gineer for the MIT Computer-Aided Design Project. His activities include:

- directing development of an integrated system of program development, integration, and testing tools for production of large-scale military programs;
- creating and directing the development of the APT system for automatic programming of numerically controlled machine tools, now an international standard; and
- leading the MIT Computer-Aided Design Project, including research and development in language theory, language design, generalized compiler construction, computer graphics hardware and software, and design applications.

In 1969, SofTech was founded. Ross was president from 1969 to 1975 and is now chairman emeritus of the board of directors.

He was an organizer and participant in the NATO Software Engineering Conferences in Germany (1968) and Italy (1969).

Ross received the Joseph Marie Jacquard Award from the Numerical Control Society in 1975. The paper "Theoretical Foundations for the Computer-Aided Design System" coauthored with J.E. Rodriguez, then senior vice president of SofTech, received the Prize Paper Award at the AFIPS Spring Joint Computer Conference in 1963. The paper "AED Approach to Generalized Computer-Aided Design" shared the Prize Paper Award at the ACM 20th Anniversary National Meeting in August 1967. In 1980 Ross received the Distinguished Service Award from the Society for Manufacturing Engineers.

Martin Greenberger.* Greenberger was born in Elizabeth, N.J., in 1931. He received BA, MA, and PhD degrees in applied mathematics from Harvard University. Before joining the MIT faculty in 1955, Greenberger was manager of Applied Science Cambridge, the IBM group that cooperated with MIT in the establishment and operation of the MIT Computation Center. Greenberger is the coauthor of the 1962 book, *Microanalysis of Socioeconomic Systems — A Simulation Study*.

John McCarthy. Born in Boston on September 4, 1927, John McCarthy holds a BS in mathematics from CalTech (1948) and a PhD in mathematics from Princeton (1951). At Princeton he was the first Proctor fellow and later Higgins research instructor in mathematics. He has also been associated with the faculty of Dartmouth College and MIT, and is now Charles M. Pigott professor of computer science and director of the Artificial Intelligence Laboratory at Stanford University.

McCarthy participated in the development of Algol 58 and Algol 60, and created the Lisp programming system while at MIT, where, with Marvin Minsky, he organized and directed the Artificial Intelligence Project. In the same time period he was a significant instigator of the work on time-sharing at MIT.

He was chosen as the 1971 ACM Turing Award recipient and awarded the 1988 Kyoto Prize for outstanding accomplishments in advanced technology.

J.A.N. Lee (editor). "JAN" has been active in computer-related studies since the mid-1950s and has served as the director of computing and head of department of three major North American universities. He has spent the past four years on leave from his position at Virginia Polytechnic Institute and State University, serving as the director of the Institute for Information Technology of the Virginia Center for Innovative Technology. His principal research interests have been in the design and implementation of compilers, testing techniques, and the history of computing. Lee serves as the editor-in-chief of the *IEEE Annals of the History of Computing* and is a past vice president of ACM.

Robert F. Rosin (interviewer). Rosin earned a BS in economics, politics, and engineering at MIT (1957) and MS and PhD degrees in communication sciences at the University of Michigan (1960 and 1964). He taught computer science at the university level until 1976 and has since worked in the computer-telecommunications industry. He is currently vice president and principal architect of Enhanced Service Providers, Inc., a start-up company that is developing software to integrate voice, text, and image information. He also provides consulting services in the areas of communication and information system architecture and education.

Rosin is a founding editor of the *Annals*, and he served on the program committee for the first ACM SIGPLAN History of Programming Languages (HOPL) Conference held in 1978. He will repeat that role for the second conference (HOPL-II), planned for 1993.

* Adapted from Greenberger, 1962.[25]

Time-Sharing at MIT

Introduction

J.A.N. LEE

Time-sharing as an operating systems implementation methodology is, in many minds, synonymous with MIT [illegible] the names of Fernando Corbató and Robert Fano. Even [illegible]ugh the technique existed and was implemented in pre-1960 special-purpose systems, numerous other implementations arose in the late 1960s in conjunction with interactive computing, so it is almost inconceivable today for a vendor to deliver a mainframe computer system without some measure of time-sharing and interaction. CTSS and Project MAC are clearly identifiable pioneer efforts. In 1988, the Laboratory for Computer Science at MIT invited participants to a two-day symposium to celebrate the 25th anniver[illegible]y of the laboratory. That celebration — the MIT Computer Science Research Symposium, held October 26 and [illegible] at MIT's Kresge Auditorium — took the form of an exemplary set of presentations on the future of computer science.

The activities of the laboratory in 25 years, since its inception as Project MAC by Robert Fano, have covered a wide variety of topics far broader than just time-sharing and interactive computing:

- Computer-aided design
- Time-sharing
- Mathlab and Macsyma
- Artificial intelligence
- The development of editors (TECO, Runoff, Script)
- Lisp
- Theory
- Labware
- Parallel Computing
- Education
- The emergence of companies — Prime Computer, SofTech, Thinking Machines, and many others
- Abstraction and specification

Building on the good fortune of having several participants in the early development of CTSS and Project MAC present in Cambridge, the *Annals* sought the cooperation of the laboratory through the good offices of Michael Dertouzos and Albert Meyer to record interviews with the attending pioneers. These interviews were transcribed by Kellie Ross of the Virginia Center for Innovative Technology, and edited by Robert F. Rosin and J.A.N. Lee. Each participant was also afforded the opportunity to edit his presentation, comment on the statements of others, and provide footnotes of further explanation. False starts, blind ends, and unfocused asides have been deleted from the transcript, and some minor reordering of the proceedings has been undertaken by the editors. The original audiotapes are in the possession of the MIT archives. Copies of these tapes and the original transcript, unedited (but one-pass corrected for accuracy), have been deposited with the Charles Babbage Institute of the University of Minnesota, Minneapolis.

The interviews pointed to numerous supportive articles and reports that have documented the technological history of the development of both CTSS and Project MAC, as well as the subsequent design of the Multics system. We have attempted in this special issue of the *Annals* to supplement the interviews, which concentrated on the more human and administrative side of the history, by reprinting (and publishing for the first time in many cases) excerpts from the relevant documents, together with our commentary, which we hope will provide the bridges in time and activity. Unfortunately, during the development of this special issue, one of our pioneers died — Joseph C.R. (Lick) Licklider. We dedicate this special issue to his memory. We only met him briefly, but it was clear that the participants regarded him with great love and admiration for his contributions and support.

Definition

Like terms such as "compiler" and "operating system," the term "time-sharing" has been applied to a set of developments over a period of 30 years, differing in 1985 from an early application of the term in 1955. Initially the term was applied[2] to a mechanism by which the processing power of the central computer was shared between a number of different activities of differing speeds, especially activities related to the operation of resources such as input-output devices. By this means, the processing time of a central processor that would be wasted in waiting for a slower device to complete its task is assigned to another task that is ready to execute. This technique was used in the SAGE system to implement the air defense system and later the SABRE airline reservation system. With the development of the CTSS system by Corbató and others at MIT, and subsequent extension to Multics through Project MAC, the technique for the implementation of an interactive multiterminal system became synonymous with "interactive computing."

Thus time-sharing is a methodology for scheduling a computer so several users can interact with it simultaneously and without apparent interference from each other. Although time-sharing was initially perceived by many as a programming convenience for debugging,[33] the perception soon was extended to include the provisions of a wide variety of on-line services and the availability of a large central memory shared among the user community. In the strict sense of the terms, it is clear that "interactive comput-

1058-6180/92/0100-0001$03.00© 1992 IEEE

Scanned

1959	1960	1961	1962	1963	1964
Time-sharing					
McCarthy memo	Licklider publishes "Man-Computer Symbiosis" Strachey publishes UNESCO paper	McCarthy Utility Model Long Range Study report PDP-1 given to MIT	BBN system operational		PDP-1 TSS by Dennis
		CTSS			
		Version 1 by Corbató on IBM 709	First paper by Corbató et al. IBM 7090 arrived	CTSS copied onto MAC IBM 7094 Summer session	Kludge operates on CTSS
				Project MAC	
			Fano and Licklider ride train together Proposal submitted to ARPA	Multics request for proposal sent out	GE acccpted as Multics vendor

1965	1966	1967	1968	1969	1970
Segmentation concept by Dennis		Widespread implementation on a variety of equipment			
					CTSS turned off three years later (in 1973)
Bell Labs joins Multics FJCC presentation on Multics				First operational Multics at MIT Bell Labs withdraws	Unix available at Bell Labs

Time lines for time-sharing, CTSS, and Project MAC.*

ing" was one goal of Project MAC and "time-sharing" was one way to implement that goal. Indeed, in those days using a single large computer was the only economical way.

Shortly, with Project MAC and projects elsewhere, those two terms became confused, and "time-sharing" came to include "interactive computing" for many people. Later, of course, "interactive computing" was also achieved through personal computers. As personal computers have been used as the remote, intelligent terminals for mainframe systems, the terminology has become even more confusing. Here we generally use the term "time-sharing" to apply to that style of computer usage which uses the technology to implement "interactive computing." In the next article, we briefly trace the claims to the early uses of the term time-sharing and the different meanings implied by each user.

Closed shop versus open shop

In the mid-1950s relatively reliable commercial computers were becoming available and high-level languages were moving the task of programming from the hands of the professional programmer to the problem poser, a move which was accompanied by the concurrent need for more accessible computer systems. Meanwhile, systems were becoming smarter, and professional programmers were available to construct ever larger programs. Computing center staffs were being pressured to make more efficient use of the still expensive equipment.

One solution to this problem was the emergence of monitors, supervisory systems, and, eventually, operating systems.[32] Under this scenario, programs and data for a collection of jobs were prerecorded (often off-line) on magnetic tape. Then, under the supervision of a monitor program, the jobs would run sequentially without operator or programmer interaction until they terminated or, as was often the case, encountered an error condition. This processing mode was really a means of improving the effectiveness of a closed-shop operation. Although it also cut down on the idle or wait time of the processor, it did little to improve the ability of users to debug their programs. The turnaround time for a closed-shop operation was reduced very little for the users, and the problem was aggravated as larger and more ambitious programs were attempted.

Conversely, in an open-shop operation, where users operated the computer themselves in a manner we would now refer to as "personal computing" and very definitely in an "interactive" mode, the ratio of available clock cycles to utilized cycles was extremely high. The percentage of wait time for user actions to useful operation often exceeded 80 to 90 percent during debugging activities. Open-shop users did not want to give up their "personal, interactive computing," and the administrators of closed-shop systems were unwilling to give up the advantages of machine efficiency gained through the use of operating systems.

One other element that contributed to the attractiveness of time-shared, interactive terminal systems was the interfacing of computers with communication systems — especially through the public telephone system. As early as 1940 at a mathematics conference at Dartmouth College, the Stibitz relay computer at the Bell Telephone Laboratories was operated remotely by a single user a few hundred miles away.[29] Communications were obviously essential to the development of the massive SAGE system, which involved several remote control sites with multiple users, each at a specially designed terminal interacting independently with information displayed on cathode-ray tubes. Similarly, the IBM and American Airlines development of the SABRE system involved communications with hundreds of terminals distributed geographically.

These mid-1950s systems provided special-purpose services to the users through a single software system where the terminals were used for data entry and output rather than for program development, debugging, and execution. What was different in John McCarthy's 1959 proposal[33] (an influential but unpublished internal memorandum to Philip Morse, director of the MIT Computation Center) was the vision of a computer used independently by different persons for entirely different programs. Perhaps the major step that McCarthy suggested was to optimize the use of human time rather than computer time. The optimal use of computer time was to be transparent to the user.

Technology

While the feasibility of the basic concepts of time-sharing, communications, and interaction had been verified in individual projects, two technology improvements provided the key steps toward practicality. These were the replacement of the vacuum tube by the transistor and the availability of large-capacity memories. Transistors provided the reliability of systems that were to be inserted into an environment which was tantamount to a public utility. Large-scale memories provided a medium for the storage and rapid retrieval of the large variety of software packages used by individual users, as well as their personal programs and data. The timing was just right! McCarthy[33] proposed the key hardware modifications to an IBM 709 computer that would allow time-shared debugging by multiple users. Corbató said that "it was McCarthy's early advocacy of time-sharing which inspired much of the interest in developing such systems."

In the history of time-sharing and interactive computing at MIT, two major events followed the early developments of the 1950s: The demonstration of the Compatible Time-Sharing System (CTSS) in 1961 and its extension into the Multics system within Project MAC. These occurred against the backdrop of planning and design politics, interactions with computer vendors to supply the needed hardware features, and attempts to maintain the viability of the ongoing services of the Computation Center. ■

* Adapted from the time line for the Project MAC 25th anniversary by Peter Elias.

Claims to the Term "Time-Sharing"

J.A.N. LEE

time-share - v - To interleave the use of a device for two or more purposes.

time-shared system - n - A specific system in which available central-computer time is shared among several jobs as directed by a scheduling plan or formula.

time-sharing - adj - 1. The apportionment of intervals of time availability of various items of equipment to complete the performance of several tasks by interlacing. (Contrasted with multiprogramming.) 2. The use of a device for two or more purposes during the same overall time interval, accomplished by interspersing the computer component actions in time.

Charles J. Sippl, Computer Dictionary and Handbook, Howard W. Sams, New York, 1965

The emergence of a term is not always coincidental with the initial development of that particular technology or system. Even in the case of computers, the word initially used was "calculator." The word "computer" was originally applied either to humans who did computations using mechanical calculators[10] or to the computational element of a bomb aiming device.* In 1966 Fano and Corbató[21] published a paper in *Scientific American* which attributed the origin of time-sharing to Christopher Strachey in a 1960 paper[43] presented originally at the 1959 UNESCO conference. This resulted in a letter to the editor from Robert Bemer[6] in which he claimed that his article in the 1957 issue of *Automatic Control*[4] was the original reference to time-sharing. Bemer also pointed out that it was reported in the *Journal of the Franklin Institute*[5] that he had used the word in a presentation in the same year. In turn, Robert Dodds wrote to Bemer quoting a 1949 letter in which he described his (unnamed) invention:

> A system consisting of the following: one or more input-output devices...each conveying its information to a common location distant from one or all of the input-output devices; one or more devices which generate the electrical impulses that convey information and that control the various operations of the several devices: one or more scanning or gating devices to segregate the information originating from the several input-output devices...[18]

This description appears to accentuate the confusion between the process of "time-sharing" that Bemer described and "interactive computing" that was prevalent in 1966. At the time of Bemer's Franklin Institute lecture, time-sharing was already being implemented as part of the SAGE system,[2] and shortly thereafter IBM and American Airlines developed the SABRE system for on-line passenger scheduling and ticketing. When Strachey[43] used the word in 1959, adding the concept of on-line, he added also debugging for programmers. Donald Knuth wrote to Strachey 25 years later:

> John McCarthy thinks he invented [time-sharing]. So does K.D. Tocher. I [have] read that Lord Halsbury says...that the idea of time-sharing was "very much in the air." I hope you'll have time to write me a letter explaining the history of time-sharing as you see it...

Strachey responded:

> The paper I wrote called "Time-sharing in Large Fast Computers" was read at the first (pre IFIP) conference at Paris in 1960 [sic]. It was mainly about multiprogramming (to avoid waiting for peripherals) although it did envisage this going on at the same time as a programmer was debugging his program at a console. I did not envisage the sort of console system which is now so confusingly called time-sharing. I still think my use of the term is the more natural.

Campbell-Kelly[3] also quoted Strachey as saying:

> "Time-sharing in Large Fast Computers" was probably the first paper to discuss time-sharing and multiprogramming as we know them. It is a matter of history that the time-sharing idea became extremely fashionable in the middle sixties and dominated much of the work on computing at the time. When I wrote the paper in 1959, I, in common with everyone else, had no idea of the difficulties which would arise in writing the software to control either the time-sharing or multi-programming. If I had, I should not have been so enthusiastic about them.

Strachey filed an application for a patent on time-sharing in 1959, which was eventually granted (British patent 924672) in 1963. Campbell-Kelly suggests that "the idea of interactive time-sharing is plainly there in embryo." In the same year (1959), John McCarthy wrote a memo to Philip Morse* suggesting modifications to the IBM 709, which was being contributed by IBM to the Computation Center, in order to develop "a time-shared operator program."

In 1962 McCarthy suggested that the concept, if not the term, goes even further back: "The subject was also touched on in the lectures by Kemeny and Perlis. In 1945 Vannevar Bush[8] discussed a system for personal information retrieval called Memex, which probably requires a computer system of the kind that I am going to discuss for its realization."

It is apparent that there have been two major uses of the term corresponding to two major periods of the history, with two different spellings (with or without the hyphen):

- Before 1960, time sharing described a method of implementing multiprogramming (Astrahan and Jacobs,[2] Bemer,[4-6] Dodds,[18] and Strachey[43]).
- After 1960, time-sharing described a technique by which interactive computing was developed (McCarthy[33] and Fano and Corbató[21,22]). ■

* London *Times*, July 8, 1944.

* The date on the memorandum is January 1, 1959; there has been a suggestion that the McCarthy memorandum was actually written in 1960, the misdating being a common error in the early days of a new year. See the next section for the content of the memorandum and McCarthy's comments on this suggestion.

1058-6180/92/0100-0016$03.00 © 1992 IEEE

The Beginnings at MIT

Two key elements were necessary to implement time-shared, interactive computing: interfaces with communications facilities, and a machine design that supported interrupts, memory protection, and a large fast-access external store. Each of these elements was feasible in 1959, as demonstrated by George R. Stibitz in 1940 and by the SAGE development in 1957.

The first public demonstration of remote operation of a digital computer and of computer-telephone communications was conducted as part of the 1940 annual meeting of the Mathematical Association of America. Problems were entered into a teletypewriter located in McNutt Hall on the Dartmouth campus in Hanover, N.H. They were then transmitted via standard Bell System telecommunications facilities to the Bell Telephone Laboratories in New York City, where they were solved on-line by the BTL Model I Complex Number Calculator. The results were immediately returned on-line via the same telecommunications link to the teletypewriter located on the Dartmouth campus, 250 miles away.

Stibitz, who was behind both the demonstration and the digital computer used in it, was then a mathematician on the technical staff of the Bell Telephone Laboratories. (Now he is professor emeritus of physiology at the Dartmouth Medical School.)

Stibitz describes the ability to do remote computing as a natural and integral part of the design of his system. It was natural and "no big thing" to demonstrate this device at Dartmouth by leaving the computer where it was and operating it remotely via a teletypewriter link, rather than going to all the trouble and expense of moving the machine itself from New York to Hanover. After all, teletypewriters in other departments in BTL had already tapped into the system on occasion to get their computations done. The length of the teletype line, whether it was 25 feet or 250 miles, involved no major change in operational techniques or concept.[19]

In their 1957 paper, "SAGE — A Data Processing System for Air Defense" (*Proc. EICC* © 1957 IRE (now IEEE)), R.R. Everett, C.A. Zraket, and H.D. Bennington[20] described the working of the SAGE system and not only used the term "time-sharing" but also provided their own description:

> The central computer performs air-defense processing in the following manner (see figure 1 [reproduced on the facing page]). The buffer storage tables, the system-status data, and the system computer program are organized in hundreds of blocks — each block containing from 25 to 4000 computer words. A short sequence-control program in the central computer's core memory transfers appropriate program and data blocks into core memory, initiates processing, and then returns appropriate table blocks (but never programs) back to the drum. To take advantage of the in-out break feature, operation of each air-defense routine is closely coordinated with operation of the sequence-control program so that programs and data are transferred during data processing.
>
> By *time-sharing* [emphasis added] the central computer, each of the air-defense routines is operated at least once every minute — many are operated every several seconds. One interesting feature is that the frequency of program operation is locked with real time rather than allowed to vary as a function of load; during light load conditions the sequence-control program will often "mark time" until the real-time clock indicates that the next operation should be repeated. Such synchronization with real time simplifies many of the control and input-output functions without causing any degradation in system performance.

Both of these demonstrations showed the feasibility of the concept, but it took an environment (at MIT) and a brilliant mind (John McCarthy) to put two and two together into a far-reaching system.

Digital computation* began at MIT in 1947 with the Whirlwind I computer. Whirlwind was four or five times faster than its predecessors by virtue of its excellent electronic design and parallel operation. By 1953, core memory, developed by Jay Forrester and his associates at MIT, yielded another factor of two. The standards of speed and reliability set at that time have now come to be expected of modern computing equipment.

While Whirlwind satisfied the needs of many MIT users, the pressures of special requirements led to the acquisition of other computers as well. In 1955 the Instrumentation Laboratory obtained an IBM 650 computer, and in 1956 the Naval Supersonic Laboratory installed a Bendix G15 computer for processing wind tunnel data.

In 1956, IBM provided an IBM 704 to be located at MIT with one shift per day for the use, without charge, of educational and research projects at MIT, and a similar amount of time for the use of universities and colleges in the New England area. The Computation Center, under the direction of Philip M. Morse, was formed for the administration and efficient use of this facility. Subsequently, the TX-0 computer was loaned to the Electrical Engineering Department by Lincoln Laboratory.

The need for computer capacity at MIT continually increased. In 1960 the Computation Center IBM 704 was replaced by the more powerful IBM 709 computer, and several smaller computers also were installed by various MIT groups. Even more ambitious expansion was scheduled for the immediate future.

* Of course in the field of analog computation, Vannevar Bush had led the world with the development of the Differential Analyzer.

1058-6180/92/0100-0018$03.00 © 1992 IEEE

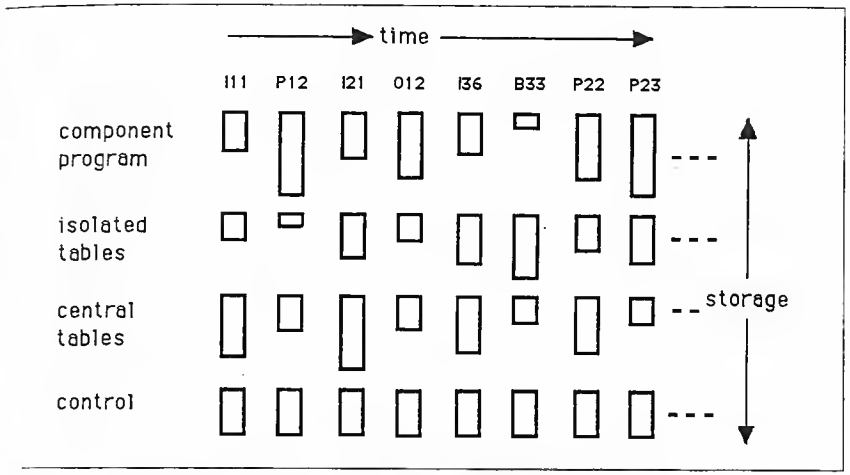

Figure 1. Dynamic program operation.[20] Reproduced with permission (© 1957 IRE (now IEEE)).

Reminiscences on the History of Time-Sharing

John McCarthy

At the time of our interviews with the CTSS and Project MAC pioneers, we invited John McCarthy to join the group which provided this record following the 25th anniversary celebrations of the founding of the Laboratory of Computer Science. McCarthy was unable to extend his stay in the Cambridge area but graciously wrote his own memories of this era.

