Trends in Microprogramming: A Second Reading

LOUISE H. JONES AND RICHARD E. MERWIN, SENIOR MEMBER, IEEE

Abstract—Microprogramming is a rapidly emerging technology for the implementation of the control section of modern digital processing systems. An assessment was made of this approach to control unit design three years ago in a series of papers derived from the Third Annual Workshop on Microprogramming and published as a special issue of this TRANSACTIONS. This technology has matured in the ensuing three years, and an updated assessment has been prepared again as a special issue of this TRANSACTIONS based upon the papers presented at the Sixth Annual Workshop. The extent of realization of past predictions is reviewed and current trends are discussed. Finally, an attempt is made to assess future developments that may be expected to occur in the applications of microprogramming.

Index Terms—Applications, automata, compilers, control storage, microprocessors, microprogramming, minicomputers, operating systems, programming languages.

INTRODUCTION

THREE years ago, the July 1971 issue of the IEEE TRANSACTIONS ON COMPUTERS was devoted to microprogramming. The papers in that issue were largely drawn from the Third Annual Workshop on Microprogramming, held in October 1970 at Buffalo, N.Y. A perspective and overview [1] of microprogramming, prepared by Profs. Michael J. Flynn and Robert F. Rosin, the Guest Editors of that issue, addressed the then contemporary aspects of microprogramming, and suggested possible future trends. In the past three years, applications of microprogramming have increased markedly, and there is promise of an even greater increase in the future. On the other hand, certain aspects of this technology have been dormant. This second special issue of the IEEE TRANSACTIONS ON COMPUTERS contains a selection of the papers presented at the Sixth Annual Workshop on Microprogramming, held in September 1973 at College Park, Md., under the sponsorship of the ACM Special Interest Group on Microprogramming. These papers reflect the current status of microprogramming, and provide a revised perspective on microprogramming as a technique for the implementation of the control function in digital computers.

Both the current status of microprogramming, including an assessment of the degree to which earlier expectations for this technology have been achieved, and the current trends suggested by recent papers in the field are discussed in this Foreword. In addition, as in the Foreword [1] to the previous special issue, predictions of future trends which can be expected as this technology continues to mature are made. The prediction process, with its inherent fallibility, can only indicate rather general and perhaps obvious future trends in microprogramming. Its value stems mainly from stimulating the readers to make their own assessment of the future, and thereby to bring about the reality.

STATUS OF MICROPROGRAMMING:
PREDICTIONS FROM THE PAST

Although microprogramming technology has fulfilled many predictions made several years ago, a number of trends have developed which were not predicted, and still other trends have failed to materialize. The application of microprogramming techniques, including microdiagnostics, in the implementation of control circuitry of digital machines has been widespread. IBM has employed this in the design of the System 370 mainframes and input–output controllers, and most other manufacturers have added microprogrammed machines to their product lines. The minicomputer manufacturers have exploited the name in reference to their basic machine language programming, and have urged their customers to customize their hardware to special applications through “microprogramming.” The trade journals contain frequent references to the use of microprogramming by the “plug-to-plug compatible” manufacturers to tailor their hardware to a particular mainframe interface. New applications are being found to use microprogramming to implement such activities as display tube character generation, signal processing systems, customized graphics systems, interrupt and hardware fault processing, and implementation of compiler and operating system functions.

While the application of microprogramming in computer systems is increasing, the major manufacturers of microprogrammed mainframes are still resisting the expected trend to user microprogramming of their equipment. This resistance takes the form of restricted access to current microprograms and the use of special input systems for microprograms. There is no intention to supply support to the user brave enough to attempt to microprogram his application. This has caused some distress in the academic community where the reticence of the manufacturers to reveal their mainframe implementations and microprogramming techniques has effectively stifled what had been expected to be a major trend in this field. The few user microprogrammable machines that are available, such as the Burroughs B-1700, the Micodata machines, and the Digital Scientific Corporation Meta-4,
have started to provide the academic community with the tools required for research in microprogramming [2].

