# PROBLEMS WITH THE IMPLEMENTATION OF STANDARD MUMPS

Jerome C. Wilcox and Richard F. Walters

University of California at Davis
Department of Human Physiology, Davis, CA 95616

This discussion focuses on the implementation of MUMPS in multi-language environments, rather than considers only dedicated systems. There are two reasons for this choice. First, the passage of Standard MUMPS at the recently concluded MDC (MUMPS Development Committee) meeting makes a discussion of the difficulties which we encountered with our implementation of the Standard timely. Second, the paper by Sias and Covert (1975) describing an implementation of Standard MUMPS under the UNIX operating system (Ritchie and Thompson, 1974) prompts us to complement their report with a discussion of our own efforts.

As long as Standard MUMPS is being implemented on a machine totally dedicated to MUMPS and MUMPS is, in fact, the operating system, the Standard can be implemented in its entirety. However, within the environment of a general operating system, there are restrictions which may cause problems in implementation of the complete Standard. Sias and Covert show that it is possible to implement MUMPS on a minicomputer under a general operating system; hence implementation problems may exist outside of the large computer environment with which we are most familiar. In both situations, modifications of the operating system to support MUMPS are impossible; the implementor is forced to accept certain features provided by the operating system. These include control of user access to the system itself, conventions for input/output, terminal handling, file access, and security. As an example, the Standard originally contained references to specific characters from the terminals being used to control specific actions. These specifications were replaced by more general ones when it became known that the characters selected were used for other purposes by the operating systems of several large computers.

Additionally, in the multilanguage environment, the user community is extremely diverse, there is competition for resources of the computer and, quite often, users must pay for the resources used. Of course, the simplest solution to the problem is to buy or lease a MUMPS "machine", but in many university and corporate systems this is not always practical. Increasingly, restrictions are being placed on computer acquisition for specialized purposes, forcing the use of existing equipment. Many institutions also permit or require sharing of a data base in a manner facilitated by having easy access to languages and programs other than MUMPS. Whatever the reasons, many users are finding it necessary to turn to a MUMPS implementation under a general-purpose operating system. At the University of California, Davis, we have been involved in three implementations of Standard MUMPS on large mainframe computers: the IBM System/360 and System 370, the Burroughs 6700 and the DEC PDP-10. In each of these implementations certain problems have arisen requiring compromises with Standard MUMPS. These problems may be

categorized into three groups: conceptual difficulties, impossible features and incomplete specifications.

Conceptual difficulties are those features specified in the Standard which do not readily fit within the general operating system environment or within the specific hardware. The most obvious of these is the specification in the Standard of the use of the ASCII character set, as defined by ANSI X3.4-1968. It is important to realize that ANSI standards are voluntary, not mandatory standards. In the computer arena, minicomputer manufacturers have generally adopted the ASCII standard, while large mainframe manufacturers have not, preferring to use BCD, as on the CDC computers, or EBCDIC as on IBM and Burroughs equipment. Users of these large mainframe computers are generally familiar with the BCD or EBCDIC character sets and find them comfortable to use. Conversely, they are not at ease with ASCII and would prefer to be spared the unnecessary mental effort of switching character sets. Although ways may be found to operate in simulated ASCII on each of these machines, these techniques frequently introduce a high level of system overhead, increasing the execution costs. The practical effects of this problem involve portability of routines, both in terms of interchange and in terms of use of the $ASCII and $CHAR functions, and the Follows and Pattern Match operators. In general, interchange is not a problem since ASCII media may usually be both read and written. However, use of the above-mentioned functions and operators does create a problem.

Paralleling the use of ASCII, MUMPS, being an operator-loaded language, relies quite heavily upon the use of all of the punctuation characters of ASCII. Many of these characters do not exist on other devices. For example, the IBM 2741 terminal does have the caret, the backslash, the left bracket or the right bracket characters. It was therefore necessary to find substitute characters which could be used. This was done, but again the user must learn to adapt.