I remember thinking about time-sharing at the time of my first contact with computers and being surprised that this was not the goal of IBM and all the other manufacturers and users of computers. This might have been around 1955.

By time-sharing, I meant an operating system that permits each user of a computer to behave as though he were in sole control of a computer, not necessarily identical with the machine on which the operating system is running. Christopher Strachey may well have been correct in saying in his letter to Donald Knuth* that the term was already in use for time-sharing among programs written to run together. This idea had already been used in the SAGE system. I do not know how this kind of time-sharing was implemented in SAGE. Did each program have to be sure to return to an input polling program or were there interrupts? Who invented interrupts anyway?** I thought of them, but I do not believe I mentioned the idea to anyone before I heard of them from other sources.

My first attempts to do something about time-sharing were in the fall of 1957 when I came to the MIT Computation Center on a Sloan Foundation fellowship from Dartmouth College. It was immediately clear to me that time-sharing the IBM 704 would require some kind of interrupt system. I was very shy of proposing hardware modifications,

* See Strachey's response to Knuth in the previous section.

** Alan Scherr reported that the patents on interrupt-driven I/O are held by three IBM researchers.

John McCarthy's 1959 memorandum

To: Professor P.M. Morse
From: John McCarthy
Subject: A Time-sharing Operator Program for our Projected IBM 709
Date: January 1, 1959

Introduction

This memorandum is based on the assumption that MIT will be given a transistorized IBM 709 about July 1960. I want to propose an operating system for it that will substantially reduce the time required to get a problem solved on the machine. Any guess as to how much of a reduction would be achieved is just a guess, but a factor of five seems conservative. A smaller factor of improvement in the amount of machine time used would also be achieved.

The proposal requires a complete revision in the way the machine is used, will require a long period of preparation, the development of some new equipment, and a great deal of cooperation and even collaboration from IBM. Therefore, if the proposal is to be considered seriously, it should be considered immediately. I think the proposal points to the way all computers will be operated in the future, and we have a chance to pioneer a big step forward in the way computers are used. The ideas expressed in the following sections are not especially new, but they have formerly been considered impractical with the computers previously available. They are not easy for computer designers to develop independently since they involve programming system design much more than machine design.

The format of the original memo was changed for reproduction here.

A quick service computer

Computers were originally developed with the idea that programs would be written to solve general classes of problems and that after an initial period most of the computer time would be spent in running these standard programs with new sets of data. This view completely underestimated the variety of uses to which computers would be put. The actual situation is much closer to the opposite extreme, wherein each user of the machine has to write his own program and that once this program is debugged, one run solves the problem. This means that the time required to solve the problem consists mainly of time required to debug the program. This time is substantially reduced by the use of better programming languages such as Fortran, Lisp (the language the Artificial Intelligence Group is developing for symbolic manipulations) and COMIT (Yngve's language). However, a further large reduction can be achieved by reducing the response time of the computation center.

The response time of the MIT Computation Center to a performance request presently varies from 3 hours to 36 hours depending on the state of the machine, the efficiency of the operator, and the backlog of work. We propose by time-sharing, to reduce this response time to the order of 1 second for certain purposes. Let us first consider how the proposed system looks to the user before we consider how it is to be achieved.

Suppose the average program to be debugged consists of 500 instructions plus standard subroutines and that the time required under the present system for an average debugging run is 3 minutes. This is time enough to execute 7,000,000 704 instructions or to execute each instruction in the program 14,000 times.

Most of the errors in programs could be found by single-stepping or multiple-stepping the program as used to be done. If the program is debugged in this way, the program will usually execute each instruction not more than 10 times, 1/1400 as many executions as at present. Of course, because of slow human reactions the old system was even more wasteful of computer time than the present one. Where, however, does all the computer time go?

At present most of the computer time is spent in conversion (SAP-binary, decimal-binary, binary-decimal, binary-octal) and in writing tape and reading tape and cards.

Why is so much time spent in conversion and input output?

1. Every trial run requires a fresh set of conversions.
2. Because of the slow response time of the system it is necessary to take large dumps for fear of not being able to find the error. The large dumps are mainly unread, but nevertheless, they are necessary. To see why this is so, consider the behavior of a programmer reading his dump. He looks at where the program stopped. Then he looks at the registers containing the partial results so far computed. This suggests looking at a certain point in the program. The programmer may find his mistake after looking at not more than 20 registers out of say 1000 dumped, but to have predicted which 20 would have been impossible in advance and to have reduced the 1000 substantially would have required cleverness as subject to error as his program. The programmer could have taken a run to get the first register looked at, then another run for the second, etc., but this would have required 60 hours at least of elapsed time to find the bug according to our assumptions and a large amount of computer time for repeated loading and re-runnings. The response time of the sheet paper containing the dump for any register is only a few seconds, which is OK except that one dump does not usually contain

(continued on the following page)

especially as I did not understand electronics well enough to read the logic diagrams. Therefore, I proposed the minimal hardware modification I could think of. This involved installing a relay so that the 704 could be put into trapping mode by an external signal. It was also proposed to connect the sense switches on the console in parallel with relays that could be operated by a Flexowriter.

When the machine went into trapping mode, an interrupt to a fixed location would occur the next time the machine attempted to execute a jump instruction (then called a transfer). The interrupt would occur when the Flexowriter had set up a character in a relay buffer. The interrupt program would then read the character from the sense switches into a buffer, test whether the buffer was full, and if not return to the interrupted program. If the buffer was full, the program would store the current program on the drum and read in a program to deal with the buffer.

It was agreed (I think I talked to Dean Arden only) to install the equipment, and I believe that permission was obtained from IBM to modify the computer. The connector to be installed in the computer was obtained.

However, at this time we heard about the "real-time package" for the IBM 704. This RPQ (Request for Price Quotation was IBM jargon for a modification to the computer whose price was not guaranteed), which rented for $2,500 per month, had been developed at the request of Boeing for the purpose of allowing the 704 to accept information from a wind tunnel. Some element of ordinary time-sharing would have been involved, but we did not seek contact with Boeing. Anyway, it was agreed that the real-time package, which involved the possibility of interrupting after any instruction, would be much better than merely putting the machine in trapping mode. Therefore, we undertook to beg IBM for the real-time package. IBM's initial reaction was favorable, but nevertheless it took a long time to get the real-time package — perhaps a year, perhaps two.

It was then agreed that someone, perhaps Arnold Siegel,* would design the hardware to connect one Flexowriter to the computer, and later an installation with three would be designed. Siegel designed and built the equipment, the operating system was suitably modified (I do not remember by whom), and a demonstration of on-line Lisp was held for a meeting of the MIT Industrial Affiliates. This demonstration, which I planned and carried out, had the audience in a fourth-floor lecture room and me in the computer room and a rented closed-circuit TV system. Steve Russell, who worked for me, organized the practical details, including a rehearsal. This demonstration was called time-stealing, and was regarded as a mere prelude to proper time-sharing. It involved a fixed program in the bottom of memory that collected characters from the Flexowriter in a buffer while an ordinary batch job was running. It was only after each job was run that a job that would deal with the characters typed in would be read in from the drum. This job would do what it could until more input was wanted and would then let the operating system go back to the batch program. This worked for the demonstration, because at certain hours the MIT Computation Center operated a batch stream with a time limit of one minute on any job.

Around the time of this demonstration, Herbert Teager came to MIT as an assistant professor of electrical engineering and expressed interest in the time-sharing project. Some of the ideas of time-sharing overlapped some ideas he had while on his previous job, but I do not remember what they were. Philip Morse, the director of the Computation Center, asked me if I was agreeable to turning over the time-sharing project to Teager, since artificial intelligence was my main interest. I agreed to this, and Teager undertook to design the three-Flexowriter system. I'm not sure it was ever completed. There was a proposal for support for time-sharing submitted to the National Science Foundation, and money was obtained. I do not remember whether this preceded Teager, and I do not remember what part I had in preparing it or whether he did it after he came. This should be an important document, because it contains that year's conception of and rationale for time-sharing.*

Besides that, IBM was persuaded to make substantial modifications to the IBM 709 to be installed at the MIT Computation Center. These included memory protection and relocation, and an additional 32,768 words of memory for the time-sharing system. Teager was the main specifier of these modifications. I remember my surprise when IBM agreed to his proposals. I had supposed that relocation and memory protection would greatly slow the addressing of the computer, but this turned out not to be the case.

Teager's plans for time-sharing were ambitious and, it seemed to many of us, vague. Therefore, Fernando Corbató undertook an "interim" solution using some of the support that had been obtained from NSF for time-sharing work. This system was demonstrated some time in 1961, but it was not put into regular operation. That was not really possible until the ARPA support for Project MAC permitted buying a separate IBM 7090.

Around 1960 I began to consult at BBN on artificial intelligence and explained my ideas about time-sharing to Ed Fredkin and J.C.R. Licklider. Fredkin, to my surprise, proposed that time-sharing was feasible on the PDP-1 computer. This was DEC's first computer, and BBN had the prototype. Fredkin designed the architecture of an interrupt system and designed a control system for the drum to permit it to be used in a very efficient swapping mode. He convinced Ben Gurley, the chief engineer for DEC, to build this equipment. It was planned to ask NIH (National Institutes for Health) for support, because of potential medical applications of time-sharing computers, but before the proposal could even be written, Fredkin left BBN. I took technical

* Siegel had engineered a Flexowriter input device on Whirlwind for Doug Ross' Data Reduction Project in 1956.

* Regrettably, we have been unable to locate this proposal, but a later report on the work is included in this issue.

information enough to get the entire program correct.

Suppose that the programmer has a keyboard at the computer and is equipped with a substantial improvement on the TX-0 interrogation and intervention program (UT3). (The improvements are in the direction of expressing input and output in a good programming language). Then he can try his program, interrogate individual pieces of data or program to find an error, make a change in the source language and try again.

If he can write the program in source language directly into the computer and have it checked as he writes it, he can save additional time. The ability to check out a program immediately after writing it saves still more time by using the fresh memory of the programmer. I think a factor of 5 can be gained in the speed of getting programs written and working over present practice, if the above mentioned facilities are provided. There is another way of using these facilities which was discussed by S. Ulam a couple of years ago. This is to use the computer for trial and error procedures where the error correction is performed by a human adjusting the parameters.

The only way quick response can be provided at a bearable cost is by time-sharing. That is, the computer must attend to other customers while one customer is reacting to some output.

The problem of a time-sharing operation system

I have not seen any comprehensive written treatment of the time-sharing problem and have not discussed the problem. This treatment is certainly incomplete and is somewhat off-the-cuff. The equipment required for time-sharing is the following.

a. Interrogation and display devices (Flexowriters are possible but there may be something better and cheaper);
b. An interrupt feature on the computer; we'll have it;
c. An exchange to mediate between the computer and the external devices. This is the most substantial engineering problem, but IBM may have solved it.

In general the equipment required for time-sharing is well understood, is being developed for various advanced computers, e.g., Stretch, TX-0, Metrovich 1010, Edsac 3. I would not be surprised if almost all of it is available with the transistorized IBM 709. However, the time-sharing has been worked out mainly in connection with real-time devices. The programs sharing the computer during any run are assumed to occupy prescribed areas of storage, debugged already, and to have been written together as a system. We shall have to deal with a continuously changing population of programs, most of which are erroneous.

The major problems connected with time-sharing during program development seem to be as follows:

1. Allocating memory automatically between the programs. This requires that programs be assembled in a relocatable form and have a preface that enables the operator program to organize the program, its data, and its use of common subroutines.
2. Recovery from stops and loops. The best solutions to these problems require:
   a. Changing the stop instruction to trap instructions. This is a minor modification to the machine. (At least it will be minor for the 704.)
   b. Providing a real-time alarm clock as an external device.
3. Preventing a bad program from destroying other programs. This could be solved fairly readily with a memory range trap which might not be a feasible modification. Without it, there are programming solutions which are less satisfactory but should be good enough. These include:
   a. Translations can be written so that the programs they produce cannot get outside their assigned storage areas. A very minor modification would do this to Fortran.
   b. Checksums can be used for machine language programs.
   c. Programming techniques can be encouraged which make destruction of other programs unlikely.
   d. There is an excessive tendency to worry about this point. The risk can be brought down to the present risk of having a program ruined by operator or machine error.

Summary

a. We may be able to make a major advance in the art of using a computer by adopting a time-sharing operator program for our hoped-for 709.
b. Such a system will require a lot of advance preparation starting right away.
c. Experiments with using the Flexowriter connection to the real-time package on the 704 will help, but we cannot wait for the results if we want a time-sharing operator program in July 1960.
d. The cooperation of IBM is very important, but it should be to their advantage to develop this new way of using a computer.
e. I think other people at MIT than the Computation Center staff can be interested in the systems and other engineering problems involved.

charge of the project as a one-day-a-week consultant, and Sheldon Boilen was hired to do the programming. I redesigned the memory extension system proposed by DEC and persuaded them to build the modified system instead of the two systems they were offering, but fortunately had not built. I also supervised Boilen.

Shortly after this project was undertaken, DEC decided to give a PDP-1 to the MIT Electrical Engineering Department. Under the leadership of Jack Dennis,* this computer was installed in the same room as the TX-0 experimental transistorized computer that had been retired from Lincoln Laboratory when the TX-2 was built. Dennis and his students undertook to make a time-sharing system for it. The equipment was similar, but they were given less memory than the BBN project had. There was not much collaboration.

My recollection is that the BBN project was finished first in the summer of 1962, but perhaps Corbató remembers earlier demonstrations of CTSS.** I left for Stanford in the fall of 1962. I had not seen CTSS, and I believe I had not seen Dennis' system operate either. BBN did not operate the first system and did not even fix the bugs. They had few computer users and were content to continue the system whereby users signed up for the whole computer. They did undertake a much larger follow-on project involving a time-shared PDP-1 that was installed in Massachusetts General Hospital, where it was not a success. The computer was inadequate, there were hardware and software bugs, and there was a lack of application programs — but mainly the project was premature.

At the same time that CTSS, the BBN system, and the Electrical Engineering Department systems were being developed, MIT had started to plan for a next-generation computer system. The management of MIT evidently started this as an ordinary university planning exercise and appointed a high-level committee consisting of Philip Morse, Albert Hill, and Robert Fano to supervise the effort. However, the actual computer scientists were persuaded that a revolution in the way computers were used was called for. The lower level committee was chaired by Teager, but after his ideas clashed with those of everyone else, the committee was reconstituted with me as chairman. The disagreement centered around how ambitious to be and whether to go for an interim solution. Teager wanted to be very ambitious, but the rest of us thought his ideas were vague. He wanted MIT to acquire an IBM 7030 (Stretch) computer as an interim solution. As it turned out, acquiring a Stretch would have been a good idea.

Our second report to MIT proposed that MIT send out a request for proposals to computer manufacturers. On the basis of the responses, we would then ask the government for the money. The RFP was written, but MIT stalled, perhaps for two reasons. The first reason was that our initial cost estimates were very large for reasons of conservatism. Second, IBM asked MIT to wait, saying that they would make a proposal to meet MIT's needs at little or no cost. Unfortunately, the System/360 design took longer than IBM management expected, and along about that time, relations between MIT and IBM became very strained because of the patent lawsuit about the invention of magnetic core memory.

As part of the stall, president Stratton proposed a new study with a more thorough market survey to establish the demand for time-sharing among MIT computer users. I regarded this as analogous to trying to establish the need for steam shovels by market surveys among ditch diggers, and I did not want to do it. About this time George Forsythe invited me to come back to Stanford with the intention of building a Computer Science Department, and I was happy to return to California.

In all this, there was not much publication. I wrote a memo to Morse dated January 1, 1959, proposing that we time-share our expected "transistorized IBM 709." It has been suggested that the date was in error and should have been 1960. I do not remember now, but I believe that if the memo had been written at the end of 1959, it would have referred to the 7090, because that name was by then current. In that memo I said the idea of time-sharing was not especially new. I do not know why I said that, except that I did not want to bother to distinguish it from what was done in the SAGE system with which I was not very familiar.

Most of my argumentation for time-sharing was oral, and when I complained about Fano and Corbató crediting Strachey with time-sharing in their 1966 *Scientific American* article, Corbató was surprised to find my 1959 memo in the files. Their correction in *Scientific American* was incorrect, because they supposed that Strachey and I had developed the idea independently, whereas giving each user continuous access to the machine was not Strachey's idea at all. In fact, he did not even like the idea when he heard about it.

Teager and I prepared a joint abstract for an ACM meeting shortly after he arrived, and I gave a lecture in an MIT series called *Computers and the World of the Future*.[34] In this lecture I referred to Strachey's paper "Time-sharing in Large Fast Computers"[43] given at the 1959 IFIP Congress in Paris. I had read the paper carelessly, and supposed he meant the same thing as I did. As he subsequently pointed out, he meant something quite different that did not involve a large number of users, each behaving as though he had a machine to himself. As I recall, he mainly referred to fixed programs, some of which were compute bound and some input-output bound. He did mention debugging as one of the time-shared activities, but I believe his concept involved one person debugging while the other jobs were of the conventional sort.

My 1959 memo advertised that users generally would get the advantage of on-line debugging. However, it said noth-

* Earl Pugh of the Electrical Systems Laboratory, which was given the *responsibility* for the PDP-1, did the original installation and operation of the PDP-1. [Note added by Doug Ross during review.]

** See Time Line, pp. 14-15.

ing about how many terminals would be required and where they would be located. I believe I imagined them to be numerous and in the users' offices, but I cannot be sure. Referring to an "exchange" suggests that I had in mind many terminals. I cannot now imagine what the effect was on the reader of my failure to be explicit about this point. I'm afraid I was trying to minimize the difficulty of the project.

The major technical error of my 1959 ideas was an underestimation of the computer capacity required for time-sharing. I still do not understand where all the computer time goes in time-sharing installations, and neither does anyone else.

Besides MIT's NSF proposal, there ought to be some letters to IBM and perhaps some IBM internal documents about the proposal, since they put more than a million dollars worth of equipment into it. Gordon Bell discusses DEC's taking up time-sharing in the Bell/Newell book,* but I do not recall that they discuss Ben Gurley's role. Fredkin and perhaps Allan Kotok would know about that.

After I came to Stanford, I organized another PDP-1 time-sharing project. This was the first time-sharing system based on display terminals. It was used until 1969 or 1970 for [Patrick] Suppes' work on computer-aided instruction.

Excerpts from "Man-Computer Symbiosis"[28]

About the same time McCarthy was writing his memorandum to Morse on the mechanics of time-sharing, J.C.R. Licklider was investigating the concept of man-computer symbiosis, a premonition of man-machine cooperation which is even today not wholly achieved. The following selection from Licklider's "Man-Computer Symbiosis" is reprinted with permission from IRE Transactions on Human Factors in Electronics, *March 1960 (© 1960 IRE (now IEEE)).*

Man-machine symbiosis is an expected development in cooperative interaction between man and electronic computers. It will involve close coupling between the human and electronic members of the partnership. The main aims are 1) to let computers facilitate formulative thinking as they now facilitate the solution of formulated problems, and 2) to enable man and computers to cooperate in making decisions and controlling complex situations without inflexible dependence on predetermined programs. In the anticipated symbiotic partnership, men will set the goals, formulate the hypotheses, determine the criteria, and perform the evaluations. Computing machines will do the routinizable work that must be done to prepare the way for insights and decisions in technical and scientific thinking. Preliminary analyses indicate that the symbiotic relationship will perform intellectual operations much more effectively than man alone can perform them. Prerequisites for the achievement of the effective, cooperative association include developments in computer time-sharing, in memory components, in memory organization, in programming languages, and in input and output equipment.

In one sense of course, any man-made system is intended to help man, to help a man or men outside the system. If we focus upon the human operator(s) within the system, however, we see that, in some areas of technology, a fantastic change has taken place during the last few years. "Mechanical extension" has given way to replacement of men, to automation, and the men who remain are there more to help than to be helped. In some instances, particularly in large computer-centered information and control systems, the human operators are responsible mainly for functions that it proved infeasible to automate. Such systems are not symbiotic systems. They are "semi-automatic" systems, systems that started out to be fully automatic but fell short of the goal.

It seems entirely possible that, in due course, electronic or chemical "machines" will outdo the human brain in most of the functions we now consider exclusively within its province.

It is often said that programming for a computing machine forces one to think clearly, that it disciplines the thought process. If the user can think his problem through in advance, symbiotic association with a computing machine is not necessary.

However, many problems that can be thought through in advance are very difficult to think through in advance. They would be easier to solve, and they could be solved faster, through an intuitively guided trial-and-error procedure in which the computer cooperated, turning up flaws in the reasoning or revealing unexpected turns in the solution. Poincaré anticipated the frustration of an important group of would-be computer users when he said "The question is not 'What is the answer?' The question is 'What is the question?'" One of the main aims of man-computer symbiosis is to bring the computing machine effectively into the formulative parts of technical problems. The other aim is to...bring computing machines effectively into the processes of thinking that must go in "real time," time that moves too fast to permit using computers in conventional ways.

Later Licklider was instrumental in not only initiating the concept of an extensive project which would have similar aims to those expressed in his 1960 paper, but also he was in a position to provide the funding for that work! His train ride with Robert Fano from Hot Springs, Va., to Washington, D.C. (probably aboard the "Crescent") gave them the opportunity to realize that ARPA needs could be met through the capabilities of the MIT group. In our interview with Licklider and Fano we asked them of their recollections of this trip.

Teager's recommendation for an IBM 7030[48]

The two studies that preceded the formalization of the MIT activity, which led first to the development of the Compatible Time-Sharing System (CTSS) and eventually to Project MAC, have been mentioned both by McCarthy in his recollections and later by the respondents in the interviews. The first study group report, written by Herbert Teager, recommended the acquisition of an IBM 7030 (Stretch), although Teager recognized that he was unable to obtain all the necessary data regarding its capabilities, and the support software was an unknown factor. Retrospectively, Teager's arguments were sound; his choice of the IBM Stretch was unfortunate.

MIT should obtain, within the next two to three years, an ultra large capacity computer, develop time-shared remote input-output facilities complete with display and graphical input capability, and begin an intensive effort to develop advanced, user oriented programming languages for this system. This policy would seem the best possible way for MIT to obtain a research facility to multiply the intellectual effectiveness of the experimental and theoretical research workers and teachers in all fields. The policy would at the same time provide a necessary experimental tool for frontier research in many fields involving physical simulation, information processing, and real-time experimental work, which would otherwise be unfeasible.

The machine should have sufficient capacity and be of sufficiently advanced design so that it would have a useful life of at least six years, before any further change need be contemplated.

Based upon presently-published specifications for existing machines, the IBM 7030, Stretch Computer is a suitable candidate for the recommended computer, by virtue of its speed, memory size, and overall capacity and capabilities.

Published data for Stretch, while not including some important, but presently unknown factors such as reliability and mean, error-free running time, does indicate that the commercial version of this machine will come very close to meeting its specifications. Several have been built, and programs are operating on this machine as the last of the "bugs" are being wrung out. Other announced machines are either unsuitable or else are in such an early stage of development that specifications and costs are too vague for an objective appraisal.