The position of microprogramming in the mushrooming computer science curriculum remains obscure. Academic computer scientists tend to consider microprogramming as a topic best subsumed in a hardware design or programming course, and evidence for this lack of interest is the absence of textbooks devoted to this subject. The only book [3] on microprogramming was prepared over four years ago and needs revision, but it still remains the only choice for a textbook in this area. Although some universities offer special courses in microprogramming and most universities include it somewhere in the computer science curriculum, the subject remains relatively submerged in the academic landscape. The ambivalence of the academic community toward microprogramming has retarded the development of the theoretical foundations for microprogramming and microprogrammable processors. Papers, such as that by Ramamoorthy and Shankar [4], dealing with the important issue of the proof of correctness and equivalence of certain classes of microprograms have started to appear only recently. Similarly, analyses, such as that by Thomas [5] on alternative computer organizations for providing user microprogramming, and that by Hoefel [6] on the conditions under which a nontrivial interpreter will reduce the space and time complexity of program evaluation in a two-phase processing system, are the first of their type. Only a few papers [7], [8] have even attempted to define the central problems of microprogramming and microprogrammable processors, and there has been little work linking microprogramming to automata theory, combinatorics, or computer architecture. No technical journal has a section devoted exclusively to microprogramming. The only publication devoted to microprogramming is the SIGMICRO Newsletter, which is published by the ACM Special Interest Group on Microprogramming and contains original and reprinted technical articles on this subject as well as reviews of papers presented at conferences or published in various journals.

Another reason for the slow growth of theoretical studies in microprogramming is the cloak of secrecy held over this technology by the major computer manufacturers. Unlike the details of compilers and operating systems which were developed openly and cooperatively by manufacturers, the academic community, and user organizations, details of microprogrammable hardware design and implementation are completely concealed. Only one description of a microprogramming design automation system [9] has appeared, and it provides only the operational concepts. Results of the studies that have been made by computer manufacturers to develop compilers, assemblers, and control storage load modules have not been revealed. Although microprogrammed functional memory [10] was one of the major topics of interest at the Third Annual Workshop on Microprogramming, no further work on this development has been reported. The reticence of manu-
The CDC STAR 100 adopted a microprogramming approach for control of the streaming unit, as shown in Fig. 2. An initial implementation with conventional logic circuitry proved unwieldy in trying to manage the nearly 100 cases of interrupt processing required for the streaming unit. In the microprogrammed design, a sequence of microinstructions located in control storage directs resolution of the interrupts. These sequences are selected from 1536 control words, each with 224 bits, which are issued every 40 ns.

Once the decision had been made to control interrupt termination and recovery, it was determined to enhance the STAR microcode processor to provide for control of all vector/string instruction initiation and shutdown, regardless of the presence of interrupts. In addition, certain control lines and a special interface to the STAR Maintenance Station were provided to permit the use of the microcode processor for diagnosis in depth of portions of the stream unit.

The microcode processor is extremely simple and reliable, and has the capability of controlling functions at the 40-ns clock rate of the STAR ALU. The control storage is writable from the STAR maintenance station if one possesses the key to the write lockout switch, and thus many elaborate diagnostics or system performance codes can be loaded for special uses of the machine to investigate its design integrity and potential speed. The microcode processor is disabled during the execution of STAR scalar instructions. The microcode processor, further, takes no part in the actual control of data movement or the floating-point arithmetic, except in a couple of unusual and infrequently encountered cases.

In both of the applications of microprogramming noted above, the simplicity of design and inherent operational speed possible with semiconductor control storages provided the rationale for selecting the microprogrammed approach. As large machines grow more architecturally complex, microprogramming is a logical choice to implement the control function.

In summary, many of the predictions of several years ago have been fulfilled, several have not, and some new trends have developed which were unexpected. The use of microprogramming in a wide range of applications by a large number of manufacturers is well established, even though the advent of user microprogramming has failed to materialize. The paper by Agrawala and Rauscher [16] describes the current status of microprogramming and gives some perspective to the current problems of microprogramming, such as lack of easily used languages for horizontal microprogramming. It should be noted that the paper by Ramamoorthy and Tauchiya [17] is a contribution toward a resolution of this problem. In spite of the wide acceptance of microprogramming by industry both in the United States and in Europe (an International Advanced Summer Institute on Microprogramming was held at St. Raphael, France in 1971), the American academic community has remained relatively aloof to the discipline. Finally, it is interesting to note that many of the players have changed; few of the contributions to the Sixth Microprogramming Workshop were made by participants who also contributed to the Third Workshop.

