WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

COMPUTER EDUCATION IN A GRADUATE SCHOOL OF MANAGEMENT

D. N. Ness - R. S. Green
W. A. Martin - G. A. Moulton

382-69

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139
COMPUTER EDUCATION IN A GRADUATE
SCHOOL OF MANAGEMENT

D. N. Ness - R. S. Green
W. A. Martin - G. A. Moulton

382-69
Computer Education in a Graduate School of Management

D. N. Ness - R. S. Green - W. A. Martin - G. A. Moulton
Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts

Abstract

We discuss the philosophy, design, and implementation of a program in management information systems at the Sloan School of Management, M.I.T. An essential element of this program is the PRISM simulated computer system.

Introduction

For the past several years those responsible for teaching courses in computers and management information systems at the Sloan School of Management have been evolving and practicing a philosophy of a computerization. The historical antecedents of our current program have been discussed by Ness in "Some Comments on the Role of Computers in Management Education" (21).* In this paper we consider the objectives of this program to educate prospective managers, prospective leaders of management, and others about the use of computers in management.

The experience which has shaped the current program has come from diverse sources. We have been teaching courses to graduate students in management for several years; we have also had the experience of teaching

*Refers to bibliography at the end of this paper.
practicing managers in our middle management, senior executive, and summer short-course programs. This experience has led us to some conclusions about the way that this subject should be taught.

Information Technology and Managerial Problem Solving

A central purpose of our program is to teach the technology of information processing and its application to the problems of management. A basic assumption is that an understanding of the technology is vital in solving a wide range of managerial problems. We suggest that if this understanding of the technology is to be of real value, it must be much more than superficial. Although we do not expect every manager (managerial problem solver) to be a programmer, we think that anyone who must deal in a significant way with an information processing problem must at least understand the task of programming. He must also be able to understand the power and scope of the technology, for otherwise it will be difficult or impossible to evaluate technological advance realistically. In a field where technological advance is the rule rather than the exception this would surely be unfortunate.

In our experience this view has been gaining currency in the recent past. We find an ever increasing number of practicing (and often quite senior) managers come to us to learn programming and technology. These managers report that while they do not anticipate doing any significant programming work themselves, they have come to recognize that it is vital
that they understand what programming is all about. They feel that learning some programming is the only way to do this, and we agree.

Level of Penetration

For several years the Sloan School has had a requirement that all graduate degree candidates demonstrate a proficiency in FORTRAN programming. The basic idea behind this "benchmark" requirement (which is supported by a non-credit lecture course) is that the faculty should feel free to assign problems and term projects which require computer work. This is a clear recognition of the potential importance of computers in our own problem solving environment.

The level of understanding of computers and computer systems required of our students proves to be of use to almost every one of them. However, it is by no means sufficient for some. If we consider the growing domain of managerial problem solving, it is clear that such a low level of understanding is not sufficient for such tasks as evaluating and employing computing systems in new and creative ways.

In FORTRAN, as in most higher level languages, many issues are not faced directly by the user. This is the strength of such languages. From our standpoint it is also a weakness. Some languages face some of these issues, while others face different ones. It is clear that no language, other than the language of the machine itself, allows exposition of the great majority of the significant issues which must eventually be faced in many important problems. Among these are the problems of information
systems design. Not only does machine language allow these issues to be exposed; they can also often be expressed concisely, and in terms which are precisely defined.

As new technology emerges this seems to be more and more the case. Few higher level languages deal with real-time considerations, segmentation, paging and the like. Yet, many research efforts indicate that knowledge and understanding of such principles* may prove to be of singular importance in the design, specification, and construction of effective and efficient systems.

It seems clear that in the long run these specific issues will gradually cease to be of importance. This obsolescence is a characteristic of much technical material. We find it important, nevertheless, to teach this material for two distinct, but related, reasons.

First, the useful lifetime of this kind of information is often longer than might at first be imagined. While it is clear that we no longer have much occasion to use the instruction codes which we learned in the middle 1950's, the model of computers that we developed during the learning process is still in regular use. Second, much of the value associated with mastering a technical discipline comes from the understanding of its approach. One of the reasons that many non-scientists find the study of mathematics or physics useful is that while they may or may not have

*The recent research in the effects of paging is one example.
occasion to use the facts learned in such a study, they certainly will have occasion to use the methodology.

**Basic Assumptions**

It is a fundamental tenet of our educational program that anyone who wishes to deal in a significant way with issues which center on the computer must understand its actual operation. Understanding only at some higher language level will prove insufficient. It is vital that the student be acquainted with the real currency of a computer, the machine language or instruction code. Many of the basic issues of system construction and efficiencies may be considered adequately only in such real terms. In a FORTRAN program, for example, A*B represents the product of A and B. A**B, which is as easy to write, represents A raised to the B power; but invokes a process which is at least ten, and probably several hundred times as time-consuming.

By placing strong emphasis on the understanding of the computer at the machine language level, we do not mean to suggest that it will be necessary for everyone to have the world-view gained by such understanding or that it will be common for managers regularly to practice computer programming at the machine language level. Our point is that anyone who expects to do significant work using the computer in a direct and creative way must possess such an understanding. Otherwise the model of the computational environment will be so inadequate as to lead to grossly wrong conclusions.
The notion that we must begin to teach machines at the machine language level has been incorporated into the basic courses in our curriculum. The central course in this curriculum for computer specialists and management systems designers is Management Information Technology, and this course teaches machine language programming.

Choice of Machine

If the basic course in our curriculum is to teach machine language programming it is vital that we select carefully what we are to teach. Any candidate for machine and system should fulfill several criteria; it should:

1) Raise as few spurious issues as possible.
2) Allow, without undue effort, the solution of interesting problems.
3) Be capable of exposing all outstanding issues of significance, within the chosen machine.
4) Be capable of pursuing issues in great depth when appropriate.
5) Not be committed to the equipment provided by any manufacturer.
6) Be able to provide the student with diagnostic aids to a great depth.
7) Allow the student ready access to the machine.
8) Be capable of extending the environment to expose new issues and techniques as they evolve.

These points suggest certain aspects of a system design appropriate to the task of teaching about a machine and its language. We will treat these subjects in sequence.

First, one of the most confusing issues to many students is decimal
versus binary/octal/hexadecimal. This is totally spurious, in that one can understand machines completely without any regard to a number base. Therefore, we choose to teach a decimal machine: placing our emphasis on all of the issues associated with the machine itself. When a non-decimal machine is encountered it is only necessary to master the new number system. When we require mastery of a new number system at a time when the machine itself is not well understood, we are asking more than is necessary. Another issue which often confuses beginners is the presence of "quirks" in a machine design. These confuse to no real purpose.

Second, it should be possible to give the student interesting and significant problems early in the learning process. Such problems provide not only motivation, but also tie down significant points and provide the student with a benchmark to measure his own progress. One of the most interesting aspects of learning computer programming is that it is so easy to convince oneself that a given problem solution or a given model of a situation is correct and sufficient, when this is far from the case. Interesting problems allow the student to test the depth of his understanding and the adequacy of his models.

Third, while it is fine and appropriate for the machine to present a simple face to the student, certain issues are inherently complex. It does little service to provide an environment where it is not possible to deal with such issues. To assume away the problems makes it impossible for the student to learn about them. Yet it is exactly these complex problems that
are the most important for him to study. The environment thus must be capable of evidencing such issues. As we will see below, some of the prior work in this area is subject to criticism in this regard.

Fourth, complex issues must be describable, and it must be possible to pursue such a description in depth. The good student who senses a problem must be able to approach the logic of the machine and thus obtain an answer. This answer may or may not be completely logical, but it is important that the best students have a place to turn.

Fifth, different environments will have the equipment of different manufacturers. Even a specific place will have different computer equipment as time passes. Any direct emphasis on the equipment of a specific manufacturer is (rightly) suspect. One of the purposes of education is to eliminate ill-founded prejudices. If any specific equipment manufacturer receives undue attention this goal may be subverted.

Sixth, it should be possible to provide the student with diagnostic aids which educate as well as point out error. Traces and snapshots provide the student with a valuable overview of the state of the system and how this state changes from one operation to the next. Such information gives the student a feeling of how the machine really operates as well as indicating errors in his program. It contributes directly to the student's building of an appropriate model of the computation process.
Seventh, and a point implicit in much of the above, the student must really have access to the machine. A real interaction with a computer is worth many long pages of description of one. Real interaction is also a vitally important motivating factor. It is frustrating to write a program and then not see it run. We find it useful to have the student face early the psychological problems involved in having the computer system reject a program that he is "sure" is absolutely correct. It would be impossible for any teacher to look at the student's programs closely enough to provide this kind of feedback.

Eighth, and finally, it is important that the environment be capable of extension and elaboration as new issues arise. It also must be possible to demonstrate facilities which are not actually physically available at a given installation. For example, we want to be able to describe, and allow the students to program, a device like a display even when such a device is not available at the installation. Similarly it may be important from a motivational standpoint to allow a student to write programs for stock exchange tickers or some such device which is not readily available on most machines (for any reasonable cost).

The PRISM System

We have designed a system which incorporates all of these features. This system is a set of programs which simulate a machine which meets the criteria presented above. This system can provide all of the flexibility
which is required. To obtain this flexibility we must pay a price in execution time, but we feel that this is clearly a price worth paying in our circumstances.

In our experience with graduate students we have found that the average student executes between 5,000 and 50,000 computer instructions during a term's study. This suggests that efficiency of instruction execution is not a paramount consideration; even if the simulator is a little slow we will not be too demanding of real computer resources. The flexibility and careful control of errors that we obtain by simulation seem to be well worth the price.

In our implementation of the system we also provide facilities like loaders and assemblers. While we provide "documentation" copies of these programs in PRISM machine language, they are actually written to operate on the real machine we have available. Thus we do not get involved in simulating the action of the assembler. This is one reason that each student executes so few simulated instructions during his term's study.

The documentation copies of programs exist to allow the students to probe into the complexities of the assembler and loaders should they desire. Thus it becomes possible for the students to pursue their own understanding to whatever depth they deem appropriate. This is important in a field where some students find learning the material much easier than do others. Rather
than becoming bored with the slow pace of a course (by their standards),
the exceptionally able students are able to pursue many topics in much
greater depth than is possible in normal class work. We are thus able
to teach a class with substantial variation in background and/or ability
which is common in this area.

Before considering more detailed aspects of the PRISM system it is
appropriate to look at other attempts to provide the kind of facilities
that we have suggested are important. There have been several attempts
to capture some of these objectives in a number of different publications.
We will review these attempts in some detail.

Other Attempts

We may divide other attempts into two broad classes: real machines and
pseudo machines. Both approaches have been used in several contexts,* and
each has significant problems with respect to our objectives.

Let us look first at the real machines. Since the days of the IBM 650,
texts and primers have been available which describe real machines. Given
our objectives using any of these would present severe difficulties. The
books (and manufacturer's manuals) are of varying quality, and it is unreason-
able to consider buying a machine because the primer or manual which describes
it is a good teaching and learning vehicle.

*Strictly speaking there is a third category -- real machines which were
built for expository reasons. One of the simplest and earliest, a two bit
binary machine called SIMON, is described in Berkeley (1, 2). Another, APEXC,
was much more elaborate and a useful computer in its own right (3, 4). History
seems to indicate that this approach is less efficient than either of the others.
More practically, real machines have other difficulties. It is usually impossible to allow the student to observe the real-time behavior of a machine. We can trace a machine as it executes its instructions (for example, the IBM 1130 has wired-in facilities to aid this), but since tracing significantly lowers the rate of real program execution fewer real program steps will occur between any two real-time events. Thus the program may behave very differently. This can be avoided only if the program and its real-time events are simulated in the same time frame.

In the few circumstances where tracing might be feasible we would still fail to meet at least three of our goals. First, few real machines present a student with a simple interface. Most real machines have a significant number of "quirks" or "kludges". While an experienced programmer learns to accept these as a common and normal part of his environment, the novice often finds it very difficult to separate the important and carefully considered aspects of a machine design from those which arose inadvertently. Quirks become a source of real confusion and directly distract from the real purposes of learning about computers. We do not think it is sound reasoning to suggest that because such quirks are common in real life they should be a part of an instructional environment. This is tantamount to suggesting that one should learn to swim in a heavy sea.

A second difficulty with using a real machine is that it represents a commitment to a specific manufacturer. While it is clear that many people regard all computers as "IBM machines" (remember in the old days they were "UNIVAC's") this seems to be a very bad prejudice for an educational program
to reinforce. It seems most appropriate, to provide an environment in which it is possible to be relatively neutral to the question of manufacture. We think neutrality leaves the student in a better position to perform the evaluations which may be a part of his future responsibility.

A third, and perhaps the most significant, point is the ability of the simulated environment to provide facilities which could not be afforded in any other way. In the context of a management school we have found it useful to simulate such devices as stock tickers and airline reservation consoles. In a real environment purchase of such devices for purely instructional purposes would surely be out of the question. Even simulation of such things in another environment would normally prove very expensive.

Let us look at the arguments that are used by authors who have chosen to take the real machine approach. Leeds and Weinberg (16, p. ix) make the revealing comment in their preface:

Perhaps a few words on the choice of computers for text examples is in order. The first question was whether or not we should "invent" a machine, so as to supply a more perfect pedagogical instrument than any actual computer might present and at the same time give academic purity to the book. We decided against this for several reasons:

1. An actual machine would give the student recourse to information other than that covered directly in the book.

2. In an effort to gain purity, we would probably have attained sterility. We felt that the naming of an actual machine would lend authenticity to the book and give the student some feeling for the compromises that often have to be made in the design and use of a real computer.
Two points are raised which must be dealt with by any proposal to use a pseudo-machine; we will return to them later. This is not to suggest that we see no advantage in studying a real machine. Clearly real machines run much faster than simulated ones. A student who learns a real machine in a course is "one machine up" (i.e., he knows one more real machine than he otherwise would). Our experience suggests that neither of these advantages is very significant.

First, the actual number of computer instructions that a student will execute in the course of a term's work is not so great as to make efficiency of execution an important consideration. Second, being "one machine up" does not seem to be important for very long. Real machines disappear quickly. Often a student will learn a machine which will be obsolete by the time he graduates. Finally, it is improbable that he will go into an environment which has the same kind of equipment as that on which he was trained.

For these reasons we feel that use of a real machine is the wrong approach. We want to control the educational environment and insulate it from capricious change (as machines and peripheral equipment come and go, for example). We also want to insure that it will be possible to investigate and expose issues which center on equipment not physically available. This suggests the simulation of a facility which comes close to meeting all of our objectives. Let us now consider some previous attempts at pseudo-machines.
Pseudo-Machines

Pseudo-machines have a long history. The earliest one that we know of is 1953 (27). Since that time many other authors (5, 6, 7, 8, 9, 10, 11, 12, 14, 18, 19, 24, 25, 26, 28) have come forth to present their machines. Almost all suffer from some common problems which we feel makes them unsuitable for our use. A recent book by Knuth (15) presents a much more adequate and careful design, which we will discuss separately below, but even this presents difficulties in our context.

By our standards all of the books and pseudo-machines we have considered (other than Knuth) make a severe mistake in the way that they simplify their hypothetical machines. We realize that this is done with the best of motives; nevertheless, we feel it is detrimental to the students.

A rich environment cannot be described easily using a simple machine. We feel that unless the objective of teaching in this area is to describe a rich environment, it would be better for the typical student to learn a higher level language (such as FORTRAN) rather than machine language. As we mentioned above, all our students are required to have a FORTRAN background, thus our courses deal with a more complex environment.

If the student does not have the time and/or inclination to pursue the topic of computation in some depth, we feel that he should devote what time he has to learning something of relatively immediate value, like FORTRAN. If he does, there is little sense in providing a simple machine and a simple
language. A higher level language is simply not a suitable tool for discussing many interesting and significant issues.

We contend that anyone who is going to be directly concerned with computation must form an accurate model of a computer at some time. Today his model must allow the investigation of such issues as real-time processing, random access device handling, telecommunications applications, time sharing, paging, and segmentation.

It is easy to be led down the path of misguided simplicity. For example, consider a pseudo-computer described by Gregory and Van Horn (10, 11) in their widely used texts. The operation codes recognized by the hardware of their machine are the actual mnemonics for the operations. Thus an add instruction might appear in memory as

+ADDA021

This was clearly done to avoid the problems of talking about the machine in a different language from the one that the machine itself recognizes. Perhaps this is a valid objective. The choice, however, leads the student to form a model of the machine operation which is inappropriate to almost every machine which has ever been built. The student who feels that this machine adds because it "sees ADD" may well be confused when he encounters a machine which has no such direct relation between the mnemonics and the machine operation codes. This kind of simplification can produce confusion.
Another interesting example of the simplification of issues is the introduction of a "repeat" instruction (24) to make the explanation of looping very simple. It seems to us that it would be better to explain the FORTRAN do-loop, thereby teaching the student something that will be useful in his problem solving. The model of a computer that the student develops from a machine with a repeat instruction* and no other looping facilities is not more appropriate than the one obtained from understanding the "FORTRAN machine", and a FORTRAN machine can do useful work.

Knuth's book, the first of a projected seven volume series**, is excellent. Apparently he has concluded that it is necessary to deal with real issues at the machine language level (15), and we heartily agree! Unfortunately we cannot use it as a basic text because in each volume Knuth's scope is too narrow and his penetration too deep for the pace and tone of our basic course. The first volume has served as a useful reference for the advanced student. Furthermore, many problems in management information technology are concerned with issues of large volume file maintenance, and random access file management. Exposition of these issues requires that the input/output system of the computer be very carefully constructed so as to allow experimentation with a wide variety of peripheral devices. Knuth's machine design and implementation make this difficult.

*Only a few real machines have such an instruction, and the modern ones have a much more elaborate one.

**Some of the volumes are not yet available, and may not be for several years.
In summary, we have found that none of the books we have reviewed are adequate for our purposes. The systems which they present are either too simple to allow us to expose the very issues that causes us to introduce machine language, or they are aimed at issues which are not our direct concern.

The PRISM Machine and System

PRISM is a pseudo-machine designed with all of the objectives that we have been considering in mind. It has been constructed to allow us to expose all of the issues that we feel are relevant, and at the same time to neglect those issues which are not central to our goals. Let us look at the design of the system and consider how they meet the objectives.

PRISM is a 10 digit, decimal, sign-magnitude, word orientated computer with 10,000 words of memory. This allows us to focus initial classroom attention on the computer itself, rather than on the question of representation. We begin to consider the problem of representation when we present floating-point numbers and deal with the manipulation of alphabetic information. At that time issues of binary versus decimal and complement versus sign-magnitude representations can be mentioned. This seems to us a much better method for presenting such material; the student usually will have developed a reasonably robust model of the computer itself by this time.
The instruction set of PRISM is very large, but at the same time simple. Many basic instruction types can be 'modified' (to change a relationship in a logical test, for example). Let us mention two examples. The JUMP instruction tests the contents of an accumulator against zero and then transfers control or not depending on a condition. The programmer can specify any one of eight conditions (always, greater than, greater than or equal to, equal to, less than, less than or equal to, not equal to, never). Another instruction in PRISM is SKIP. This tests a memory cell against zero. The conditions that can be specified are the same as for the JUMP instruction. Each of the various JUMP and SKIP operations* can be qualified by any of the eight conditions. Although PRISM has a large number of such instructions, they are not hard to remember or use, because the student can simply remember that all jump and skip operations can occur with any condition modifier. (On many real machines this is not the case. One must remember that "Branch if Index Low" does not exist while "Branch if Index Low or Equal" does.)

PRISM also provides a full range of byte manipulation instructions. Current trends in computer hardware indicate strongly that the manipulation of quantities of less than a full word in size is becoming more and more important. The provisions of instructions to handle such quantities is thus

*There are other operations in this class to perform conventional indexing (Add One and Jump) etc., all of which use the same set of condition modifiers.
reasonable from a hardware standpoint. It is also important from the educational standpoint. Such instructions allow character manipulation to be performed easily. This allows the student to write programs which perform all of the tasks associated with solving some problem, including input and output formatting. Thus he may develop a realistic notion of the amount of effort to perform all of the activities associated with a given job.

The most elaborate and complex part of the PRISM machine is its input/output system. Many of the important problems in management information technology center on the efficient use of input/output facilities. For example, many real time and data management applications must use a computer input/output system efficiently. For this reason we thought it vital to provide an environment which was rich enough to allow the full range of such issues to be explored in considerable detail.

From the standpoint of teaching there is one real problem associated with a complex I/O system. Either students are required to learn a great deal of confusing material about complicated I/O processes at a time when they are just beginning to form their models of the computer itself, or, alternatively, issues of input/output have to be shrouded in a cloak of mystery until rather late in the learning process. To avoid this difficulty, PRISM has some very simple input/output devices which behave in an easily comprehensible way. These devices can be used by the student in the earlier part of his training; later his focus can be shifted to the more complex facilities. In this way we manage to expose the issues at a reasonable pace.
without ever lying to the student about what is going on in the system. The simplified I/O facilities are perfectly realistic and behave like the complex facilities except that they do not raise the question of real-time processing or simultaneous I/O processing and computation. Thus the new material can be exposed simply by going into issues which were not considered earlier, without it ever being necessary for the student to "unlearn" something that was wrong in his model of the environment.

This characteristic, never unlearning, is basic to our whole approach with the PRISM system. The material presented in our programming primer (22) is designed to introduce students to the subject of computers and machine language programming. This primer is not a reference manual for the PRISM machine. In the primer we feel under no obligation to expose more of the machine than is necessary for the immediate purposes of exposition. Although a substantial proportion of the instruction codes of the PRISM machine are discussed, many others are not mentioned at all. The student who is having difficulty with the basic material can concentrate on the primer, knowing full well that he will never be required to understand more of the machine than is presented there. At the same time the exceptionally quick and able students will find it possible to investigate the "advanced" facilities and options which are really available to them in the PRISM Reference Manual (22). They thus maintain their interest even though they are not necessarily challenged by the basic material.
The PRISM environment, which is being simulated on some real computer, is available for experimentation in many different dimensions. From an educational standpoint we can observe the interactions which our students have with the system, and as a result modify our presentations of the material to test the real difficulties as they are exposed. We can use the environment to experiment with issues which would not otherwise be feasible. For example, we can easily raise the question of what kind of hardware instructions would be appropriate to the task of implementing some higher level language. As micro-programming "firmware" becomes more and more common, this may turn out to be an important option.

