Implementing the UCSD PASCAL System on the MODCOMP Computer

T. Wolfe
DSN Data Systems Section

This article describes the UCSD PASCAL system developed by the University of California, San Diego, now available on the MODCOMP computer. The system includes a Pascal compiler and many useful utility programs. A BASIC compiler and a FORTRAN 77 compiler are also available. There is currently a large amount of software available written in UCSD PASCAL, including a data base system, word processing systems and a MODULA compiler.

I. Introduction

The UCSD PASCAL* System is a complete interactive software development system consisting of an operating system, compilers (Pascal, BASIC, FORTRAN 77), two text editors (screen editor and line-oriented editor), a linker and many useful utility programs. The system is written and maintained in Pascal. The compiler generates code for an idealized processor known as the "pseudo-machine." The "pseudo-code" (p-code) generated by the compiler is interpreted at runtime by a program (known as the interpreter) which emulates the pseudo-machine. Thus, the operating system, compilers, editors and all user programs are executed using the same interpreter. By writing a new interpreter, the complete programming system may be moved to a new computer.

Because the Pascal and FORTRAN 77 compilers generate the same p-codes (i.e., use the same interpreter), Pascal programs may call FORTRAN subroutines and vice versa. The system also contains an adaptable assembler which is easily modified to produce assembly language for a new computer. Assembly language subroutines thus produced may also be called from either Pascal or FORTRAN. Currently the assembler has not been modified to produce MODCOMP machine code. A system library program may be used to build libraries of useful subroutines and the system linker used to link them to user programs.

Interpreters currently exist for Z80/8080, PDP 11/LSI-11, 6800, 9900 and 6502 computers and are being developed for other computers. Thus, software could be developed on an 8080 micro or any existing system and run on a MODCOMP.

An interpreter for the MODCOMP II and the MODCOMP IV computers has been written in MODCOMP II assembly language. The complete Pascal programming system has been run successfully on a MODCOMP II and MODCOMP IV under both the MAX II/III and MAX IV operating systems. A copy of the system was sent to the Oak Ridge National Labs and was installed and running within an hour.

The interpreter requires approximately 4.5K words of memory and a work area. The minimum memory requirements

*Trademark of the Regents of the University of California.

for the complete system is 16K words plus the interpreter. The maximum is 64K words.

An error was discovered in the MODCOMP teletype handler and several patches for both the MAX II/III and the MAX IV operating systems were recommended by MODCOMP. These patches are documented and available to anyone who wishes to implement the system.

A large portion of the University of Tasmania's Pascal validation suite has been run by JPL, and several errors in the interpreter were discovered and corrected.

II. The Pseudo-Machine

The UCSD PASCAL program development system is an interpreter-based implementation of Pascal. The compiler generates code for a pseudo-machine which is emulated by the interpreter at runtime. The pseudo-machine has a stack architecture with a stack being built from high memory downward (Fig. 1). The interpreter sits in low memory with the heap being built upward from it.

Large programs that normally would not fit into memory may be partitioned into segments, only some of which need be resident in memory at a time. User programs may be partitioned into a maximum of 7 segments, each with up to 127 procedures or functions. During the running of a program each segment is loaded onto the stack as it is needed and then discarded. The first segment to be loaded upon executing the Pascal system is segment zero of the operating system. It contains all the IO functions for the system and user programs and is never discarded from the stack.

The Pascal system has available to it several IO units which may be assigned to MODCOMP logical units. In general it has a console, printer and up to 6 Pascal disk units. (A Pascal disk unit is a MODCOMP disk partition.) Each Pascal disk unit has a directory and may contain up to 77 Pascal files.

A special IO unit (unit 0) is also available to the user. This unit performs a read or write via a simple REX service call. In this manner the user may read or write data not in the Pascal format.

In addition there are three other Pascal IO units which are undefined at this time. They are GRAPHICS, REMIN and REMOUT. These could be defined and an appropriate driver written into the interpreter.