CURRENT TRENDS IN MICROPROGRAMMING: DIRECTIONS FOR THE FUTURE

Several current trends in the applications of microprogramming can be identified. Briefly, these trends include microprocessors, language interpreters, operating systems, and replacement of support software with microprogrammed hardware. Because of the proprietary nature of some of these activities, the literature is sparse, but enough is available to indicate clearly the role microprogramming is to play.

Microprocessors are an outgrowth of the tremendous strides being made in the technology of semiconductor large-scale integration (LSI). A processor on a single silicon wafer is a reality today, and the exploitation of this technology is soon to begin. Most microprocessors
employ microprogrammed control units [18] since they are easy to implement in LSI and can be easily changed if a different control sequence is desired. LSI chips, while inexpensive to manufacture, are expensive to design, and the use of complicated control logic instead of a simple writable control storage unit would be impractical. The developers of microprocessors envision arrays of these devices with the possibility of making rapid changes in the functional characteristics of each processor as the application varies. The future of this technology appears promising, and microprogramming will certainly play a significant role in the evolution of these systems.

Programming languages have in the past been supported with extensive compilers which require large storage space and consume considerable processing capacity while compiling. Direct interpretation of either higher level programming languages or of intermediate languages can be implemented in terms of primitive machine operations through microprogramming. This approach eliminates compiler storage requirements and reduces processing time for programs in a debug stage. In principle, optimization [12], [19] can be built into a microprogrammed interpretation. It is likely that direct interpretation of programming languages by means of microprogramming will appear in the next generation of computers as a natural extension of the microprogrammed interpretation of the machine instruction set.

Operating system programs are a major component of all large computer centers. Microprogramming of operating system functions offers a convenient way to reduce conventional storage requirements for this function and provide enhanced performance. The Venus Machine [18] is an experiment in the direction of using microprograms to interpret an instruction set with primitives which allow definition of a simple operating system. The ISPI System [19] is another experiment in which a microprogrammable processor permitted concurrent design of a language, operating system, and machine architecture. The appeal of this approach is obvious, and it is reasonable to expect major portions of software to be shifted into firmware in future systems. It is quite likely that the next generation of mainframes will employ microprogrammed interpretation of operating systems, including both input–output control and language translators.

Another possible trend which is an outgrowth of LSI technology is to replace operating systems and language translators with inexpensive hardware. The attraction of this approach is the replacement of software with less expensive hardware which can be easily modified in function through microprogramming. The customer is never concerned with operating system functions which are buried in the hardware and modified by maintenance personnel. The SYMBOL [22] proposal is indicative of this approach, and is just one more example of the changes in computer architecture that may result from the availability of cheap hardware.

The current trends toward implementing language interpreters and operating systems in microcode may generate increased interest in microprogramming from members of the academic community. Current studies on defining various classes of computing tasks in terms of corresponding classes of program schemata can be extended to the study of classes of microprogrammable machines which can effectively interpret these schemata. Similarly, work on the representation of languages and processors by means of the Vienna Definition Language can be naturally extended to the study of the relationships between the Vienna objects representing programming languages and the microprogrammable machines which interpret them. The study of distributed computing systems can be extended to the study of networks of microprocessors; this work has already been started [23]. In addition, as noted by Flynn and Rosin [1], there is no effective theory of computer system construction; the architectural flexibility afforded by microprogrammable systems may provide the basis for some of the experiments needed to develop such a theory. It is possible that attempts will be made to extend the current work on structured programming into the hardware area, and that criteria for implementing computer systems which will permit realization of certain basic control constructs at many levels will be developed. Finally, we can expect to see quantitative studies of the basic microoperations allowed by a given data flow structure, the minimal resources needed to execute certain tasks, and the efficiency with which various tasks can be executed in a variety of resource environments.

While the reader may have reservations as to whether many of the above trends can be realized, there is evidence in the literature that work is under way in all of these fields. The major computer manufacturers are not revealing their product strategy, but the word “microprogramming” is being mentioned frequently as playing a major role in the next generation systems. The almost wholesale adoption of microprogramming by the minicomputer manufacturers and the near necessity of the use of this technique by the developers of microprocessors provide assurance that the impact of this technology on computer developments will increase considerably.