A final example of conceptual difficulty is illustrated by the file-handling capabilities of MUMPS. The portability requirements for MUMPS specified a string length of 255 bytes. This choice may be adequate for internal string processing, but when MUMPS is to be used for file processing it is totally inadequate. We have files with record lengths in excess of 7,000 bytes, and would like to find some way in which to process these files using MUMPS.

In addition to these conceptual difficulties, there are several features defined in the MUMPS Standard which are either impossible or extremely expensive to implement. The most obvious are the timeouts specified as part of several commands. These features proved impossible to implement on the System/360. They can be implemented on the B6700 but only at an unreasonable cost. On that machine, programs which use the timer features do not run as normal timesharing jobs; rather, they are run as batch jobs which communicate with terminals. This has two impacts on the user. First of all, there is an additional 10% surcharge imposed by the accounting

system for jobs of this type at our campus. Second, severe response time degredations may occur as the job is under the working set manager of the operating system. This feature may thus result in response delays as long as ten minutes, clearly unacceptable to the user at the terminal.

An additional problem is posed by the single-character read, as specified by the Standard. On most timesharing systems, a message discipline is imposed by the operating system, usually requiring some sort of end-of-message character before any of the transmitted characters are seen by the object program. This characteristic makes implementation of the single-character read impossible. Also, a large number of the CRT terminals now available make use of local buffering of input so that editing and error correction may be done prior to transmitting any part of the message to the computer. Since the author of a MUMPS program may have no idea what type of terminal is to be used, inclusion of the single character read is a serious threat to program portability.

The final class of problems to be dealt with consists of those created by incomplete specifications in the Standard. Thus far, we have encountered two areas in which such problems occur. The first is the general area of input/output. In most cases, the entire problem may be summarized by stating that the Standard lacks a complete and cohesive specification of input/output in the MUMPS language. Partially, this lack stems from the failure to consider computers other than dedicated minicomputers as vehicles for the MUMPS language. Even most important, however, is the extreme difficulty of completely specifying input/output in a computer language. Most other languages which have been standardized have chosen deliberately to omit specification of I/O in this reason. It is our opinion that this is not a viable approach. Instead, we believe that the difficult task of specifying complete I/O syntax and semantics must be undertaken to avoid problems. As an example, consider the ambiguity of the meaning of the MUMPS READ and WRITE commands when the device addressed is other than a terminal. The commands are non-reflexive; that is to say READ A,B is not necessarily the reverse of WRITE A,B. The difficulty in this case stems from the lack of definition within the Standard of what constitutes a record from an I/O device.

The other area in which we have found the Standard lacking is that of mathematical functions. Although we have seen a continual growth of the MUMPS language in terms of non-medical uses, as evidenced by the larger sections devoted to such papers in recent MUG meetings, the Standard has not kept pace. Even though floating-point arithmetic was included in the Standard, functions to make use of this capability were not included. These features must be included in each implementation if the value of the MUMPS language outside the restricted medical community is to be increased.

In conclusion, we would like to emphasize that we are not offering these constructive comments about the MUMPS Standard in a negative vein. On the contrary, the University of California, Davis, is proud to have been involved in the effort of standardization, and we are committed to continue

efforts to develop the language. We do feel, however, the issues which we have raised must be faced at this time. One of the major reasons for undergoing the time-consuming and difficult task of developing a standard language was to enhance portability of MUMPS applications. Clearly, each implementor is going to have to address some or all of the problems mentioned in this paper. In most cases, the only solution permitted by the Standard is use of the implementation specific commands, functions and special variables. If each implementor does solve the problem in this manner, there is a very real danger that we will end up having had little impact on the existing situation. Instead, we will merely have created another dozen or so MUMPS dialects, each masquerading as "Standard MUMPS" and each equally as non-portable as the dialects which existed prior to standardization.

ACKNOWLEDGEMENTS

This work was supported in part by NIH Contract PHS NO1-CL-4-2183.

REFERENCES

Ritchie, Dennis M. and Ken Thompson, "The UNIX Time-Sharing System", Communications of the ACM, 17, 7, pp. 365-375, 1974.

Sias, F.R., Jr. and J.R. Covert, "A New Implementation of Standard MUMPS in a Multi-Language Environment", these proceedings, 1975.