TRANSPORTABILITY OF SOFTWARE APPLICATIONS ON MICROCOMPUTERS

W. Porter Welbourne, M.D.
418 North Main Street
Penn Yan, New York 14527

The problems relating to porting microcomputer software from one machine to another are becoming increasingly important as the technology matures and the range of hardware options broadens. This paper describes the p-System (tm) operating system features which make it an excellent choice for microcomputer applications when transportability and conservation of software investment are important.

The choice of microcomputer "hardware" is seemingly limitless at the present time, with a new microcomputer being introduced weekly. However, production of software to support this ever expanding small-computer market is complicated for both the user and developer by the fact that there is no industry-wide agreement as to the best host microprocessor to use in a desktop computer. In fact, it is unlikely that there will ever be such agreement in this rapidly expanding and competitive field. The question of which brands of microcomputer and which operating system(s) will dominate in the next few years has become a hot topic in the industry.

Because the dozen or so most popular microprocessor chips have incompatible machine instruction sets, software developed for one microcomputer has not generally been transportable to another without considerable effort. Thus, the programmer and the end-user alike are limited to one hardware configuration (or at best one microprocessor series) for software development and distribution.

Since computer software is "highly-skilled" labor-intensive, conversion of programs to run on many different machines is both difficult and expensive. Several approaches to this problem have been taken and are discussed below.

Language Standardization

There have been two major approaches to the problem of transportability of software in the past. The first was high level language standardization, and the second was to choose an operating system which would run on a large number of microcomputers.

Language standardization is an attractive idea from a theoretical standpoint, but more difficult from a practical one. Language implementations often approach programming problems from different viewpoints, and often handle certain functions differently even within a particular language. Anyone who has tried to contend with the hundred or so dialects of BASIC can plead for standardization to no avail.

In those higher level languages where some standardization has been achieved (such as FORTRAN and Pascal), the problem remains that new compilers must be written for each new hardware environment. Writing a high level language compiler, even a standard one, is a difficult and time consuming task for the systems programmer. Because of this, it is often a year or two between the introduction of a new microprocessor and the availability of a standardized language compiler for the new instruction set.

The end-user of software may still be unable to transport his programs since transporting via standard language compilers assumes access to the original source coding for the program, a valuable commodity which he may not possess.

The Operating System Approach

The second means which has been used to achieve portability is to use a standard operating system. The current widespread use of the CP/M operating system on the first and second generation of microcomputers, and its temporary status as the "industry standard" for about two years, gives evidence to the success of this approach.

One set of language compilers (standard or not) can be developed under the operating system. Programs developed under the standard operating system, often including the compilers themselves, may then be transported with little difficulty to any microcomputer which can be made to run that operating system. Source code is not necessary to transport software under the same operating system environment in most instances.

But most operating systems like CP/M, even though they can be made to run on dozens of brands of microcomputers, will not work on more than one family of microprocessor chips. Thus, the user is locked into one generation of machines (in the case of CP/M, the 8-bit 8080, Z-80 generation). Conversion for use with the 6502 eight-bit processor used in the Apple II, or the 8088 processor found in many IBM PC compatible machines requires a new operating system and a new compiler.

MS/DOS seems at present to be the new "standard" for the 8088/86 chipset, but many proclaim that the higher performance of the Motorola 68000 processor will make the UNIX and UNIX-like operating systems a dominant force over the next few years. But choosing any of the three "standards" (CP/M, MS/DOS or UNIX), limits portability to a great extent.
The Design of a Portable Operating System

All microcomputers use some type of low level software to control the execution of "higher level" applications programs. Normally this low level software is written in the "assembly language" specific for a given microprocessor chip. Since it is processor specific, it is not usually transferable to another microprocessor.

Over the past decade, a new concept in the design of this low level, or operating system software has evolved. The basic idea is to make the operating system itself portable to new microprocessor architectures.

Through a project initiated at the University of California at San Diego (UCSD) under the direction of Dr. Kenneth Bowles, and continued commercially by SoTech Microsystems, the p-System has emerged as the first truly portable and universal software development system for microcomputers.

A portable operating system needs a way to isolate applications programs and the majority of system software programs from the specifics of the host computer. Additionally, software components which must be processor specific should be easily adapted to differing instruction sets.

The group at UCSD did this by designing a hypothetical microcomputer instruction set (called pseudo or p-code) consisting of an assembly language instruction set optimized for executing high level languages (the p-machine). They then wrote a high level language compiler (for Pascal) to generate object code for this hypothetical pseudomicrocomputer. The rest of the operating system software, including file handler, editors, utilities, etc., was written in Pascal and compiled into p-code (the native code for the p-machine). The p-code could then be executed by a low level p-machine emulator written in the native assembly code of the actual microprocessor host.

