DOCID: 3928859 Approved for release by NSA on 12-01-2011, Transparency Case# 63852 

# The UCSD p-System

STATUTORILY EXEMPT

*This paper discusses the UCSD p-System, an operating system for small computers developed at the University of California at San Diego. The discussion includes the overall system, file managing, editing, and programming in Pascal on the system.*

## INTRODUCTION

The UCSD p-System was developed at the University of California at San Diego to support Pascal programming on microcomputers. Similar to MS-DOS, the p-System is an operating system for small computers but is, in many ways, very different. The p-System is written in Pascal and now supports not only Pascal, but also FORTRAN, COBOL, BASIC, and Modula-2.

The concept was to have an operating system that, once it was implemented on a machine, allowed any program written under that operating system to be truly transportable from computer to computer. That is to say, the p-System compiler would not actually translate the program into a language that was specific for, say, an 8088 chip on the IBM-PC, but rather would translate it into a "pseudo" language that, when used with an operating system designed for the PC, would run correctly. Similarly, if the operating system were implemented on a Digital Equipment Corporation (DEC) computer, this same pseudo code would still work properly with no modifications.

The particular version of UCSD p-System tested was written for the IBM-PC and requires two single-sided double-density disk drives and at least 128K of memory.

## OPERATION

### *Startup*

A user's first introduction to the p-System could possibly be described as overwhelming. It comes in a good-sized box containing five books and six disks. What are you going to do with six disks, you say? You've only got two disk drives! Read on, things aren't so bad.

The books themselves come in the typical IBM format, roughly 8 ×9½ inches in the usual grey slipjackets. Furthermore, they are *very readable*. To give the reader more encouragement after the initial shock of seeing five books and six disks, it is only necessary to read half of one of the books to become a useful programmer *and* only two disks are really necessary (see, aren't you glad you read on?). The various disks are used depending on what sort of accuracy you are

striving for in your numerical calculations. UCSD Pascal will support up to 4-byte accuracy.

Probably the most important manual to read is the *Beginners Guide for the UCSD p-System*. It will advise you to put the *Startup* disk in the left drive and the *Pascal* disk in the right. The names are printed on the disk jackets. You then bring up the system in the usual manner (ALT-CTRL-DEL).

Some introductory remarks will come up on the screen with some exercises to get the user familiar with the use of the p-System. A prompt at the top of the screen advises the user to push **Q** to leave the tutorial mode and enter the Command mode.

## USING THE OPERATING SYSTEM

### *Command Mode*

In contrast to MS-DOS, where you are simply shown `A>` on the screen, the UCSD system provides you with a prompt as to what options are available to you. This prompt will always appear across the top of the screen. You will be shown

```
Command:  E(dit, R(un, F(ile, C(omp, L(ink, X(ecute, A(ssemb?
```

The operating system is telling you that in the command mode you can press **E** to edit a file, **R** to run a program, **F** to look at your list of disk files, etc. If you are a little hesitant to push the buttons, the users manual gives short, well-written explanations of what each button push will do.

If, for example, you push the **E**, the editor will look for a file with the special name SYSTEM.WRK.TEXT. If the file does not exist, the system will, once more by a series of prompts at the top of the screen, give you the option of creating the work file, editing a file of your choosing, or, by pushing **Q** to quit, get you back out of edit mode and into the Command mode.

### *The Editor*

If your only experience in editing program files has been with EDLIN, the IBM program editor which comes standard with MS-DOS, you will be very pleasantly surprised. This editor is much quicker and more versatile. It can handle up to 700 lines of code, all of which is kept in main memory. The net result is that any command you choose to give is executed almost immediately. Often by the time your finger is off the key, the command has been executed.

Once you have indicated to the editor that you wish to edit a program and whether it is SYSTEM.WRK.TEXT or some other file name, your program will be read in and you will be given a series of prompts at the top of the screen:

```
Edit:  A(djst C(py F(ind I(nsrt J(mp R(place Q(uit X(chng
```

Because your entire program is already in memory, these commands are performed amazingly fast.

If you are writing a new program, you simply push **I** which will get you into the **I(nsrt** mode and you can start typing. If you already have a program and have recalled it to make some changes, you use the cursor controls, the **J(mp** command, or the **F(ind** command to locate the area in which you want to make your changes.

When you choose to edit your program, the first 20 lines are displayed on the screen. If the area that you are interested in changing is on the screen, the easiest thing to do is simply move the cursor to that area and then push the **I(nsrt** command. Note that the text will immediately part for you, with a large blank area suddenly appearing to the right of the cursor. You may then type in your changes.

If the area needing to be changed is toward the end of the program, then from the Edit mode you can push **J(mp**. A prompt will appear at the top of the screen asking where you wish to jump to, beginning or end. If you push **E**, the last 20 lines of your program will appear on the screen as fast as you can take your finger off the key. You can then push the cursor controls to move your cursor to exactly the position that you wish to make changes and then push **I**. Once more you will be in the insert mode.

Another option is the **F(ind** command. If you push the **F** from the edit mode, you will be prompted for the string of characters that you wish to find. You can type in a string and, once more, almost as fast as you have completed typing, you will have a 20-line display with the character string in question at the center of the screen.

There are many more options available to allow you to go immediately to any area of interest in the program, and the enjoyable thing about this is that there is not a lot to remember. The options are always displayed at the top of the screen. If you have a question regarding an option, refer to the User's Manual which explains the options very well.