As a third example, consider how software systems might use some of the hardware facilities that are commonly being developed today. Hardware generated "faults" during the addressing cycle of a machine is one example. Such facilities are built into available hardware but there are few reported experiments using such hardware. With PRISM we can create an environment which includes the hardware features we wish to evaluate, and we can experiment with these features in a realistic device and problem context. Without an environment within which such ideas could be implemented and tested, however, this kind of investigation tends to be sterile and uninteresting.

The PRISM system provides the essence of a laboratory for software and hardware experimentation. It is direct support for our educational efforts in the computer area; but it fulfills more than this simple direct educational goal.
The business community, which makes huge annual investments in both hardware and software, has only in rare circumstances been able to afford the time and effort required to investigate the potential applicability of advance hardware and software concepts to their information processing problems. Machine structures, such as that proposed by Iliffe (13), which differ in a significant way from the usual, are difficult to evaluate in the context of a management information system.

Few practicing managers have the time or the required background to understand the implications of such ideas, and few machine designers have the time or required background to understand the problems of information systems development.

Similarly, many software notions have never been adequately studied for their applicability in the information processing environment. Lombardi (17) suggested several years ago a structure which might prove to be useful in solving some kinds of problems. To the best of our knowledge these ideas were never tested extensively. This is not surprising, given the pressing nature of the data processing problems in most industrial environments. Most companies have been very short (in some cases desperately short) of programming resources.

The academic environment has the kind of resources needed to perform such research, and the responsibility to do it. It also has the people who could investigate such issues if the appropriate kind of laboratory can be
developed. We believe that PRISM can provide the basis for such a laboratory. It will give our students a common background and experience over a substantial period of time (the time of their studies at the Sloan School). This will allow them to concentrate their efforts on the problems rather than on learning a succession of machines and systems. We expect PRISM to provide a focus which will thus help us conserve our programming resources.

The Current Implementation of PRISM

Presently PRISM operate on the CP-67/CMS (IBM 360/67) system at M.I.T.'s Urban Systems Laboratory. The system will be run on a regular basis under OS/360 on an IBM 360/65 at M.I.T.'s Information Processing Services Center. Other implementations of the system will be developed as time becomes available and as needs dictate.

Conclusion

To conclude, let us return to the points raised above by Leeds and Weinberg. They suggest two reasons for using a real as opposed to a pseudo-machine. The first point, that the student would have recourse to material other than that covered directly in the book, is answered by making available a substantial amount of information about the pseudo-machine. We are preparing a Reference Manual, a Case and Problem Book and reference implementations of several major programming languages. The Reference Manual* will be written in the style of a standard "principles of operation" manual. The Case and Problem Book will contain examples of

*The Reference Manual will also be used to introduce our students who have previous experience with computation to PRISM.
programs written by people with good programming style. These programs will be carefully documented, and they will be just long enough to expose significant issues of taste and technique. We feel that this will make more pertinent and instructive documentation available than might be typical of many real systems. The point, surely, is not the quantity of information available, but rather how well this material exposes important and relevant issues.

The second point, that a pseudo-machine will probably achieve a form of sterility, is more difficult to respond to. All we can say is that we have paid a substantial amount of attention to the problem of a hardware realization of the machine that we describe. We have done this for two reasons. First, the discipline of requiring that we be clear about how much hardware our machine would require guarantees that we do not attempt to solve every software problem by building hardware which makes the problem disappear. This would surely lead to exactly the kind of sterility which Leeds and Weinberg fear.

A clear hardware implementation of PRISM is desirable because this too fits into our educational program. We are in the process of developing courses which will study such problems as hardware/software trade-offs in the design of information systems. In such courses we must clearly deal responsibly and at a reasonably detailed level with the problems of hardware construction. It is natural to want to do this in an environment which is already familiar.
The current implementation of PRISM has been used for the first time in the Spring Term of 1969. Success with earlier implementations, which have been in use over the past several years, leads us to believe that we are accomplishing at least some of our objectives. We are confident that this new implementation represents a substantial improvement over the earlier versions.
REFERENCES