Once a compiler for the p-machine was available, the Pascal compiler itself was rewritten and compiled into p-code. With all of the operating system converted to p-code, the entire set of software could be transported to any host processor with a p-code emulator (sometimes called a p-code interpreter). Likewise, any applications programs or even additional compilers for other languages could be easily moved to different microprocessors once a p-code interpreter was installed on a given machine. Currently, compilers are available for Pascal, BASIC, and FORTRAN-77 under the p-System.

Implementing the p-System

Object p-code is designed to originate from any of the source language compilers written for the system. As long as a program compiled into p-code has no native assembly code elements attached to it, it is totally isolated from the host computer, and may be executed without recompilation in object code form on any microcomputer which supports a p-code emulator or interpreter.

Writing a p-code emulator for a new processor is a task measured in man-months, rather than in man-years, as is the case with writing compilers. After a p-code interpreter has been written for a given microprocessor series, a set of interface protocols (known as the basic input-output system, or BIOS) needs to be written to install the interpreter on a particular microcomputer's peripheral system of disks, terminals, printers, and so on. The task of writing these disk driver and screen handler routines is done in the assembly code of the host processor, but again is a task measured in days or weeks.

Once a p-code interpreter and a BIOS (together referred to as a runtime or 'turnkey' support module) have been completed, the entire operating system as well as all applications programs may be immediately transported to the target microprocessor and executed, often without change or modification.

Thus, if a software developer or an end-user has chosen the p-System environment initially, transporting his software to a different microcomputer will involve a good deal less time and effort.

In the ideal case, a p-code runtime system will already exist for the target machine(s). The only expense is purchase of the runtime system from a commercial supplier, and the only effort involved is that required for media conversion (uploading or downloading the software). Properly written applications may be fully transported across the full spectrum of eight and sixteen bit microprocessors in an afternoon. (As an example, the author has compiled a program on a Radio Shack Z-80 based 8-bit machine, and transferred and executed the object code on an Apple II 6502-based 8-bit machine, an IBM PC 8088-based 16-bit machine, and a Corvus Concept 68000-based 16/32 bit machine without touching a line of code or even recompiling the source code.)

There presently exist p-code interpreters for nearly all of the microprocessor chips on the market, including the 8080, Z-80, 6502, 6809, 8086, 9900, 68000, HP-65, LSI-11, and the VAX. There are commercially available p-code runtime systems for most of the popular microcomputer brands including the Apple II, Radio Shack, IBM (on the PC/XT and the Displaywriter), Digital Equipment Pro 350 and Rainbow, the TI Professional, Hewlett-Packard, Commodore, Victor, Altos, Sage, Terak, Wang, Olivetti, Osborne, Columbia, Zenith, NCR, Phillips, Toshiba, Corvus, Spectravideo, Northstar, Data General, and many others.

If a p-code runtime system is not commercially available, but the interpreter is, an experienced programmer may have to spend several days to several weeks writing a BIOS for the target microcomputer. In the worst case, the target microcomputer has a processor for which there is no p-code interpreter. Then a new p-code emulator must be developed along with a basic I/O system. This task generally takes
between two and nine months, depending upon the similarity of the native instruction set to p-code and the expertise of the programmer.

Given any of the above situations, the total time and investment needed to transport software across the range of hardware configurations is a small fraction of that required to transport programs by any other existing method.

Implications of Object Code Portability

The ability to transport software in object code form means that the software developer can produce and distribute one version of a program for dozens of different microcomputers. But more than that, he can do so on a single microcomputer. Thus, the total cost of his computer hardware and his software development and maintenance is minimized.

The end-user can preserve his software investment, since in many cases he needs only the specific p-code runtime system for the target machine to transport and run his programs. He does not need access to the source code. And because the entire operating system is designed for portability, chances are good that he will be able to move his programs to newer, more efficient microprocessors in the future.

Since the file and directory structures of the p-System are identical for all computers, it is quite possible to share a single copy of a program among several different types of microcomputers using shared hard-disk subsystems. The advantages of this feature to local area networks (LANs) is obvious, and will become more apparent as interest in LANs develops in the future. One is not locked-in to one brand of microcomputer on a network. In fact, work is proceeding on a transportable p-System based networking protocol. Scheduled for release later this year, this p-System net is intended to permit transport of programs across different hardware network architectures (Arcnet, Ethernet and Omninet) as well as across different microprocessors.

The Universal Medium Concept

The problem of media conversion, that is, of converting from one computer's diskette size and format to another has been a nuisance for anyone who has faced the problem. Transfer of code, text, or data files over an RS-232 serial line or modem is tedious and requires special equipment and software.

The p-System developers have addressed this problem directly by adopting a standard 'universal' diskette format for 8" and 5 1/4" diskettes. A utility program is provided with most p-System software which temporarily reconfigures the computer's programmable disk controller circuitry to read or write the standard formats. The diskette files may then be transferred to a diskette format optimized for the specific machine.