The final decision for the central computer should be made on a basis of its installation within a maximum period of perhaps *three years* [emphasis added]: a minimum set of speed, reliability, memory size, and overall capacity requirements; and finally the relative cost for such unit capacity. Presently unknown commercial machines might conceivably prove a better choice if they could meet these criteria, and in any event other manufacturers should be given a chance to bid.

In order to properly prepare for such a machine, it is believed that the machine design and programming specifications would have to be firm within a year at most. Before IBM or any other computer manufacturer can be directly approached at a high level for performance specifications and bids, it is believed that an early policy decision with respect to MIT's intentions is necessary.

The present trend towards additional separate computers would appear an unsatisfactory overall solution to MIT's problems in this area from a standpoint of both cost and overall performance. Far more serious, however, is the fact that the chance to provide the needed capacity in the form recommended by this Report may well become impossible in the face of the developments and acquisitions of these many divergent groups unless a decision is made very soon.

Making the capacity of a powerful machine more directly available to each user will multiply its effectiveness and eliminate the drawbacks of a large central machine.

Analyses based upon costs and characteristics of existing commercial machines made during the course of this study tend to show that computer capacity is far more economically provided by a single, very large capacity machine, rather than by many separate medium- to small-scale machines. In addition, there are vital research problems scattered in all fields of interest to the Institute which must have vast memory sizes and extremely high processing speeds if they are to be solved within feasible running times. Separate machines which may have either a vast memory or high speed are thus highly inefficient and overly costly both in total time and money for the solution of these problems. Providing a capacity of the type necessary can most economically be achieved by the rental or purchase of the largest, fastest available commercial machine such as the IBM Stretch computer. There is, at present, no other existing machine or combination of machines that can achieve comparable computation costs and rates.

Making the capacity of a powerful machine more directly available to each user will multiply its effectiveness, and eliminate the drawbacks of the use of a large central machine such as have been experienced at MIT. The MIT Computation Center is as well administered as any other comparable facility with a load of similar nature and magnitude. Nearly 400 distinct problems are active at the Center at any given time, and to fairly distribute the available time, amounting to less than half of that requested, among large numbers of anxious users, leads to difficulties. These become even greater as nearly 100 competing users attempt to use the machine on an average day. The net result from the user's viewpoint is excessive paper work to achieve inadequate time grants, long turnaround times between program submission and results, and thus, an overall slowdown in research. There would be little justification for an extremely high capacity central facility which compounded these shortcomings on a larger scale. Fortunately, however, there is strong technical evidence that it need not.

While some of the faculty members are aware of computers and programming developments with a potential application in their field, they are even more aware of the present usage difficulties. Programming for them, even in the so-called advanced languages, tends to be tedious, not easily applicable to their own work, and requiring knowl-

* More likely: C.G. Bell, J.C. Mudge, and J.E. McNamara. *Computer Engineering*, Digital Press, Bedford, Mass., 1978.

edge of far too many conventions and exceptions. Other obstacles are the uninterrupted hours required to program and debug before finally getting any results, in the face of an overcrowded faculty day, and an overworked facility. They cannot afford to wait for the presently necessary weeks of elapsed time between problem formulation and machine solution.

The difficulty of administering a large central machine can only be overcome if a time-shared central machine is brought out to many simultaneous users in the form of remote input-output "consoles" which have all the characteristics of a user's own personal computer with respect to access, a known available capacity, and a minimum of usage formalities. Such a console would provide far more capacity and capability than the user could afford for the same cost in a machine of his own. [See Figure 2.] Such capacity, in addition to speed, would allow the user to "program" in languages that are closely allied to that of his field, and in addition, eventually allow him to use pencil and paper should he dislike or be unable to use a keyboard to communicate his problem to the machine. It would also provide graphical display and a permanent record of all work; such capability would go a very long way towards attracting a sizable fraction of the faculty who could use computers to extreme advantage, but who cannot spare the protracted time requirements of present usage, as well as realizing the benefits of computers as a classroom aid.

During evenings, nights and weekends, the system described would have adequate capacity to run very large problems, which might require the total capacity of the central machine, or the work of groups which might not be compatible with or desirable for time-sharing. This would be a minor restriction upon such groups.

Although development programs are already underway for the high capability, low-cost console described, time-sharing techniques and remote input-output stations equipped with typewriters presently exist. A large part of the access problem described can be solved today with very little further development, and thus the entire program does not hinge on the possible gamble that to develop the high-capability languages and facilities might take longer than the indicated three-year time period. To provide the central facility will be costly, as will the programming effort to provide the needed languages, and to a lesser extent, the development and procurement of a large number of high-capability remote facilities. But it would seem the cheapest way of providing the needed capacity, and the only way of providing the necessary, close man-machine interrelationship which is felt vital for research in the years to come.

One of the most common complaints made by users in the course of the survey of all MIT research projects was that it was extremely difficult to find competent programmers, who could translate their desires into machine programs. This is a symptom of a much longer range problem, i.e., non-faculty usage, and calls for a longer range solution.

For any intellectual tool to be used, it must have a response time roughly comparable to that of the person using it; otherwise it may well slow down, rather than speed up, the overall rate of the man and machine. The net result will be that the user does not use the tool, even though it *could be* of material assistance to him.

It is believed that the relatively small percentage of the MIT faculty who are using computers directly, and the rather limited amount of such usage, is due primarily to the tedious nature of present day programming languages, and the inescapable fact that it requires a minimum of 2-4 weeks to get a moderately complex program written in the best available language to operate... For most senior researchers, this time lag provides a sufficiently strong barrier to keep them from using machines at all.

Much has already been made in this study of the fact that MIT faculty are not directly using computers but are rather working through middlemen, in the form of research staff and graduate student programmers. More will be made of the ineptitude of present languages, which results in many "debugging" and compiling runs before a problem is finally ready for the "production" which was its raison d'être. Due to present administration policies and computation loads, most of the total elapsed time in getting to production is spent in the presently *inevitable* [emphasis added] waits of turnaround times between submissions in the succession of reruns.

The net result is to force highly skilled personnel to remain idle for long periods of time, unless sufficient machine time is granted so that they are working on several problems at the same time, but having to constantly "shift mental gears" and rethink programs.

As their efficiency decreases (due to the long turnaround times and small time grants) so does morale, and not having the satisfaction of seeing their work come to early fruition, the best of them leave.

No study was made of this area, but from many personal contacts and discussions it is estimated that no group at MIT can expect to hire and keep good qualified programmers for much longer than about 1½ years. Many groups have suffered almost 100 per cent casualties inside of three years. This is not, it is believed, a question of salaries.

The effect of such a turnover rate, the constant loss of the best and most talented people, and the training of far less capable ones to take their place are highly detrimental and costly to the research work of MIT. This problem too, it is believed, can be alleviated.

In the past, many cogent and pressing arguments have been presented for separate small- to medium-scale computers for use by many of the research groups around Tech. Although a variety of reasons have been given for this form of private enterprise, the basic reasons it is believed are ones of access and possession. A researcher in general regards a computer as a tool (ranging in power from an intelligent assistant to a desk calculator, depending upon his inclination and familiarity) and, as with all other tools, feels that its effectiveness is measured in part by his degree of access to it, and the degree to which he can count on using it when he wants to. If it is located in close proximity to his other work, all the better.

It is thus believed that the only fair and workable arrangement to divide the large machine's capacity is to assure each large user a fixed minimum share of the large machine's capacity, on call to him at any time he needs it, so that it has all the *properties* of a personal machine. If properly combined with remote facilities and languages, the user will be much happier with a percentage of a larger, and far more efficient data processor, than he would be with a much more limited capability machine of his very own. After all, as far as he is concerned, there would be no way of telling the difference, outside the fact that the time-shared facility would be far less costly to him, far easier to administrate, and much more powerful.

There is every indication that MIT will need at least a 20-100 fold increase in its available computer capacity within the next three years if it is to continue and accelerate its use of computers as an intellectual tool for research and teaching. The major alternatives for providing this capacity are the following:

1. Continue the present policy of allowing separate groups to provide their own separate facilities as present computer facilities become less and less able to meet their requirements.
2. Purchase or rent an extremely large central machine with adequate remote facilities to satisfy the projected needs and requirements of the various user groups.
3. Design or build our own large computer that would have the same or better capacity as would be provided by a machine that is available commercially, using presently existing components and machine organization techniques.
4. Designing or build our own large computer, using "state of the art" components, and the most advanced concepts of machine and system organization that can be obtained from the foremost experts in the field, whether they are at MIT, Lincoln, Rand, or elsewhere.

Of these major alternatives, buying a suitably modified commercial machine, and in particular Stretch, seems by far the most advantageous.

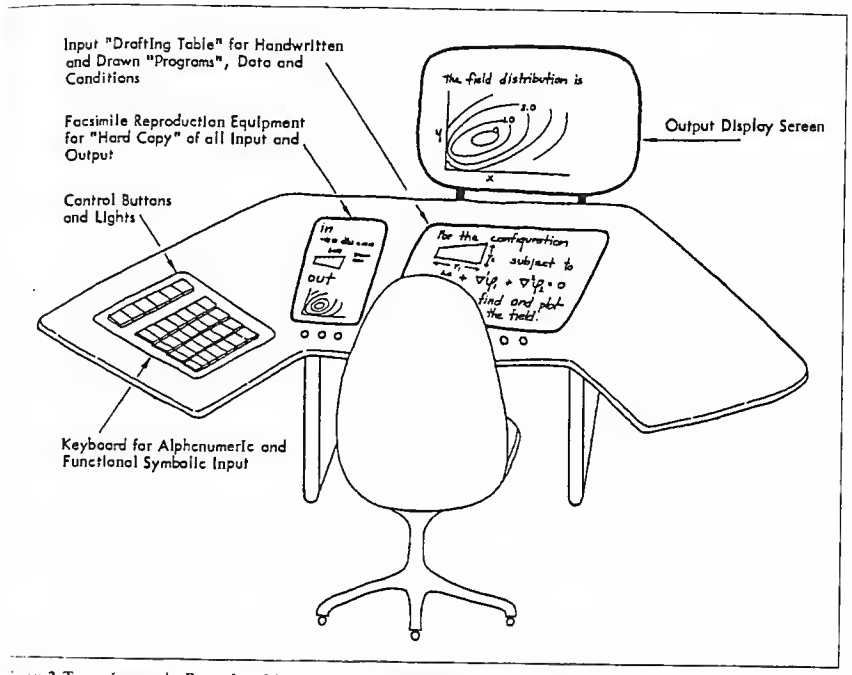

Figure 2. Teager's console. Reproduced from "M.I.T. Computation — Present and Future."[48]

Minsky and Teager

Interestingly, in the workshop which formed the basis for the book *Computers and the World of the Future*[25] by Martin Greenberger, McCarthy says

> The material I shall present on computer-system design was developed jointly with Marvin Minsky. I also want to acknowledge the stimulating effect of discussions with Professor Herbert M. Teager and Dr. F.J. Corbató, who are developing time-sharing systems for the IBM 7090 at the MIT Computation Center.

This contains one of only a few references to Marvin Minsky's contribution to the development of the concept; one other reference is in one of the translations of "MAC" provided by Peter Elias:

Machine Aided Cognition
Man And Computer
Minsky Against Corbató

Others have also provided Multiple Access Computer.

McCarthy also implies, and this was confirmed during the interviews, that Teager was proceeding with his own plans in parallel with Corbató. Teager's work was described in the *Communications of the ACM* in a 1962 reprint[49] of an earlier research report: "Real-Time, Time-Shared Computer Project," Computation Center and Research Laboratory of Electronics, MIT, Cambridge, Mass. Reported by Herbert M. Teager (Nov. 1961). Some excerpts follow (copyright 1962. Association for Computing Machinery, Inc., reprinted by permission):

> The objective of this project is to develop devices, systems, and languages for the fruitful interaction between scientists and computers, using the computer as a powerful, on-line aid to understanding. Due to cost and computer capacity considerations, this ability can best be provided within the context of time-sharing a slightly modified, standard memory size computer, equipped with random access files, among many low-cost simultaneously operating remote consoles, each equipped with low data rate graphical and character-producing input-output devices.
>
> The major accomplishments of this project over the past calendar year are the following. On-line programming and computation utilizing a system of multiple independent typewriters has been tested. An existing digital plotter has been connected to the IBM 709 computer, and a special-purpose computer to control multiple independent plotters is being constructed. A prototype of a high resolution graphical input device for hand-drawn figures and symbols is being built. Design modifications to the 7090 computer have been proposed and are being incorporated in our forthcoming machine. Scheduling systems for time-sharing and memory allocation nave been simulated and found satisfactory. An information retrieval system for programs and data is being designed. Programming systems to allow handwriting as a valid input are being checked out, and new graphical languages for several major problem classes for input and output have been partially specified.

REFERENCES:
Teager, H. Dec. 1960. Marriage of on-line human decision with computer programs. *Naval Res. Logist. Quart.*, 379-383.

As indicated in the interviews later, Teager apparently concentrated on the design of interactive input-output devices connected to the IBM 709 at the same time as Corbató was implementing CTSS on the same machine.

In the above communication, Teager refers to a prior paper presented at George Washington University in 1960, which appears to provide the motivation for his work. This motivation, typified here by the abstract from that paper, has a strong resemblance to Licklider's man-computer symbiosis concept.

> As the logistics problems of current interest become more and more complex, it becomes apparent that heuristic methods hold more promise than exhaustive or algorithmic methods of solution. It is also becoming clearer that heuristics, per se, are useless in approaching large combinatorial problems (via pattern recognition, for example), unless some general techniques are developed to draw out the specific "best" heuristics required for each problem. It is a fact that humans are, at present, far more efficient in developing and applying such heuristics than a machine. It is therefore of considerable value to explore how the best features of both human and machine decision can be married. The general problem requires the development of common meaningful languages, specialized input-output display and entry facilities, and a means of time-sharing the computer, in order to efficiently match the machine versus the human-decision time.[47]

Long Range Computation Study Group's recommendation for a time-sharing system.[1]

The conclusions and recommendations of the Teager report did not reflect the views of the majority of the committee. Thus, a second report was submitted that recommended the acquisition of not just a large computing system, but also a system capable of supporting time-sharing.

The second report of the Long Range Computation Study Group was presented to Albert Hill in April 1961.

The major part of MIT's computational needs, a few years from now, can best be met hy a single very high speed large-capacity computer system with provision for time-sharing through a number of remote consoles.

Present estimates of the cost of such a system lie between $8 and $25 million fnr a suitable central machine, including the development and procurement of remote input-nutput equipment, and the development of programming languages and systems appropriate for time-shared nperation. However, this system would be the cheapest way of providing the needed capacity and the only way of providing the necessary, close man-machine interrelationship that we feel is vital for research in the years to come.

The committee is cnnvinced that the hulk of scientific calculatinn is certain tn increase rapidly and that it would he uneconomical, even if feasible, to try to keep pace with this growth hy the addition of more and more standard machines, although there are areas in which separate machines may be valuable. Therefore, steps should be taken to acquire a giant machine meeting the specifications presented subsequently.

The conclusion that a central cnmputer with remote consoles is required is based not only on the important cennomy ohtained, both in running expenses and in programming expenses, but also on the importance of establishing close communication between the scientist and the machine. Thus, the committee feels that it is essential to provide for operating the machine in a time-sharing mode to provide immediate service to each of a large number of users simultaneously operating remote consoles throughout the Institute.

It is remarkable that, despite the many views and needs represented by members of the committee, there were essentially no comprnmises involved in coming to this general conclusion. Such a high-speed computing system having a very large memory and remote time-sharing nperation meets the needs nf very different kinds of projects, ranging from the numerical computations of physics, engineering and meteorology, to the symbolic computations of language translation and artificiat intelligence projects.

Feasibility of proposed system

All components nf the proposed central computer system either exist or can be developed during the time necessary for the procurement of the central computer. For example, time-sharing arrangements using remote input/output stations equipped with typewriters presently exist. Further study is needed in the areas of console specifications, programming languages and system administration, but the Study Group is confident that the present recommendations are not dependent upon the results of these studies. In defining these areas, it is merely recognizing the need to present a full picture of future work.

The final choice for the central computer should be based on whether it can be installed in three years and will meet the set of general specifications [presented in Part III of this repnrt]. There are two feasible alternate ways to acquire such a computer. The first is to purchase or rent one of the very large commercially available computers, such as the IBM 7030, cnmmonly known as "Stretch." With minor modifications and provisions of time-sharing facilities, this computer will meet many of the requirements. The second alternative is to have a manufacturer construct a computer according to our specifications, making full use of any advanced components which he may have.

It is not possible at this time to make any kind of firm cost estimate, since the committee has not yet been in direct contact with manufacturers. A suitable system based on the IBM Stretch computer would, at rumored prices, cost between $20,000,000 and $25,000,000. Much of this cost would be the cost of memory. Some of the members of the committee believe that the cost of a suitable system may be as little as seven or eight million, but others are not convinced.

The importance of time-sharing

The main conclusion of this report is not just that MIT acquire a very large computer system, but that the computer system be a time-sharing one, accessible to its users through remote consoles and accessible to laboratory apparatus through suitable data transmission facilities. The reason for this emphasis on time-sharing is because the greatest shortage at MIT is not in computer speed but in interaction capability with the users and with the users' laboratories. In fact, the increase in interaction capability made possible by the time-shared system will have as much impact on MIT research as the introduction of automatic computation in the first place.

A single, very large, time-shared system is recommended for the following reasons:

1. Only a very large computer can satisfy the needs of users with very large problems.
2. Only a time-sharing system can make possible efficient use of large capacity by many small users.
3. Only a very large secondary storage system can contain all programs and data and make them rapidly available to the computer under the control of simple consoles.
4. Only a single large system with many users can provide a sufficient base to support the large effort required to provide advanced public programs to all users on an adequate scale.
5. Only a time-sharing system can provide the interaction capability which, especially when combined with advanced programming and control languages, is so essential to improved computer utilization in research.

Specifications for central computer

The general scale and characteristics of the central computer are indicated by the following specifications:

1. An advanced indexing and addressing system, fixed and floating point arithmetic, and a full complement of logical and variable length operation.
2. Average program running speed of at least 10^6 instructions per second.
3. 250,000 to 1,000,000 words of directly addressable memory, each word at least 48 bits.
4. Full time-sharing facilities including memory protection, interrupt capability, and non-stop operation.
5. Provision for a system of remote consoles.
6. Capability for real-time interaction with laboratory apparatus at a rate of 250,000 words/sec.
7. Secondary storage of more than 10^7 words with an access time of about .1 sec. and a data rate of 250,000 words/sec. ■

CTSS — The Compatible Time-Sharing System

Immediately after — if not before — the presentation of the Long Range Computation Study Group report to Albert Hill, Fernando Corbató began to develop an experimental system that would show the feasibility of a time-shared system such as that envisaged by several members of the group. Within a very short time, he was able to demonstrate the first steps in this direction, as shown in the following excerpts from a 1962 paper he coauthored with Marjorie Merwin-Daggett and Robert C. Daley. The selection from the paper, which reported work completed in late 1961, is reprinted from *Proc. Spring Joint Computer Conf.*, Vol. 21 (© 1962 AFIPS).

Excerpts from "An Experimental Time-Sharing System"[14]

Before proceeding further, it is best to give a more precise interpretation to time-sharing. One can mean using different parts of the hardware at the same time for different tasks, or one can mean several persons making use of the computer at the same time. The first meaning, often called multiprogramming, is oriented towards hardware efficiency in the sense of attempting to attain complete utilization of all components. The second meaning of time-sharing, which is meant here, is primarily concerned with the efficiency of persons trying to use a computer. Computer efficiency should still be considered but only in the perspective of the total system utility.

An experimental time-sharing system has been developed. This system was originally written for the IBM 709 but has been converted for use with the 7090 computer.

The 7090 of the MIT Computation Center has, in addition to three channels with 19 tape units, a fourth channel with the standard Direct Data Connection. Attached to the Direct Data Connection is a real-time equipment buffer and control rack designed and built under the direction of H. Teager and his group. This rack has a variety of devices attached but the only ones required by the present systems are three Flexowriter typewriters. Also installed on the 7090 are two special modifications (i.e., RPQ's): a standard 60-cycle accounting and interrupt clock, and a special mode which allows memory protection, dynamic relocation and trapping of all user attempts to initiate input-output instructions.

In the present system, the time-sharing occurs between four users, three of whom are on-line each at a typewriter in a foreground system, and a fourth passive user of the background FAP-MAD-MADTRAN-BSS Monitor System (FMS) used by most of the Center programmers and by many other 7090 installations.

Significant design features of the foreground system [for the user] are [that he can]:

1. Develop programs in languages compatible with the background system.
2. Develop a private file of programs.
3. Start debugging sessions at the state of the previous session, and
4. Set his own pace with little waste of computer time.

The foreground system is organized around commands that each user can give on his typewriter and the user's private program files which presently (for want of a disc [sic] unit) are kept on a separate magnetic tape for each user.

The commands are typed by the user to the time-sharing supervisor (not to his own program) and thus can be initiated at any time regardless of the particular user program in memory. For similar coordination reasons, the supervisor handles all input/output of the foreground system typewriters. Commands are composed of segments separated by vertical strokes: the first segment is the command name and the remaining segments are parameters pertinent to the command. Each segment consists of the last 6 characters typed (starting with an implicit 6 blanks) so that spacing is an easy way to correct a typing mistake. A carriage return is the signal which initiates action on the command. Whenever a command is received by the supervisor, "WAIT" is typed back, followed by "READY" when the command is completed. (The computer responses are always in the opposite color from the user's typing.) While typing, an incomplete command line may be ignored by the "quit" sequence of a code delete signal followed by a carriage return. Similarly, after a command is initiated, it may be abandoned if a "quit" sequence is given. In addition, during unwanted com-

mand typeouts, the command and output may be terminated by pushing a special "stop output" button.

Although experience with the system to date is quite limited, first indications are that programmers would readily use such a system if it were generally available. It is useful to ask, now that there is some operating experience with the 7090 system, what observations can be made. An immediate comment is that once a user gets accustomed to [immediate] computer response, delays of even a fraction of a minute are exasperatingly long, an effect analogous to conversing with a slow speaking person. Similarly, the requirement that a complete typewritten line rather than each character be the minimum unit of man-computer communication is an inhibiting factor in the sense that a press-to-talk radio-telephone conversation is more stilted than that of an ordinary telephone. Since maintaining a rapid computer response on a character-by-character basis requires at least a vestigial response program in core memory at all times, the straightforward solution within the present system is to have more core memory available. At the very least, an extra bank of memory for the time-sharing supervisor would ease compatibility problems with programs already written for 32,000-word IBM 7090's.

Conclusions

In conclusion, it is clear that contemporary computers and hardware are sufficient to allow moderate-performance time-sharing for a limited number of users. There are several problems which can be solved by careful hardware design, but there are also a large number of intricate system programs which must be written before one has an adequate time-sharing system. An important aspect of any future time-shared computer is that until the system programming is completed, especially the critical time-sharing supervisor, the computer is completely worthless. Thus, it is essential for future system design and implementation that all aspects of time-sharing system problems be explored and understood in prototype form on present computers so that major advances in computer organization and usage can be made. ■

The CTSS Interviews

Location:
Laboratory for Computer Science
Fifth-Floor Conference Room
Massachusetts Institute of Technology
545 Technology Square
Cambridge, Mass.

Date:
October 18, 1988

Participants:
Fernando J. Corbató
Allan L. Scherr
Douglas T. Ross
Martin Greenberger

Interviewers:
Robert Rosin, Editor, IEEE Annals of the History of Computing
John A.N. Lee, Editor-in-Chief, IEEE Annals of the History of Computing

On the day following the celebration of the 25th anniversary of Project MAC held in Cambridge on October 16 and 17, 1988, two small groups of participants in the developments of CTSS and Project MAC met to exchange recollections about their activities. These interviews are separated into two parts, concentrating on each of the two developmental stages of time-sharing, although it was impossible to strictly maintain the separation since the discussions naturally overlapped the time periods. By choice, the interviewers guided the discussion to concentrate on the more personal and background aspects of this history, since the technological history has been well documented in the open literature. (See the References and Bibliography section on pp. 52-54.) The footnotes and reference citations were added during editing.

Biographical sketches — Corbató, Ross, and Scherr*

Corbató: The war broke out in 1941 for the United States; our high school went into long hours and I saw a chance to get out in a hurry. I went to UCLA as a student with the threat of the draft looming over me. Suddenly some people came by who were concerned about the ability of the Navy to maintain and repair the incredible amount of electronic equipment it was getting. There was a program called the Eddy Program, and they gave me the opportunity to join the program and get an education as an electronic technician. One of the benefits, of course, was one did not get drafted or get assigned to be a cook or something worse. So I enlisted at the age of 17 in the Navy and went through a year-long program as an electronic technician. I got exposed to some of the earliest and largest electronic systems then deployed in the Navy; mostly radars, lorans, and sonars. I got [exposure] both on land and on ship, and it was an incredible background experience. I did not realize until later how important that was.

Lee: And how that experience paralleled that of other people in some respects in radar and other analog applications? [Such as the wartime experiences of Maurice Wilkes.[53]]

Corbató: Yes, and it gave me a large-system perspective which was very, very important, and I view that as seminal in my own career. After the war I got a chance to go back to college. I went to Caltech. Since everyone wanted to be a physicist in those days, I wanted to be a physicist. Then I came to do graduate work at MIT, and I was still a physicist. I even got my doctorate in physics, but along the way I got exposed to digital computers by the late Philip M. Morse [later to be director of the MIT Computation Center]. He recruited me to take on a new research assistantship in the use of digital computers. That was approximately 1951. The people who signed on with this initial ONR-sponsored re-

*Greenberger provided his biographical notes during the Project MAC interviews, which will appear in the next issue.

1058-6180/92/0100-0033$03.00 © 1992 IEEE

search assistantship program were exposed to punched cards in the spring, and in the summer we were exposed to Whirlwind. Whirlwind was just barely coming up: it had about 1,000 words of 16-bit memory.

Rosin: Core memory by then?

Corbató: No, a very fancy, custom-built electrostatic memory that had a reliability [such that the] mean time between errors on a good day was about 20 minutes.

Lee: Was that the William's Tube?

Corbató: No. Jay Forrester was very prescient in recognizing that the William's Tubes were flaky in design, and he had actually gone back basically to first principles to design a very elaborate mosaic storage tube. Somewhat similar in storage retention (charge retention), but it was a tour de force in electronics to make it work — and they had to.

Ross: They built them in the basement at MIT. Patrick Youtz headed the group.[52]

Corbató: They designed, built, and fabricated everything. The engineering design was good: it was tough — they had to rewrite everything every few hundred microseconds, I think. The cycle time on the memory was 24 microseconds. But it was a relatively fast computer for its day, and it had a nice lean order code — a RISC machine without realizing it. [Laughter.] There is no such thing as a non-RISC, right? That's right. It had a 5-bit field for instructions so there was a maximum of 32 instructions.

I got to know people like Doug Ross who were also the users of the day: I was a graduate student. I used the computer in my physics doctoral work and got full exposure to using operating systems.

Rosin: Whirlwind, of course, was government sponsored and had some really important work assigned to it. Was there a lot of campus computing ("nondefense computing" in today's vernacular) going on at that time? How many graduate student colleagues were there who were using the machine?

Corbató: Well, the people who ran it recognized the need to have unclassified work and [be able] to use it in a free way. So the deal they worked out was, to my recollection, approximately three to four hours a day, once in the late afternoon and once from 2 to 4 a.m. In the morning were the times when the nonclassified people were allowed a crack at using the machine. In fact, I did not really know very much about what was going on at that spot in time. This meant one worked all day and all evening preparing these Flexowriter tapes that were the input [media]. Then one took a quick shot at the machine. And if you were really fast, you might get two shots in [during] that two-hour period. But it was perhaps one shot a day. Very frustrating, but on the other hand, the machine was not that big; the programs were not that huge yet. But the hours were all full.

Lee: But did you regard it as a personal computer?

Corbató: It was a personal computer; there were many things about it that got lost in the next few years, such as graphical display output and even audio output, in the sense that there was a probe put on the circuit of the accumulator which gave you a signature of every program you were running. You actually could hear where your program was, you could sense the loops, and you could even get the tempo of how the program was running. Occasionally, people recognized from the audio that they had a program that was mishehaving because it did not do what they thought it should do at that point.

After I got my doctorate in physics, Phil Morse recruited me into staying on with the newly formed Computation Center that he had begun in 1956. But the space and machine did not really come together until the fall of 1957. For approximately one year, those of us who were working on the project with the new center were quartered in the Barta Building where the old Whirlwind was. The focal point of the new center was an IBM 704. Phil Morse had worked out a package deal with IBM, where in return for the use of the machine on campus for one shift, MIT would also make it available to a cooperative group of New England colleges for the second shift. Then the third shift would be retained for use by their [IBM's] own local scientific office, and the fourth by a group in Cambridge which computed the orbits for the first sputnik. That group was formed and directed by Martin Greenberger.

To make a long story short, I played various roles ranging from initially supervising a research assistantship program to later becoming assistant director, associate director, and finally deputy director of the Computation Center.

Rosin: I think this is an opportunity to have Douglas [T.] Ross and Allan [L.] Scherr describe their backgrounds and how they got to be users of this same system that Corby has just been describing from his point of view as a service center. Doug, do you want to kick it off? Where did you come from and how did you get to the same point?

Ross: Actually, I did undergraduate work at Oberlin. We have just finished having the 25th Anniversary Celebration of Project MAC, and I was telling people that my honeymoon night, the first night of my marriage, was spent in that very same Boston Sheraton Hotel used for this conference. [Laughter.] I came over to interview about coming here for math graduate school [laughter] in the fall of 1951. I had a better [financial] offer from Carnegie, but I decided MIT was the better [technical] place to be.

I had a teaching assistantship on the way to a doctorate in pure mathematics in the Math Department, and my wife was the first "computer" hired at Lincoln Laboratory — meaning the first female punching a desk calculator! We're very proud of her badge number (161). They worked out of Building 20 and also used a mechanical correlation computer designed by Norbert Weiner in the Servo Lab (Building 32). They were using it for analyzing radar noise. She had brought home a manual about it, and when the summer of 1952 came along I needed a summer job. I do not know how I got the chutzpa, but I called up Al Sise, who was the executive director of the lab. I said, "Hey, if you've got an electrical engineering student, I'm a math student, and by the end of the summer we could give you a much better calculator than that old ball-and-disk integrator thing! Would you be interested in that?" He got back in a couple of days and said "Well, we can't quite do that, but how would you like to punch a calculator like your wife does?" So I said, "Fine, a summer job is a summer job."

One of the first things I had to do was check out the range results that they got with that analog computer where they set the smallest separation between the two tracing points. Two girls would each trace the data record with a point, and the computer would compute one point on the autocorrelation function. For a small shift, it kept coming out unexpectedly. The engineers did not understand why it was so far off. I found out that there were a lot of other correlators around, both mechanical and electronic, but all of them were either broken or busy.

Somebody said, "Why don't you check it out on Whirlwind?" I said, "What's that?" [Laughter.] I did not know it and spun off from the same lab and was now up the street in the Barta Building. Someone had told me that Jules Charney in the Meteorology Department had had a correlation program written by Jack Arnow (and I still have a copy of that program). That was the first program I ever saw! It was Jack's program, and that same day I went downstairs and got the programming manuals from Donna Neeb and met John Frankovich (who just retired from Lincoln Lab this year). Within a couple of weeks I was a programmer.

Scherr: That's all it took in those days.

Ross: Yeah, right! [Laughter.] Anyhow, the upshot of it was that by the end of the summer I had completed an autocorrelation program which I was trying to debug. At that point Whirlwind was completely shut down for a couple of months while they yanked out all of the input/output system and put in a brand new one, in order to support the multiple displays and push-button inputs and so forth for the pre-SAGE Cape Cod System.[53]

So I had time on my hands, and I started writing a Fourier transform program in order to be able to complete the job and actually get power density spectra out of the autocorrelations. By the end of the summer, I had two programs that were partially debugged. I still had my two classes of Freshman Calculus to teach, and I had a full schedule of graduate school, but John Ward asked, "Do you really want to go back teaching freshmen, or would you like to consider making a career here?" I was the only mathematician in the lab. Doing that sort of thing I could continue my work, study as a special student, and get my degree — and everything would be wonderful. I said, "Yes, I think it is a good idea." So that's what I did, and by the time Whirlwind was back up, I was hooked.

I think it's also appropriate that I give some thoughts here in the context of time-sharing. As a matter of fact, my memory is very faulty, so whenever I do these historical undertakings, trying to document them, I only put in what I can find in my own stuff. I'm not a scholar. I do not go to look at other people's stuff except my own copies of the Comp Center reports and so forth, which are published. I do not go digging around elsewhere. I do try to only say

"Within a couple of weeks I was a programmer."

"That's all it took in those days."

things that I can actually document. Usually I find I am off by a couple of years. Things happened earlier than I thought, and I'm surprised by the sequencing and so forth. It's really a fascinating, but time-consuming, chore to do this. Out of this particular early sequence came things that do feed directly into time-sharing.

Rosin: Before we get to time-sharing, Doug, I would be interested in your reflections on the Comp Center as it emerged at MIT: it sounded, from what you were saying when Corby was talking, that you were a user, but when did you get to the other side of the input tray?

Ross: I did not get to using the Comp Center yet. This was well before the Comp Center existed. I'm still four years before that.

The key thing was that the job I was doing concerned very large scale data reductions for Air Force data. As soon as I found out about the classified work that Lincoln Lab was doing with Whirlwind, I managed to eventually get a tour to see all the multiple scopes and so forth, and decided that that was just what we needed to control our massive data reduction programs. They were so big and so multifaceted that it was really hard to cram them through what had gone from 1K of memory (16-bit electrostatic memory) to 2K of magnetic memory. But they also had added a drum which had six more 2K banks worth of stuff, so you really had a lot of new things that we could use. Also, they doubled the speed of the machine when core memory went in.

So we had these very large programs to run, and I commissioned John Frankovich and Frank Helwig (who were actually under Jack Porter, under Charlie Adams in the Whirlwind programming group) to write what later became called the Director Tape program — which is, I claim, the first operating system! That came about because the Flexowriter that was attached to Whirlwind had its paper tape punch, paper tape reader and printer hooked up, but when they installed a photoelectric tape reader, the mechanical reader was left unused. We had a tremendous shuffle of tapes to make our runs. So I had this idea that we could put those same manual instructions on a tape in the mechanical reader while we spliced together all our programs on one big reel for the much faster photoelectronic reader — eliminate the manual operations and get more reliability. So Frank and John built such a system called the

Director Tape system. It was used also for controlling post-mortems, dumping the machine state. So that was, in effect, the first operating system, working along with the drum. I also wrote a mistake diagnosis routine that swapped chunks of memory to the drum and so forth, and that was part of the system.

But then as the system grew, we started working on the use of the manual intervention consoles, with push buttons and analog displays. We switched from a 3D computational programming emphasis to what is now called "interactive" programming, but I then called it "Gestalt programming," with man-machine conversation being the thrust of it. We had a couple of big demonstration meetings on it. But we still were working just with the push buttons that went with the Cape Cod control consoles that were in the secret Room 222 in the Barta [Building].

I did the first hand-drawn input to a computer (graphic input) in 1954. That was one of the few programs I wrote that worked the first time. I have a quotation of the operators in the main control room saying, "What kind of a program do you have there?" It sounded so strange," when they looked up on the scope and saw my handwriting with my name coming out.

We were under way early using all this stuff, but we needed more language capability — ability to label displays and label plots and so forth — to get some true language control over what went beyond the push buttons. Starting in February 1956, I started working with the Whirlwind engineers to propose hooking the keyboard of a Flexowriter into the console, so we could put alphanumerics into the program on line. Until then the keyboard had not been used, as near as I can figure out. Arnie Seigel was an engineer as well as a programmer, and so he did a logic design, using relays. We had a Flexowriter on its own rolling table, and a plug. We were not allowed to change any of the wiring on Whirlwind because of the importance of the Cape Cod work, so we did it without affecting anything in the electronics.

I believe that same physical keyboard input hardware, as well as the design, was used by Herb Teager and his group, when they decided to put a Flexowriter on the [Computation Center] IBM 704. In my Daily Résumé book I mention the meetings between Dean Arden, Jack Gilmore, John McCarthy, myself, Arnie Seigel, and whoever would be in charge of the Lincoln [IBM] 709 programming (no name given), sometime between December 29, 1957, and January 1958 in order "To consider using the real-time package (which was IBM's terminology) on the 709 to simultaneously service about 20 Flexowriters in TX-0 fashion. We are going to start off with one Flexowriter on the 704 real-time package." I say in the Résumé that keyboard as well as light-pen input was standard on the TX-0 all along. That was the start of MIT's time-sharing development. That's the tie that I thought would be appropriate to make here about how the Computation Center time-sharing efforts got under way.

Lee: This [following] quotation is from the book *A Century of Electrical Engineering and Computer Science at MIT*[2] — it talks about 1960: "A proposal was worked out by John McCarthy and Marvin Minsky; other advanced ideas were proposed by Doug Ross, Jack Dennis, and Daniel Goldberg." And then it goes on to say, "the report mentioned many students involved in the study, especially Allan Scherr." That was one of the reasons I felt that it was so important that we have a student of the era in this session to say, "How I got involved in this project."

Scherr: I came to MIT in the fall of 1958 as a freshman. I came with a general interest in electrical engineering that had started as a tinkerer with hi-fi systems, automobile electrical systems, and so on, when I was a high school student. In the second semester of my freshman year (this would be the February 1959 semester) John McCarthy and Nat Rochester, who was a visiting professor from IBM, put a freshman course together called something like "Introduction to Automatic Computation." In fact, I can even remember the course number; it was 6.41. A bunch of people got really excited about being in that course, so I signed up for it and got accepted. Apparently there was a much larger number of people who had signed up than were accepted, but I got in!

John and Nat wound up teaching us assembly language starting with a pseudomachine that Nat had invented called the "Rochester machine." Then later [they taught] FAP or SAP, I forget which, on the [IBM] 704, then Fortran I (or II), and then John taught us Lisp. We ran class problems in all those languages. I recall *almost* being bitten by the programming bug. Most of the people in this course became instantly addicted to writing programs. A couple of the people flunked out [of school] either that semester, or later, because all they did was spend all their time writing computer programs and did not go to [their] physics or math classes. As it turned out, I came away with a fascination for the hardware. So I did not do much with software for a while. Dean Arden was my freshman advisor, so Dean figures in a lot of this.

It was not until the following year (which would have been 1960) that two things happened. One was that I met Herb Teager who was looking for skilled [chuckle] undergraduates to hire for a pittance to do computer work. I needed the money, so I applied. In those days there were two salary scales for undergraduates: if you were skilled you got $3.40 and if you were unskilled you got $3.20. If you knew how to program, that was considered skilled. Herb hired me to do some work for him. One of the things was to perform some of the measurements that were associated with the study that was going on about future systems. I sat at the console of the [IBM] 704 with a one-card self-loading program that Herb wrote. After every job was run, I would stop the machine and load this card in, and it would display how many zeros were in main (core) storage on the console. I'd write that down, and by the end of the day we would have all these numbers that would show how big the programs were. [Laughter.] By subtracting the number of zeros and taking the 7's complement (or the 8's complement), you could figure out how big the programs were. I produced a histogram on how big the programs were, and that was one of the things that went into the study.

From the audience: There's one point of confusion which is [illegible] that there were two long-range studies — Herb Teager [illegible] the chairman of one. Herb produced a large report[48] which got very little circulation — later we can get into what the problem was — and I think that the data that Allan was alluding to was gathered by Herb as reinforcement for his large report. There was subsequently a short report which was written by everyone but Herb, almost a rump report[1] that had many of the same conclusions, but also differed in critical ways. That short report had almost no data to my recollection.

Corbató: The data that Allan was referring to was part of a large body of data which was produced by the committee, and it's sort of in the record but on the other hand it may not be noticed.

Scherr: Interesting, because I was for the most part unaware of the controversy that was going on...

Audience: Heavy politics.

Scherr: Well, as we go through the day, I'm sure it will come out because there were some things that I wound up doing which were political also, at least as I found out later.

Let me just say a few words about what it was like to use the Comp Center in those days. As a student, basically what you did was to write programs to do the usual kinds of things. In fact, I remember one of the exercises in this course [6.41] was to write a floating-point ADD in assembly language without using floating add instructions. We were supposed to use all the other instructions to simulate a floating-point ADD, and that was probably the hardest of the homework problems. I do not recall whether I ever got it to work, but basically all the students would submit their [program] decks, and they would be batched together and run under a main program that the instructors provided. That program would check to see whether you got the right answers and produce score sheets for each of the students. You'd wind up using 30 seconds of computer time for an entire homework assignment.

I'm actually not convinced that any operating systems since then have been faster than the ones we used in those days. The thought of assembling and running a program in a few seconds still boggles my mind. In those days there were peripheral machines that IBM provided that were basically card-to-tape, tape-to-printer, and so on, that were run as special-purpose machines. They were all in a glass house with the [IBM] 704. It was probably a year later — 1959 or 1960 — that we got [IBM] 1401s. The [IBM] 1401s came in, and there was a peripheral operation down in the basement of Building 26 that I vividly remember going down to. It really was a *remote* operation. The input/output station was basically physically remote from the computer by a floor, so we would submit jobs down there and the output would come back somewhere else. I do not remember where anymore. The whole idea was that programmers did not go into the machine room.

Rosin: What was turnaround time on this?

Scherr: A day — if you got two shots a day, it was a good day.

> **"I'm actually not convinced that any operating systems since then have been faster than the ones we used in those days."**

Rosin: Did you have to go back to find out whether your job was run?

Scherr: Yeah, there would be people who would forecast when you should come back. Usually you would ask the operators, "How's turnaround time?" They would say, "Well, it's pretty good today. Why don't you come back at 4 o'clock?" You would come back at 4 o'clock to see if you had got it and they'd say, "Come back later." Then you would call up and say, "How is it going?" Mostly they didn't know, so what I would do [would be to] go to my classes and come back in the afternoon and see if it had come back, and if it had I would say "Great!" and if it had not, I'd go home. Writing programs in those days was a labor of love, and you wound up spending a lot of time at your desk because you did not want to waste the precious shot you were going to get once a day.

Computing at MIT before CTSS

Corbató: There were quite a number of people who were part of the initial new creation. One of them was John McCarthy, who first came as a visitor (from Dartmouth) as part of the New England Colleges Program and [who] then decided to switch to MIT as a faculty member. Dean Arden was on the faculty, and had been part of the Whirlwind staff. Then he became part of the Computation Center staff. Herb Teager, who was a young assistant professor, was also part of the staff.

In that environment, Phil Morse had tried to organize a dual role for the Computation Center. It was first a center of computer science research activities largely carried out by the faculty, and it was also a service center for both the MIT and New England college campuses. Part of the goal in those days was to try to encourage people to use the [IBM] 704 in shrewd innovative ways. At first it was a little uphill. We had to sell people on the idea of how to use the machines. Fairly rapidly, as you would expect, we built up a good head of steam, and people began to use the computer.

One of the first things that happened was that we began to get caught with the congestion of too many people trying to use the computer at one time. In the very early days, people tried to use the machine as we did Whirlwind, where you could go up to the machine with a deck of cards, read

them in, and run your job. A few minutes later, probably, you would take a core dump (which had an incredible number of octal digits) and then go back and try to sort out what had happened. That was too inefficient for mass use and gradually we, and almost everybody else in the country using a machine of that character, switched to a mode which got to be called *batch processing*, where the input decks were collected in a group and prerecorded on a magnetic tape using an auxiliary computer, so that the magnetic tape was the input to the computer. It would run job after job. After each job finished, it would take a core dump, putting each out on magnetic tape, or when the [program] failed, which they usually did...

Scherr: ...At the exorbitant cost of 100 locations in lower core, which everybody complained about.

Corbató: The result was you had this kind of mail-order business, where you could drop off your deck. The closest to the computer you ever got was putting your deck into a tray! Someone would pick the contents of the tray up and record them on tape. Then you would go and pick up your output in a file bin and ponder what in the world had happened. That style of operation persisted all the way through 1964 or 1965.

Rosin: Did you write your own primitive operating system or did you pick one up per se? You have to remember that IBM did not provide an operating system.

[Here there was background discussion about whether that statement was true!]

Corbató: Let me see if I can give you *my* recollection of what happened. At first there was no operating system. Then there was something called the Fortran Monitor System (FMS). What happened was that just around the same time that we were to get our first computer, Fortran made a big splash. It had a major impact, of course. It opened the door to people who were intimidated by machine language. It enlarged the user base by a factor of 10 at least. One of the by-products of that system was that the people who developed it organized a very simple-minded operating system called the Fortran Monitor System (FMS), and I do not know whether they, or whether we, inserted the assembler, which was called FAP (Fortran Assembler Program), into the same system. In that Fortran compiler implementation, Fortran source was translated to assembler language and assembled by FAP. SAP was the predecessor (Symbolic Assembly Program). The original effective name was very confusing. What was it assembling? It was assembling a library routine at first, but in fact the result was this subset, or successor, called FAP, specifically organized around linking the Fortran program. That became our de facto machine language.

Rosin: How many users did you have? How many times could you run it in a day? What was turnaround time like?

Corbató: The number of registered users probably was in the several hundreds. Here I'm guessing, but we have the old manuals and one could dig it out of the old progress reports.

Lee: Let me read something to you. This is from John McCarthy in Martin Greenberger's *Computers and the World of the Future*[25] [page 224] — so this is about 1961, John says, "Since the Computation Center is overcrowded, delay is more likely one or two days."

Corbató: Let me explain that. We were using somewhere on the order of three to five minutes for each job. So you can figure out for yourself how many jobs that amounts to per day.

Lee: [McCarthy] says 125 to 160 a day.*

Corbató: We tried to use all kinds of pressure and persuasion (and just plain ordering) on people who had runs of longer than an hour (an hour was a big run) to run that kind of stuff in the middle of the night and off-peak hours. But debugging runs were [different] — occasionally people would bomb in a minute and be in and out, but that was in the beginning. By 1964 or 1965, things sped up a little bit, and you could scale all those numbers down by a factor of 2 to 4. People were running faster jobs, they were in and out faster, but that's a detail. The critical problem was that the number of people trying to use the machine was sufficiently large that unless you had some pull or priority, which was very reluctantly given out...

Audience: I'll say! [Chuckles.]

Corbató: ...you were faced with a minimum of two to four hours of waiting and [it was] much more likely to be an eight-hour wait before you got a repeat run — even if you were ready to turn the job back in instantaneously. Now frequently what would happen is that you would not be sitting there waiting at the output bin. You would probably show up two hours after your job ran. You would then discover that you had to do some repair and/or analysis. It would take you maybe two to four hours to do that, even if it was pretty obvious. Finally, you would resubmit, and now you suddenly got to the back end of a queue and you could wait another four to six hours before it ran again and you got it back and saw the results. You can see how that could easily escalate into 24 hours. In fact, that's what happened and that's what was driving people crazy.

Rosin: [to Ross] Were you also a Comp Center user?

Ross: Oh, yes — my group was. I myself never even learned how to turn on a keypunch! During this time we were doing the APT system, work which started in 1956.[37] Although we started on Whirlwind, from the beginning the cooperative industry APT project was targeted toward the IBM 704 series, because that's what all the airplane manufacturers had. We were a very heavy user of the Comp Center right from the very beginning, even using some of IBM's time, on occasion.

* He probably meant *per shift*.

Rosin: What was your view of Comp Center service capabilities and so forth?

Ross: Well, we had two roles really. I had known Corby and the others who were saddled with this responsibility. They had all sorts of things to balance that had never been done before. Also, they inherited the style [of programming] that had built up around Whirlwind — of having an in-house system programmer group (nowadays we would call them a system group). They were the center programmers, and then there were the user programmers. There was a big difference because priorities were given to the inside people that the outside people could not get. So my group always was in the crack between the privileged and the ordinary users. We continued to push for more things because we were so big. We had a big job to do, so we did manage to wrangle deals. I did not get involved in that very much except just to say, "Yeah, you pinched us too hard. We've got to have some relief," but normally we would be able to get some runs on an express channel. But also we did our best to stay out of the way and play the game the way that they had to have it.

Rosin: By express channel you did not mean a physical device?

Ross: No, no, no. It was just that we occasionally got faster turnaround — a priority type of thing.

Audience: An analogy is the grocery store. You had a quick checkout counter.

Ross: Yeah. We would adjust what we were doing to try to fit that [schedule]. I think we always were straining the system — just with the magnitude of what we were doing.

Corbató: Maybe it's worth my interjecting for the moment the chronology of the machines, because the way it is coming out sounds confusing, and we should get it more straightforward:

Whirlwind first came on-line around 1950. The non-Lincoln Lab use ran as a tool for the academic community. While it was only limited in use, it ran until (according to my recollections) about 1958 — maybe a little later was when it actually had to be shut down.

The Computation Center began in 1957 before Whirlwind shut down. It started with an IBM 704. At some point it was upgraded to an IBM 709 — in 1959 or 1960.

The IBM 7090 — which of course was the first transistorized machine in that family. Then that subsequently became an IBM 7094 and that sequence of machines. That machine family would run at the Comp Center until about 1973, when they were finally able to turn it off and get rid of it.

[IBM System/360 family:] Meanwhile, in about 1965, the first OS/360 machine was installed. Those became the main work force for batch processing.

So that was the chronology of it all.

Rosin: I have to interrupt Corby — I graduated from here in 1957, and my first program here was on the [IBM] 650 in what was, I think, then called the Computation Center?

"The number of people trying to use the machine was sufficiently large that unless you had some pull or priority...you were faced with a minimum of two to four hours of waiting."

Corbató: Frank Verzuh was one of the people who was running a small service group beforehand, and he ran an IBM 604 which was a vacuum tube, 50-step, hard-wired program machine. He ran a punched-card shop; he ran a CPC too. In the Lab for Nuclear Science, they ran an LGP-30 series machine. There were all kinds of small machines floating around, including [IBM] 650s and later [CDC] 1604s, all on the periphery of the center.

Audience: Frank Verzuh was the first associate director.

Corbató: MIT has always been a diverse place, in kind of an anticipation of the minicomputer era. Many small centers were successful because they had their own small machines. So even though the central machine was the largest, it was not unique.

Another key but minor point — the Whirlwind sped up by a factor of 4, so it went from 24 microseconds to 6 microseconds when they put in the core memory.

Ross: Yeah, but the throughput only went from 20,000 to 40,000 instructions. [Laughter.]

Corbató: My understanding was it really sped up. The other thing that I picked up on was that Teager (I believe) built all of his own hardware from scratch. He may have been influenced by what had been done before, but he designed all his typewriter controller stuff himself, and that was all after McCarthy wrote a pivotal memo to Morse in 1959.[33]

Lee: Let me quote from John McCarthy again.[25] He's talking about the turnaround time. He says, "This has a very bad effect on student theses." You [Scherr] were talking about projects in the classes, but John is talking about theses. "When an ambitious student proposes to undertake something substantial, his advisor often must say, 'I do not believe you can finish in time for a thesis.'" He continued, "I have several students this semester who are in this kind of trou-

ble." Then he goes on to conclude that the solution is clearly to have a private computer. Did these restrictions really have an impact on the graduate program as well as the undergraduate program?

Corbató: It had an impact on everybody. In fact, it seems today with the rapid response by personal computers and good time-sharing systems, people just do not understand how bad it was. It's like trying to have a conversation through the mail. You can't do it.

Scherr: I can give you an example. I took a course from Marvin Minsky on artificial intelligence back in the 1960 or 1961 time frame, maybe 1962. We had to do a term project, and I proposed to write a program that would bid a bridge [hand]. So I was going to take Gorin's book and turn it into a program. Or part of it anyway — just the opening bids. I asked for and received an hour of computer time for the project.

Lee: For the whole project?

Scherr: The whole project.

It was worth a small fortune. Oh yes, it was worth several hundred dollars. An hour of computer time in those days was considered a fair chunk and that was good for maybe 10 or 20 shots, and that was typical. People would do a bachelor's or master's thesis and get allocated four or five hours. That's clocked time on the maintrame, from the time your batch job starts to when it ends. I do not know whether they accounted for every second, but...

Rosin: They charged somebody for it! [Laughter.]

Lee: [To Corbató] Were you actually charging people real "honest-to-God" green money?

Corbató: No, that was one of the interesting aspects. One of the terms of IBM's donation for the use of the equipment was that we were not to charge for it. It was free all right. We nevertheless kept hooks and did the accounting to try to introduce a note of moral persuasion, so that we could keep people from abusing the amount of time they used and help them recognize how much of the resources they were using. The problem was that when we became overloaded and we clearly should have been acquiring more machine power, we had no money to do it. Nobody was in the position to pay for anything. That did not get sorted out until about 1965, when people began to [really] charge for machine time. That created the cash flow to acquire machines.

Ross: In the meantime that's why my projects and Professor John Slater's physics project under Michael Barnett, and I forget who else, formed the Cooperative Computer Laboratory at MIT. We had money. They tried to get me to take over the whole [IBM] 709 and I said, "No, I'm not in the computer business; I'm not going to run the computer." But Slater and Barnett did, and that gave us much better turnaround to do our large projects and also took considerable pressure off the regular Comp Center.

Rosin: That was done with sponsored research funds, I imagine.

Corbató: Let me explain where that [IBM] 709 came from. When the Computation Center was switching from the [IBM] 709 to the [IBM] 7090 in 1962, Michael Barnett was then working as a research associate for John Slater in the Physics Department. He persuaded Slater that there was need for additional computation facilities where you could get a little more high-class service. So he convinced MIT, in fact, to underwrite the cost to install that machine.

Audience: Fourth floor, Building 10.

Corbató: I forget exactly where. That machine ran in parallel with the [IBM] 7090. It may have been convenient for the physicists, but it was also an economic bust.

Rosin: Lest we beat ourselves too much with this, I wanted to offer a slightly different point of view. I had been an undergraduate at MIT — I was a graduate student at the University of Michigan. I did my first year of graduate work in psychology. From our point of view, as people interested in computing and interested in doing other things involved with computers, we felt there were never enough resources. I think we have to reflect on how much the advent of computing support of academic research was changing the nature of research in universities.

For example, in psychology, not a mainstream science or discipline at MIT but certainly so at many other institutions, prior to the advent of even small computers a student could earn a PhD by doing two factor analyses using desk calculators. All of the sudden that became a three- or four-minute shot on a large computer, and psychologists went back to doing psychology instead of doing computation.

So there were lots of changes going on at the same time. Demand was building up not just because there was not enough computer time, but, as you say, Corby, there were people coming out of the woodwork who had real work that they could do, changing their lives with computers. We were doing a lot of good things.

Corbató: Phil Morse was a pioneer at MIT in terms of trying to encourage, feed, and grow the use of computers for research in other fields. The universities led the way because we were quicker to acquire new applications for computers than the industry in general, and so we became extreme "have nots." There were far more demands than there were resources.

That's why time-sharing sprang out of the university environment — because we were the ones who were really hurting. People were trying to do more programming jobs and [fewer] production jobs, [with] more innovations. Programming was the thing that was the hardest to do in a heavily congested environment. It seems to me, in retrospect, that it was very obvious that time-sharing would be first recognized in the university. What was not obvious was that industry would be slow in recognizing how important it was.

[illegible]: That brings me to two questions regarding the introduction of more generally available time-sharing at MIT. One would be with respect to the Long Range Study Group, and the other to outside influences. There has been an allusion to Strachey's paper[43] and to things that McCarthy was stimulating.[25] There must have been things going on in the computing community in addition to what was going on in Lincoln Labs. Is it worthwhile focusing on outside influences, or do you want to look at the Long Range Study Group?

[illegible]: On the technical side I think the study group not only included the studies of MIT's own needs. We went around and visited all the major manufacturers, or was that later for the Multics project?

Corbató: Do you remember the date on the report?[1] I thought 1961 maybe. I'll give you my own recollection of the chronology. It's a little tricky. In some ways the ideas were [illegible] the air — [from] people who had used computers back in the Whirlwind days when they had direct use of the machine. They had not had any intermediaries or operating systems or all that. Whirlwind in a sense was the first personal computer. That had not been forgotten by people. I think that the person who deserves the most credit for having focused on the vision of time-sharing is John McCarthy. I do not know quite what led John, beyond the frustration of the day. John McCarthy wrote a very important memo[53] where he outlined the idea of trying to develop a time-sharing system.

John was very prescient — he recognized hardware [illegible] — and to my recollection, he had two key goals: [1] One was to make it possible for people to have timely interaction with programs for the small computational needs of actually debugging a program in contrast to running programs. He thought of time-sharing [as] acting against the buffer of larger programs. [2] He also had a desire for an individual to be able to use the largest machine possible — for an individual who could not have afforded it for himself. So it was a chance to use a very large machine [illegible] problem which was justified by having a very large [illegible] community of people using it at the same time. I do not know for sure whether he invented it, although I think he first wrote down the vision of how to do it in a fairly precise way. It was, in a sense, a kind of an invention.

I have read Strachey's paper. Although he was given credit for kind of coimagining some of the ideas, when you read it carefully, Strachey had in mind the notion of debugging between jobs or during jobs; it was a much more limited view of what we were trying to accomplish. He was trying to develop "parallel debugging." Strachey was a modest fellow [illegible] he did not argue that he had the grand vision, but he certainly was innovative and deserves some sort of cocredit. But McCarthy discovered [the concept] (it's too bad John is not here), and I believe McCarthy discovered Strachey's work after he had done his work — recognized the similarity and acknowledged it was a duality.

It was interesting [to note] that when Project MAC got started a few years later, a number of people came out of the woodwork and said, "Oh, I invented time-sharing," and "Did you read my paper?" and this or that. The problem was that everyone had kind of dreamy visions of people interacting with equipment. You can even trace it back to Vannevar Bush,[8] who had written a paper in the late forties. The thing that John did was to spell out the particulars of how you go about accomplishing the notion of having to be able to interrupt, [and] the notion of having boundary registers. He really began to think it through. He was an evangelist in part. He was a little cocky about how easy it was to do and tried to get IBM to do things, and they kind of humored us. We were having a hard time getting off the ground because IBM was just humoring us. It was in that environment that the Long Range Study Committee was created.

"With the rapid response by personal computers and good time-sharing systems, people just do not understand how bad it was."

Rosin: Let me interrupt because you've raised some interesting points. Allan [Scherr], you said you had McCarthy as a freshman.

Scherr: Yes.

Rosin: I want to pick up on this idea of vision — what kind of vision did he [McCarthy] give you folks as freshmen? Did he talk about computer utilities?

Scherr: No.

Rosin: Did he talk about large systems of interacting processes and things like that? The AI visions?

Scherr: A lot of the AI stuff was in there. He talked about the chess-playing program project that they were working on. It was almost like that course was a recruiting program for people working on the chess problem.

Rosin: I guess I was just trying to find out if John himself had something in mind and said, "Gee, this individual use of large machines is necessary for chess programs," for example.

Corbató: John articulated that best in the Greenberger book.[25] But there would be early memos. The early memo, in particular, was arguing [that] we ought to try and do something right now. The long-range study was set up in part to shut John up! He was making enough of a row within the administrative circles that they thought [that] we ought to

do something. The person who formally commissioned us to do that was Al [Albert G.] Hill, who was then, I believe, the vice president for research at MIT. He created a committee with Hill, [Robert M.] Fano, and [Jerome B.] Weisner, and there was a fourth person, who might have been Carl Floe. They in turn selected a committee to explore the ability of MIT to do something about getting a large machine.

Teager chaired this committee, which held several meetings. John [McCarthy] and others held forth, and in large part I think we all quickly agreed that we wanted a time-sharing computer all right, but the question of how to go about doing it was the part that was not unanimous. In particular, Herb Teager, as the chairman, had his own set of ideas, and he did not relate well to the rest of the committee on that score. After it was time to write up the results. Herb went out, holed up for a month or two, and wrote the report without interacting much with the committee. Then he came back and presented the draft to the committee. The committee said, "That's fine, but this is not quite what we meant to say." They started saying how we have yet to change this, you ought to change that. They disagreed most importantly with his conclusion, which was [that] the best thing MIT could do was get an IBM 7030 Stretch computer. So Herb went off and stewed about this. He sent a letter to everyone on the committee saying, "I cannot deal with this." I forget whether he said he resigned from the committee or not, but we did not let him resign. [Laughter.] He, in effect, was very upset because he felt we were in rebellion against his role as chairman. He tried hard to reflect what he thought was a reasonable set of conclusions, but we had a headstrong committee.

Rosin: Not unusual at MIT.

Corbató: Not unusual! [Laughter.] So the result was that the committee decided the only way to deal with it was that we would have to write another report by everybody except Herb. That other report is the more widely circulated one, which was shorter and basically agreed upon [by MIT]. Much of what was said was the same as what Herb said — namely, that time-sharing was the way to go and that's what we needed — but the conclusion said that we should shop around and look for a vendor who would be willing to create or interested in creating such a system. It argued for a computer, it put some specifications on the computer, [including that] we ought to go for a computer that had a million words, which, in those days, sounded grandiose.

Ross: Six megabytes. Was that in the report? I remember that was McCarthy's big thrust. It certainly became rapidly John McCarthy's focus.

Corbató: It had some sort of semigrandiose specifications. Not totally unattainable. Thinking big. It did not try to use off-the-shelf hardware, and that was the big difference between Teager's [report] and the one we produced. It tried to commission the machine — we were supposed to go out and look for one.

Lee: If you had been doing the work in the 1950s era, you would have done this whole thing in hardware and thought about how to build another Whirlwind to do a job. When did you move over to saying, "We can do this in software"?

Corbató: I think from the beginning it was recognized that you needed a little hardware help to build the interrupt programs — the ability to have clocks, the ability to set boundary registries to prevent one program from straying into another space. There were a few hardware things that were vital.

Rosin: You said John [McCarthy] saw that in the process early.

Corbató: John was quick to spot the need for all those things. We got IBM's help in part. They did a certain amount of retroengineering on their old machines, but the older machines were miserably engineered, because from the point of view of meeting these requirements they had been sloppy. If you used a certain sequence of undefined operations on an IBM 704 or IBM 709, something would happen. It was not what you wanted maybe, or maybe it was. Illegal operations were not trapped; individual users could give executive operations. It was nightmarish from the point of view of trying to share the machine. So those things were hard to rectify. It was recognized there was some key hardware [that would] help, not a lot, but key. It had to be available early enough in the design [cycle] that people could get it in right. Then the rest of it was reorganizing [the] software. It was never in doubt, I think, that there was a software problem to get organized. No one thought of it as just hardware.

Scherr: Let me ask a couple of questions. After I got to IBM, I was asked a lot about how all this started. One of the questions that came up, that I never quite got an answer to, was about the relationship between CTSS and what was going on at Dartmouth with the Basic system — which was more or less the same time period — and JOSS at Rand. Both systems were originated in the early sixties and were time-sharing-like systems. Was there any influence from those?

Corbató: I make reference to those [systems] in my 1962 paper on CTSS.[14] I did kind of a quick thumbnail survey of contemporary systems of the time. My recollection is that Dartmouth's was influenced by our early activity at MIT. They saw the idea of time-sharing, but they quickly latched onto the idea of a special-purpose system and language — that was the influence of [John] Kemeny. Now it's possible that there was some other path to that origin because [John] McCarthy had been an assistant professor at Dartmouth. He had been recruited by Kemeny to go there, and he was a visitor to MIT from Dartmouth. He then finally switched over [to MIT], but he obviously still had some ties [to Dartmouth].

Rosin: You could imagine John visiting up in Hanover and trying to spread the same gospel in a very different setting.

Greenberger: But I do not think John's thinking had gelled to the score yet — on time-sharing on interactive terminals.

Rosin: In what year?

Greenberger: 1957.

Rosin: It was before John was here at MIT.

Greenberger: The JOSS system was developed at Rand by Cliff Shaw as the hardware and software developer. His vision was largely to try to develop a system which was a computerized calculator to support engineers and physicists. Much like the Dartmouth system, he viewed the goal as creating a computing environment that was wholly unto itself, and it had a single language. So you put a lot of attention into a language that was kind of a sophisticated desk calculator. You could program in it, but it was alien to everything else that you might be doing. Particularly, you could not write a Fortran program because that just was not a known language [to that system]. Now whether Shaw got his inspiration from hearing about the time-sharing ideas that were flying around, I do not know. I'm not sure where he got the idea. It started out from the notion of a special-purpose system.

Rosin: Was Dartmouth something like this?

Scherr: Well, it was a simple system: [fundamentally] they invented the Basic language and limited the usage to that. That again was later [than CTSS].

Greenberger: I guess my own recollection of the era was that everybody who began one of these multiple-access systems had been influenced by John or at least had heard about him.

Ross: I've been really itching to fill in because I've just been tracking [the discussion] here in my Résumé. McCarthy and Minsky came [to MIT] essentially at the same time in 1958. From 1956, when we in the Servo Lab had success with getting our first specially built console attached to the ERA 1103 computer down at Eglin Air Force Base in Florida, our new work concentrated on intimate man-machine working together, and preceded the sharing of the machine. And so there's a whole section about how, if we're going to move our work from Whirlwind to the [IBM] 704, "we've got to have this kind of equipment and it will be extremely desirable to have the larger storage capacity and longer word length to the [IBM] 704 for use in SLURP" (Servo Lab Utility Routine Program — our system in which we did all this experimental programming). "The SHARE Assembly Program (SAP) and the algebraic coding system being developed for the [IBM] 704 installation by the Comp Center personnel, which they never did (it says here), will be more modern and flexible than the Comprehensive System on Whirlwind and so forth, so we really have got to have it. The possibility of detailed research into automatic process control and managerial business decisions should provide ample justification for the active consideration of these techniques as an integral part of the MIT Computation Center facility." This was in a memo from me to them.

Rosin: To whom? To the study group?

Ross: No, no, this was 1956! Way before. In fact, this was my first memorandum in the Computer Applications Group series. "Servomechanisms Laboratory Requirements for Computing Facilities, '56-'57." So it was addressed to my laboratory head over at the Comp Center. Our need to have continuing man-machine problem-solving capabilities and what that would mean to the Comp Center were ringing bells on both sides. Here [referring to the Résumé] I have on November 1, 1956, "F.J. Corbató called to inquire about several of our people giving talks at the seminars and so forth." In January 1958, Peter Elias (then EE Department head) suggested to John Ward that I should get together with McCarthy, Minsky, and Dick Marcus (who is still here at MIT, by the way) because they had just come into RLE (Research Lab for Electronics). Here on January 30, 1958, "I finally arranged to have lunch with McCarthy, which then transferred to Minsky's home in mid-afternoon and we talked some more until around 2:30 in the morning." Sounds typical. [Laughter.]

> **"We all quickly agreed that we wanted a time-sharing computer all right, but the question of how to go about doing it was the part that was not unanimous."**

Anyway, that was the first real ding-a-ling between us when they arrived on campus. It was right immediately after that, in January 1959, that we had this meeting between [Dean] Arden, [John T., Jr.] Gilmore, McCarthy, Seigel, myself, and whoever, to address the questions of using the Lincoln Lab [IBM] 709 and the sharing of IBM's real-time package.

Corbató: But that would have come out of McCarthy's memo.

Rosin: I sense what you're saying, Corby, but I wonder if you'll agree that John, independent of where the ideas were coming from...

Ross: He's the one who put them in focus.

Greenberger: He was the catalyst.

Ross: Right here [in notes], "McCarthy is calling a series of meetings to consider operator and compiler programs for the [IBM] 709 which will be transferred to the [IBM] 7090 which will replace the [IBM] 704."

Rosin: He pestered the administration to get this — which resulted in the formation of the committee?

Corbató: I have finally put my finger on what the key issue was. You'd have to ask the JOSS people whether they were influenced by John. My guess is they were; they'd heard about it. But what John was advocating was something that was harder than what they were trying to do. He was advocating a general-purpose system where you could program in any language you wanted. That's where Dartmouth and all of the others were taking out a slice of the problem but not the whole thing. John was very clear on that. Later, when we were trying to explain it to people, we would say we were trying to create a system where you could write the system in itself. That was not true of most of those interactive calculator types.

Greenberger: In the late fifties and early sixties there were many other thrusts in the direction of on-line multiaccess systems, but they were not general purpose. The [AN/FSQ-7] defense system was a multiaccess system, and so was the airline reservation system [SABRE], which was started in the same time frame. But these were special-purpose systems built to perform specific functions.

Corbató: One of the arguments we had with IBM engineers and executives was that they kept saying, "You do not need time-sharing. We are already doing that; we have our airline reservation systems." We tried to point out to them that that was a transaction system and had nothing to do with general-purpose programming.

Rosin: You're about to answer a question that I wanted to ask. Why was it called the "compatible time-sharing system"? [Spoken in unison — laughter.] What was compatible?

Corbató: In retrospect, it was a very modest problem. [Laughter.]

Scherr: Well, it was important for us who were trying to write programs.

Corbató: That actually is a good lead-in because it ties together with Teager's activities. Teager was the one who tried to start building a time-sharing system within the context of the Computation Center. Phil Morse helped him get funding, and he tried to get a group together — he had a small group. Herb's problem was he was too good a hardware engineer to want to take things off the shelf.

Greenberger: Even though he was advocating taking things off the shelf!

Corbató: Yes, but he proceeded to sketch out a system where he was going to do everything over. It was going to be a total operating system. In this sense, he did not think of it as general purpose, but you had to do it with *his* languages. He was going to build the hardware for the terminal attachment and the multiplexing of the terminals, and he was even going to address the fact that some people are better at writing than typing — he was going to allow input by handwriting analysis.

Greenberger: He invented what became the Rand Tablet at that period in time. Teager Tahlet. He may even have gotten a patent out of it — did he?

Ross: No, as a matter of fact Rand has got it.

Corbató: Herb tended to be a little close to the vest in terms of the way he managed and operated, and he did not find it easy to attract people to work with him. He had this grandiose vision and maybe a half dozen people trying to work with him. It looked like the timetable was going to stretch out a long time before anything came out of the door. Furthermore, he had a fundamental flaw — namely, that if you were going to use Herb's ultimate system, you had to abandon everything you had done before.

From The Development of CTSS to Project MAC

Corbatñ: I started up with just a couple of the key staff people, Marjorie Daggett (who was then Margaret Merwin) and Bob Daley. We hammered out a very primitive prototype.

Rosin: Why don't you give us a date, approximately?

Corbató: We started thinking about it in the spring of 1961. I remember that by the summer of 1961 we were in the heat of trying to work out the intricacies of the interrupts.

Rosin: What was going to be the user interface?

Corbató: We were going to borrow Teager's engineered typewriter interface. We were going to have a few Flexowriters initially as the keyboard input.

Rosin: This was on the [IBM] 7090?

Corbató: The [IBM] 709.

Lee: This was using Computation Center funding?

Corbató: It was all out of the same pot, sure — if you mean computer time.

Lee: No, no. I'm presuming you had some graduate students and other people who needed some salary and other support.

Corbatñ: No, it was just two staff people. There was no external funding.

Rosin: And the Long Range Study Committee report had not yet been released, is that correct also?

Corbató: That probably was true. I do not know the chronology there. But we were acting on the vision we had already internalized. I sketched out what we would try to [get] Marjorie, Daley, and I worked out the hairy details [of] trying to cope with this kind of poor hardware. By November 1961 we were able to demonstrate a really crude prototype of the system. What we had done was [that] we had wedged out 5K words of the user address space and inserted a little operating system that was going to manage the four typewriters. We did not have any disk storage, so we took advantage of the fact that it was a large machine and we had a lot of tape drives. We assigned one tape drive per typewriter.

Rosin: Weren't you sharing main memory with these people, or were you rolling in and out their programs?

Corbató: We had to roll in/roll out programs.

Rnsin: One at a time?

Cnrhatб: On tape. We may possibly have used a drum for some of that, but...

Greenberger: There was no drum on the [IBM] 709, as I recall.

Corbató: I think you're right. It was pretty miserable. We were just trying to get a demonstration system going to convince people that it was a good idea. A lot of people did not understand what it meant to interact. It was amazing.

The fact is that this system could operate [effectively] as long as nobody wanted an "all-the-core-memory" type job [ran] under the Fortran (FAP) monitor system. This system could coexist with that kind of an operating system and could run jobs. So we could run *compatibly*; we could run while ordinary work was being done on the [IBM] 709.

Greenberger: The feature that I thought "compatible" meant was that you could more or less take a program that ran on the normal FMS system and run it under CTSS. It meant [compatible] in both senses; it used the programming style, and languages it was using.

Scherr: A year or so later when Project MAC started, I had done an immense amount of programming to do some simulation work, and they told me that from now on I'm to get all my computer time on the time-sharing system. I said, "Oh, I'm going to have to rewrite all this stuff." Later I found out I did not, because everything I had written would more or less still run.

Greenberger: That was probably the primary meaning of compatibility. It was not only compatible with running together with the old system, but it meant you did not have to start over. So in fact we were able to get versions of Lisp running on it.

Corbató: Let me just get in a few dates. Somewhere in November 1961 we were able to give a seminar and demonstration; that's the date that's branded in my mind. It was only a four-Flexowriter system because it had four [magnetic] tapes, and it would just barely demonstrate. But you could have a live dialogue, and it would show that you could

"You're about to answer a question that I wanted to ask. Why was it called the 'compatible time-sharing system'? What was compatible?"

get a typewriter to respond. There was a background stream going on as well.

Rosin: This work was going on at the same time as people were trying to use this overloaded Computation Center to get their work done?

Corbató: Well, initially we would commandeer the machine for periods of maybe a half hour to do a trial run of this time-sharing system. Because of [the use of] batch processing, we could get away with that. [Laughter.]

Ross: And because you were the in-house crew. [Laughter.]

Corbató: Oh, there's no question about it; we took advantage of the fact that we had access.

Ross: This is the same time that I was feeling so squeezed, and I was making deals with IBM to get time outside out of their place. [Laughter.]

Corbató: We could not abuse it, so we had to be prudent as to how much time we used. A lot of the time was on weekends, which was when we managed to get the hardest bugs out.

Rosin: So, then in November you ran a demonstration — but you still were not providing service to anyone.

Corbató: Absolutely not. It was a demonstration. Then we went through a long hiatus where we had to make some fairly massive changes. I forget whether we had the memory protection properly installed, but in any case it was not used and we did not get it sorted out cleanly until we got the [IBM] 7090. The [IBM] 7090 hardware arrived in the early spring of 1962 and there was then a transitional period. The reason I'm so sure of this is that I wrote the paper about it, saying that we were running it on [IBM] 7090 hardware, thinking surely we would have it running by the time I gave the paper. [Laughter.] Well, it was an embarrassment, and I still live with that embarrassment. The paper said we were

running on the [IBM] 7090, but we in fact had not got it running yet.*

Rosin: Was there any effect by your demonstrations at the end of 1961 on the Long Range Study Group? The Long Range Study Group report was in the process of emerging at that time. There was the study group looking ahead into what MIT should launch for the future. [While the future was right there in front of them!]

Corbató: People were pleased that there were finally examples surfacing from [the work]. They did not view it as an answer to anybody's problem.

Rosin: They did not see what you were doing as the first step?

Corbató: Absolutely not.

Ross: 1959, January 12 to 21: "McCarthy is calling a series of meetings to consider operator compatible programs for the [IBM] 7090 computer which will be a transistorized version of the [IBM] 709, which will replace the [IBM] 704. We had a meeting yesterday."** So the things were showing up in the Teager report, one or two years later.

Corbató: Let me explain what had happened. We did the demo and we had some friends at IBM who worked very hard to help make this be a more viable system — Loren Bullock, in particular, who was the IBM liaison. Phil Morse was able to get us some money from NSF because there were three key upgrades we had to have besides switching to the [IBM] 7090: getting the proper memory protection and trapping the I/O instructions. Those were really critical hardware helps, as was getting a real-time clock that could interrupt. The first critical upgrade was a disk memory — I forget whether it was an IBM 1301 or 1302 — but it was a big one for the day. And that got us away from this absolutely ridiculous restriction that we improvised [assigning] one tape unit per terminal. Secondly, the compromise we had made by trying to steal 5K of memory from user program space was intolerable. We got IBM to engineer a second bank of memory so that we could put our supervisor program in it, and at the same time prevent users from writing over the supervisor easily.

The core memory was the second [critical upgrade]. A second bank of core memory, and the third key modification was that we needed something more extensive than a two-year-old home-built typewriter interface channel.

Ross: Was there a Comp Center [IBM] 7090 before the [Project] MAC Center [IBM] 7090?

Corbató: Let me spell it out. I know the chronology pretty well. We made the [first] demo in November 1961 on an [IBM] 709. The switch to the [IBM] 7090 occurred in the spring of 1962 at the Comp Center. These changes that I'm discussing are with respect to the disk memory and the core memory, and the [IBM] 7750 was the big typewriter interface unit — physically gross, but it did the job logically.

Greenberger: And a lot more capable than what you were supposed to use it for [laughter] — [it was] built to be a massive switch, but in the end it was never used for that.

Corbató: It was the only thing we could get our hands on that could take 32 terminals at once. Today is much different.

All those upgrades were in place by the spring of 1963. Now this was coming together fast enough, and the schedule was good enough, that when Bob Fano began to put together Project MAC, and his thinking began to gel during the Thanksgiving of 1962, it was part of the planning of Project MAC. It was possible to consider making CTSS a platform for Project MAC. The plan was to get a duplicate of the Computation Center [system] for Project MAC — and that was ordered approximately at the turn of the year. The intent was to use the system during the summer school, which was in the summer of 1963, but of course IBM could not deliver it quite so fast. So the [Project] MAC machine did not show up actually until October [1963]. But we had got the system up and functional in the spring of 1963 with all of these changes I described, and we were able to use it as the first platform in the summer of 1963.

Lee: I ran across several people who remember the MIT summer 1963 event — the CTSS summer school. Were those the first people from outside MIT who really got to use this system? What was their reaction?

Corbató: The intent of the summer session was to make a splash. We were absolutely frustrated by the fact that we could not get any of the vendors to see a market in this kind of a machine. They were viewing it as just a special-purpose gadget to amuse some academics. (They were humoring us.) They did not see it as affecting the productivity of people in trying to get things done. That was a major problem.

Scherr: I was at IBM during most of that period as a cooperative education student and later as an employee. That was not the case. That may be your impression.

Corbató: That was your impression. That was not the case from my point of view.

Scherr: Why?

Corbató: In fact IBM, being a big organization, had every thread of thought you could imagine, and we had some people working within IBM who were rooting as hard as they could and trying like heck to get the company to move in our direction more. They succeeded in part, which is why we got as far as we did. In fact, CTSS would not have been possible without a lot of help from IBM.

Scherr: As a matter of fact, when I got to IBM after I left Project MAC, I discovered that they had been doing a time-sharing system in Mohansic, in New York, on the exact same hardware that you had at Project MAC. It was called TSM (Time-Sharing Monitor), and some guys basically built it as a monitor. It had 30 to 40 users, and they were doing very much the same kind of stuff as was going on at Project MAC.

Greenberger: It was not until much later that people started using it seriously.

Corbató: Well, in fact, they did not take it seriously until we ultimately joined forces with GE [General Electric Company]. GE was really not that sophisticated as a computer company, but what the IBM people were afraid of was the deep-pocket financial help that the GE Company could give.

Greenberger: It was also the fact that you were soon thereafter joined by Bell Telephone Laboratories.

Corbató: And again we could argue the same thing.

Let me return to the impact of the summer study. We had approximately a 16-active-ports-at-a-time type system. Although in principle we could run a background stream, I think that in those days to make the maximum use of time-sharing, we did not have significant background work.

Rosin: So how did you run the Comp Center in those days — many hours of time-sharing, so many hours batch?

Corbató: In the Comp Center we had a background stream and we worked against that. I [had] jumped ahead to the time when we had our own machine. That was, in a sense, a heavy compromise because basically we took advantage of the fact that the batch processing people could not see the impact of interactive users. We got away with it, so to speak.

The result was that of the several hundred people who were involved through during the summer, most of them were persuaded — we did not pretend that we had a complete system. We did have a general-purpose system. We would write arbitrary programs, and since by that time we had fairly rapidly got a set of language choices, we were beginning to show the potential to be able to use any language. I do not remember when we got Lisp, but we certainly had MAD.* We had a version of Fortran based on some of your own work, Bob [Rosin].

People began to say, "Hey, this does seem to be like the right direction." I think we made a lot of converts, and people for the first time began to break out of this shell that had unfortunately descended on the computer industry (or the computer users) — namely, that the vendors knew best. People began to really seriously think that the architecture of machines might be different than it had been before.

Rosin: I'm curious about the effects inside the institute. You said that you were getting away with a lot. I've got some questions in the back of my mind. What was a significant application of CTSS by someone who had no involvement, no investment at all in the project? Were there people using time-sharing during the day during the spring of 1963, while you were getting ready for the summer session? Did people actually get work done in that environment?

"We took advantage of the fact that the batch processing people could not see the impact of interactive users. We got away with it, so to speak."

Corbató: I do not think we had [any of those] in the pre-summer session period. First of all, we did not have it debugged. We were conscientious about not taking too much time from the other job streams. And I do not think that we ramped up enough so that there were any major applications that I recall during that period.

Ross: We were running on the Cooperative Computer Lab [IBM] 709 machine. I do not believe we got onto the [IBM] 7090 until it was the Project MAC [IBM] 7094 in October 1963, because I have the printout of the first successful run of our AED compiler [on that machine], which was within two weeks of when they brought the system up. That's how long it took us to sift our way through to get a successful compilation on the Project MAC second machine. My impression was that it was supporting internal beta tests more than supporting [real] users.

Corbató: In one sense the Project MAC situation was one extended demonstration — except that it was a self-participating demonstration, where users really got to get on the machine themselves. Project MAC's first use of CTSS was during the summer session.

Rosin: You had a wild card naming system** for some files. How was that dreamed up? The idea of an interactive editor may have been obvious, but where did it come from? The idea of logging on and logging off? A runoff capability? Electronic mail?

Greenberger: Too many questions at once. [Laughter.]

Rosin: I'm trying to get the user's view of this strange thing.

* Editor's note: A prime example of why historians cannot even always trust the primary sources of information on an event!

** From Ross's Résumé.

* Michigan Algorithm Decoder — a dialect of Algol 58 — implemented by Bruce W. Arden, Robert M. Graham, and Bernard A. Galler.

** The ability to use a generic indicator for a name, or part of a name, usually denoted by *.

Corbató: Sure, we knew all of those things were pretty much cooked up by ourselves, but I hesitate to say we invented them — we felt all of those ideas were kind of state of the art. We were just picking ideas that were floating around and which were kind of the obvious thing to do if you were trying to put together a system. So the wild card idea — I find it hard to believe we invented the wild card because it was kind of a natural — and I must say it's a very ad hoc design. We just sat down and said, "Well, it would be reasonable to do," and "What would be nice?" and "How would you like to see this organized?" — and that's the way we cooked it up.

Greenberger: Some of the editing stuff came, as I recall, from TX-0 in a program called the Expensive Typewriter.

Corbató: So there were some [prior] prototypes of people interacting with machines which I'm sure were in the back of our minds. On CTSS, the first editors were card image editors. We did that for a reason. It was because we wanted the metaphor of a card as a quick and easy communication means, so we could explain to people quickly how [the system] was used. Once we got rolling, card images were an uninteresting artifact. The next round of editors quickly got rid of card images.

Rosin: That next round of editors — were those done by students, by staff, or did they just pop up?

Corbató: Well both. Jerry Saltzer, who was then a graduate student, did *typeset* and *runoff* — a document preparation system. He, in turn, borrowed ideas right and left from pattern matching and string matching. He did a good engineering job of pulling together all the things that were kind of recognized [as applicable]. In turn, as soon as people saw how successful [the system] was in doing basically what we now call word processing, it was clear that preparing a program was just yet another form of the word processing. Unfortunately, because [MIT] is the sort of place where people like to diddle and there are a lot of strong opinions, we soon had more editors than we knew what to do with.

Rosin: *In*compatibility! [Laughter.] Of course, your file system provided compatibility. You still had a single format to your file system.

Scherr: Well, but the different editors had different internal formats.

Corbató: We made one early decision — in fact, I believe I made it — to make the file system neutral to the contents. So the file system did not understand what it was filing. It was quite a deliberate strong boundary, and it meant that any language could store material in any format it wanted.

Ross: Also, any command could be stored as a program in any language, because they just had to have a common boundary and a common start point. That came from ancient Whirlwind. There was always this standard of starting your program at register 40 (octal), from which you would jump to the actual program start. That technique was just carried along with all this history, and so you had a general framework into which you could put data or be ready to run programs, no matter where they came from — you just worried about putting them on the disk.

Corbató: That was quite deliberate. We wanted a simple-minded system that was basically straightforward to use, and the notion of having a command library — with arguments — seemed obvious.

Scherr: The file system is kind of interesting. What I wrote [in my thesis] is that each track had 466 words, which apparently was an IBM parameter. The [IBM] 1301 we started with had nine million words on it. The files were stored with each track being chained to the next, so that when the disk was reorganized they put things pretty much in proximity, so that it went very fast. As the day, or the week, wore on and files were copied over, deleted, or created, the performance of the system would degrade as the distance between the pieces of the file got larger and larger. As a matter of fact, when I was doing a performance model a couple of years later, it turned out that the parameter measuring how well the disk was organized was the single most influential one in the simulation. It was almost a knob you could turn to get everything to match in terms of performance.

Greenberger: That's interesting. Because that's, of course, the same effect [found in] personal computers. [Laughter.]

Scherr: Exactly, the organization is very similar to what the PCs do — it was a scheme that we tried very hard to get away from when we went to the [IBM System] 360/370 systems.

Lee: Can you identify a time when people began to think of this as being a system for communication or a system for word processing, as opposed to a system for getting [computational] work done? Or did that not happen until the Project MAC era?

Corbató: Word processing started very early. The Expensive Typewriter example that was mentioned earlier was clearly just that. It was obviously an immediate application that people began to use for Saltzer's typeset and runoff. They were aimed at exactly that.

Lee: When did word processing become communication — e-mail?

Corbató: The communication aspects were a little later — one of the things that [Robert] Fano was responsible for. He correctly saw that a time-sharing system was more than just a set of people using a common resource; it was also a means of communicating and sharing ideas. We began to build [that system]. I do not think that was in our initial versions of CTSS, as much as we began to put it in the later evolution of it.

Rosin: During the Project MAC years?

Corbató: Yes, because we kept developing it [CTSS] for [illegible] a year or so within the context of Project MAC.

Rosin: My recollection, in looking at the early CTSS manuals, was that it was unusual for a user to be able to refer to some other user's files. [You were] logged in and you had your user [space]. There was a system for a universal file directory or whatever it was called, and then an individual user file directory. Was there a command in the early days to be able to refer to a file that was done by, say, Doug?

Scherr: [There was] a public file area [that] you could copy [illegible] to and from.

Corbató: That's right, we shared through a public file. In fact, we came up with a quick solution which was to create a public domain. That was the original thinking for libraries, handy programs, and the like. And then, when we got into the context of operating CTSS as part of Project MAC, we continued to develop this [concept]. In fact, as we began to think about our next time-sharing system, we decided to try [illegible] the ideas on CTSS, and so the second file system developed for CTSS had the concept of linking.

Ross: Until then, that public area was the same thing as the Compool was, in the programming context, from the old SAGE days. In order to go from one user to another, you put it in the common area...

Scherr: Looking at some of the statistics we took in those days, the typeset runoff facility that Jerry Saltzer did was [illegible] on the average about once per logon. This data was [illegible] toward the end of 1964 during about a four-month period,* so it was after the system had matured a little bit. Basically, what was going on was that typeset and runoff accounted for about 1 percent of commands issued and logon and logoff were about 1.5 percent.

Typeset was a mark-up language — when you used typeset you put in a document mark-up language to do things like centering, headings, etc.

Rosin: What was that editor called?

Greenberger: There were two versions — *ed* and *edm*.

Scherr: ed would [constitute] 4.5 percent of the commands. The *edit*, which was the card image editor, was 4 percent. So it was still being used after the shift that had already started to occur toward this more free-form editor.

Rosin: Was there a mail command?

Scherr: No, there was no mail command.

Ross: But you could find out who was on the system!

* See excerpts from Scherr's report in the next issue.

Corbató: No, no, no, no. But we were working in that direction, and there were a lot of the pre-Multics ideas during the last versions of CTSS. We had the aspiration of going in those directions, but it was a tough environment to work in. As we began to wind down, we put our effort into trying to put all the things we wished we had into Multics.

"A time-sharing system was more than just a set of people using a common resource; it was also a means of communicating and sharing ideas."

Rosin: A little bit about the terminal equipment. Two kinds of questions come to mind. (1) with respect to the kinds of equipment used and how that evolved in the early days, and (2) the first system for dial-up use of CTSS, during those early days?

Corbató: OK, good point.

The terminal system was very unsatisfactory. Everything was ugly in one way or another. Flexowriters were noisy, unreliable, and slow. The IBM equipment that we were able to get them to adapt always seemed to be designed for some other purpose. So even though they had Selectric typewriter elements in them, we had trouble getting type balls that had reasonable characters. We were also fighting another fight at the time, which was to get people to admit that upper- and lowercase were part of the computer vocabulary! We kept insisting that we wanted both. That was tough. Most IBM keypunches did not allow both, and Teletype did not allow both.

Scherr: If IBM and AT&T did not allow them, they must not be needed. [Laughter.]

Corbató: That's right. Even in the early manuscripts, I think there were a dozen terminals that we managed to test — almost all of them [were] unsatisfactory. Even the Teletype model 37 [which] was an upper- and lowercase unit (which was later followed by the 33, which was a cheaper version) was pretty clunky and noisy with slow response.

Rosin: Slow response, five characters per second? Ultimately IBM came out with some pretty decent Selectric units which were capable of being a good terminal.

Lee: The [IBM] 2741?

Ross: The [IBM] 1050 was a heck of a step up from the Teletype. That was the big thing around Project MAC in those days! *Who is going to get a 1050?*

Rosin: What about off-site systems? I recall some early anecdotes about people using CTSS from home.

Corbató: The next thing we did was we got the late Carlton Tucker to assign us some [telephone] lines, and we got AT&T to cooperate in giving us modems.

Rosin: Lines on the MIT campus?

Corbató: On the campus PBX. People could have them in their office, and they could dial up to CTSS. In fact, when we got two CTSSs on campus, it was convenient because you could then dial the one you needed or were trying to use at the moment.

Ross: Or the one that was not out. But before the dial-up came, I used to tweak Corby and Fano while they were setting all this stuff up with Tucker and deciding how to go about it. I had my own Computer-Aided Design Project money, so as soon as I realized that was going on, I moved [to CTSS]. I had not been able to program all the years we were getting off the Whirlwind and onto IBM. I never learned how to turn on a keypunch, and I had never touched an IBM machine before. I did some TX-0 programming — that's all. So I saw my chance to get back in and show my programmers what I had been trying to get them to do. So I just picked up the phone and called Teletype (which was a separate division of AT&T) and said, "Let's put one in!" They spent three months getting a line out to my home in Lexington. It was a great huge box in the basement. Although American television ignored it, when the MIT press office said this was the first one [in a home], Canadian Broadcasting Corporation [CBC] and British Broadcasting Corporation [BBC] sent film crews [to my home] in the spring of 1964 or 1965. So that was the first Teletype at a home, connected to a computer. It was a direct line though: I did not dial at all.

Rosin: Was it hardwired to the machine or hardwired to the switchboard?

Ross: Hardwired all the way to the computer. I had it there for two to three years, and once it was up it never made an error.

