TEACHING COMPUTER ORGANIZATION IN A LABORATORY

C. L. Seitz

Rapporteurs: Mr. A. Alderson
Mr. C.R. Snow

In his first two lectures, Professor Seitz described a one-year undergraduate course in computer organization, which he teaches at the University of Utah. The course is required of all computer science majors, and most take it in the second or third year of a four year program. The course also attracts many students from electrical engineering, arts and sciences, and other areas.

About half of the 130 students registered for the computer organization course, including most of the computer science majors, are concurrently enrolled in the "Switching Circuits Laboratory." The use of this laboratory, coordinated with the computer organization course, is the magical ingredient in the receipt for this course which makes it appetizing to the students.

Overall Organization

Largely because of the coordination between the classroom and laboratory activities, a "bottom-up" approach is followed. The students learn first how to design simple logical circuits from gates and flip-flops, which are the most basic elements available to them in the laboratory. The student soon discovers that certain circuit configurations -- decoders, adders, multiplexers, registers, counters, shift registers -- reappear often, and are available "ready to use" on the more elaborate modules found in the laboratory. He then begins to design at the functional unit level. This approach has the advantage that each hierarchical building block of a digital system can be understood as a synthesis of devices whose behaviour is already understood. The student also learns by this process how to design those parts of a system -- particularly timing and control -- which tend to be irregular.

Although the goal of the course may be only to show the student "how a computer works," and how it is built up from parts, it seems that the only way to do this effectively is place the emphasis on design. The classroom activities are design-oriented, while the laboratory activities are project-oriented.

The particular <u>style of design</u> that is encouraged is relatively formal. The main interest is however in teaching the <u>use</u> and <u>meaning</u> of the various representations for logical machines and computing processes rather than vote manipulation. In this way, the students learn to use the available formalisms to <u>organize</u> themselves to the design task.

<u>Autumn Quarter -- Switching Circuits</u>

Combinational design is taught first, including switching algebra, classical simplification of switching expressions by means of Karnaugh maps or Quine's method, and AND-OR or NAND realizations. The canonical representations and simple decompositions of switching functions are then studied, in order to formalize the design of combinational nets built from decoders, multiplexers, and read-only store. The students are later encouraged to make good use of random-access store (either read-only or read-write) to implement complicated switching functions, and this second view of the implementation of combination functions is an important tool.

The state diagram representation is used for (clocked) sequential circuits. The students first learn the art of constructing a state table or state graph representation which corresponds to verbal description of a process. Most of the students have had enough background in programming to relate state graphs to flowcharts, and tend to regard state diagrams as simply another form of flowchart. Of course, they discover the situation of state equivalence, and teach the reduction of state diagrams to minimal state representation. From this point, the students have no difficulty in learning to implement the state table, because the assignment of binary codes to the states reduces the problem to a familiar one of combinational design. Some time is spent discussing the relationship between the assignment and the efficiency of the implementation, even though this is not particularly important if the combinational logic is implemented with storage. The approach used is an informal version of the structural analysis of Hartmannis, in which on attempts to recognize independent submachines.

Finally, a return is made to the state diagram model to study its character independent of implementation. The first outstanding feature of finite state machines is that their reduced representation is unique -- i.e. the minimal-state representation for a process is canonic, thus permitting a notion of machine equality. In order to develop the students' insight into this model, studies are made (quickly) of the gedanken experiments and (not so quickly) of the language of regular expressions.

Concurrently, in the autumn quarter, the students do a sequence of eight weekly laboratory "exercises". These are "cookbook" labs, in the sense that the student is fairly constrained about what he is to do, but there is a good deal of design effort involved as well. The first laboratory exercise concerned with the electrical behaviour of switching devices, and is actually the only laboratory concerned with the fact that the devices happen to be electronic. The students tour several "stations" in which simple experiments are set up to illustrate (1) determination of the switching threshold, (2) transfer characteristics of inverters, (3) effects of loading, and (4) propagation delay measurements. From this exercise the students seem to learn all they need to know about electrical measurement techniques and the electrical characteristics of the logic elements.