III. Design of the Interpreter

At the start of the project there was no documentation available for the design of any of the existing interpreters. The implementer was also unfamiliar with the MODCOMP computer. The source code (with very poor to nonexistent comments) for an 8080 microcomputer version of the interpreter written by UCSD was available. This source code was used as the design for the MODCOMP interpreter. A mapping of the functions within the 8080 interpreter into MODCOMP II assembly language was the method used to code the interpreter. This method allowed the implementer to learn how the system was designed and to code the interpreter simultaneously. It also allowed the implementor to become familiar with the MODCOMP. The method worked rather well but would have been easier if the source code were properly documented. For example, the carry was used as a flag in various widely separated places in the code. No comments indicated when the carry was set, what it meant or where it was used.

The 8080 interpreter is in reality two separate programs. The first loads the interpreter and initializes the Pascal system and the second is the interpreter. These were combined into one program for the MODCOMP interpreter.

The only interface between the MODCOMP operating system and the Pascal system is REX service calls. This allows the Pascal system to use the flexibility of logical file assignments within the MODCOMP operating system and yet maintain its independence.

The MODCOMP interpreter has three major functional areas. They are:

(1) The BOOTER, which initializes the work space and loads the Pascal operating system. It is executed only once at the beginning of the interpreter.

(2) The BIOS (Basic IO System), which performs all IO operations for the Pascal system.

(3) The p-code interpretive routines which make up the bulk of the interpreter. Program control cycles through these routines as various programs are interpreted.

In order to reduce complexity and the time needed to code the interpreter, the floating point intrinsic routines such as SIN, LN, COS, LOG, FLOAT INTEGER, etc., were taken from the MODCOMP FORTRAN library verbatim. P-code interpretative routines which perform these functions make calls to these routines.

All IO operations in the BIOS are done by IO unit number only. The Pascal operating system maintains all necessary

parameters for reading or writing files by name, all buffering, and directories on all disk units. The BIOS receives only a unit number, number of bytes, a memory address and a disk block number if a disk unit is being accessed.

In the BIOS the unit number is used to index into a jump table and the appropriate IO driver executed. Each type of IO unit has its own driver, depending on its characteristics. For example, each character sent to the printer is saved in a buffer until a carriage return; then the complete line is printed out. All characters sent to the printer are also converted to uppercase because the standard DSN MODCOMP printer cannot handle lowercase characters.

IV. P-Code Conversion

The interpreter would be useless without p-code programs to interpret. The UCSD PASCAL system consists of approximately 20 programs, all in p-code form. These programs were transferred from an 8080 microcomputer to the MODCOMP and converted. Three special programs were needed to convert the byte-sex (see below) of the p-codes to run on the MODCOMP. It should be noted that none of the standard system programs use floating point. This would require a different type of conversion.

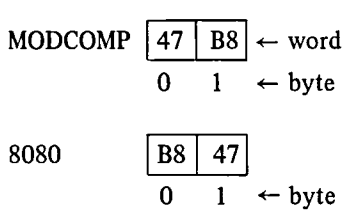
The 8080 micro and the MODCOMP use different ordering of bytes in memory (different "byte-sex"). The difference is in the way the two computers store 16-bit integer values in memory and is due mainly to the 8080 being a byte machine and the MODCOMP being a word machine.

The two different ways of ordering bytes in memory are:

(1) Byte ZERO is the byte containing the least significant half of the word. Byte ONE contains the most significant half.

(2) Byte ZERO is the byte containing the most significant half of the word. Byte ONE contains the least significant half.

For example: Storing the hex value 47B8.



This requires all integer values in the p-codes to have their byte-sex changed when run on the MODCOMP. It was decided to write conversion programs rather than to change the byte-sex at runtime in the interpreter to avoid slowing the runtime execution speed.

As shown in Fig. 2, two programs are necessary to change the byte-sex of a Pascal disk unit to a form usable by the MODCOMP. Each Pascal disk unit contains a directory and files, only some of which may be code files.

The first program (DSKMOD) converts the disk directory and the segment dictionary of all code files. The second program (CODMOD) converts the p-code segments of all code files or any non-code files containing p-codes. For example, the operating system (SYSTEM.PASCAL) is not a code file, although in all other respects it looks like a code file. Library files are also noncode files, but contain code segments.

A third conversion program was also found necessary to convert the op-code data file (OPCODES.II.0) used by the p-code disassembler program. Conversion is not necessary if you never use this program.

