	4	ea	40-pin SOCKET
SK0009	1	ea	64-pin SOCKET
SK0010	2	ea	16-pin SOCKET --J6,J7
SW0000	2	ea	16-pin DIP SWITCH
SW0001	1	ea	8-pin DIP SHUNT BLOCK --U49
SW0002	1	ea	RESET SWITCH, SPDT, PB, MOM.
SW0003	1	ea	RESET SWITCH END-CAP
TC0000	3	ea	74LS00 NAND GATE --U32,U45,U50
TC0001	5	ea	74LS04 BUFFER --U1,U3,U33,U37,U72
TC0002	1	ea	7406 BUFFER --U65
TC0003	1	ea	7407 BUFFER --U48
TC0004	2	ea	74LS08 AND GATE --U28,U41
TC0005	1	ea	74S11 AND GATE --U15
TC0006	1	ea	74LS12 NAND GATE --U52
TC0007	1	ea	74LS14 BUFFER --U4
TC0008	2	ea	74LS20 NAND GATE --U24,U27

SAGE #	QTY	U/M	DESCRIPTION
TC0009	1	ea	74LS21 AND GATE --U14
TC0010	1	ea	74LS27 NOR GATE --U25
TC0011	1	ea	74LS38 NAND GATE --U53
TC0012	1	ea	74S74 FLIP-FLOP --U34
TC0013	2	ea	74LS74A FLIP-FLOP --U38,U51
TC0014	1	ea	74LS138 DEMUX --U19
TC0015	1	ea	74LS139 DEMUX --U47
TC0016	1	ea	74LS148 PRIORITY ENCODER --U36
TC0017	1	ea	74LS151 MUX --U29
TC0018	2	ea	74LS161A COUNTER --U70,U71
TC0019	2	ea	74LS164 SHIFT REGISTER --U20,U54
TC0020	1	ea	74S240 OCTAL BUFFER --U44
TC0021	8	ea	74LS240 OCTAL BUFFER --U2,U10,U11,U12,U16,U26,U35,U4
TC0022	1	ea	74S241 BUFFER --U46
TC0023	3	ea	74LS245 OCTAL TRANSCIEVER --U43,U55,U56
TC0024	2	ea	74LS280 OCTAL TRANSCIEVER --U31,U64
TC0025	2	ea	74LS390 COUNTER --U67,U69
TC0026	1	ea	74LS393 COUNTER --U13
WC0000	18	ea	TMS4164-15NJ 64K RAM MEMORY CHIP
WC0001	1	ea	FDC9216 DATA SEPARATOR --U5
WC0002	1	ea	MC1488 BUFFER --U60
WC0003	2	ea	MC1489 BUFFER --U59,U61
WC0004	1	ea	75160A BUFFER --U7
WC0005	1	ea	75162A BUFFER --U8
WC0006	1	ea	70-260-370NS & 25-60NS DELAY --U42
WC0007	4	ea	MC8T28P BUFFER --U23,U30,U62,U63
WC0008	1	ea	16 MHZ CLOCK --U66

USUS MEMBERSHIP APPLICATION

(Individual only. Not Corporate.)

JSUS ("Use Us"), the UCSD p-System Users Society promotes and influences the development of the UCSD)-System. USUS also fosters UCSD p-System education and the exchange of UCSD p-System experiences.

The UCSD p-System is a machine-independent microcomputer operating system which provides complete software portability. UCSD Pascal*, assembly, FORTRAN-77, BASIC, Pilot, LISP, and Modula II languages are all available.

JSUS chapters hold regular meetings throughout the United States, the United Kingdom and Europe, where members participate in technical presentations and discussions. USUS also publishes a quarterly newsletter and supports a Software Exchange Library as well as Special Interest Groups (SIGS). (See below for a listing of current SIGS.)

NOTE: USUS is independent of all vendors and is a non-profit organization.