Next follows a sequence of three exercises on the logical behaviour of gates and the design of cominational nets. The last of these three exercises introduces "wired-logic" and the use of "buses". The fifth laboratory on pulse and timing circuits is particularly difficult for the computer science students, but here they develop some mastery of the use of an oscilloscope. The remaining laboratory exercises are concerned with clocked sequential circuits -- both as implemented formally from a state diagram and heuristically designed circuits based on counters and shift registers. The grading in the laboratory in the autumn quarter is on the basis of reports written by the students on their "experiences" in each exercise.

Winter Quarter -- Machines with Storage

Beginning in the winter quarter, the coupling between the classroom and the laboratory activities becomes weaker, and is based more on ideas and concepts than on particular techniques. The central theme of the classwork during the winter quarter is machines with storage. Physical storage services are discussed, ranging from punched cards through integrated circuit stores. The physical mechanisms involved for this wide variety of storage devices are in each case traced to the form of stored energy or eneryy barrier employed to separate the stored 0 and 1. The students' understanding of the physical nature of the store makes it very easy for them to appreciate, for example, that a periodic store such as a disc or shift register may be very advantageously in a situation with complementary periodicity, but has a significant latency when used as a random-access store. Also, the student can compare the physical addressing mechanism used (i.e. looking at the output of a disc at the right time versus special decoding to look in the right place) with the terminal characteristics of the storageddevice. A start is made by explaining addressless mechanisms -- stacks and queues -- and continue into location- and content- addressed stores. As part of the review of content-addressed (or symbolically-addressed) storage, it is also natural to include a discussion of searching and sorting, including hash-coding.

It is interesting that at this point in the course the inevitable question arises: "Well, isn't this software you're talking about? I thought this class was about hardware!" This question provides an excellent opportunity to develop in the students a maximal degree of confusion about how to tell the difference between hardware and software.

Since the students by this time have a fairly thorough understanding of storage and also the finite-state model, the first "computing machine with storage" that is introduced is the Turing machine. Many of Professor Seitz colleagues criticize this pedigogical use of Turing machine, but it seems to be a very natural model for serial information processing, which contains no mysteries, no pragmatism, and uses a representation which is already familiar to the students. After showing the students some of the basic things that Turing machines can do, they are shown that a many-symbol Turing machine can be simulated by a machine with only two symbols, and that a machine with two tapes can be simulated by a machine with a single tape.

Next, a universal Turing machine, is designed which raises so many issues fundamental to the design of computers that only a few can be discovered. First of all, this is the first occasion in which the students have seen a machine deliberately designed to be "general-purpose". There is no trick to this; all one does is to design a special-purpose interpretive finite-state control, and pay dearly in terms of performance. This is also the first time the students have seen "program", in the form of a representation of the state table, intermixed with the "data", in the form of the simulated tape, intermixed in a store. By reorganizing the universal Turing machine to have two tapes, one to include the representation of the state table, the structure becomes remarkably like a microprogrammed computer, with two levels of control. Since at this point the students have all of the necessary preliminaries, it would seem a waste not to expose them to Turing's hypothesis -- namely, that any computation that can be performed by any machine can be performed by a Turing machine. The not-so-obvious consequences of this hypothesis, that even for Turing machines there are limits of effective computability, are demonstrated through the undecidability of the halting problem. It may seem a bit absurd to teach such an "advanced" idea at the sophomore level but Professor Seitz believes that it is an important concept that should be taught early.

This final topic of the "machines with storage" part of the course is the "separated date-control" schema model. This representation comes in two parts. First, there is a collection or registers (data storage) and operators (combinational elements) interconnected in a network. This picture corresponds roughly to the "block diagram" of a computer's registers data paths, and function logic, except that the schema may be left uninterpreted. Second, there is a representation for the control. At first a finite-state machine is used as the representation for the control, but later this is replaced by a precedence graph, in order to illustrate the possibilities for concurrent computation. In this latter case, it can be shown how a given schema may be non-determinate. This exposure to schemata presents to the student the essential ideas in style of design that Gordon Bell probably best represents, but without such a direct implementation of the schema as Bell's register transfer modules allow.