THE FUTURE IMPACT OF MICROPROGRAMMING

Predictions of future trends in microprogramming are not likely to be any more reliable than the recent predictions of U.S. economic growth have been. Nonetheless, a few conjectures are offered which are based upon the current trends noted above. These predictions stem from the expected impact of future hardware and software developments and how these will affect the role of microprogramming in computer technology. Finally, to spare the reader, our predictions will be brief.

The two main forces molding the future developments in computer technology are the emergence of LSI and the necessity for the automation of programming. If the
The present rate of increase, the demand for programmers will increase twentyfold by 1985, sufficient trained personnel will not be available, and the present trend of cost of software exceeding that of hardware [24] will accelerate. Automation of software generation will be necessary, even though this places a large burden on operating systems and language translators and requires the development of new procedures for organizing and structuring programs. The user will be expected to supply only his application routines, while the hardware and software will provide the required data management, generate machine code, maintain an input–output file system, and support any required interactive terminal usage. The use of microprogramming to implement these systems on the next generation hardware seems certain. In a similar vein, all of the proposals to replace software with hardware require microprogramming to simplify the original design and to provide flexibility to implement changes in engineering requirements.

Most of this activity will be carried out in the laboratories of computer manufacturers under a shroud of secrecy, with little opportunity for participation by the academic community. This means that the academic community will continue to show relatively little interest in the specific problems of microprogramming or the more general issues of interpretive programming [25]. However, there will be a modest academic research effort on the theory of microprogramming from either an automation-theoretic or graph-theoretic approach, on integrated system design, on directly executable intermediate languages, and on the relation between multiprogramming and microprogramming.

The minicomputer industry will continue to call the machine language programming of their systems “microprogramming,” and may augment the present support software consisting of assemblers, simulators, and loaders with compilers and simple operating systems. The range of application of mini- and microprocessors will continue to expand rapidly, and this will extend the use of “microprogramming” as a method to customize hardware to various applications. While some question could be raised as to the suitability of the use of the word “microprogramming” in this instance, the die has been cast and the future will be one of growth.

Finally, the development of automatic optimization of microprograms will continue primarily under the aegis of mainframe producers who will use this technique to improve their marketing position by offering “optimized” hardware for particular applications. This trend may prove to restrict the user to expensive software which can only be run on a specific machine. Replacement and modernization of hardware may become more costly unless efficient microprogrammed emulators can be generated algorithmically. In conclusion, the use of microprogramming will have a profound impact on data processing technology in the next few years, and, as predicted above, this impact will be in the direction of improved support for the user at the cost of greater complexity of hardware and support software. The cost of software, including microprogramming, will continue to exceed the cost of hardware, and the user will have even less control over his application program in the future than he has today.

ACKNOWLEDGMENT

The authors would like to express their appreciation to the referees who reviewed the papers for this special issue, and to N. Lincoln of Control Data Corporation who provided the description of the STAR microprogramming applications.

REFERENCES

Abstract—Several methods of evaluating user programs are analyzed with respect to space and time requirements. The concept of an "ideal" directly executed language is introduced and it is argued that the "ideal" directly executed language for a contemporary computing system will not be either its source language or the language accepted by its base machine.

Index Terms—Directly executed languages (DEL's), emulation, interpretation, language-oriented-machines, machine-oriented-languages, microprocessor.

INTRODUCTION

In a broad sense, any computer system evaluates user-written (or source language) programs in two distinct phases. First, a user program is converted into an "equivalent" program in some intermediate language during an initial translation phase. The resulting program then becomes a "surrogate" for the original user program, being executed in its place as often as desired, over any number of subsequent interpretation phases. In real systems, there may be several "levels" of intermediate languages involved in the translation phase (e.g., "parser-tree" code emitted by a syntactic scanner, "object" code emitted by a code generator, " relocatable " by a loader). Similarly, there may also be multiple "levels" involved in an interpretation phase (e.g., the microprogrammed CPU of a model 65 emulating a 360-machine which is executing a 360-code program that is interpreting a LISP source program).

In this paper, such complex computing systems are abstracted into "Two-Phase Processing Systems," in which all translational processes are composed into a