Name and Title: _____

Company: _____

Address: _____

_____

_____

Phone: (_____) _____-_____ ext. _____

____ Do NOT print my phone number on USUS roster.

____ Print ONLY my name and country on USUS roster.

____ Do NOT release my name on mailing lists.

COMPUTER SYSTEM

RAM Memory Size: ____ K-Bytes ____ Disk Memory Size ____ M-Bytes

8" Floppies: ____ IBM ____ Western Digital ____ Other

5" Floppies: ____ Apple ____ Other

Processor: ____ Z80 ____8080 ____ PDP-11/LSI-11 ____ 6502 ____ Apple ____ 6800
____ 9900 ____MicroEngine ____ 8086 ____68000 ____ 6809 ____ Other

Manufacturer/Description: _____

_____

Terminal(s): _____

_____

I am interested in the following Committees/Special Interest Groups (SIGS)

____ Real Time	____ Standards	____ Foreign Affiliates
____ Apple	____ Advanced Planning	____ Users Services
____ CAI	____ S/W Exchange Library	____ Meetings
____ European	____ Commercial Vendor	____ Compatibility & Communications
____ Publications	____ Word Processing	

For one year membership, mail check for $20 payable to USUS, Inc. to:

USUS, Inc.
Secretary
P.O. Box 1148
La Jolla, CA 92038

*UCSD p-System and UCSD Pascal are trademarks of the Regents of the University of California.

SAGE COMPUTER TECHNOLOGY
195 N. EDISON WAY, SUITE 14. RENO, NV 89502 (702) 322-6868

AGE II **PROBLEM REPORT** **DATE**_____

AME _____ SAGE II SERIAL# _____
OMPANY_____ RAM SIZE:_____
DDRESS_____ FLOPPYS: 96 TPI OR 48 TPI?____
_____ TERMINAL:_____
_____ PRINTER: _____
HONE: (____)_____ OTHER: _____
_____

he problem is in [__] SOFTWARE [__] HARDWARE [__] OTHER
hat module? _____ _____ _____
_____ _____ _____

rror message (if any)_____
ystem error (if any)_____--Seg_____P#_____O#_____

his problem [__] is not a problem, but an idea or suggestion.
[__] is a nuisance.
[__] has a serious impact on my project.

lease describe the problem being as specific as you can. Listings and xamples of how to duplicate the problem will help us solve it.

_____

_____

_____

_____

_____

_____

_____

_____

lave you an idea on how to solve the problem?

_____

_____

_____

_____

_____

_____

Check here if additional information is attached: [____]

CABLE WIRING NOTES:

TERMINAL CABLE, TELEVIDIO 925, #CC0000, CC0001, & CC0002

```
SAGE II                    TERMINAL
2 ---------------------> 2
3 ---------------------> 3
7 ---------------------> 7
```

Dummy pins can be inserted in positions 1, 13, 14, 25 if these positions are unused.

TERMINAL CABLE, MULTIUSER, TELEVIDEO 925

```
SAGE II                    TERMINAL
2 ---------------------> 3
3 ---------------------> 2
7 ---------------------> 7
5-->6-->9
```

Dummy pins can be inserted in positions 1, 13, 14, and 25.

PRINTER CABLE, TI, SERIAL, #CC0003 & CC0004

```
SAGE II                    PRINTER
2 ---------------------> 3
5-->20
                           6-->8-->9
6 ---------------------> 11
7 ---------------------> 7
```

Dummy pins can be inserted in positions 1, 13, 14, 25 if these positions are unused.

PRINTER CABLE, QUME, SERIAL

```
SAGE II                    PRINTER
2 ---------------------> 3
6 ---------------------> 20
7 ---------------------> 7
```

Dummy pins can be inserted in positions 1, 13, 14, 25 if these positions are unused.

PARALLEL PRINTER CABLE, # CC0005

The parallel printer cable consists of 3 components. A 34 pin female header connector, a 36 Pin Centronics printer connector and the appropriate length of 34 conductor ribbon cable.

The 34 pin female header connector is placed on the cable such that the pin 1 identifier on the connector is on the same side as the pin 1 identifier on the ribbon cable. Also the connector shall be placed such that the polarizing tabs on the connector will be facing the ribbon cable.

The 36 pin centronics connector is placed on the cable such that pin 1 of the connector is on the same side as and all the way towards the pin 1 identifier of the ribbon cable. This means that pins 35 & 36 of the connector will be left blank. If the cable is laid on the table such that pin 1 is to the right the connector should be installed such that its main body is up.