It should be noted that these conversion programs are not needed unless the user is transferring a code file from the 8080 micro to the MODCOMP.

A copy program, called appropriately enough "COPY," was written to simplify the transfer of the Pascal system from one MODCOMP computer to another. It copies a MODCOMP disk partition containing a Pascal disk unit (directory and files) to another disk partition or tape. This simple program and the interpreter are all that are needed to install the system on a new MODCOMP computer.

V. System Software Problems

Two system programs were found to have problems. Both were corrected by making small changes in their Pascal source code and recompiling them.

The program PATCH is used to display and modify memory directly. The program displays memory in reverse order for the 8080 micro. Fortunately, the source code for this program contained comments indicating what to change to correct this problem.

The second program with a problem was the system linker (SYSTEM.LINKER). This problem was due to the design of the program. It was designed with the architecture of the 8080 micro in mind. The code was therefore not machine-independent Pascal. Several patches were added to the code, modifying it to work on a word machine. No effort was made to redesign the program.

VI. Conclusion

The UCSD PASCAL programming system is now available on the MODCOMP II and MODCOMP IV computers. It provides a useful program development environment for its users. Using the interactive editors, compilers, linker and libraries, software in Pascal or FORTRAN 77 may be developed quickly. The ability to develop software on a microcomputer and transfer it to a MODCOMP has the large advantage of relieving the overworked MODCOMP computers from software development work.

Follow-on activities should include modifying the assembler to produce MODCOMP machine code and looking into code generation and optimization from the p-code.

References

1. *UCSD PASCAL System II.0 User's Manual*, Institute for Information Systems, University of California, San Diego Campus, March 1979.
2. *UCSD PASCAL System Synchronous Input/Output Subsystem Implementation Guide* (Release Level II.1 Preliminary), University of California, San Diego Campus, 10 April 1979.
3. Bowles, K. L., *Beginner's Guide for the UCSD PASCAL System*, Byte Books, 1980.

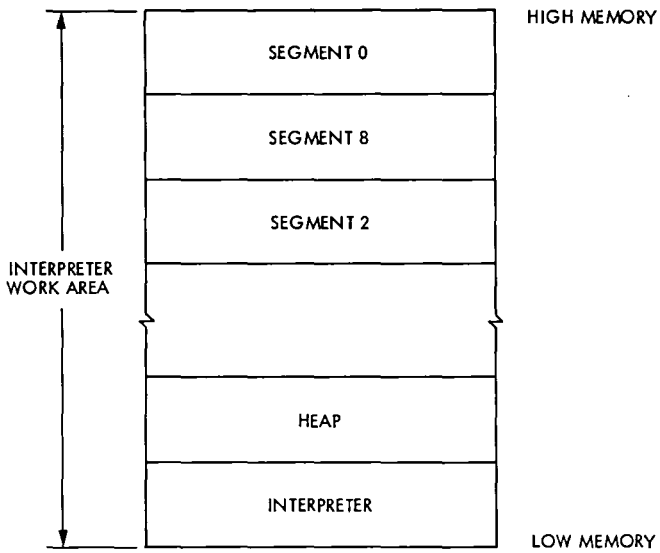


Fig. 1. The general configuration of memory. Segments are loaded into memory as they are needed

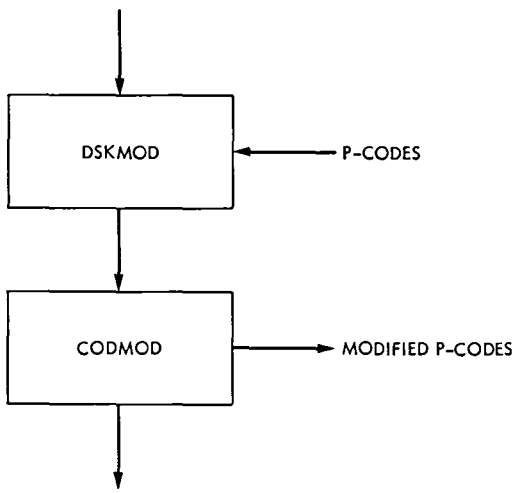


Fig. 2. Flowchart showing the general flow of data and programs necessary to change the byte-sex of a Pascal disk unit containing p-code files