UCSD PASCAL with a VAX as a floppy disk

JAMES L. FOX, JR., RAYMOND J. McKNIGHT, and RICHARD V. WOLF
Learning Research and Development Center, University of Pittsburgh, Pittsburgh, Pennsylvania 15260

The increasing sophistication of laboratory microcomputers has brought an attendant need for efficient programming systems. One such system, UCSD PASCAL, has been made commercially available as an adaptable package for many popular microcomputers. This paper addresses the implementation of UCSD PASCAL and describes a powerful extension of the package to use a host timesharing system as a virtual floppy-disk facility.

A few years ago, the typical laboratory microcomputer contained only the essential hardware needed to run an experiment. To the credit of those who used these elementary devices, a great deal of work was done and a large volume of data was collected. Before long, however, it became obvious that the efficient use of these powerful and relatively inexpensive machines was heavily dependent upon the number of support peripherals available to the system. As each new hardware device was added to the system, software had to be written to conveniently use it. These primitive software tools ultimately evolved into the sophisticated microcomputer operating systems available today.

Most of these operating systems are written in the assembly language of a specific microprocessor family, and all of them naturally depend on the type and format of the peripheral hardware with which they are used. This central processor and peripheral hardware dependence severely limits the number of different hardware configurations with which a given operating system will run. At least one system, however, developed at the University of California at San Diego (UCSD), was designed to overcome these restrictions and to facilitate program transportability between many different types of machines.

THE UCSD SYSTEM

All of the UCSD system software is written in a standard, higher level language. With an appropriate compiler, these programs are translated into an efficient intermediate code (pseudocode, or P-code) that is independent of any particular microprocessor instruc-

tion set. All that is needed to implement this software on any given system is a P-code interpreter and an input/output (I/O) handler, both written in assembly language for that system.

Development of the System

The UCSD system was developed initially on the PDP-11 minicomputer. P-code interpreters were then written for specific LSI-11-, 8080-, and Z80-based systems. It is now available in an "adaptable" form for use with most of the 8-bit microprocessor-based systems; support for the emerging 16-bit machines is in the planning stages. A special microprocessor chip set that executes the compiler output code directly is even available. Thus, it is now possible to install this software on a wide variety of totally different hardware systems and have some assurance of software transportability.

Important Advantages

Two major advantages are gained by using this system. First, the software package is complete. That is, it contains all of the system and utility programs necessary for software development without any additional programming on the part of the user. Second, PASCAL was chosen to build the system. Although originally designed to teach students how to program, PASCAL has been found to be very useful as a general-purpose programming language. In addition to being easy to learn, it forces the programmer to apply a structured approach to program design. This leads to an almost self-documented listing that is easily read and understood.

Hardware Requirements

Although all of the system and utility programs can be run on any computer system once the appropriate interpreter has been written, there are certain minimum memory and peripheral requirements that must be met. For example, the adaptable system will run on any microcomputer with at least 48K bytes of read/write memory and one floppy-disk drive. As will be discussed later, the memory and disk requirements may be more or less severe, depending on how the system is used.

This report was supported by the University of Pittsburgh Learning Research and Development Center, which is supported in part as a research and development center by funds from the National Institute of Education (NIE), U.S. Department of Health, Education, and Welfare. The opinions expressed do not necessarily reflect the position or policy of NIE, and no official endorsement should be inferred. For reprints or more detailed information, contact the authors at Learning Research and Development Center, 3939 O'Hara Street, Pittsburgh, Pennsylvania 15260.

Software Requirements

The most important and complex software required is supplied by Softech Microsystems of San Diego, California. Their package includes: (1) an interactive operating system with disk file handling capabilities, run time support routines, and block I/O service routines (PASCAL); (2) a one-pass PASCAL compiler for producing universal P-code; (3) a screen-oriented editor that runs in both programming and text-editing modes; (4) a character-oriented editor for context editing with hard-copy terminals; (5) a file handler to manipulate disk files; (6) a conditional MACRO assembler that produces code that can be linked with PASCAL programs; (7) a linker for link-editing executable PASCAL object code and assembly code modules; (8) a compact interpreter for executing P-code; and (9) a library of program modules and other utilities.¹

The PASCAL interpreter and system bootstrap are provided in machine-specific executable code; the user must provide a device handler package called a "basic input/output system" (BIOS) tailored to his particular hardware configuration, unless he has one of several specific systems for which Softech distributes handlers.