Corbató: I would say it had a couple of effects. It opened up people's eyes to the fact that distance did not need to be a factor. Of course, hard lines were feasible, but the dial-up was [the next logical step], and that led some people to try some real gutsy demos which came in from Europe and [elsewhere]. They discovered [that] they had nightmares with the telecommunications [in] trying to pull off demos through all those foreign switchboards. It was really wild.

So there are all kinds of stories around that. There was a fellow by the name of Joe Wegstein,* who used to work at the Bureau of Standards. He was visiting one day, and I was kind of showing off where we were. I walked up to a pool room where we had a bunch of typewriters. I dialed up a number, and I accidentally dialed the Project MAC machine. I meant to dial the one downstairs in the Computation Center. It was not until the login masthead came up that I realized the mistake and said, "Whoops. I made a mistake. I'm not dialed into the [Project] MAC we were just looking at: I'm dialed into another one." He said, "Where is it?" I said, "It's over there." I pointed across the street to the other building and he looked at me somewhat incredulously — I don't think he believed me. Nowadays that's taken for granted, but in those days that was really [surprising].

Scherr: I remember being in the [Project] MAC room one night. In those days whenever anybody logged on or off, the [activity] was printed on the big line printer that was part of the [IBM] 7090. Somebody from England had dialed in and was clunking away at five characters per second with [Project] MAC... I forget who it was.

Lee: Maurice Wilkes?

Scherr: It was Wilkes, as a matter of fact! We knew who it was because it said "Wilkes" and somebody said, "Oh, that's the guy from England."

Rosin: What year was that?

Scherr: 1964 or 1965 or thereabouts.

Rosin: So things were really starting to happen after that summer session?

Scherr: I was a graduate student in those days, and I had just got my master's degree. I had spent a year as a teaching assistant and most of my career learning how to do logic design. I had decided that there was no future in trying to get a PhD by building hardware, because it was so hard to get funding to build things. So I decided I'd do some programming for my PhD work, and what I had chosen was logic simulation. I had written some programs to do that. I had been an IBM co-op since my sophomore year (which was 1960), and by the summer of 1963 I was a summer employee of IBM in the Cambridge Scientific Center, which in those days was up by Harvard Square. I spent that summer writing software to do logic simulation.

So now I came back to school in September of 1963 after the summer session. Herb Teager was my thesis supervisor. He said, "Everybody who's doing anything with computers is in Project MAC. That's where the research money is, so that's where we're going." My reaction was, "OK, as long as there is money and as long as I get computer time." Also, the question was, "Do I have to convert my programs [again]?" I had completed almost all my course work for a PhD, so the next job was to get my topic picked. The committee I had was [Robert M.] Fano and Ron Howard, who was the operations research guy, and Dave Liu.

Herb tried to talk me into doing modeling of time-sharing systems — performance modeling. I always thought that what he really wanted me to show [was] how terribly inefficient, slow, and unwieldy this whole thing was. I took some measurements of reality to compare [with] the model. I asked, "What is being done on CTSS to measure its performance, its efficiency, its characteristics, etc.?" As it turned out, there really was not any. Tom Hastings had done some very early, very good work. But that was about it. It was not even close to what I thought was needed. Here I was, probably early in 1964, more or less committed to a course of research that required me to either discover measurements wherever they were or make them myself.

Corbató: That's a very interesting... I'm glad you brought that up. There was a difference of viewpoint between Teager's thrust and ours, which in hindsight is easy to see. Herb was trying to engineer it from the ground up. He wanted to do it just right; he wanted to see the system running efficiently; he wanted to see good user resources; and, if he could not get it that way, he was willing to wait. We were trying desperately to get a prototype going to get people's attention. So Herb's project ultimately never came off because it kept disappearing into the future, and ours would shitt in a very ad hoc fashion and [a] "let's get it going" [way]. Clearly it became the thing that we built upon.

Scherr: And it gave you something to measure. [Laughter.]

Corbató: We would try to [develop a] prototype because we saw a major watershed in the way people looked at computers in terms of whether they could interact in real time or not. So it was important to get a machine that did that — whereas Herb was still focused on trying to justify [the fact] that he was making good use of all the resources.

Rosin: There's an interesting contrast between different people's views of reality in what you just said. You said *resources*. You see, if what you're trying to do is optimize technical resources (physical resources), Herb's point of view was exactly right. If you try to optimize the use of human resources, then the point of view you were taking was a lot closer to reality.

Lee: This ad hoc development — it sounds like it was almost independent of the [study] committees!

Ross: As you debate the different views between Teager and Corbató and Fano, I would like you to realize that you're hearing from two insiders with respect to us outsider groups. The fact is that availability of a Project MAC [IBM] 7090 [using] CTSS, with all this eagerness to invite outside groups to make it fly — make it useful and so forth — had quite an impact on us. We were on the crack; we were neither inside nor outside, but were not the proper kind of outside either, because we were providing a service to other people. That was our mandate as the Computer-Aided Design Project. I always say, "You can't design an interface from just one side."

Corbató: In terms of what influenced us — it was basically the thinking that got caught up in the small Long Range Study report. That was clearly what was on my mind, and I was the leader of the CTSS development. It reflected completely an attempt to implement those ideas using contemporary tools. I never claimed to be more than a person who brought into development the ideas we had been all talking about together. So I was trying to [develop a] prototype, but I was not working on an independent theme. My inspiration was the same inspiration as the Long Range Study report. ■

*Joseph Wegstein, Chairman of the CODASYL Short Range Committee, — staff member, National Bureau of Standards (now NIST), see *Annals*, Vol. 7, No. 4.

Prolog to the Future

By the early 1960s, work on different implementations of the computer utility vision had begun at various places. As we have seen in the previous pages, the Cambridge, Mass., area was particularly active, initially because of the influence of John McCarthy. Besides the Compatible Time-Sharing System (CTSS) of F.J. Corbató, initially on an IBM 709 (1961) and later on the IBM 7090 and 7094 (1963), some of the other first working prototypes were also at MIT, for example, the DEC PDP-1 system of J.B. Dennis. At the Bolt Beranek and Newman Company in Cambridge, a time-sharing system based on a DEC PDP-1 was developed by J. McCarthy, S. Boilen, E. Fredkin, and J.C.R. Licklider. Other early influential prototypes were the Dartmouth College BASIC system of J. Kemeny and T. Kurtz, initially implemented on a GE 234; the JOSS system implemented at the Rand Corp. by C. Shaw; and, at the System Development Corp., a time-sharing system developed by J. Schwartz for the AN/FSQ-32 military computer. These developments were tracked for a short period by Lewis Clapp, then president of Computer Research Corp. of Belmont, Mass., in the "Time-Sharing System Scorecard." The last of these, published in fall 1967 (reproduced here, in modified form, with Clapp's permission), clearly shows the proliferation of systems that had occurred in the few years since Corbató demonstrated the first version of CTSS.

Time-Sharing System Scorecard

This guide is prepered periodically to keep the reader abreast of the rapidly increasing number of time-shared computer systems which ere bringing man and machine together in close partnership for the pursuit of intellectual and edministrative ectivities. By glancing at the following charts the reader can judge for himself the progress which is being made in this new end dynamic field. There are several different definitions of time-sharing. No single definition is edequate for all purposes. We heve limited this survey to systems which have at least two independent, remote and simultaneously operable consoles (from the user's point of view). If the language capabilities of the system ere extensive and general so thet e user can create new languages while working on-line, we have denoted this as a general-purpose time-sharing system. Where the languege capabilities ere more restrictive, permitting the user to work in only one specific problem area, we heve used the term special-purpose time-sharing system.

The number of commercial and research time-sharing systems has grown so rapidly in the past several months that it is no longer possible to list each individuel system in a brochure of this size. Therefore, we heve listed only the first or major occurrences of any time-sharing system as has been supplied by the organization involved. In some cases the numbers may be unduly optimistic, but we have let the data stand as supplied.

Recent development

The continued growth of commercial time-sharing is perhaps the most striking feature in this edition of the scorecard. Almost every major city in the country now has local access to at least one time-sharing service. Most organizations that now offer service are expanding geographically, and several new companies are entering the scene. In addition, specialized time-sharing services are beginning to emerge such as the financial analysis service by White, Weld & Co., which will be built upon an SDS 940. Meanwhile, General Electric's MEDINET division has started to offer time-shared service to some hospitals in Massachusetts and New York on an experimental basis.

It is interesting to observe the almost universal slippage in the development of the newer time-sharing systems. Project MAC, which expected to be in normal operation with a time-shared GE 645 in June of 67, now estimates that it will be operational in the summer of 68. Performance of the IBM 360/67 has also been disappointing, and some organizations that originally hoped to be operational by this time have declined to make any further predictions as to when their time-sharing capability will be fully realized. The SDS 940, which was reported as a fully operational 32-user system some time ago, has also been beset by software problems. SDS now estimates a 32-user capability in March of 1968.

New machines that have been announced with a time-sharing capability include the Digital Equipment Corp. PDP-10, the RCA 70/46 and the Burroughs 5500.

Research-oriented time-sharing systems

(Adapted with permission from the "Time-Sharing System Scorecard" prepared by Computer Research Corp. in fall 1967.)

Organization	Status	Type	Computer(s)	Languages	Terminals	Main Storage	Secondary Storage	No. of Users	Remarks
Bell Telephone Laboratories,[1] Murray Hill, N.J.	D (1/68)	G	GE 645[2]	FORTRAN IV COBOL, PL/I, SNOBOL	TT-37 IBM 1050 CRT (10)	256K	DK (40M wds.) DR (4M wds.) Tape loop (100M wds.)	100	Highly interactive system for research and production computing.
Bolt Beranek and Newman, Inc.,[3] Cambridge, Mass.	D (6/64)	G	PDP-1D[4]	MIDAS, TELECOM[5]	TT-33 (90)	24K (4K)	DR (128K wds.) DR (2 units, 25M wds. each) MT (2 units)	64	Medical information and communications system for hospitals. Also used for computational and data management facility.
CSIRD, Canberra City, Australia	D (7/66)	G	CDC 3600	CIDER, INTERP, STATIST, 3600 FORTRAN, COMPASS	CDC 210 (6) CDC 250	32K (2K)	DR (2 units, .5M wds. each unit) DK (12.5M wds.) MT (8)	7	General-purpose scientific computations.
Dartmouth College, Kiewit Computation Center, Hanover, N.H.	D (9/67)	G	GE 635 Datanet-30 (4)	BASIC, ALGOL-30, (1/68), FORTRAN (1/68)	TT-35 (55)	64K (24K)	DR (768K wds.) DK (2 units, 4M wds.) MT (6 units)	200	Datanet-30s have 16K core memory each. Educational and research use.
Edinburgh Univ., Dept. of Machine Intelligence and Perception, Edinburgh, Scotland	D (6/67)	G	Elliot 4120	POP-2	TT (20)	32K (16K)		8	POP-2 language suitable for list processing and numerical computation using FORTRAN type statements.
General Electric, Research & Development Center, Schenectady,	O (7/66)	G	GE 265	BASIC, ALGOL, FORTRAN, LISP, and others	TT (45) CRT (3)	16K (6K)	DK (20M char.) MT (6 units)	21 (TT) 3 (CRT)	Uses include scientific programming, data acquisition from experiments, system programming development.
Lawrence Radiation Laboratory, Univ. of California, Livermore, Calif.	Partial D (7/67) Complete D (6/68)	G	PDP-6 (2) CDC 7600 CDC 6600 (3) CDC 3600 IBM 7030 IBM 7094 (2)	FORTRAN, LISP, assembly languages	TT (200) PLT (10)	256K[6] wds.	DK (8×10^8 bits) DC (3.2×10^9 bits) Photo-digital store (1×10^{12} bits)		Use is mostly scientific computation.
Lincoln Laboratory (MIT), Lexington, Mass.	D (2/66)	G	TX-2	CORAL, VITAL, MARK 5, LABGOL	TY (5) CRT (4) Rand Tablet PDP-338 Remote terminal	105K	DR (20M wds.)	6	System features fast response time for on-line graphical communication.
[illegible] Laboratory [illegible], Lexington, [illegible]	D (9/67)	G	IBM 360/67	Macro assembler, FORTRAN IV	IBM 2741 (30)	128K	DR (1M wds.) DK (56M wds.)	20	Establishment of a large computational facility for scientific and engineering research.
Lockheed Georgia Co., Marietta, Ga.	O (7/65)	G	IBM 360/50	FORTRAN IV	IBM 1050 (36) IBM 2200 (6)	64K (20K)	DK (6.45M char.)	42	System named RAX, developed from earlier 360/40 system, used mostly for engineering.
Lockheed Palo Alto Research Laboratories, Palo Alto, Calif.	O (12/66)	S	IBM 360/30	360 assembly language (7/68)	IBM 2260 (4) Sanders 720	64K bytes	DK (2 units, 2.75M bytes each) DC (418M bytes) MT (1 dual)	12	System named LACONIC. Information retrieval and updating, research.
MIT, Dept. of Civil Eng., Cambridge, [illegible]	D (6/68)	S	IBM 360/40	COGO, STRUDL, ICETRAN	IBM 2741 (5)	128K bytes	DK (2)	10	System named ICES. Uses include engineering, science, management.
[illegible] Dept. of Electrical Eng., Cambridge, Mass.	O (5/63)	G	PDP-1	Macro assembler	TY (5)	12K (8K)	DR (88K wds.)[7] MT (6 units)[8]	5	Experimental time-sharing system for student use in thesis and research projects.
National Bureau of Standards, Washington, D.C.	O (4/66)	G	MOBIDIC B[2]	DESCAL CLA, CAS, EDIT	TT-33, 35 (4) CRT	16K (6K)	DK (1M wds.) MT (4 units)	6	Uses include research in the design of on-line systems and terminals.
Northern Electric Co., Ltd., Research & Development Laboratories, Ottawa, Ontario, Canada	O	G	CDC 3300 (2)	FORTRAN, PL/I, COBOL, COMPASS	TT (70) 8130 (4)	82K 65K (4K)	DK (12 units, 8.2M char. each) MT (10 units)	35	Scientific and business use.
Ohio State Univ., Columbus, Ohio	D (9/68)	G	IBM 360/50 IBM 360/75	PL/I, FORTRAN IV	IBM 2741 (20) IBM 2260 (8)	512K bytes 1024K bytes	DR (1 unit) DK (1 unit)	14	
Perkin Elmer Corp., Norwalk, Conn.	O (10/67)	G	SDS 9300 SDS 930	SPEED (text editor), FORTRAN IV, META-Symbol, SLIP (12/67)	TT-33, 35 BR IBM 2741 IBM 1050	32K 16K (22K)	DK (67M char. 1/68) MT (4 units) DR (4 units, 8M char.)	16	Uses include leo design, circuit analysis, scientific engineering, and research.

(Table continued on the following page)

("Research-oriented time-sharing systems" continued)

Organization	Status	Type	Computer(s)	Languages	Terminals	Main Storage	Secondary Storage	No. of Users	Remarks
Project MAC, Phase I, MIT, Cambridge, Mass.	O (10/63)[9]	G	IBM 7094	ALGOL,[10] FORTRAN, MAD, LISP	TT-35 (54) IBM 1050 (56) TLX (1) CRT (2)	64K (32K)	DK (36M wds.) DR (.5M wds.) MT (12 units)	30	Project MAC is an MIT research program sponsored by the Advanced Research Projects Agency (ARPA), DOD, under a contract with the Office of Naval Research.
Project MAC, Phase 2, MIT, Cambridge, Mass.	D (1/68)	G	GE 645[2]	ALGOL,[11] COBOL, FORTRAN IV, PL/I	TT-37 IBM 2741	256K	DK (40M wds.) DR (4M wds.) MT (8 units)	100	Initial limited system operation by early 1968 with continued development thereafter.
Purdue Univ., Computer Science Dept., Lafayette, Ind.	O (9/67)	G	IBM 7094	File generation, TEXT-90 (12/67), FORTRAN	TT-33 (2) IBM 1052 (2)	32K (16K)	DK (9M wds.) MT (9 units)	4	System named PTSS.
Rand Corp., Santa Monica, Calif.	O (11/65)	G	PDP-6	JOSS II	TY (30)[12]	32K	DK (6M wds.) DR (1M wds.)	30	Interpretive system with compact conversational language for small numerical problems.
Stanford Univ., Stanford, Calif.	O (8/64)	G	PDP-1	Assembly language	Philco CRT TT-33, 35	32K (12K)	DR (131K wds.) DK MT (2 units)	100	An IBM 360/67 will be operational in 1968.
System Development Corp., Santa Monica, Calif.	O (1/64)	G	AN/FSQ-32 PDP-1	TINT, IPL-TS, JOVIAL, LISP	TT-33, 35 TY, CRT, TLX IBM 1052	65K (47K) 16K buffer	DR (5 units, 139K wds. each) DK (1 unit, 4M wds.)[13] MT (12 units)	30	Oriented to command and control experimentation and other general uses.
TRW Systems Group, Redondo Beach, Calif.	O (1/65)	S	Bunker-Ramo 340	Culler-Fried system for mathematical analysis	4 consoles[14] BBN console	16K	DR (48K wds.) MT (2 units)	4	Highly flexible system for on-line manipulation, specification, and execution of mathematical and symbolic operations with graphical display of results.
UCLA Western Data Processing Center, Los Angeles, Calif.	O (11/64)	G	IBM 7740[15] IBM 7040/7094		IBM 1050 (12)	32K	DK DR	12	Jointly financed by UCLA and IBM, the system services UCLA and 88 other California schools.
United States Military Academy, West Point, N.Y.	O (12/65)	G	GE 225 (3) Datanet-30	CADETRAN[16]	TT (15)	8K 16K (6K)	DK (18M char.) MT (2 units)	15	
Univ. of California, Irvine	O (1/66)	G	IBM 1410 IBM 1440	JOSSI[9] Coursewriter	IBM 1050 (18)	100K char.	DK MT (5 units)		Uses include computer-assisted instruction and the administration of student enrollment.
Univ. of California, Irvine	D (1/68)	G	IBM 360/50	ISIS, CAL	IBM 2741 (28) IBM 2260 (3)	512K bytes (8K bytes)	DK (12 units, 7.25M bytes each) MT (4 units)		Instruction, administration, and research.
Univ. of California, Project GENIE, Berkeley, Calif.	O (4/65)	G	SDS 930	FORTRAN II, ALGOL, LISP, SNOBOL, CAL, DDT, QED, ARPAS, QSPL	TT-33 (8) TT-35 (8) CRT (2) Dataphone (6)	48K (38K)	DR (1.3M wds.) MT (2 units) DK (144M wds.)	16	Features hardware address mapping. The SDS 940 system is based on the results of this ARPA-sponsored project.
Univ. of California, Santa Barbara	D (1/67)	G	IBM 360/50	Culler-Fried system, FORTRAN IV	20 consoles[14] Rand tablet IBM 1050 (3)	64K	4 OK (1.8M wds.) DR (1M wds.) Core (.5M wds.)	16	Extension of the Culler-Fried system now operating on the RW400. The 360/50 system has simultaneous background processing.
Univ. of Illinois, Urbana	O (1/66)	G	Illiac II PDP-7	FORTRAN	TT-33, 35 (8) CRT	8K (6K)	DK (10M wds.) DR (64K wds.)	7	Experimental time-sharing system for general university research.
Univ. of Massachusetts, Amherst	O (9/67)	G	CDC 3600 PDP-8/680	BASIC, SNOBOL, COGO, SMALL, FORTRAN IV	TT-33, 35	32K (8K)	DR (2 units, 2M char. each) DK (2 units, 8M char. each) MT (4 units)	32[17]	Uses include education and varied research programs in diverse fields.
Univ. of Pennsylvania, Philadelphia	O (6/65)	G	IBM 7040 PDP-8	FORTRAN, MULTI-LANG, MAP, ALGOL, LISP, SNOBOL	TT-35 (4) BR (2)	32K (24K)	DK MT (6 units)	6	Uses include information retrieval, research, and multiprogramming experimentation.
Univ. of Pittsburgh, Computer Center, Pittsburgh, Pa.	O (3/66)	G	IBM 360/50	ALGOL, PIL,[5] FORTRAN IV, PL/I (1/68), Assembler	IBM 1050 (3) IBM 2741 (20)	128K bytes 1M bytes LCS (32K bytes)	OK (2 units, 7.5M bytes)	24	General university research and education. On-line LINC-8 for medical research available in late 1967.
Univ. of Utah, Salt Lake City	D (12/67)	G	Univac 1108	FORTRAN V, TRAC, COBOL, ALGOL	TT-35 (20)	131K (65K)	DR (6 units, 1.5M wds.) MT (8 units) FastRand II	20	Plans include the addition of more displays and bulk core memory.

Notes

1. Development in cooperation with Project MAC, MIT.
2. Multiple-processor time-sharing system.
3. Developed with the Massachusetts General Hospital under contract from the National Institutes of Health.
4. Based on an earlier five-station PDP-1 system operational 9/62.
5. Based on the Rand JOSS language.
6. The 256K core storage applies only to the PDP-6s.
7. To be replaced by a larger model early in 1968.
8. Units have been installed but are not operational.
9. Initially time-shared in 1961 at the MIT Computation Center.
10. Other languages include FAP, SLIP, COGO, SNOBOL, STRESS, GPSS, COMIT, OPL-1, and OPS-3.
11. Most Project MAC Phase 1 languages will be implemented later.
12. Selectric with JOSS keyboard and paging.
13. Additional unit available in January 1968.
14. Each console consists of two keyboards and a storage tube display.
15. System currently utilizes five computers in addition to the control IBM 7740.
16. CADETRAN is an extended teaching dialect of FORTRAN.
17. 64 with added communications ports.

Characteristics listed in charts

Status
- O Operational system; number in parentheses is the approximate date that the system went on the air.
- D System under development; anticipated date that operations will begin.

Type
- G General purpose
- S Special purpose

Computer
Manufacturer's name and number of central computers in system.

Languages
Basic languages available on the system at present.

Terminals
Type of terminal equipment available; number of such terminals in parentheses.
- TT Teletype; number following denotes terminals and model number.
- TY Typewriter
- TLX Telex console
- CRT Cathode-ray tube display
- BR Bunker-Ramo Series 200 display consoles
- IBM IBM 1050, 2741 keyboard consoles; IBM 2250, 2260
- Philco Philco display consoles
- PLT Plotter

Main Storage
First number denotes total core storage in words on the system; second number in parentheses, if given, denotes maximum core storage available to an individual user.

Secondary Storage
- DR Magnetic drum
- DK Disk file
- DC Data cell
- MT Magnetic tape
- Core Bulk core
- CF Random-access card file (K = 1024, M = 1,000,000 words per unit).

No. of Users
Maximum numbers of users who can operate simultaneously at any given time.

Commercial time-sharing systems

Users could purchase remote, on-line, and interactive computer services from the organizations listed below. (Adapted with permission from the "Time-Sharing System Scorecard" prepared by Computer Research Corp. in fall 1967.)

