COMPUTER PROGRAMMERS AT LRL

A somewhat alarming public-service advertisement which recently decorated Bay Area buses and billboards featured an array of cogs, wheels and wires plus an ominous question: "When this circuit learns your job, what are you going to do?"

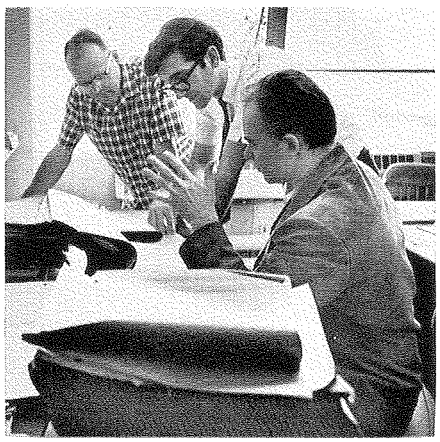
The *Chronicle's* Herb Caen, never one to resist a challenge from man or machine, volunteered several suitably spunky replies ("Become a circuit breaker," "Blow a fuse," etc.), but probably the best response to the question and its implications is the one that is being chosen by an increasing number of young college students today: Become a computer programmer, and be the one who teaches the electronic circuits *their* jobs in the automated world round the corner.

A New Profession

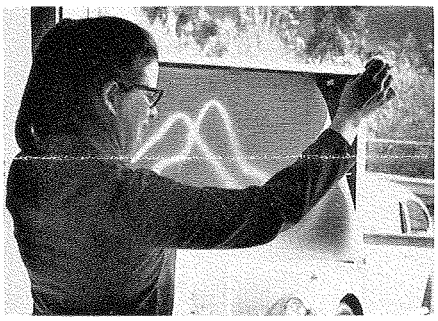
The vigorous growth of the new profession of computer programming over the past decade becomes clear when you look at LRL employment statistics. As recently as ten years ago, the job classification simply didn't exist at the Laboratory. In the early fifties, when computers began to come into general use, programming (in the difficult machine-language mode) was done by scientists or by specially trained mathematicians provided by the computer manufacturers. The first year in which programmers appeared on LRL employment rolls was 1958. By early 1961, soon after the LARC was installed in Livermore, the number of programmers at the Lab had already grown to 120. Today, LRL employs more than 200 programmers (about 90 in Berkeley, 120 in Livermore), and an active recruiting program keeps that figure rising steadily.

Two Myths

In the eyes of the general public, the new profession has become surrounded by two well-developed, and mutually contradictory, myths. One myth, based on the arcane mathematical vocabulary to be found in the field, and its close relationship with the advanced sciences, holds that computer programming is a formidably difficult business suited only to bespectacled mathematical wizards. The opposing myth holds that any housewife with a modicum of common sense can learn to be a programmer by attending a three-week short-course and doing her homework conscientiously. Both myths, curiously enough, turn out to be true—simply because the profession is wide and varied enough to fit almost any definition you care to make. Some simple programming *can* be done with only minimal training; some, on the other hand, requires extensive knowledge not only of programming techniques but also of complex scientific subject-matter.



WEAPONS DEVELOPMENT programming pool, headed by Harry Nelson (l.) is one of Livermore's largest. Above, Harry checks output data with programmer John Levan (r.) and A Division scientist Howard Wilson.



CONVERSION OF RAW DATA to computer language begins at a sunny window in Berkeley's Building 55, where programmer Ruth Hinkins traces a curve onto graph paper from a photograph.

UNDER THE ENIGMATIC EYE of computer-produced Mona Lisa, Bud Wirsching discusses current work with one of his "clients"—M.E.'s Glenn Moxon. Bud heads programming pool for Livermore's engineering groups.

Who Are They?

Programmers in general (and perhaps LRL programmers in particular) tend to be a mixed bag of people—as unstereotypable a group as you're likely to find anywhere. This is especially true of those who have been in the profession since its earliest days. When most of these people went to school, computer programming as a profession, or as a topic for study in a university, was unheard of, and so they took their degrees in everything from animal husbandry to French.

Among the younger programmers, however, it is becoming more common to find someone who actually majored in the subject at school (UC and many other universities now offer majors in computer science) and who decided upon programming fairly early in his academic career. Another noteworthy aspect of the profession is the large number of women in it—in fact, programming is beginning to shape up as the number-one new job frontier for college women. It's well-paying, challenging, and, in many cases, it offers the flexible hours so important to working wives and mothers.

A Young Group

The typical LRL programmer—if there is such a thing—is in his late twenties or early thirties. He majored in math at a regular four-year college or university, and came to the Laboratory directly after receiving his B.A. or B.S. Often, he may be continuing his studies during his off hours, either as a regularly enrolled graduate student or through selected job-related training courses. He may see programming as his lifetime career, or he may regard it as a stepping stone to related professions in mathematics or computer engineering.

LRL's Ed Schoonover, who as head of Livermore's Computer Systems Operations Section (SOS) supervises the training of most new programmers at that site, feels that a logical mind, common sense, and patience with detail are the essentials of a good computer programmer. Loren Meissner, Schoonover's counterpart in Berkeley's Math & Computing Group, agrees, but adds that in his experi-

ence the best programmers seem to be the kind of people "who enjoy solving puzzles." Meissner feels that the outstanding programmers on his staff are by no means the ones who make the fewest errors; instead, they're the ones who have the knack of finding original and creative ways to solve problems. Livermore's Hans Bruijnes, who supervises a staff of "systems development" programmers, adds that "willingness to work 25 or 26 hours a day" can come in handy too.

Training

Both Berkeley and Livermore hire programmers with no previous working experience, but suitable background *is* required. Usually, this means a major in mathematics or computer science with an excellent grade-point average—"about the same," says Meissner, "as the standards for admission to graduate school." Occasionally, however, both Meissner and Schoonover say they may bend the rules a bit to hire someone with little formal academic background in mathematics but an obvious aptitude for and interest in the programming game.

At LRL Berkeley, new programmers are usually assigned to the programming pool in the Mathematics and Computing Division, headed by Kent Curtis. (Three Berkeley groups, however—Alvarez Physics, Data Handling, and Administrative Data Processing—have their own independent programming staffs.) In Livermore, all programmers are members of the Computation Division, headed by Sidney Fernbach.

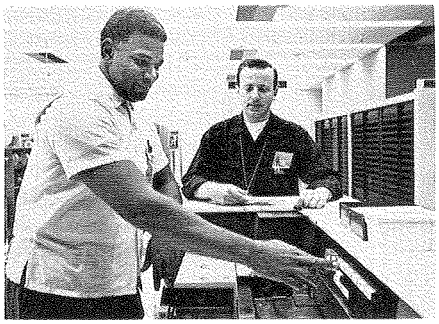
At both sites, informal on-the-job training is the rule. Livermore, however—faced with the unavoidable three-to-six-month clearance delay before new employees can begin work on classified problems—utilizes this time by putting newcomers through a set course of instruction, which includes programmed-training manuals, a FORTRAN short-course, lectures, and slides. Berkeley, which doesn't have the clearance problem, prefers to let newcomers plunge right in under the close supervision of a senior programmer.

Specialization

Training usually begins with assigning the beginner a small subroutine that is part of a larger problem. Meissner and Schoonover agree that it generally takes from six months to a year of on-the-job training before a beginner is ready to work independently on advanced problems. And, of course, the need for learning never stops. Even after a programmer has mastered the essentials of his profession, he still has to hustle to keep up with new machines, new developments in the computer-language field, and new problems in his scientific specialty.

After training, programmers are assigned to a particular subject-oriented section within the programming pool. In

SCIENTIFIC PROGRAMMER Pat Paoli, of Berkeley, prepares input data for OMNIBUS, a program used in the design of magnets.



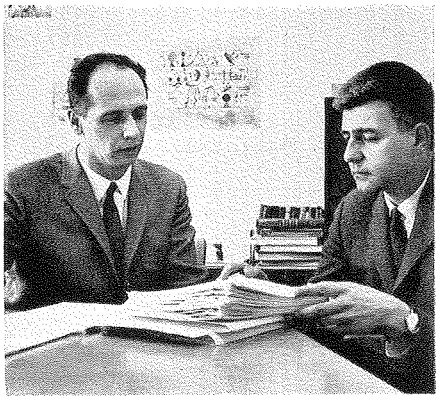
SYSTEMS PROGRAMMERS Bob Hughes (l.) and John Ranelletti, of Livermore's FORTRAN compiler group, feed revisions into a card reader.

PUBLISHER is a program that generates printed reports from Livermore's 6600 and 3600 computers. Checking a recent PUBLISHER print-out are two of the program's authors, Virginia Smith and Hans Bruijnes, of the Systems Group.

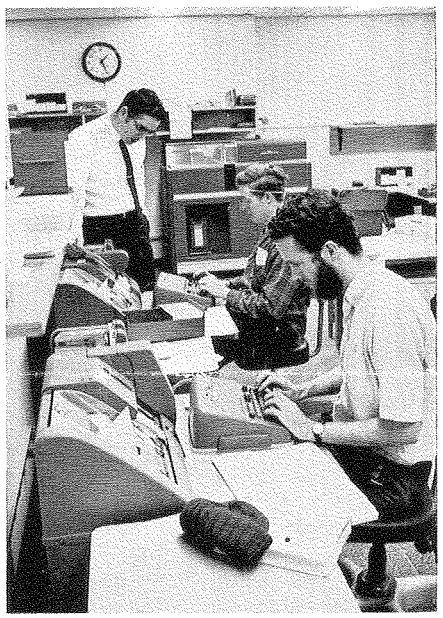
Berkeley, there are six of these groups: a systems branch and five "applications" sections: data acquisition, biological applications, accelerator study, bubble-chamber data analysis, and general applications. Each section has a group leader and about five to ten programmers. In Livermore, there are three basic groups: systems, systems operations (which handles training, information, and consulting and library services) and scientific. The scientific group, in turn, breaks down along programmatic lines in support of each of Livermore's research efforts.

Scientific Expertise

The degree of scientific expertise required of a programmer varies according to the individual and his subject area. At the very least, the programmer needs a



MEDICAL RECORDS of Donner Laboratory diabetes patients are coded by Berkeley programmer Mark Horovitz (l.) and Donner research scientist Henry Stauffer. Horovitz and Stauffer are working together on developing techniques for automated, computer-controlled retrieval of significant data from medical charts.



READY ROOM of Berkeley's 50B computer area offers self-service key punch and other facilities for the Hill's programmers. L. to r., above, are Loren Meissner, Eric Beals, and Bill Dempster.

working familiarity with the vocabulary of the discipline and an understanding of the problem that his program is designed to deal with. Many programmers, however, go beyond these basic requirements to take courses in their subject area, read the literature, and become extremely knowledgeable in their fields. A good example of such individuals is Berkeley's Mark Horovitz, who, as leader of Math & Comp's biological applications group, has an office at the Donner Laboratory, spends part of his time training graduate students in the use of computers, and works closely with Donner scientists on such projects as the automated processing of medical records, mathematical models of biological processes, and automated approaches to problems like chromosome counting.

Continued on Page 10

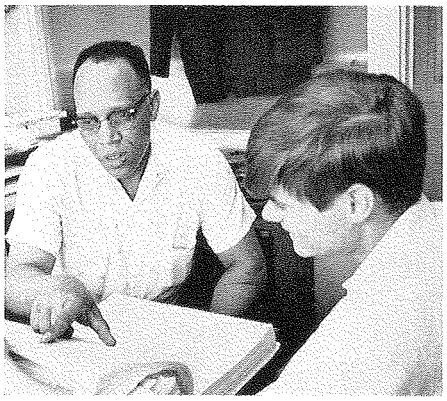
Computer Programmers at the Laboratory . . .

Continued from Page 7

Systems programmers—the people who write the programs that actually run the computers—are a particularly specialized, highly trained, and harried group—sometimes considered the prima donnas of the field. Hans Bruijnes, who supervises a staff of 20 systems programmers in Livermore, attributes this image to the intense pressures in systems programming. "If a scientific programmer makes an error," Bruijnes points out, "nothing is lost except his own time. If a systems programmer makes an error—especially in a major program—the whole computer could screech to a halt and *nobody's* work would get done." Another thing complicating life for the systems programmers, says Bruijnes, is the fact that so many LRL computers are "Serial Number 1"—first of their kind. "We have a secret little fantasy," Bruijnes admits wistfully, "where we sneak the next super-duper, first-of-its-kind computer into our offices in the dead of night without telling a soul, and keep it all to ourselves until we've got the systems programs working perfectly." Needless to say, Bruijnes' fantasy isn't going to come true, and systems programmers are going to have to continue to do their best while jostling for time with other impatient computer users.

Schedule

Many programmers tend to be night owls by inclination, and a late-night visitor to programming offices is sure to find at least a few busy at their desks or hanging around the computer waiting for output. Schedules tend to be flexible—often depending on the availability of computer time. This tradition, in part, has its roots in the days when the man-computer relationship was more direct than it is now, and programmers operated somewhat as "on-line" components of data-processing systems.



PACKAGE—the comprehensive bubble-chamber track reconstruction program—is "like a second language" to Alvarez Group programmer Cecil Draper (l.). Here, he reviews data with physicist Dick Hubbard.

