III. DESIGN CONSIDERATIONS

This section will describe the more significant decisions made in the design of the SCAMP prototype software and the considerations which prompted them.

A. Scamp Software Overview

The SCAMP 1130 APL is a single user system, and as such there is no need to overlap I/O. When the APL Interpreter issues an 1130 XIO, it waits until the I/O is complete. This eliminates the need for a complex software monitor and instead allows the very simple block structure shown in Figure 5. Whenever the APL interpreter requires more input, control is passed to the input software until a character is typed in by the user. The output software is passed control until output has been completed to the display and/or the printer. The cassette recorder software is invoked by the APL interpreter whenever a)LOAD,)SAVE,)COPY,)PCOPY, or)TAPEID SCAMP APL workspace system command is issued (See section III.F. for further discussion on SCAMP system commands). As mentioned previously, the 1130 Emulator is used to provide an architectural match between 1130 APL software and the PALM instruction set.

B. 1130 Emulator

Figure 6 shows a block diagram of the 1130 Emulator written for the PALM host machine. The emulator is passed control by the system initialization routine DATAINIT. Once the emulator has been initialized, it assumes control of SCAMP and only temporarily relinquishs it to various PALM I/O routines. 1130 APL is the data the emulator works on. During the 1130 Instruction Fetch and Decode phase (i.e. ICYCLES), the emulator gets the next 1130 instruction to be emulated and then control is passed to one of several 1130

Instruction Execution routines (i.e. ECYCLES). The 1130 Instruction Execution routine may call the Effective Address Generation Subroutine, used to determine the exact PALM storage address the Instruction Execution Routine is going to use. After execution of the 1130 instruction is complete, control is passed back to the ICYCLES portion of the emulator and the process continues.

The design of an emulator depends upon which features are available in the architecture of the 'host' machine in which the emulator is being written. The ICYCLES portion of an emulator is particularly critical in terms of speed. This portion of the emulator will be executed during the emulation of each 1130 instruction. The SCAMP 1130 emulator is the third 1130 emulator/simulator written at the Scientific Center and the design of ICYCLES has been different in all three cases. Since the last action performed during ICYCLES is normally a multiple way branch to an ECYCLE routine, the most important design consideration is the proper number of possible branch destinations. The tradeoff is emulation speed versus the storage required to emulate the various ECYCLE routines. The first 5 bits of each 1130 instruction indicate which 1130 instruction is being emulated, while the next 3 bits contain the flag and tag fields that pertain to the instruction length and the 1130 index registers used. A 256 way branch is a very fast way of decoding the first eight bits of the 1130 instruction. This is the strategy used for the 1130 simulator described in Appendix A of this report. The SCAMP 1130 emulator branches on only the first 5 1130 instruction bits and then decodes the remaining 3 bits by branching to the Effective Address Generation Subroutine. This was deemed to be the best compromise of speed versus storage.

Key 1130 machine registers are maintained in the PALM registers in exactly the same format as in the 1130, except for the Instruction Address Register (i.e. IREG). In the 1130, the IREG points to the halfword memory location from which the next instruction will be fetched. PALM storage is byte addressable and if the emulated 1130 IREG were maintained in halfword format it would have been necessary to convert the halfword value to a byte value whenever there was a requirement to get the next 1130 instruction.

Occasionally it is necessary to convert the IREG to halfword format when its contents are to be stored in program storage (i.e. as in the case of an 1130 BSI or STX instruction).

Fortunately the 1130 APL interpreter does not address the low order 256 halfwords of 1130 memory (which corresponds to the directly addressable portion of PALM memory), so there is no conflict between PALM and 1130 address space and thus no need to offset 1130 address space from PALM address space. 1130 storage address 0000 corresponds to PALM storage address 0000, 1130 storage address 0001 corresponds to PALM storage address 0002 etc. If an offset had been required, it would have been necessary to add a constant value to each 1130 storage address generated. Not having an offset automatically eliminates any problems in emulating the 1130 storage wraparound feature. The PALM 1130 Emulator is tailored to run 1130 APL and should not be used indescriminately with any other 1130 software.

The SCAMP 1130 Emulator differs from the standard 1130 architecture (5) in the following areas: (1) The 1130 index registers are not maintained in 1130 memory locations 1, 2, and 3; (2) Some additional opcodes were added; (3) Some new I/O commands were added while others are not emulated; (4) no interrupt structure is supplied; (5) The WAIT opcode is interpreted as a no-op; and (6) The load and store double commands do not require the storage address to be on an even word boundary.

The 1130 emulator runs in PALM Interrupt Level 0; on that level PALM registers 0 through 15 have the meanings shown in Figure 7. The 1130 machine state is maintained in PALM registers since this eliminates fetching the information from memory prior to processing it.

The report listed in reference 6 is recommended for those who wish a deeper understanding of the tradeoffs involved in writing an 1130 emulator. As indicated in this report, reflecting index registers into/ memory causes a degradation in processor performance of about 10 per cent. Since the emulator is for a specific 1130 program, system performance was enhanced by modifying 1130 APL not to take advantage of the registers being in memory. Speed was the consideration

for not imposing the normal even word boundary restrictions on the 1130 load and store double instructions.

To compensate for not reflecting the registers in memory, some new 1130 opcodes were implemented. These new opcodes provide the capability that is possible when the registers are reflected in memory. Thus the new opcodes allow for the direct loading of one register into another register or into the accumulator. Registers can also be stored indirectly, and subtracted or added to each other. In addition to restoring old capabilities, a new opcode was added which is similar to the S/370 instruction Move Long (i.e. MVCL). This provided a significant performance enhancement in implementing an APL garbage collection algorithm.

New 1130 I/O commands were added to provide a mechanism for communicating with the cassette recorder. The 1130 APL disk was not used, so there was no need to emulate any disk I/O. I/O emulation was provided for the 1130 console typewriter and printer.

The implementation of an interrupt structure in the 1130 emulator would have required a test at the beginning of ICYCLES, and resulted in a minimum of two more PALM instructions to be executed with the interpretation of each 1130 instruction. This overhead is unnecessary since there is no I/O overlap in 1130 APL and whenever an 1130 XIO command is interpreted, the I/O is completed before control is returned to the emulator at the next 1130 instruction. This eliminates the need for an 1130 interrupt structure and the need for an 1130 WAIT opcode, therefore WAIT is interpreted as a no-op.

Emulation of the other 1130 opcodes is fairly straightforward with the exception of multiply. Since the PALM lacked a shift instruction while the prototype was being developed, a ternary multiplication algorithm was used to eliminate the need for shifting an intermediate product by any amount smaller than 8 bits. As a result of this project a new PALM processor with shift capability is being built. The new PALM will be approximately 2 1/2 times faster than the PALM processor used in the prototype.

The lack of shift capability in the prototype PALM processor led to implementation of a small shift adapter that is addressed like an I/O device. The shift adapter is used for emulating several of the 1130 instructions. This adapter will not be used in the released product due to the new PALM's shift capability.

C. OUTPUT INTERFACE SOFTWARE

1) Display Interface

The 16 line by 64 character display used on the prototype reflects the contents of a 1024 byte display buffer area of main memory. The display adapter cycle steals 8 bit characters from the display buffer and uses them as an index into a display ROS, which contains the dot patterns seen on the screen. Because the display ROS contains only 256 possible display patterns, there are combinations of keys that can be pressed on the keyboard that will not result in the same displayable output as would be present at a terminal. Keying in the following combination 'A' backspace 'B' will result in a composite character meaningless to APL. These meaningless combinations result in an 'error' character being presented on the display. The hardware is capable of displaying legal composite APL characters which are legitimate to other APL interpreters but not to the 1130 Type III version of APL. Also the complete basic character set can be displayed.

The display ROS was designed to be indexed by the internal code representation of characters used in 1130 APL. This avoids the use of two translation tables which would be required if some other representation such as EBCDIC were used. Of course APL overstruck characters are not presently defined in the EBCDIC character set. APL overstruck characters.

A major design decision was to place the newest line of output on the bottom line of the display and 'scroll' whatever was previously on the display up by 1 line. This presents a typewriter output-like appearance. The user knows where to look for the next line of output and he has available on the display the preceding 15 lines. If output to the display had been written from the top line down, it would be necessary either to clear the display after writing the bottom line and prior to writing the top line, or suffer possible user confusion as to the ordering of the information on the screen.

Another major defect of the top-down approach is the difficulty in editing a function line, when the function line is on the bottom display line and the modifications to it are displayed on the top line.

Each input character is displayed as it is keyed into SCAMP. Scrolling of the display occurs whenever the enter key is pressed or a multiple of 64 characters is encountered (i.e. There is no more room on the bottom display line for displaying new characters). The Enter key is used in place of a terminal carriage return key, and results in the display being turned off until APL is ready to display a response to the previous input statement. Since the display is refreshed by cycle stealing from memory, turning the display off allows all the processor time to be used by APL for processing. A cursor character is used to indicate where the next input character will go. Whenever SCAMP is in input mode and the attention key is pressed, all characters displayed to the right and bottom of the cursor are erased from the screen. This is in keeping with the spirit of APL, where an attention on input results in the logical elimination of any input beyond that point. It illustrates an advantage of display over terminal output since terminal output cannot be erased.

2) Printer Interface

The printer, when it is enabled, prints everything that is displayed. Because of the printer mechanics, it is virtually impossible to print a line of output a single character at a time the way a typewriter would. The entire line must be buffered and printed at one time. It was decided to print directly from the last line position of the display buffer and save storage. Whenever a line of output on the display is complete, a check is made to determine whether the printer is enabled, and if it is, the line is printed. The software takes advantage of the printers ability to print from either left to right, or right to left. Prior to printing the next line the printer software positions the print element by determining the present position of the print element,

and deciding whether it requires less time to move the print element to the beginning or end position of the next line.

D. Input Interface Software

1) Overview

The SCAMP prototype keyboard is capable of being physically disabled such that none of the keys will cause an interrupt, with the exception of RESET. Since several function keys were required and they had to be capable of interrupting at any time, the keyboard is never physically disabled. Instead, it is sometimes logically disabled through software. When pressing the ENTER key to signal APL that an input line is complete, the input software logically disables the APL portion of the keyboard and discards any input character other than a special function.

2) Special Function Keys and Indicators

The SCAMP prototype has six defined function keys, these are: Display Hold, Next Line, Display Enable, Clear Display, Printer Enable, and the normal APL function of Attention. There are indicator lights associated with the Display Hold and Printer Enable functions which indicate if these functions have been activated. A third light is used to signal the user that the keyboard is logically enabled and awaiting his input.

Display Hold is an on/off switch which allows for the 'freezing' of display information on a line basis. This allows the user to stop the displaying of information at a desired line so it can be studied at leisure. If a user is displaying a 50 line function and wants to hold the display to look at line 20 through 25, he would push the Display Hold key at the appropriate time and the Display will stop scrolling. Pushing Display Hold again will allow the display to resume scrolling.

Next Line is a function key for use whenever Display Hold is active; it allows the display to be scrolled one line at a time. In the previous example if only lines 20 through 24 were present on the display, pushing Next Line would allow line 25 to be presented.

Pushing Clear Display will cause blanks to be propagated throughout the display buffer; the screen will be blanked and information previously in the display buffer is lost. Any new output will be displayed in the normal manner. Pushing Display Enable will also cause the display screen to be blanked, but without destroying the information contained in the display buffer. The Display remains blank until Display Enable is pushed a second time, which reenables the display with no loss of information. The display buffer is continuously updated regardless of whether the display is enabled or not, this allows the printer to continue operation if it is enabled, since it prints from the display buffer.

Printer Enable is an on/off switch that operates in a manner similar to Display Enable. This function key

allows the user to select information he wants a hardcopy record of. An indicator light is turned on when the printer is enabled.

E. Cassette Recorder Software

The SCAMP prototype uses the same OEM cassette recorder used by the System/7 and a conscious effort was made to follow a similar data format. A common data format allowed tapes to be written on a System/7 and read into SCAMP. This was a significant step in the development process since it provided the ability to write the systems software on a S/370 and transfer the text to SCAMP via a S/7. The use of System/7 as a development tool is covered in Appendix A.