In this way a diskette created on an IBM Displaywriter may be read directly by a Radio Shack Model 4 or a DEC LSI-11 running the p-System. Likewise, a p-System diskette made on a Radio Shack Model 4 may be inserted into an IBM PC and read directly. No other operating system can approach this degree of portability.

Dynamic Memory Management

Another problem commonly encountered when transporting programs between microcomputers involves the differences in available memory space for program execution. Most microcomputer operating systems provide a fixed amount of RAM (random access memory) with which to work. If one wishes to write an application which effectively utilizes 128 K of RAM, he must give up the hope that his program will run in the 64 K RAM limit imposed by 8-bit processor architectures. This clearly can limit portability.

The p-System uses program segmenting and dynamic memory management to overcome this type of problem. The ability to break large programs into partitioned overlays provides a means for running programs which are much too big to fit in the available RAM of the host computer. The operating system automatically fits as many segments of a program as possible into the available memory space, while the rest of the segments are stored on disk. When segments are needed that are not in memory, the system "swaps" inactive segments out of RAM back to disk and loads the needed ones. In fact, the operating system itself is segmented, and will dynamically remove unused portions of system software if program execution requires additional space.

A large program will run more slowly on a 64 K RAM machine than on a 128 K machine because more swapping is necessary. The important point is that the "virtual" memory capability of the p-System works to optimize the memory available while preserving portability. Hence, very large programs may be constructed under the p-System, even for a computer limited to 64 K of main memory.

An additional construct of the p-System is that of separately compiled program "units". These are utilities which are stored in the system library and which may be used commonly by any p-code program. An example is a screen control unit which controls terminal cursor movements. The specific ASCII codes for arrow keys may differ with the hardware, but the specific control sequence is recorded and stored in the "screenops" unit found in the target machine's library. An application program needs only to send out the standard instruction for 'up-arrow'. The screenops unit in the system library does the conversion to the correct hardware code. In this way a program can be made to execute correctly even if the original author never ran the software on the target microcomputer.

Once again, the system design for transportability works in a very practical manner.

Speed versus Space versus Portability

The p-code for a given program is considerably smaller than the corresponding native code for most microprocessors (in the range of 25% of the size),
which means that p-code lets store more instructions at once. While it is true that the translation through p-code often produces a somewhat slower execution speed than does native code, it often is nearly as fast since more instructions are present in memory. For most applications, the speed differences are not significant.

When performance is a critical consideration in the software execution, the p-System provides the option of trading object code transportability for improved execution speed with a set of "native code generators". These can produce processor specific native code for any of the procedures within an application program.

The programmer can designate native code for those portions of the program which must run extremely fast, and can leave the rest of the program in compact p-code. Native code generation destroys compatibility across machines with respect to object code, but it does not affect p-code portability. If the p-code is moved to a new microprocessor, it can be run through the native code generator specific to that processor as well.

Problems with the p-System

The p-System addresses the major problems of transportability among microcomputers quite well. Device directories and text files are readable on all microprocessors running the p-System. Object p-code files are executable on all supported microcomputers as well. Screen handling, printer support, etc. are handled at the operating system level, and need not be addressed by program developers within their applications.

There are some data file portability problems that remain unsolved, however. Data which is not stored in an ASCII textual representation is generally not transferable between processors. Thus, data files containing real numbers or long integers cannot be converted for use on another microprocessor without effort.

Many p-System programs aim at the "generic" microcomputer. Implementations often neglect certain hardware "bells and whistles" which may have attracted the user to a specific brand of computer in the first place.

As the p-System has matured, the RAM space required to support the operating system has increased, leaving less space for program execution. From a practical standpoint, a minimum of 64 K RAM is necessary to run even a trivial application, and disk drives are mandatory. This eliminates many low end computers presently being offered for the home.

As the RAM space available for program execution decreases, so does the speed of execution, since considerable "swapping" of program and operating segments between disk and RAM is necessary. This can make a very large application unacceptably slow.

Presently, the system can effectively use only 128 K of RAM because the p-machine is a hypothetical 16-bit stack machine. The remainder of the RAM in large systems is usually allocated to a RAM disk emulation. A p-machine based on 32-bit architecture is under development, but will not likely be released before mid-1984.

A shortcoming for some types of medical applications is that the p-System is not ideal for handling complex file constructions, nor for extensive text handling in the same manner as MUMPS for example. It is probably a poor choice for artificial intelligence applications as well, again because of the design constraints of the p-System.

The release of the p-System advanced file system (AFS) in the fourth quarter of 1983 may correct at least some of these complaints.

Summary

The design of the p-System as a vehicle to achieve greater portability of software applications in the microcomputer world has largely been successful. Despite some remaining concerns, for a sizable percentage of users and developers, the p-System provides an excellent development and execution environment that is transportable across the entire spectrum of the microcomputer industry.

References: