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
(REVISED)

September, 1969

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
(REVISED)

September, 1969

title: Computer Education in a Graduate School of Management

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

address: Sloan School of Management
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

abstract: Several years of experience have led us to believe that the creative design and evaluation of management information systems requires a thorough understanding of the related computer technology. Concepts such as paging and priority interrupt systems can best be explained at the machine language level. Any machine used for exposition 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 useful for 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 extension to expose new issues as they come along.

We have constructed a simulated machine and its associated software which meets these criteria. This system, called the PRISM system, is documented by a primer and a reference manual.

key words: education, simulation, machine language, management information systems, interpreters

CR categories: 1.52, 4.21, 4.13, 3.51

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

Introduction

For the past several years we at the Sloan School of Management have been evolving and practicing a philosophy of education in computers and management information systems. In this paper we consider the objectives of a program* to educate prospective managers, teachers, and others about the use of computers in management.

Our current program has been shaped by diverse experience. We have been teaching graduate courses in management for several years; we have also taught 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

The central purpose of our program is to teach the technology of information processing and its application to the problems of management. An understanding of this technology is vital in solving many managerial

---

*Historical antecedents of our current program are discussed by Ness (24).

problems. We suggest that if an understanding of the technology is to be of real value, it must be much more than superficial. Although we do not expect every manager 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 be able to understand the power and scope of the technology, for otherwise it will be difficult to evaluate technological advances and technical personnel realistically.

In our experience this view has been gaining currency. We find an increasing number of practicing, and often quite senior, managers come to us to learn technology. These managers tell us that they do not expect to do any significant programming work themselves, but they have come to recognize that an understanding of programming is vital to their function. They feel that learning some programming is the only way to do this, and we agree.

Scope of the Program

For several years the Sloan School has required 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.

The level of understanding of computers and computer systems required of our students proves to be of use to most 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 this degree of understanding is not sufficient for 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 and purpose of such languages, but from our standpoint it is also a weakness. No language, other than the language of the machine itself, allows exposition of many significant issues which arise, for example, in the design of information systems. Not only does machine language allow these issues to be exposed; they can 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, 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.

In the long run these specific issues will gradually cease to be of importance. Obsolescence is a characteristic of much technical material. We find it important, nevertheless, to teach this material for two distinct reasons.

---

*In FORTRAN A*B and A**B are about equally easy to write. This may lead a user to feel that they are equally easy to calculate.

First, the useful lifetime of this kind of information is often longer than might at first be imagined. While 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. Even though non-scientists may have little occasion to use the facts learned in a study of mathematics or physics, they certainly will have occasion to use the methodology.

Choice of Machine

For these reasons the basic course in our curriculum teaches machine language programming. Advanced courses build on and apply the material presented in this basic course. Therefore, it is vital that we select carefully the machine and system we are to teach. Any candidate 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.

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.

We will consider these criteria in sequence. First, one of the most confusing issues to many students is decimal versus binary/octal/hexadecimal. This is spurious, in that a manager can form an effective model of a machine without regard to its 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. Requiring mastery of a new number system when the machine itself is not well understood is asking more than is necessary.

Second, it should be possible to give the student interesting and significant problems early in the learning process. Such problems not only provide motivation, but also tie down significant points and provide the student with a benchmark to measure his own progress. One of the most revealing aspects of learning computer programming is that it is so easy to convince oneself that a problem solution or a model of a situation is correct 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. It is exactly these complex problems that are the most important for him to study.

Fourth, complex issues must be describable, and it must be possible to pursue such a description in depth. The student who cannot understand, for example, the exact effect of a certain sequence of instructions must be able to turn to a description of the logic of the machine to obtain an answer. The answer obtained may or may not be completely logical, but it is important that inquisitive students have a place to turn.

Fifth, installations will have different equipment. Yet any direct emphasis on the equipment of a specific manufacturer is suspect. One of the purposes of education is to eliminate ill-founded prejudices.