Sid Fernbach, head of Livermore's Computation Division, recalls those days—and looks ahead to the future of man-machine relationships—in a recent article published in *Computer Group News,* a publication of the Institute of Electrical and Electronics Engineers.

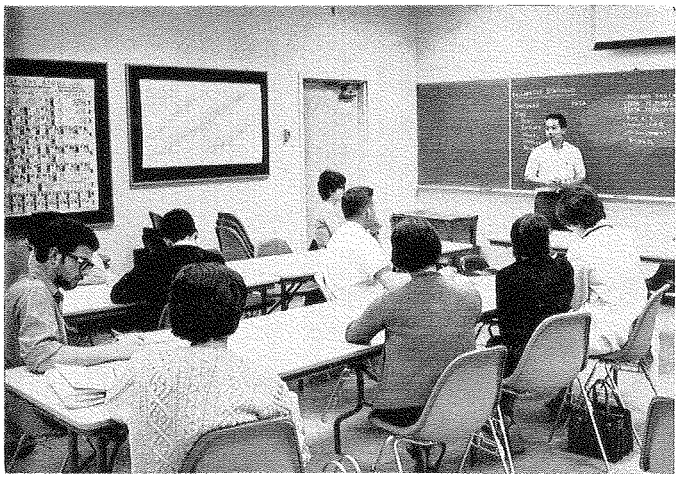
"Early machines," says Fernbach, "were slow enough that a man sitting at the console of the computer could keep up with the various steps of the calculation. Knowing what he had programmed the computer to do, he knew at each instant of time whether the computer was proceeding properly. He could stop the calculator and request some intermediate results on the console typewriter; he could add, delete, or modify data. Very often the computer stood idle while some changes were being made on punched cards or paper tape, or even magnetic tape. Thus, the man had a very close relationship with the machine; in a sense he was 'on-line' with it . . .

"As users of the equipment learned to fill up all available time and newer, faster systems became available, the mismatch of the human with the computer system became somewhat intolerable. Monitor or batch processing systems were designed so that problems were grouped to utilize every possible unit of available time. The user was thrown 'off-line.' He had to prepare cards, which, when they left his hands, concluded his interaction with the machine . . . This again became intolerable, especially when the system became so loaded that the user could not get back on the computer for a considerable length of time, perhaps eight hours or even longer . . ."

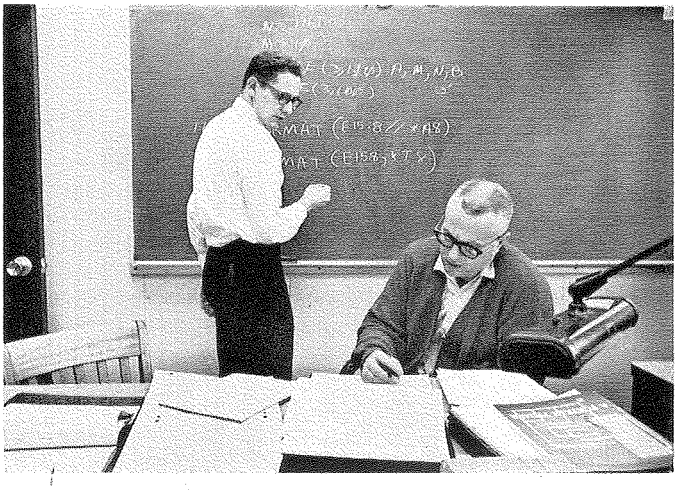
"On-Line" Again

In recent years, Fernbach concludes, the growth of time-sharing techniques and the ability of computers to run several jobs simultaneously has permitted the user to move back into a more natural "on-line" relationship with the computer. Through systems like Livermore's Octopus, a programmer can sit at his desk yards, even miles, from the computer and communicate with it via telephone or teletype lines. Even direct voice "conversations" between man and computer may soon become feasible.

These developments, plus the growth of new programming languages to replace the FORTRAN and ALGOL of today and the extension of automated techniques to many new areas of science, industry, and management, are indications that the great man-computer dialogue is really just getting started. It's a safe bet that the programming profession—the vital link in that dialogue—is in for a decade of changes and challenges at least as exciting as the last.



INSTRUCTION IN ELEMENTARY PROGRAMMING is one of the many services offered by LRL's programming groups to other staff members. This one, taught by Berkeley's Carl Quong, is intended for graduate students, technicians, and other "occasional" computer users.



LEARNING BY DOING is the method that has proved most successful in the training of new programmers. Don von Buskirk (l., above) has been a programmer for about one year, still works closely with his supervisor, John Burk of Livermore's Systems Operations group.