Current theories of good programming require that you do a certain amount of indenting. This editor supports that very well. When you do a carriage return, the cursor will come to rest directly under the first character of the line above. If the line above started in column 5, for example, the new line will also start in column 5. If this is unacceptable, you can always use the right or left arrow to adjust the cursor.

Suppose you have inserted a few lines of new text but have decided that all of the lines should be moved to the right by, say, two spaces. All you have to do from the **E(dit** mode is to move the cursor to the first line that should be adjusted to the right. The cursor may be positioned anywhere on that line. Then push **A(djst**, getting you into the adjust mode, and then the right arrow two times. The line that the cursor was on will move over two spaces to the right. Then push the down arrow. Every line that the cursor is positioned at will also immediately move two spaces to the right. When you are through adjusting, you push **Q(uit**, which will get you back to the **E(dit** mode.

When you are through editing, you simply push **Q** to quit the editor. You will immediately be given the option of (1) saving your file as SYSTEM.WRK.TEXT or some other name that you may give the file, (2) returning to edit mode (you might have pushed **Q** by mistake), or (3) exiting without saving anything (you might be really disgusted with your program).

*The Pascal Compiler*

After leaving the editor, you will be back in the Command mode, once more with the prompt at the top of the screen advising of possible options. If you have just written a program, the next option might be to push **C** to compile. Your compiler will be read in, and it will go out looking for your SYSTEM.WRK.TEXT file to compile. If you choose to save it under some other name, the compiler will ask what file you want compiled. Once you type in the name, compilation begins. The screen will continually advise you as to what procedure, function, or segment is being compiled as well as what line it is on. If (alas) there is an error, the compiler will stop and tell you of the error and the options available. Typically your options would be either to continue compilation or to make the change. If you wish to make the change, simply push **E** which returns you to the edit mode with the cursor at the position where the compiler found the error. Simply correct the error, push **Q** to get out of edit mode, and initiate compilation once more. Very simple.

The compiler is amazingly accurate at finding errors. It is usually able to point to the exact location and is rarely more than a word off.

After the program has successfully compiled, simply push **R** to start it running. Once more, the prompts on the screen are quite clear.

Other options are possible from the command mode. Pushing **F** for files will give a number of options that can be done from the filer. You can, for example, rearrange the files, add or delete them – the usual things that one would expect to be able to do on a personal computer. The unusual thing is the small number of keystrokes required to perform these operations.

*The Pascal Language*

The authors of the UCSD system have done a very good job of taking an excellent programming language and making it even better. The creator of the Pascal language, Dr. Nicholas Worth, wrote a language to teach students good programming techniques. He was not very interested in I/O techniques and, for the purposes of his language, assumed that all input would be via card reader and all output would be to a tape. Therefore, sequential I/O was all that was needed. Unfortunately(?) the IBM-PC has no idea what a card reader or tape drive is. The UCSD modification allows for random access of files, addressing the screen, string operations almost as easy as those in BASIC, and many other modifications needed to make Pascal into a useful language for a microcomputer.

*Observations*

The authors designed this system to be friendly to the programmer, and it would seem that they have succeeded admirably. Moving around through the operating system and the various utilities is indeed fast, and very rarely is the user unsure of what his next move should be.

The documentation in all five manuals is excellent. The Beginner's Guide serves as an orientation guide for people first learning to use the system and as a reference manual for these same people once they are familiar with the system.

There is also a Reference Manual, which can be used by someone who knows Pascal and is interested in learning the differences in the p-System version. The User's Guide provides a very good description of how to use the operating system. The Internal Architecture Guide describes the internal design of the UCSD p-System: the P-machine, Operating System, basic I/O and the way in which these elements are organized to support the running of a program written in UCSD Pascal. The Assembler Reference Manual describes the UCSD 8086/88 Assembler.

There were some bad points, however. If, in the course of a programming session you do continual edits and recompilations, your disk space will become very fragmented. You may have enough room for your program but not in contiguous locations. The operating system does not allow for fragmented programs in the manner that MS-DOS does. Your program must be in contiguous locations, or you will not be able to get out of the edit mode without abandoning your program. You can get around this by inserting a formatted disk into your B disk drive and writing the program to that disk and then going back and rearranging your fragmented disk. This, to say the least, is rather inconvenient.

There is not a great deal of software support for this system. If, for example, you need a communications package that will go on a p-System diskette, your choices will be limited.

The programs run slow. Although no serious attempts at benchmarking were done, a large program was written using p-System Pascal. This program had a sorting routine in which a very large amount of data had to be manipulated. This particular routine took 45 minutes. The programmers and potential users found this to be unacceptable. The program was then rewritten under MS-DOS using Microsoft Pascal. This same routine then took 5 minutes. This certainly is not a definitive speed test, but there is some indication that speed of execution should be a consideration if one is thinking about using the p-System.

## CONCLUSIONS

All in all, the price is steep ($700.00), but you do get an operating system, a very good program editor, and a Pascal compiler. The operating system is not as sophisticated as MS-DOS causing the user to quit his editing every so often to rearrange the disk.

This version of Pascal was, as mentioned, modified so that interactive programs can be written. The potential user should balance that with the fact that when the programs are written they will run as much as 10 times slower than a standard compiled Pascal, and that in the process of developing large programs, you will constantly be switching disks.

For my particular application, I chose to stick with Microsoft Pascal and the Vedit program editor.