In addition to using the cassette recorder on the prototype for loading systems software, it's also used in implementing the system commands involved with workspace manipulation. To increase reliability, a redundant data blocking strategy is employed for the saving and restoring of workspaces on tape.

(e.g. A A B B .....N N where A,B, and N are blocks.)

Workspaces are saved on tape using 254 halfword blocks with a 2 byte checksum added. Each data block is written twice to allow for error recovery. If an error is detected in reading the first copy of a data block (i.e. there is a checksum mismatch), the second copy of the data block immediately following the first can be read. Experience with the SCAMP cassette recorder has shown it to be extremely reliable and there is a very small probability of finding errors in both copies of a data block.

An alternative to using fixed blocks would have been to write variable sized blocks that contained only one functional item per block. One difficulty with this approach is the need for sizeable interrecord gaps between blocks. As the number of APL variables saved became large, it would result in a large amount of tape being used, and a significant amount of time spent in reading it.

Another advantage of fixed block sizes is the fixed size of the memory I/O buffers used to read and write the blocks. The fixed block size chosen is a compromise between the amount of memory used for buffering and the density of data on the tape.

It was decided to write only one workspace per cassette. There were essentially three reasons for this: (1) If there were more than one workspace per tape the user would be required to wait until the workspace he wants is accessed on the tape, this could run into several minutes; (2) The need for a tape directory and the ability to modify a workspace and then replace it on the tape in its correct position was a problem too complex to be addressed at this time; (3) Tape cassettes are inexpensive (i.e. approximately one dollar each).

F. SCAMP APL

"Design" of the SCAMP APL software consisted of modifying the 1130 type III APL program in those areas which were affected by the differences between SCAMP and the 1130. The relevant differences with regard to APL software included (1) the lack of a disk, (2) use of a tape cassette for offline storage of APL workspaces, (3) APL output being directed potentially to three different units - display, matrix printer, and teleprocessing adapter, and (4) the APL system being developed for a single user rather than multiple users. The work performed in converting 1130 APL to the SCAMP environment will be described below.

1) Disk functions taken over by memory

The 1130 APL code is structured into a number of overlays to enable it to fit into the 16K bytes which was its design point. Since the SCAMP APL code is memory resident, programming for bringing in an overlay was removed. Subroutines which appeared in more than one overlay module were reduced to one occurrence as were data structures and buffer areas which were not needed simultaneously.

The 1130 APL workspace is logically organized into two sections called M-space and F-space. M-space, containing user data and system information, is memory resident. F-space, containing the user's APL functions, is maintained on disk. One disk sector's worth of F-space may be present in memory, but when a different portion of F-space is needed, the sector in memory has to be written to disk and the new one read in. M-space is written to disk when a workspace is saved by the user. The disk allocations for M-space and F-space are fixed at 6 and 26 sectors respectively. In SCAMP, both M and F-spaces are memory resident. The code which pages in sectors of F-space was therefore removed. Also, the sizes of M and F were made explicit variables so that the workspace size could be changed at assembly

time from the original size to a size suitable for present or future SCAMP memory capacity. There is an upper limit on M-space in that it may not exceed 8K bytes. This is due to the 1130 APL code packing M-space addresses into 12 bit units. The removal of this limitation would be a useful although difficult enhancement.