AN IMPLEMENTATION OF UCSD PASCAL

We have implemented UCSD PASCAL in a laboratory environment of diverse applications. In this section, we describe the basic components of our system and discuss some of the implementation details that will be useful to those who eventually undertake a similar project.

Hardware Components

In a laboratory in which 10 or more microcomputers of various sizes and types are used to run experiments, it can be difficult or impractical to equip each unit with a large memory and a floppy-disk drive. As Figure 1 illustrates, our solution to this problem is to utilize our original scheme for storing microcomputer files on a larger host system (Fox, Wolf, & Lesgold, 1979), simulating the function of the floppy-disk drive on our time-shared central computer (VAX 11/780).

Three LSI-11 systems (TERAK 8510/A) running UCSD PASCAL were already in use in our laboratory. Three of our 6800-based systems were equipped with sufficient memory and at least one disk drive, allowing them to potentially support this operating system. Our strategy, therefore, was to install the adaptable system on the 6800 microcomputers equipped with disk drives and to use them and the TERAK as development stations for even smaller machines. The 6800 systems with real floppy-disk drives were then used to transfer floppy-disk images to simulated floppy disks (pseudo-disks, or P-disks) through a serial link to our host machine.

Figure 1. Stylized overall hardware configuration.

Three different hardware configurations are supported. First, a system with floppy disks and no connection to the host can be used to run experiments at a remote site, such as a school. Second, a floppy-disk system expanded by the addition of P-disks can be used for general program development and system maintenance, as well as for running experiments and analyzing data. Third, the microcomputers with a minimum amount of memory and without disks can use any or all of the software available on the larger systems to load and run experiments using the P-disks as a storage device. The minimal memory configuration that will allow the execution of small programs is 28K (not including the BIOS). If smaller configurations are desired, source code purchases from Softech will be necessary.

Software Components

As illustrated in Figure 2, the software components of our system consist of PASCAL, the BIOS, and (optionally) the user-written host disk simulator. Physically, PASCAL and the BIOS execute in the micro and the disk simulator in the host machine. PASCAL communicates with the BIOS through a well-defined "jump table" that is assembled at the top of the BIOS; the BIOS communicates with the optional disk simulator through a series interface. The package as provided by Softech should be installed successfully before the host disk simulator is added.

Figure 2. Stylized memory maps of host and micro systems.

Bringing Up the Basic System

Before describing the extension of the system to include host P-disk capability, a few comments on the installation of the basic UCSD PASCAL system are in order. The programming manual and installation guide provided by Softech are complete and, as far as we have been able to determine, accurate. System installation procedures are provided in cookbook format and, if followed religiously, will result in a running system. The implementor is advised to be particularly careful when coding the BIOS, which is completely specified in the installation guide. If the recipe for these routines is not followed to the letter, the system may lose its mind with no clear indication of what went awry. We learned this the hard way. When the BIOS has been coded, the implementor is led through a procedure to test it through execution of a Softech-provided BIOS tester program. Our BIOS passed the test with flying colors, but the system would not run. Since system source code is not provided with the standard package, our debugging operation was frustrating. After a considerable debugging delay, we discovered that one of our

Figure 3. Diagram of major components of disk handler section of BIOS.

Table 1
Specifications for the Nine Entry Points to the Disk Handler
Section of the User-Written Input/Output Package

SETDISK	The disk drive number is passed to be recorded by the BIOS.
SETTRACK	The track number is passed to be recorded by the BIOS.
SETSECTOR	The sector number is passed to be recorded by the BIOS.
SETBUFFER	The data buffer address is passed to be recorded by the BIOS.
DISKSTART	Performs any motor starting or head loading operations not done automatically as consequences of DISKREAD, DISKWRITE, or DISKINIT operations.
DISKINIT	The floppy drive is reset to its power-up state and its recording head is moved to Track 0.
DISKREAD	A sector is read from the drive, track, and sector specified by previous calls to the above "SET" routines.
DISKWRITE	A sector is written to the drive, track, and sector specified by previous calls to the above "SET" routines.
DISKSTOP	Performs motor stopping and head unloading functions not done automatically after DISKREAD, DISKWRITE, and DISKINIT operations.

handlers was returning an incorrect status code to PASCAL. This delay would not have occurred if the tester program had been more comprehensive. The moral: follow Softech's instructions to the letter and do not trust the BIOS tester program.

If the BIOS is written to function exactly as specified, PASCAL appears to function as advertised.

Extending the Disk Handler

The disk handler can be extended without having access to PASCAL's source code, because all necessary modifications can be made in the BIOS. PASCAL calls the floppy-disk segment of the BIOS through nine entry points (see Figure 3). Table 1 gives a brief description of the Softech specifications of these routines.

The call on SETDISK determines which disk drive is to be accessed by subsequent requests. PASCAL recognizes drive numbers in the range [0...5]. The implementor may arbitrarily designate one or more of these drive numbers to represent P-disks. SETTRACK, SETSECTOR, and SETBUFFER, which specify the physical disk track, sector, and memory transfer address, respectively, remain unchanged in the P-disk-extended BIOS.

The flowchart of Figure 4 illustrates the flow of the DISKWRITE procedure. Note that unless the current drive number has been designated a P-drive, the originally defined diskette handler is executed. The same approach is used to alter DISKINIT and DISKREAD. In short, the BIOS merely sends the information to the host serially that PASCAL sent by subroutine call; status codes and/or data received from the host are returned to PASCAL as though they were from a floppy-disk controller.

Serial communications. There are several subtle design decisions to be made for the serial communications software to function properly and efficiently. A few hints are offered here to cover the problems we solved with our design. Most important, all records (messages) must be packaged and checksummed to ensure data integrity. Even if the serial hardware interface is very reliable, data errors will occur. We chose a simple handshaking scheme in which each record is positively or negatively acknowledged as a function of whether its checksum matches its contents.

Figure 4. Flowchart of DISKWRITE procedure of extended BIOS.

Time-out procedures are built into the BIOS to insure against loss of the communications link. The system will never "hang," since the BIOS receive routines will wait only a finite period of time for the host to begin or complete a response. When an error is detected or a time-out occurs, the previous message is either retransmitted or return is made to PASCAL, indicating an error.

Other problems may arise if the host system translates special control characters. These problems are eliminated if a data coding scheme is adopted that results in records consisting solely of printing characters. In our implementation, a 6-bit compression scheme was used; we transmit four printing characters to represent 3 binary 8-bit bytes. In addition, a repeat format was implemented to minimize serial transmissions when the data is repetitious. If the host system will allow it, all reads from the micro should be made with echo suppressed. With random data, the compression scheme results in one-third fewer character transmissions than the common two HEX digits per byte format. The repeat format allows a 256-byte sector of repeated bytes to be specified with six characters. Performance can be significantly improved by employing these methods.

When the above additions are made to the BIOS, PASCAL will still run properly from diskettes alone. Time-outs will occur when PASCAL makes DISKINIT requests of the P-disk drives, but this merely results in off-line status being established for these drives. We found that PASCAL makes frequent requests of all six drives, so the implementor may wish to build in the ability to suppress serial message transmission when the P-disks are known to be off-line.

Switching User Context

To utilize the P-disk capability of the system, the user should have dialogue access to both the UCSD PASCAL system and the host system. If this facility is not provided, the friendly environment provided by PASCAL may be lost when host interaction is required. Two things are necessary to facilitate this dual-system access. First, both systems have to be accessible via the local terminal, and second, a means of user-controlled context switching must be provided.

Access to the PASCAL system is automatically provided by the BIOS console handler. Access to the host is provided through a dialogue/loader routine. When using the dialogue facility, the microsystem becomes no more than a terminal on the host system. Through this terminal-like behavior, the user can access the host system and start the host disk simulator to establish a usable communications link between the two systems.

The dialogue/loader routine mentioned above has been described in detail by Fox et al. (1979). Briefly, it blindly passes characters between the host and the microsystem using the host serial line or, after a certain

cue, interprets the character stream from the host as object code to be put into the microsystem's memory. Our extended UCSD PASCAL system uses two of these routines. When the microsystem is started (e.g., from power up or RESET), it uses a ROM-resident (i.e., the program is burned onto a read-only memory, ROM, chip) dialogue/loader to communicate with the host. However, once the system has been loaded, a dialogue routine embedded in the BIOS console handler is used.

Upon trapping a special character, the BIOS console handler uses its dialogue routine to carry on a character exchange between console and the host serial line. Exit from this exchange is made upon receiving a complementary special character from the host or keyboard. This signaling scheme allows the user to switch context at will for full access to both systems, and without the necessity of reloading the PASCAL system.

Host Disk Simulator: the VAX 11/780 as a Floppy Disk

The P-disk software package developed for our VAX 11/780 is a small FORTRAN program having two avenues of access and two corresponding levels of operation. Refer to the diagrams of Figures 1 and 2 and the flowchart of Figure 5 for a better understanding of this section. Using the microsystem's ROM-resident terminal dialogue routine, the PASCAL user can LOGIN to the host system and issue a RUN command string to start the floppy-disk simulator program. This start-up causes the program to run at its upper, or interactive, level of operation. At the upper level, the user is provided a menu of commands to simulate the handling of floppy disks, get a help display, exit the program, or transfer control to PASCAL in the microsystem. The second means of access is through the BIOS disk handler for the PASCAL system. This access method reaches the host software at its lower level of operation, where it behaves like a floppy-disk controller. It is at this lower level that most micro/host communication and data transfer takes place.

At the host's interactive level, P-disks can be created, mounted, and dismounted by the user. These operations are analogous to putting a label on a disk, inserting a disk into a drive, and removing a disk from a drive. Each P-disk is simply a random-access data file on the host's disk system. Creation of a P-disk is the creation of a file whose attributes match those of a floppy disk for the microsystem PASCAL. That is, the file is declared to have a maximum size (i.e., number of blocks) and the records are defined to match the sector size of the microsystem's floppy disks. A command to "mount" a specific P-disk causes an existing file to be opened for use; a "dismount" command causes the file to be closed. Appropriate error checking is performed to handle possible errors returned by the operating system and to deal with user mishaps at the keyboard.

When all necessary P-disk "handling" has been done on the host, the user can transfer control (switch

Figure 5. Flowchart of host disk simulator program.

context) to the microsystem by issuing either a LOAD or a RETURN command. If the PASCAL software has not yet been loaded into the microsystem, the LOAD command is appropriate. The LOAD command queries the user for the existence of real disks on the microsystem. If disks exist, the host software downloads a very short record to the microsystem that causes it to load PASCAL from its own floppy disks. If there are no real disks on the microsystem, the host downloads the PASCAL bootstrap from a file on its system to the microsystem. The RETURN command is used to simply return to the microsystem. This method of transfer assumes that the PASCAL software has already been loaded and is executing the dialogue embedded in the BIOS.

Once control has been transferred to the microsystem, the host software begins operating at its lower level and opens its second access point. At its lower level of operation, the host software is in communication with the microsystem's PASCAL software and the PASCAL software is, in turn, in communication with the user. Normal operations can then be performed on the P-disks, through PASCAL, as though they were the real disks. All read, write, and initialize commands for the P-disks are routed through the BIOS to the lower level entry point in the host software, and appropriate information and data are returned to PASCAL. Because this interaction with the host is taken care of by the BIOS, as far as PASCAL is concerned, local disks are being used.

The user, while in PASCAL on the microsystem, still has access to the host by way of context switching. When a switch is requested, a "dialogue request" message is sent to the host. This message is received and processed at the host software's lower level, and control is returned to the upper level, where the user can again interact with the host software.

The entire host program could be written in standard FORTRAN. However, certain system service facilities and some assembler codes were used to enhance the performance of the host software. These enhancements yielded a slight increase in operating speed, and a small decrease in the load on the VAX operating system.

On the whole, the extension of the disk handler and the host software allow our VAX to behave completely as a floppy-disk drive on the microsystem. The VAX does not interfere at all with data formats, directories, or file structures of the PASCAL system and, as a result, is completely transparent to PASCAL.

Loading PASCAL Without Floppy Disks

As mentioned at the outset, one of our primary goals was to generate a PASCAL system that would execute in a stripped-down microcomputer without floppy disks. Once the system is running as described, only trivial tasks remain to allow this facility.

The first step is to create a P-disk image containing all PASCAL system files necessary for system bootstrapping

and execution. This is accomplished "in system" using the well documented FILER, which is provided with the system.

The primary bootstrap is described in detail and an example listing is provided in the installation guide. This is a small program that merely reads the PASCAL secondary bootstrap into memory and executes it. The primary bootstrap is located at a fixed address on the diskette on Drive 0. All that is required to load PASCAL from a P-disk is to provide the host disk simulator with a loadable image of the primary bootstrap. The host disk simulator downloads this file when the user elects the LOAD option described above. Unless the implementor has elected to install the BIOS as a permanent memory chip (ROM), it will be necessary to include a loadable BIOS image in this file. When the primary bootstrap and BIOS are memory resident and executing, the system will boot itself through the transparency of the extended drive capacity of the BIOS. One bit of inelegance is necessary to fully accomplish this. We have assumed the inability to modify the PASCAL system; therefore, since it insists on Drive 0 for system file access, Drive 0 requests must be mapped to the appropriate P-drive number by the BIOS when no floppy disks are on-line.

System Performance

Table 2 gives the times required to boot the system under various hardware configurations and character transmission rates.

System throughput and general responsiveness are proportional to the speed and efficiency of the hardware employed. The reader may correctly surmise that a PASCAL system loaded over a 30-character/sec phone line will not blind the user with its speed. The initial system load delay can be eliminated completely by putting the PASCAL interpreter and the BIOS onto a permanent ROM chip. However, even at slower transmission rates, the one-shot system boot is tolerable and application program runs are limited only by output volume.

We implemented the P-disk concept specifically to support limited-configuration micros. Therefore, the intolerable delays that would result from large-program segmented procedure swapping do not apply, since we

Table 2

Hardware Configuration	Baud Rate	Time*
Terak: LSI-11/Disks	N/A	0:10
MSI: M6800/Disks	N/A	0:35
M6800	9600	1:25
M6800	4800	2:54
M6800	2400	5:32
M6800	1200	8:55

Note—The difference between the two disk systems is due not only to the difference in processor speed, but also to the slower disk controller on the 6800 system. No special arrangements were made to measure these times. They were taken during typical midafternoon weekday activity. *Minutes:seconds.

would not attempt to execute such programs in a purely P-disk environment. The ability to perform all PASCAL functions using P-disks occurred naturally as a result of the transparency of the host/PASCAL interface. Although we will not routinely utilize this total facility, it allows us to back up diskettes in a secure environment and provides us an emergency procedure in the event of floppy-disk failure.

FUTURE PLANS

A definite performance improvement to the P-disk system described above would be the implementation of the BIOS/host communications through a direct memory access (DMA) device rather than the serial line described. Although this approach requires a more significant investment in host software, the BIOS modifications would be similar.

A more interesting idea is that of implementing UCSD PASCAL as a user program on our VAX time-sharing system. If the proper licensing arrangements can be made with Softech, we would like to customize the interpreter, bootstrap, and BIOS of the LSI-11-adaptable system to make it executable within the VAX/VMS timesharing environment. If the "disk handler" of this program is adapted to access VAX direct-access disk files in lieu of floppy disks, the resulting disk files would be none other than P-disks, which are accessible from our micro-based UCSD PASCAL systems. All of the UCSD PASCAL operating system features that were retained in the previously described micro/P-disk environment are also retained in the timeshared "image" of the system. The immediate benefit brought about by this program would be that anyone with access to our VAX system could develop UCSD PASCAL programs without having to schedule time on one of our floppy-disk-equipped microcomputers. Users could edit, compile, link, and in many cases, execute UCSD PASCAL pro-

grams that could later be executed in one of the microsystems. Of course, execution of PASCAL programs dependent on micro-based hardware or including linked assembly code foreign to the VAX could be executed from an appropriate microsystem only.

SUMMARY

As computer hardware costs continue to decline, the cost of software development looms larger. Until recently, an inexpensive laboratory microcomputer required an investment of time worth several times its original cost to program it for a relatively simple task. The UCSD PASCAL system provides small computer users with a software development tool that significantly reduces the ratio of software to hardware costs. Implementing the P-disk extension to this system provides the additional advantages of secure, organized data storage, the ability to execute sophisticated programs in inexpensive machines, and program transportability. The initial software investment to implement the system is not prohibitive, due to the availability of the basic system that we have described.

REFERENCE

- FOX, J. L., WOLF, R. V., & LESGOLD, A. M. Microprocessor uses in a timeshared facility. *Behavior Research Methods & Instrumentation*, 1979, 11, 271-280.

NOTE

1. This package description was adapted from a Softech Microsystems (San Diego, California) sales brochure (1979). It is supplied on 8-in IBM 3740-compatible diskettes (single density, single sided, soft sectored). RK05 cartridge platters are also available for DEC PDP-11 and LSI-11 systems.