Sixth, it should be possible to provide the student with diagnostic aids which educate as well as point out error. Traces and snapshots provide a valuable overview of the state of the system and how it changes from one operation to the next. Besides indicating errors in his program, such information gives the student a feeling of how the machine really operates. It contributes directly to building an appropriate model of the computation process.

Seventh, the student must really have access to the machine. A real interaction with a computer is a vitally important motivating factor, and worth many pages of description. It is frustrating to write a program and then not to see it run. We find it useful to have the student face the psychological problems involved in having the computer system reject a program that he is "sure" is absolutely correct. It would be impossible for a teacher to look at 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 must be possible to demonstrate facilities which are not available at a given installation. For example, we want to allow the students to program a device like an array processor 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.

We have a set of programs, called PRISM, which simulate a machine and its environment that incorporates all of these features. We will discuss this system in detail below. Before considering more detailed aspects of the PRISM system, however, 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 psuedo-machines. Both approaches have been used in several contexts,* and each has significant problems with respect to our objectives.

---

*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.

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 unreasonable to consider buying a machine because a primer which describes it is a good teaching vehicle. Further, it is usually impossible to allow the student to study the real-time behavior of a machine.* We can trace a machine as it executes its instructions, 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 differently than it would without tracing. This can be avoided only if the program and its real-time events are simulated on the same time scale.

In the few circumstances where tracing might be feasible using a real machine we would still fail to meet at least three of our goals. First, few real machines present a student with a simple interface. Most machines have a significant number of "quirks". An experienced programmer learns to accept these as a common and normal part of his environment, but 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 confusion and distract from the important issues. 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.

---

*Notice the difference between watching a bullet hit a piece of wood in "real time", and studying a slow motion movie of the same action.

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 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. Neutrality leaves the student in a better position to perform the evaluations which may be a part of his future responsibility.

A third point is the ability of the simulated environment to provide facilities which could not be afforded in any other way. In a management school we have found it useful to simulate such devices as stock tickers and airline reservation consoles. Purchase of such devices for purely instructional purposes is surely out of the question.

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). We suggest that neither of these advantages is very significant.

First, the actual number of computer instructions that a student will execute during his introduction to programming 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. 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 (30). Since that time many other authors (5, 6, 7, 8, 9, 10, 12, 13, 14, 16, 21, 22, 27, 28, 29, 31) have come forth to present their machines. Almost all suffer from some common problems which we feel makes them unsuitable for our use. Recent books by Knuth (17, 18) and Gear (11) use a much more adequate and careful design, which we will discuss separately below, but even these present difficulties in our context.

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

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 rather than machine language. Thus, all our students are required to have a FORTRAN background, while our computer 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 or a simple language. A simple machine is not a suitable tool for discussing many interesting and

significant issues. 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.

Knuth's book, the first of a projected seven volume series,* is excellent. He has concluded that it is necessary to deal with real issues at the machine language level (see (17) page x), and we heartily agree! Unfortunately we cannot use these books as a basic text because each volume is too narrow and its penetration too deep for the pace and tone of our basic course. The volumes serve as a useful reference for the advanced student. Furthermore, many problems in management information technology are concerned with telecommunications, 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.

Gear (11) uses a pseudo machine to explain interpretation, simulation, and computer design. He develops his machine only far enough to illustrate such ideas as microprogramming, and it would be difficult to extend his design to cover all of the issues that we need to consider. If his machine had been simulated in PRISM rather than directly on a real machine, then it would be possible for students to watch it operate step by step.

---

*Some of the volumes are not yet available, and may not be for several years.

In summary, we have found that no book we have reviewed is adequate for our purposes. The systems which they present are too simple to allow us to expose all of the issues that cause us to introduce machine language.

The PRISM Machine and System

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

PRISM is a 10 digit, decimal, sign-magnitude, word oriented computer with 10,000 words of memory. Using a decimal instead of binary machine allows us to focus initial classroom attention on the computer itself, rather than on the question of representation. We begin to consider this problem when we present floating-point numbers and deal with alphabetic information. Later issues of binary versus decimal representation can be mentioned without forcing the student to become proficient in the manipulation of binary numbers. This seems to us a much better method for presenting such material.

The instruction set of PRISM is very large, but at the same time simple. Many basic instruction types can be 'modified'. Let us mention two examples. The JUMP instruction tests the contents of an accumulator against zero and then conditionally transfers control. 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 which tests a memory cell against zero. The conditions that can be specified are the same as for the JUMP instruction.* Thus PRISM instructions are not hard to remember or use, because 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.)

The most complex part of the PRISM machine is its input/output system. Important problems in management information technology center on the efficient use of such facilities. For this reason we thought it vital to provide an environment which was rich enough to allow all aspects of these problems to be explored.

Teaching complex I/O systems is difficult. Either students must master a great deal of confusing detail about complicated I/O processes before they have an adequate model of the computer itself, or issues of input/output must be shrouded in a cloak of mystery until late in the learning process. To avoid this difficulty the PRISM I/O facilities admit of a simple description and mode of operation which is adequate for the student in the first ten weeks of his training. Later issues of device characteristics, simultaneity and efficiency are introduced simply by describing the I/O facilities in greater detail. Thus the new material is exposed by going

---

*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.

into issues which were not considered earlier, and it is never necessary for the student to "unlearn" or to establish a new framework capable of supporting new concepts. His model has been correct but not sufficiently rich.

Providing a framework to be elaborated, rather than rebuilt, is basic to our whole approach with the PRISM system. The material presented in our programming primer (26) 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 it we feel under no obligation to expose more of the machine than is necessary for our immediate purpose. Many of the instruction codes of the PRISM machine are discussed, but 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. By reading the PRISM Reference Manual (25), the exceptionally quick and able students may explore the full range of facilities and options which are available to them. Thus we can maintain the interest of a wide range of students even though some may not be challenged by the basic material.

PRISM - Research Aspects

PRISM does more than support our direct educational efforts in the computer area; it provides the essence of a laboratory for hardware and software experimentation. Since PRISM and its environment are simulated

on some real computer, we have the ability to work with several versions of the system simultaneously. We can add, delete or modify features with ease and tailor versions of the system for experimentation in different dimensions.

First, we can observe the interactions which our students have with the system. We can then modify our presentation, the system, or supporting programs, to meet any difficulties thus exposed. Further, the ability to observe student interactions, coupled with PRISM's extensive diagnostic facilities allows us to experiment with the computer as a teaching tool.

Second, both the faculty and the advanced students can use the PRISM system to study the behavior of interesting I/O devices and machine structures. For example, only a program which simulates such a device need be written. Since our students are already familiar with the PRISM system in detail, they can concentrate their efforts on experimenting with the device. Thus, PRISM provides a focus and a common set of conventions which allow new programming projects to be built on the results of previous ones. This is conservation of important resources.

The major cost of using a simulator, once the software has been written, is the amount of machine time required to execute each instruction. We do not find this to be a serious problem. In our system the loaders, assemblers, etc., which the student accesses are written in the language of the "host" machine. This means that they operate at real, rather than

simulated, speeds. Since the student is not usually concerned with tracing the action of such programs he loses nothing by this process. For the student who wants to learn more about the operation of the programs, however, we do provide "documentation" copies of the programs written in PRISM.

Concluding Remarks

To conclude, let us look at some arguments used by authors who have chosen to take the real machine approach. Leeds and Weinberg (19, p. ix) make the following 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.

The first point 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* is written in the style

---

*The Reference Manual will also be used to introduce our students who have previous experience with computation to PRISM.

of a standard "principles of operation" manual. The Case and Problem Book will contain examples of 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.

It is more difficult to respond to the second point -- sterility. We pay 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 assuming hardware which makes the problem disappear. That would surely lead to exactly the kind of sterility which Leeds and Weinberg fear. Second, a clear hardware implementation of PRISM is also desirable because it fits into our educational program. We are developing courses which study hardware/software trade-offs in the design of information systems. In such course we must 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

PRISM has been running on the CP-67/CMS (IBM 360/67) system at M.I.T.'s Information Processing Center since January 1969. The system will be run on a regular basis under OS/360 on an IBM 360/65. A FORTRAN implementation is being planned and others will be developed as time and needs dictate.

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 most of our objectives. We are confident that this new implementation represents a substantial improvement over the earlier versions.

REFERENCES

1. Berkeley, E. C., Giant Brains: or Machines That Think, 1949 (Reprinted by Science Editions, New York, 1961).

2. Berkeley, E. C., Computers - Their Operation and Applications, Rhinehold, New York, 1965.

3. Booth, A. D., and Booth, K. V. H., Automatic Digital Calculators, (Second Edition) Butterworths, London, 1956.

4. Booth, K. V. H., Programming for an Automatic Digital Calculator, Butterworths, London, 1958.

5. Crawford, R. A., and Copp, D. H., Introduction to Computer Programming, Houghton-Mifflin, Boston, 1969.

6. Cutler, D. I., Introduction to Computer Programming, Prentice-Hall, Englewood Cliffs, New Jersey, 1964.

7. Davidson, C. H., and Koenig, E. C., Computers: Introduction to Computers and Applied Computing Concepts, Wiley, New York, 1967.

8. Davidson, J. F., Programming for Digital Computers, Business Publications Ltd., London, 1961.

9. Flores, I., Computer Software, Prentice-Hall, Englewood Cliffs, New Jersey, 1965.

10. Flores, I., Computer Programming, Prentice-Hall, Englewood Cliffs, New Jersey, 1966.

11. Gear, C. W., Computer Organization and Programming, McGraw-Hill, New York, 1969.

12. Gregory, R. H., and Van Horn, R. L., Automatic Data-Processing Systems, Wadsworth, San Francisco, 1960.

13. Gregory, R. H., and Van Horn, R. L., Business Data Processing and Programming, Wadsworth, Belmont, California, 1963.

14. Gotlieb, C. C., and Hume, J. N. P., High Speed Data Processing, McGraw-Hill, New York, 1958.

15. Iliffe, J. K., Basic Machine Principles, Macdonald, London, 1968.

16. Jeenel, J., Programming for Digital Computers, McGraw-Hill, New York, 1959.

17. Knuth, D. E., Fundamental Algorithms, Addison-Wesley, Reading, Massachusetts, 1968.

18. Knuth, D. E., Seminumeric Algorithms, Addison-Wesley, Reading, Massachusetts, 1969.

19. Leeds, H. D., and Weinberg, G. M., Computer Programming Fundamentals, McGraw-Hill, New York, 1961.

20. Lombardi, L. A., "Mathematical Models of File Processes", Atti Sem. Mat. Fis. Univ. Modena (12), 1963, pp. 193-214.

21. McCracken, D. D., Digital Computer Programming, Wiley, New York, 1957.

22. McCracken, D. D., Weiss, H., Tsai-Hwa, L., Programming Business Computers, Wiley, New York, 1959.

23. Moulton, G. A., "The Design of an Aid to the Teaching of Computer Technology: The STUPID System", Unpublished S.B. Thesis, M.I.T., February, 1969.

24. Ness, D. N., "Some Comments on the Role of Computers in Management Education", Sloan School of Management, M.I.T., Working Paper No. 383-69, 1969.

25. Ness, D. N., Green, R. S., Martin, W. A., Moulton, G. A., The PRISM Reference Manual.

26. Ness, D. N., Martin, W. A., Green, R. S., Moulton, G. A., The PRISM Primer.

27. Reilly, E. D., and Federighi, F. D., The Elements of Digital Computer Programming, Holden-Day, San Francisco, 1968.

28. School Mathematics Study Group, Algorithms, Computation and Mathematics, Student Text, Stanford University, 1966.

29. Sherman, P. M., Programming and Coding Digital Computers, Wiley, New York, 1963.

30. Thomas, W. H., "Fundamentals of Digital Computer Programming", Proceedings of the IRE, Vol. 41, 1245-49 (October 1953).

31. Wimmert, R. J., Computer Programming Techniques, Holt, Rinehart, Winston, New York, 1968.