Space management of F-space consists of a garbage collection scheme. Blocks of F-space are allocated from a sequential pool of unused space which is initially the entire F-space. When a block becomes outdated its space is not physically released. When F-space is exhausted, garbage collection is performed to re-claim the inactive blocks. The algorithm used in 1130 APL involves first writing the entire F-space to a temporary area on disk. Then, the list of global names defined by the user is traversed and for each name which is a function name, that function's representation in the temporary F-space is copied into the regular F-space section on disk so it is adjacent to the previous function copied. In addition to user functions, F-space may also contain representations of statements entered by the user to request APL execution. If there are any such statements present in the APL stack, they are copied from the temporary F-space to the regular one. Since SCAMP APL does not have the luxury of using a disk for temporary storage, the garbage collection code was re-written. The difficulty posed by garbage collecting in place is the list of user names did not point to the F-space entries in their physical order. The solution chosen was to add a header to all entries in F-space which was used to link F-space entries in their physical order in F-space. Garbage collection in SCAMP APL consists of traversing this list and moving entries such that there are no gaps. Another item placed in the F-space header is the length of the entry. This is used in conjunction with a MOVE LONG instruction which was added to the instruction set. As a result, moving of F-space entries is much more efficient than it would have been in the native 1130 instruction set.

2) Tape cassette used for storage of APL workspace

In 1130 APL, the user's workspace is saved on disk by copying the entire F-space from the active workspace section on disk to the disk area selected for the workspace being saved, and also writing the entire M-space from memory to the disk area for the workspace being saved. With the switch in storage media from disk to tape cassette, the format of the saved workspace was changed. Presently, four files are written to tape. The first is a workspace descriptor which names the workspace being saved and specifies the sizes of the three files which follow. These subsequent files contain M-space excluding the APL execution stack, the stack, and F-space respectively. In all three cases, only that portion of the workspace component which is presently in use is written to tape.

Use of a sequential storage media for saving the APL workspace presents a problem in supporting the workspace copy command. The basic scheme used for a copy operation is to read the workspace being copied into memory and follow the global name chain until the name of the object to be copied is found. Pointers are associated with the name which will lead to the other components of the object which must be copied into the active workspace. If the entire workspace being copied from could co-reside in memory with the active workspace, there would be no problem, but this is not possible. In fact, only one tape record worth of data is read into memory at a time. The first difficulty is the order of entries in the name chain is not necessarily their physical order. Thus, an entry in tape record n could point to the next chain entry in tape record n-i and this cannot be handled with the SCAMP tape cassette. The problem was resolved by reordering the name chain entries prior to writing them in the workspace save operation.

A similar problem existed with regard to function encoding in F-space. In 1130 APL the line representations of a function do not necessarily

constitute a single physical block. Since this posed a problem for the code which implements the copy command, the format of function entries in F-space was modified to always be a single physical block. The re-organization into the single block is performed whenever a function definition is closed. To facilitate the re-grouping, a section was added to the function header in which the length of each function line is specified.

The code modifications described in this section were made, and the commands which load and save a workspace were re-written to use the tape cassette and new saved workspace format. The copy command was designed, but not actually coded. A new workspace command,)TAPEID, is provided to be used with tape. This command tells the user the name of the workspace on the tape cassette he has mounted.

3) Multiple APL output devices

SCAMP APL output may be directed to a video display, matrix printer, teleprocessing adapter or to some combination of these units. To minimize the impact on APL I/O software of supporting multiple I/O devices, it is desirable to have APL be device independent. The greatest difficulty lies in providing a device independent method of presenting overstruck characters as output. A suitable technique was found in which the character to be output is passed to the emulator interface in a two byte form, where the code for the TP adapter is in the left byte and the code for the display is in the right byte. Thus the APL software need not be aware of what output device(s) are active at any given time.

4) Single user APL

In a single user APL system many of the system commands found in a multi-user system are meaningless. The set of commands selected for SCAMP APL is described below:

a.)SI — list halted functions (state indicator)

b.)SIV — list halted functions with local variables

c.)FNS — list names of defined functions

d.)VARS — list names of global variables

e.)CLEAR — establish a CLEAR workspace

f.)COPY Name — copy the object 'name' from the workspace on tape to the active workspace

g.)PCOPY Name — copy the object 'name' from the workspace on tape to the active workspace if it does not already exist

h.)LOAD wsid — read the workspace on tape into memory making it the active workspace. If 'wsid' is specified, it must match the workspace name on tape for the LOAD to take place.

i.)SAVE wsid — write the active workspace to tape giving it the name specified as 'wsid'.

j.)ERASE name(s) — erase global objects

k.)TAPEID — state the name of the workspace on the currently mounted tape cassette.