In the laboratory during the winter quarter, the students do three or more "miniprojects", each of which represents about three weeks of design, construction, and testing. The students may choose these three miniprojects from a collection of writeups which outline both the design task and suggest the overall organization. Some typical miniprojects are:

(1) A serial arithmetic-logical unit capable of register move, exchange, arithmetic, and logic operations, according to an "operation code" set in switches. The student gains experience here in timing logic and combinational circuits especially.

(2) A Scope character display, capable of displaying at least the digits 0-9 on the face of a scope.

(3) A bit-by-bit (successive approximation) alalog-to-digital converter, using a feedback principle from the comparison of a digital-to-analog converter output and the analog input to converge on the correct digital representation for the input voltage.

This kind of small project, whose complexity is fairly well legislated to be neither too overwhelming nor too simple, is an important bridge between the autumn quarter exercises and the full-scale projects of the spring quarter. The miniprojects are large enough to force the student to assume good habits of organization and documentation, but are small enough for him to complete in a short time.

The opportunity was missed last year to include miniprojects which employ storage, simply because there were no modules of addressable store available in the laboratory. A 512-bit high-speed store module is now available, and this year miniprojects to use them will be invented. These might include a sorting machine, an associate store employing hashing, or a small computer. For these projects, the students would use the store in a standard way -- to store data. Another class of projects employing the storage modules would be to use them as "soft" logic elements, for example to implement the finite-state machine which controls the head of a Turing machine.

Sprint Quarter -- Compromising with Reality

In the classroom in the spring quarter, material which purports to prepare the student for the realities of life is presented -- in the form of later courses and computers which he may meet. This part of the course is probably much like any number of other "computer organization classes", except that the students enter the study of real-world computers armed with an unusual mixture and amount of conceptual apparatus, and a certain amount of irreverence for the subject which has rubbed off on them from their heretic instructor. Actually, the pragmatic atmosphere of "real machines" is an opportunity to expose issues: microprogrammed control vs. wired control, serialism vs. parallelism, etc., etc.

Addressing is introduced first, from four addresses per instruction all the way to none at all. Also mentioned are machines whose storage is organized and addressable in smaller pieces than "words," i.e. in bits, digits, or characters. But it is within the most common medium of the fixed-word length single-address machine that the students go into any depth about effective addresses (indirection and indexing), instruction sets, input/output, machine language programming, and "how it works." At this point, however, they do not need to be shown "how it works;" they are good enough logicians that the internal operation of such a machine is utterly transparent. Nevertheless they are forced to design at least one machine at the schema level, and some parts of it at the logical level.

It is truly astounding how easily the students discover the essential issues and tricky details, having been prepared by two quarters of general tools.

In the laboratory, the students may devote the entire quarter to a single project of their own devising. These projects should be regarded (this year) as being somewhat unsatisfactory in that they were largely special-purpose machines, such as game-playing or display-generating machines. The limiting factor in these projects was the small amount of storage, which was available only as flip-flops, and not as addressable store. It is certain that the nature of the spring quarter projects will change dramatically next year to machines which are programmable. This development is crucial to tying these projects in more closely with the classroom studies in the computer organization and other courses.

The Role of the Laboratory

Having described his computer organization course, Professor Seitz tried to analyse the reaction of the students to the laboratory work. Professor Seitz was responsible at M.I.T. for a similar laboratory which was extremely popular amongst the Electrical Engineering students to whom it was offered. The appeal seemed to be that the students could regard what they were doing as relevant, or real, and had some opportunity to be original and creative in inventing and designing their projects. Of course, this entire idea would not work were it not for the fact that the equipment is good enough to support ambitious projects, and that the students were taught design techniques that work when applied to real equipment and practical design.

Professor Seitz regards the Utah lab as a major improvement over the M.I.T. experience. While the computer science students at Utah are somewhat naive about electronics, they have no difficulty thinking in logical and structural terms. The E.E. students, both at M.I.T. and those that take the laboratory at Utah, need to be encouraged toward logical approaches. They know all too well how the electronics works.

At this point, Professor Seitz opened a discussion on whether it was necessary to teach electronics in a course on digital machines. In the discussion which followed, it was generally felt that people have different reactions when presented with a "black box" device such as a logical gate. Some people would be quite happy to accept the definition of the "black box" on faith, whereas others would feel obliged to investigate the contents of the "black box" at greater levels of detail. A different point of view (that shared by Seitz) was that most people are willing to accept the "black box" provided that it was rational and consistent, and that its specifications were compact and clear, but otherwise would want to investigate further to discover the reason for any irrationalities.

Professor Seitz then posed the question of why we should provide a laboratory instead of a simulation program for student projects. [I posed this question with tongue-in-cheek, expecting people to consider a simulator to be a viable alternative to a laboratory. At M.I.T. a few years ago, several faculty members thought that the laboratory venture was silly, because one could simulate all of that. In spite of the fact that I have written logic simulators and used them extensively, I completely disagree with their use for educational purposes, and apparently so do most of the attendees of the meeting. It looks like simulators have slipped in popularity. The arguments against simulators presented in the discussion were very good ones. I had also two other arguments in mind -- that simulators are more expensive than laboratories; and the problem so well presented by Professor Wheeler, regarding synchronization. -CLS]. It was felt that simulation was in some sense a fraud, in that only those effects which are "of interest" are simulated. Another important advantage of the laboratory is the realism of dealing with real parts, which operate in real time, and occasionally suffer from real failures.

The Student-Laboratory Environment

The switching Circuits Laboratory is now housed in a clean, well-lit, former classroom with about 1200 square feet serves as a "factory" and parts storage room. All of the completed equipment is located in the main laboratory, and the students just take what they need. This coming year the laboratory will be open about 30 hours per week, plus two evenings, and the students may use the facilities at any open time at his convenience. This year there will be three or four teaching fellows to supervise the laboratory and help the students with their problems. [Figure 1 shows a general view of the laboratory.]

The students rightfully regard the laboratory as their own. They may work on their exercises and projects alone, or in groups of two or three, according to personal taste. There seems to be a rather unusual spirit of harmony and cooperation amongst the students, and a great deal of exchange of information and experiences between the students. The teaching fellows now come from the ranks of those who have taken this course (and done well in it), and their competence and experience is an indispensable asset. They are what makes the entire enterprise work, and also what makes the laboratory self-perpetuating.

Once the students begin project work, a very straightforward "evaluation" scheme is used: A completed project must work. When beginning a project, the student(s) receive a "project card" which labels their equipment. One of the teaching fellows must "sign-off" on the project after a demonstration of its proper operation. The project card then becomes the first page of the project report. Of course, one cannot assure that the design is clean or efficient --

although the project report is usually examined for this --, but it is felt that a good design job is its own reward, and the student who used poor design practices has certainly already been punished by the inevitable difficulty he has in making this project work.

Laboratory Equipment and Costs

All of the digital breadboard equipment used in the Switching Circuits Laboratory was designed and is manufactured at the University. Some of this equipment is pictured in Figure 2. The students plug taper-pin wires into one side of a "logic panel," and circuit cards into the other side. There are about 50 types of plug-in circuit cards, ranging from simple gate functions to complex modules such as addressable store, digital-to-analog converters, arithmetic elements, registers, switches, etc. The logic used is from the DTL/TTL family. Power is distributed around the laboratory on standard polarized plugs and sockets from two 5 volt, 50 ampere power supplies.

The investment in laboratory equipment, including benches and chairs, oscilloscopes and other instruments, is about $80,000, all of which has come from University (non-government) funds earmarked for instructional purposes. About $50,000 of this money has been spent for the special laboratory equipment developed at Utah, indicating that there is a substantial saving if the laboratory were bootstrapped on an existing electronics laboratory. The switching circuits laboratory is adequate for about 100 students at this point,* which means that about $800 per student has been invested. If this is amortized over 5 years, this means a rather substantial outlay of some $160 per student per year. This is a stiff price, but it is thought unlikely that it can be done so effectively for much less.

*Our registration for the autumn quarter 1971 is 108 students, nearly double that of last year. The registration in the computer organization class turned out to be 215.

Summary (by CLS)

What should be taught? I believe in teaching material of some enduring value.** When it comes to "computer organization," I think this means that the most worthwhile subjects are those that prepare the student with the tools and concepts that allow him to appreciate in a critical way the present computer technology, while arming him to change it if he can.

How should it be taught? My own best experiences as a teacher and student have been in design and project-oriented environments.

Why should "computer organization" be a part of "computer science" studies? First of all, it is an opportunity to expose the student to the favorable consequences of relevant formalism and "good practice." Switching theory is now a "respectable discipline," and has achieved a coherent methodology that programming is just beginning to apporach.

For most students: My objective is to show them how the mechanisms they will use operate, but in such a way as to blur in their minds the distinctions between programming, designing information machines, or designing other practical information-handling structures (such as the languages they use or the institutions they work in). For a few students: I hope that the will pursue "computer organization" somewhat as a specialization.

Final Discussion

The first point in the discussion concerned the similarity between programming and machine design. The need for good practice and a clean approach to the problem were equally important in both. However, the formalisms of machine design

**I believe that the phrase "enduring value" was used some years ago to describe a goal of the Electrical Engineering curriculum at M.I.T.

have as yet no parallels in the programming world. [I am forced simply to disagree with this statement. First of all, I really cannot see so clearly as others apparently can why the formalisms of machine design do not apply also to programming. Even a casual reading of last year's report of the Newcastle meeting shows that in fact this is happening <u>now</u>. Similarly, isn't it true that machine designers have adopted much of the methodology of programming? -CLS].

The discussion then turned to consider the relative merits of the "top-down" and "bottom-up" approaches. The motivation for using the "top-down" approach is that the students can exploit the programming background. However, it was felt that "bottom-up" approaches are equally plausible because of the need to give the learner a notion of what he is building on.

Another point raised in favour of having a laboratory is that some theoretical principles are not sound in practical situations. [My experience is that the laboratory sure keeps you honest in your lectures! -CLS].

Figure 1

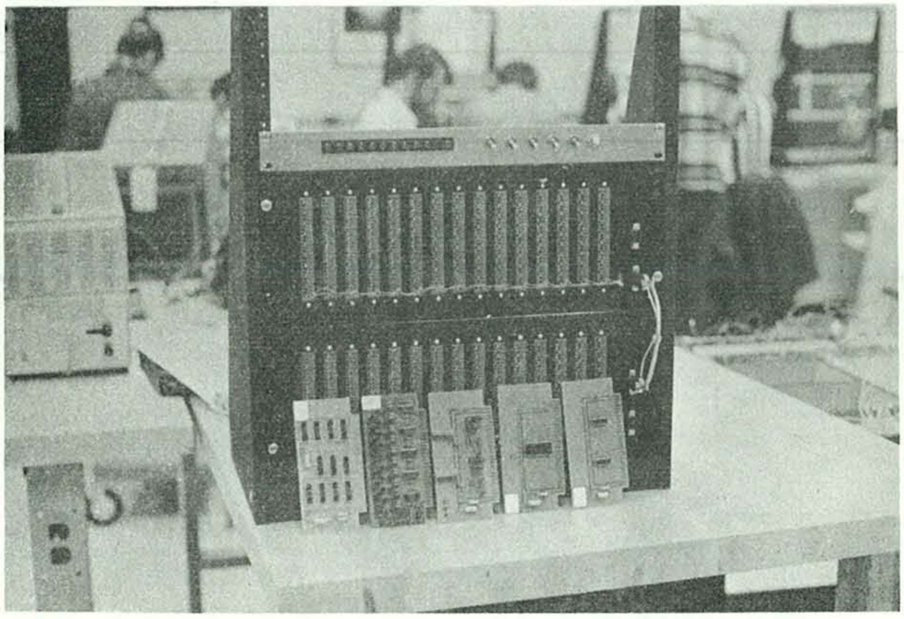
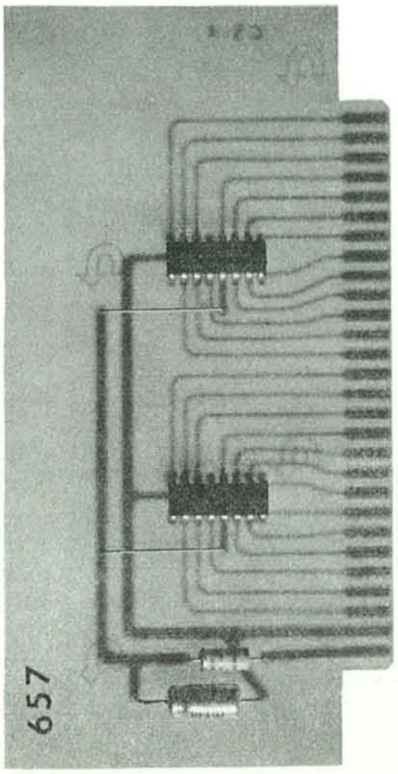
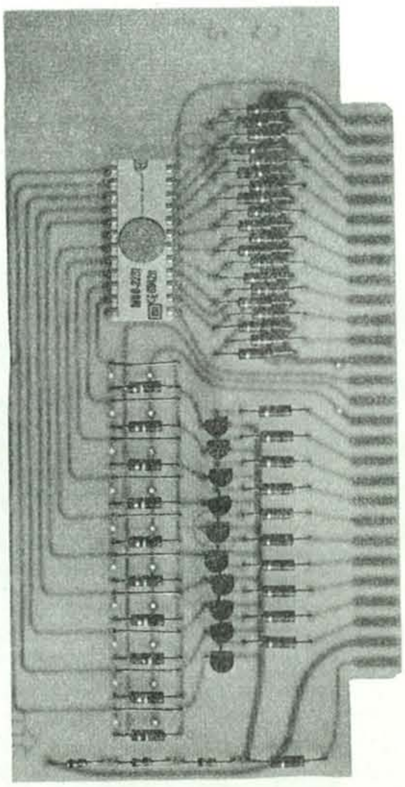
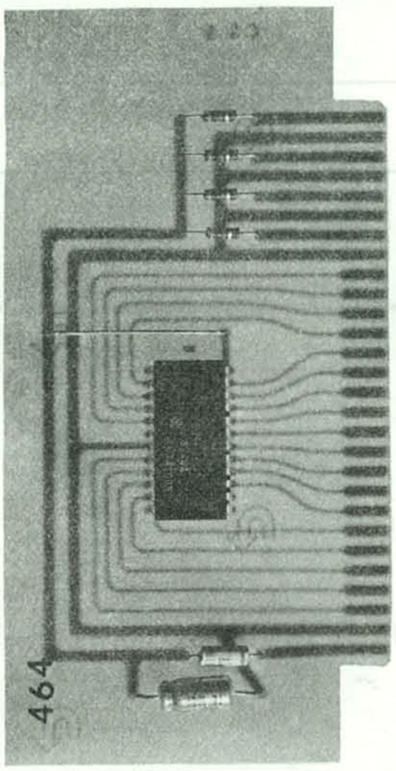
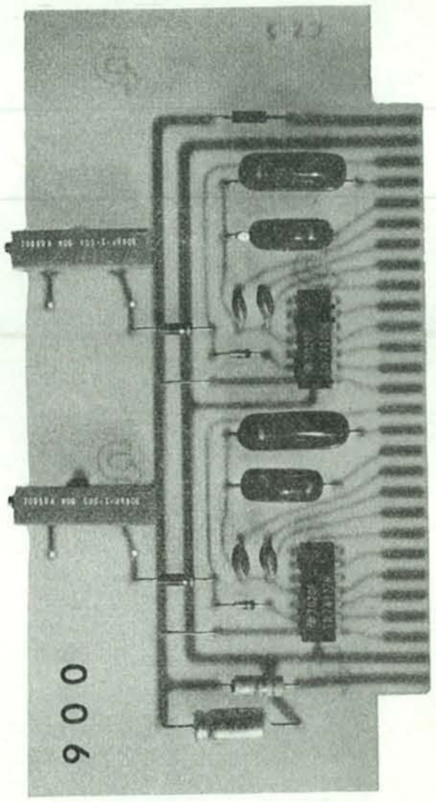


Figure 2









Circuits

Back of Logic Design Unit