Organization	Computer	Conversational Languages	Terminals	Number of Users*	Minimum Charge per Month	Average Charge per Terminal Hour	Charge per Minute of CPU Time	Disk Storage per Customer†
Allen-Babcock Computing, Inc., Palo Alto, Calif.	IBM 360/50[1]	PL/I (on-line subset)	IBM 2741 TT-33, 35, 37 Friden 7100 IBM 1050	90	$385.00	None	$5-$10[2]	100K+
Applied Logic Corp., Princeton, N.J.	OEC POP-6, POP-10 (12/67)[3]	FORTRAN IV, DDT, JOSS, MACRO-10, Compact COBOL, LISP, SNOBOL-6	TT-33, 35 CRT	30[4]	None	$5.00	$6.00	0+
Bolt Beranek and Newman, Inc., Cambridge, Mass.[5]	PDP-7/8	TELCOMP	TT-33	*	None	$12.50	None	None
CEIR, Inc., Arlington, Va.	GE 235 Datanet-30	BASIC, ALGOL	TT-33, 35	40	$250.00	$6.00	None	120K
Computer Sharing, Inc., Bala Cynwyd, Pa.	SDS 940	CAL, ARPAS, BASIC, DDT, FORTRAN IV, FORTRAN II	TT-33, 35	32	None	$30[7]	None	60K+
Com-Share, Inc., Ann Arbor, Mich.	SDS 940	BASIC, CAL, FORTRAN IV, SNOBOL, TAP, DDT, FORTRAN II	TT-33, 35	64	$100.00	$10-$20	$2.50	0+
Dial-Data, Inc., Newton, Mass.	SDS 940	CAL, DDT, QED, FORTRAN II, BASIC, ALGOL, FORTRAN IV, SNOBOL, ARPAS	TT-33, 35	32	$100.00	$13.50	$3.00	60K+
General Electric Co., Information Service Dept., Bethesda, Md.[8]	GE 235 Datanet-30	BASIC, ALGOL, FORTRAN	TT-33, 35 PLT	40	$100.00	$10.00	$2.40	0+
International Business Machines, New York, N.Y.[9]	IBM 7044	QUICKTRAN	IBM 1050 IBM 2741	80	$125.00	$12.50[10]	None	0+
Intineo Limited, London, England	Univac 418 (2)	Stockbrokers' Language	TT-33 (60)	60	11	11	11	
Keydata Corp. (Adams Assoc.), Cambridge, Mass.	Univac 491	KOP III	TT-28	200	12	12	12	
Pillsbury Occidental Co.,[13] Raleigh, N.C.	GE 265	ALGOL, BASIC, FORTRAN	TT-33, 35 PLT, CRT	40	$108.50	$10.00	$3.00	0+
Realtime System, Inc., New York, N.Y.	IBM 360	FORTRAN IV, COBOL, ALGOL	TT, TWX, CRT	15	$500.00	$15.00	$8.35	0+
Tymshare, Inc., Los Altos, Calif.	SDS 940	CAL, BASIC, QED, DDT, FORTRAN IV, ARPAS, ALGOL	TT-33, 35 PLT	60	$80.00 or $390.00	$13-$16	None	60K+
VIP Systems Corp., Washington, D.C.	IBM 1440	IBM Administrative Terminal System	IBM 2741	40	$375.00	$7.50	None	100K+

* In all cases, the number of simultaneous users can be increased by the addition of equipment or by duplicating the computer system.
† Number denotes amount allocated in characters or bytes; + indicates more available at extra charge.

Notes

1. Special operation codes for efficient conversational interaction added.
2. Dependent on amount of core used.
3. This new system will be in operation early in 1968.
4. Will be increased to 40 in late January.
5. Systems located in Cambridge, Mass.; East Orange, N.J.; and London, England.
6. Cambridge and East Orange handle 32; London handles 16.
7. For the first 20 hours; $25 per hour thereafter.
8. Service available from offices located in 33 major metropolitan areas.
9. Other systems in Chicago, Cleveland, Philadelphia, Los Angeles, and Toronto.
10. For the first five hours, $11 for hours six through 75, $9 per hour thereafter.
11. A charge of approximately $5,000 per year plus a usage charge of $.05 per inquiry.
12. For accounting and management. Charges on the basis of message transmissions, processor time, and storage used.
13. Trade name Call-A-Computer.

The emphasis differed in each case. CTSS was oriented toward a general-purpose service offered by a central computing service. The MIT PDP-1 system was organized to allow each user direct control of I/O devices in native machine languages, but with protection from other users, so that each user was presented with a virtual machine capable of running arbitrary programs. The BBN PDP-1 system was oriented toward an environment for interactive program development which included the use of a high-performance graphical display. The Dartmouth system focused on introducing computing to nonprofessionals with the constrained but easy-to-learn BASIC language; the JOSS system focused on a carefully human engineered computational programming interface; and the developers of the AN/FSQ-32 system were interested in similar objectives to those of CTSS, but in the context of developing and maintaining large programs for military applications.

These time-sharing systems, while among the earliest and more significant, were not the only ones developed in the 1960s. Rather they display some of the variety of objectives and directions taken. With hardware obsolescence, none of the systems has survived, except for the BASIC system, which, with changes of hardware, continues to evolve as the time-sharing computing facility of Dartmouth College. Nevertheless, the early systems have had direct influence on almost all the current time-sharing systems in use, frequently by the students of one system becoming the designers and implementers of the next.

By the mid-1960s, time-sharing systems, and especially those of MIT's Project MAC and of Dartmouth College, had attracted considerable attention among computer users, managers, and manufacturers. The obvious impact of time-sharing systems forced these different groups to reevaluate their roles and the desired modes of computer use. Moreover, development of extensive new time-sharing systems had begun. Among the more notable plans were those for the Multics system (by MIT's Project MAC, the Bell Telephone Laboratories, and the General Electric Company) and the TSS system (by IBM for the IBM 360/67), which were especially comprehensive in their goals. Indeed, this very comprehensiveness led to underestimations of the scale of the software engineering required. The Multics system, eventually marketed by Honeywell (which had acquired the GE Computer Department), took several years longer to develop than initially anticipated. The TSS system, implemented by a much larger group, was not as delayed as Multics, but had disappointing performance and human interfaces when first delivered. Despite these warning signs of engineering complexity, by the end of the decade, dozens of time-sharing system implementations were being developed both by ambitious users and by major manufacturers, and time-sharing was well recognized as a significant mode of computer interaction.

The series of "Time-Sharing System Scorecards" published from 1965 to 1967 by the Computer Research Corp. chronicles the development of time-sharing systems in research organizations (universities and laboratories) and the expectations for commercial offerings. The last of these scorecards (reproduced here) shows a stage of development when time-sharing was still new enough to be a research effort and not quite mature enough to be commonplace. ■

SOFTWARE—PRACTICE AND EXPERIENCE, VOL. 11, 741–744 (1981)

A Note on the Multics Command Language

A. W. COLIJN

Department of Computer Science, The University of Calgary, Calgary, Alberta, Canada T2N 1N4

KEY WORDS Operating system Command language Recursion Multics Ackermann's function Towers of Hanoi

The Multics operating system,[1, 2] currently available on a series of Honeywell machines, was developed starting in 1964 as a joint effort involving Project MAC of M.I.T., Bell Telephone Laboratories, and the computer department of General Electric Company, later taken over by Honeywell Information Systems, Inc. This operating system is very nicely structured, and it is intended for, and very well designed for interactive use. The command language, therefore, is also designed to make interactive use of the computer convenient. This is achieved through a combination of features, including the availability of powerful commands, the availability of standard and user-defined abbreviations, the availability of on-line documentation, and the great consistency in the use of arguments to the commands.

In addition to containing a number of powerful commands for interactive use and immediate execution, the Multics command language is also a reasonably complete programming language, since there is provision in Multics for defining files, called segments in Multics, containing commands; these commands are executed when such a segment is invoked by giving its name as the first argument to the *exec_com* processor. These so-called *exec_com* segments may have parameters, which are passed by value, and they may be called recursively. An extensive set of so-called active functions is provided in the operating system; these active functions allow certain operations, including arithmetic ones, to be performed, and they permit access to many system values and parameters ranging from the time of day and the electronic mail system to such things as the search rules for locating segments in the hierarchical file directory system.

Commands in *exec_com* segments fall into two categories: the regular Multics commands, and commands which can occur only in *exec_com* segments. The latter, which are distinguished by being preceded by an ampersand, include the *&print* command, and the *&if* command, which permits conditional excecution of commands, but which, regrettably, may not be nested.

A weakness in the *exec_com* facility is the fact that there are no facilities for declaring local variables, and even the facilities for declaring and using non-local variables are inconvenient, and very poorly documented. For example, since it is apparently assumed that most *exec_com* processing will be concerned with strings, if the value assigned to a variable by means of the *value$set* command is an arithmetic expression, the command assigns to the variable the string representing the expression (which may be evaluated later using the active function *value*, rather than the value of the expression.

0038–0644/81/070741–04$01.00

Received 12 November 1980

In spite of its weaknesses as a programming language, the command language, together with the *exec_com* processor, is a powerful programming facility. Of course, the validity of this claim may be proven or illustrated in a number of ways. Following an earlier paper,[3] we use two examples as illstrations: the Towers of Hanoi problem and Ackermann's function.

The well-known Towers of Hanoi problem[3-5] has an elegant recursive solution, as follows. In order to move n discs from stack 1 to stack 3, say, begin by moving the top $n-1$ discs from stack 1 to stack 2, according to the rules, and provided that $n>1$. Then move the remaining, largest disc from stack 1 to stack 3, and finish up by moving the $n-1$ discs from stack 2 to stack 3, again according to the rules, and again provided that $n>1$.

The solution using the *exec_com* facility of Multics, shown in Figure 1, follows the above recursive solution directly. The segment named *Hanoi.ec* (where the suffix *.ec*

```
&  This is the segment Hanoi.ec; it has four arguments
&  representing the number of discs and the source,
&  intermediate and destination stacks. Example:
&     ec Hanoi 4 "s" "i" "d"
&
&command_line off
&if [greater &1 1] &then ec Hanoi (minus &1 1] &2 &4 &3
&print move disc &1 from &2 to &4
&if [greater &1 1] &then ec Hanoi [minus &1 1] &3 &2 &4
```

Figure 1. Multics exec_com segment for the Towers of Hanoi problem

indicates the type of the file to the operating system) contains the necessary commands, with the first one,

&command_line off

being used only to suppress the automatic printing of Multics commands as they are being executed. It is worth noting that references to formal parameters are made by their positions—thus &1, &2, &3 and &4 denote the four formal parameters—and that *ec* is (an admissible abbreviation of) the Multics command to execute an *exec_com* segment. Hence the parts of the last and third-last lines of Figure 1 which read

ec Hanoi [*minus* &1 1] ...

are recursive calls of the *exec_com* segment Hanoi, with the first argument, the number of discs, reduced by 1. The *&print* statement causes the string which follows on the same line to be printed, but only after parameter substitution has taken place.

Ackermann's function[3, 6, 7] is a doubly recursive function which may be defined by

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m>0, n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m>0, n>0 \end{cases}$$

A Multics command language program for Ackermann's function cannot follow the definition directly, since *exec_com* segments cannot be used as functions, and they cannot, therefore, be used as arguments to *exec_com* segments or to active functions. A Multics command language program to compute Ackermann's function, consisting of

four *exec_com* segments, is shown in Figure 2. The segment *Ackermann.ec* plays a coordinating role. It creates a segment, here called *whocares*, in which the *value$set* command may allocate variables; in this case only one variable, A, is used. It then invokes the segment *p1.ec* which starts the actual computation. Upon return from *p1* the value A of Ackermann's function is printed using the *value$dump* command. The segment *p0.ec* is used to circumvent the problem, mentioned above, that all assignments using *value$set* are string assignments: since arguments are passed by value, the value of the argument &1 is assigned to A.

```
&  This is the segment Ackermann.ec;
&  it has two non-negative integer
&  arguments. Example of use:
&     ec Ackermann 2 3
&
&command_line off
value$set_seg whocares
ec p1 &1 &2
value$dump A
&command_line on

&  This is the segment p0.ec; it has
&  one argument and is used to make
&  assignments to the variable A.
&
&command_line off
value$set A &1

&  This is the segment p1.ec; it has
&  two arguments, it distinguishes
&  between the cases of Ackermann's
&  function, and it deals with the
&  simpler cases.
&
&command_line off
&if [nequal  &1 0] &then ec p0 [plus &2 1]
&if [nequal  &1 0] &then &quit
&if [ngreater &2 0]
&then ec p2 &1 [minus &2 1]
&else ec p1 [minus &1 1] 1

&  This is the segment p2.ec; it has
&  two arguments and it deals with the
&  case m>0, n>0.
&
&command_line off
ec p1 &1 &2
ec p1 [minus &1 1] [value A]
```

Figure 2. Multics exec_com segments for Ackermann's function

The segment *p1.ec* separates the three cases and deals with the simpler ones. Thus the first two *&if* statements deal with the first line of the definition of the function, i.e. $A(0,n) = n+1$, after which the *&quit* statement effects a return. Note in passing that two *&if* statements are used because of the absence of a compound statement facility, and that the active function names *nequal* and *ngreater* stand for 'numerically equal' and

'numerically greater than' respectively. The final three lines of the segment distinguish between the remaining cases, dealing with the more complicated one with a call to *p2.ec*, and with the simpler one through a recursive call to *p1.ec*. Finally, since *exec_com* segments cannot be used as functions, the segment *p2.ec* deals with the most complicated case in two steps: It first computes $A(m, n-1)$ by a call to *p1.ec*, and then it uses the computed value in another call to *p1.ec* to obtain the final result.

It remains to be observed that although few people would use the *exec_com* facility for computing Ackermann's function, or solving the Towers of Hanoi problem, the facility is very useful, and it is used frequently, especially for systems-related tasks. Of particular interest in this context are *exec_com* segments with the name *start_up.ec*. These are invoked automatically when a user logs in and they allow the creation of convenient and 'friendly' environments within Multics.

Table I. Comparative timings for Multics control language and comparable PL/1 programs on a Honeywell level 68 DPS

	Multics control language (s)	PL/1 (s)—includes compilation, dynamic linking and execution
Towers of Hanoi		
2 discs	0·272	1·158
3 discs	0·724	1·191
4 discs	1·522	1·256
5 discs	3·206	1·296
Ackermann's function		
A(2, 0)	0·861	1·166
A(2, 1)	2·362	1·169
A(2, 2)	4.501	1·172
A(2, 3)	7·454	1·172
A(2, 4)	—	1·172

Note finally that, as shown in Table I, the command-language solutions to the two problems are competitive in terms of virtual cpu time (though the measurement of this tends to be inaccurate in time-sharing environments) with the more conventional solutions using PL/1 (say), so long as the arguments are small.

REFERENCES

1. Honeywell Information Systems, Inc., *Multics Programmer's Manual—Commands and Active Functions*, 1979.
2. E. I. Organick, *The Multics System: An Examination of its Structure*, The MIT Press, Cambridge, Mass., 1972.
3. A. W. Colijn, 'Experiments with the Kronos control language', *Software—Practice and Experience*, **6**, 133–135 (1976).
4. R. F. Griswold *et al.*, *The Snobol Programming Language*, 2nd edn, Prentice-Hall, Englewood Cliffs, N.J., 1970.
5. J. E. L. Peck, 'Comparison of programming languages', *Technical Report*, University of British Columbia, Computer Science Department, 1972.
6. F. W. Ackermann, 'Zum hilbertschen Aufbau der reelen Zahlen', *Mathematische Annalen*, **93**, 118–133 (1928).
7. H. G. Rice, 'Recursion and iteration', *Comm. ACM*, **8**, 114–115 (1965).

MULTICS SYSTEM-PROGRAMMERS' MANUAL Section BX.1.00

Published: 10/29/68
(Supersedes: BX.1.00, 09/15/66;
BX.1.00, 05/27/66)

Identification

Multics Command Language
W. H. Southworth, G. Schroeder, R. Sobecki, D. Eastwood

Purpose

The Multics Command Language provides the user with a concise means of expressing his wishes to the Multics system. The language and Implementation are based on the work and suggestions of L. Peter Deutsch, E. L. Glaser, R. M. Graham, C. N. Mooers, J. H. Saltzer, and C. Strachey.

Introduction

Most time-sharing systems provide the user with various services which may be invoked from his console by means of "commands" to the operating system. A small number of time-sharing systems go beyond this and allow any user to define (with varying degrees of ease) his own commands, which may be used in exactly the same way as the system supplied commands. Nearly all command programs, whether user defined or system provided, require additional information, which must be supplied by the user, before they can complete their function. Some systems require that every command program directly interrogate the user for the additional information it needs. A more general method is to accept "arguments" in addition to the command name, at the time the command is issued by the user. These arguments are then passed on to the command program by the system. This approach permits great flexibility in the design of command programs. Information may be supplied by arguments, interrogation or a combination of both.

Issuing a command is analogous to executing a function or subroutine call in a language such as PL/1 or FORTRAN. With this view, the name of the command is simply the name of a command program to be either interpreted, if it is a user defined macro (see BX,1.01 for a description of the Macro facility), or executed if it is the name of an entry point in an executable segment which conforms to Multics standards (as defined in BD.7.02). The arguments are either used in the expansion of the macro or passed to the executed procedure in the standard manner. The link between the issuance of the command by a user and the calling of the command program is the command language interpreter. It performs many of the functions of a compiler, principally, parsing the command (which is initially a character string) into its basic elements (e.g., command name and arguments) and formatting the arguments for use by the command program. Finally, the command language interpreter calls the macro expander if the command name is the name of a macro or calls the command program directly.

The Command

A command is a sequence of zero or more elements. The first element is interpreted as the name of the command program and any additional elements are arguments. The normal command program will expect input arguments which are fixed length character strings, and return a value which is a varying character string. If the command program does not expect arguments in this form the command language interpreter, the

Shell, will convert the type of the arguments according to information contained in the symbol table for the command program. The elements of a command are separated by spaces and terminated by a semicolon or a new line character. For example, to change the name of a file from "a" to "b", using the change_name command, a user would type

(1) change_name a b

Elements

The simplest type of element is a string of characters not containing any spaces or other characters reserved by the command language. The semicolon and new line characters are reserved. Other reserved characters will be identified as they are encountered. The three elements in example 1. at the end of the previous section are the simple character strings "change_name", "a", and "b".

Any element (or part of an element) may be a command. The user can tell the interpreter to evaluate an element as a command by surrounding the element with brackets (which are reserved characters). For example in

(2) change_name [oldestfile] b

the second element is a command. When an element is evaluated is a command, the result of that evaluation (i.e., the returned value of the function) replaces the original element. Suppose that the command "oldestfile" returns as its value the name of the oldest file in the users' directory, then example 2 changes the name of the oldest file to "b". In this case, the first argument to the command program change name is the character string returned by the command program "oldestfile" and the second argument is the character string "b".

Note that the spaces before and after the brackets are necessary to indicate that the result of "oldestfile" is an element and not a portion of an element. Suppose that the user had a program "me" which returned as its value his default working directory, (e,g., "me" would return a character string of the form ">user_dir_dlr>Southworth.MAC"). While working in some other directory the user might link to a file in his default directory by typing

(3) link [me]>test.epl

This command consists of two elements, the character string "link" and the result of the command program "me" concatenated with the character string ">test.epl". Similarly a command of the form

(4) link [me]>[filename 17]

would consist of only two elements, where the second element is formed using the values returned by the two command programs "me" and "filename". Note that the command "filename" has one argument, the character string "17".

Sometimes it is necessary to use a reserved character without its special meaning. For example, a command name might contain an imbedded space. The characters quote and left and right accent are reserved for this purpose. Reserved characters within any string of characters surrounded by quotes or accents, will be treated as ordinary characters. For example,

(5) `change name' a b
change" "name a b

```
"change name" a b
change` 'name a b
```

are all acceptable methods for executing a command whose name contains an imbedded space. Also, since quotes and accents are reserved characters it may be desirable to suppress the special meaning of one or the other. This may be done by surrounding quotes wIth accents, and accents with quotes. For example, the operator could issue the command

```
broadcast  "Don't do anything!"
```

Active, Neutral, and Empty Commands

There are three types of commands which may appear in elements: active, neutral and empty commands. In order to understand how these three types differ it is necessary to have a basic knowledge of the scanning and interpretation algorithm of the shell. A command line is scanned from left to right. The shell maintains a pointer which indicates its current position in the lIne. Whenever a command has been completely scanned it is evaluated (i.e., the command program is executed). For example in

```
(6)       change_name a b ;
                          ^
```

when the pointer has reached the indicated location the shell will recognize that the end of a command has been reached and call the command program "change_name". In the case of an element which is a command, nothing to the right of the element will be scanned until after the command element has been executed and its value has been inserted back into the command line. For example, in

```
(7)       change_name [oldestfile] b;
                                   ^
```

when the pointer has reached this point the command program "oldestfile" will be called. If the returned value is "xyz", the transformed command line is

```
(8)       change_name xyz b;
                      ^
```

Note that the pointer is set to the beginning of the inserted value. This is important because the returned value will now be scanned in the same manner as the original command element. If "oldestfile" had returned the string "[myoldest]", then the scanning pointer would have encountered this string as a command because of the brackets and executed the command "myoldest".

The type of command in the previous examples, which returns a value which is rescanned we will define as an "active command". The command language recognizes two other kinds, a "neutral command" and an "empty command". A command preceded by "||[" rather than ||[' is said to be "neutral". Its value is not rescanned. This is particularly useful in defining certain macros. If our previous example with "oldestfile" had read

```
(9)       change_name ||[oldestfile] b;
```

and "oldestf1le" had returned the string "[myoldest]", then this value would have been inserted into the command line and the scan pointer set to the next character after the inserted character string, i.e.,

(10) change_name [myoldest] b
                              ^

In this case the inserted string would not be recognized as a command and "change name" would be called with "[myoldest]" as its first argument. An "empty" command is preceded by "|||[". After an empty command is executed its value is thrown away.

The vertical bar is a reserved character only in the context of "|[" or "|||[". An easy way to remember the three types of commands is to think of a command as performing the three actions: evaluation, insertion, and reevaluation. A single vertical bar suppresses reevaluation leaving evaluation and insertion, while a double vertical bar suppresses reevaluation and insertion leaving only the first evaluation.

Iteration

Sometimes the user wishes to repeat a command with one or more elements changed. The iteration facility of the command language is provided for economy of typing in this case. The "iteration set" is a set of zero or more elements enclosed by parentheses (parentheses are reserved characters). If it contains no elements it is ignored. Otherwise, each element of the set will, in turn, replace the entire set in the command line. For example

(11) print (a b c).epl

is equivalent to the three commands

```
print a.epl; print b.epl; print c.epl
```

More than one iteration set may appear in a command. All possible combinations will be executed. For instance, the compound command

```
(print delete)  xyz(.epl .eplbsa);
```

would expand into the commands:

```
print  xyz.epl
print  xyz.eplbsa
delete xyz.epl
delete xyz.eplbsa
```

Nested iteration sets behave in exactly the same manner as unnested sets. Evaluation of parentheses occurs from the outside in. The principal use of nested iteration sets is to reduce typing when subsets of an element are repeated. For example

(12) make_directory >user_dir_dir>((Southworth Martin).MAC Stone.GE)

would make three directories

```
>user_dir dir>Southworth.MAC
>user dir dir>Martin.MAC
>user_dir_dir>Stone.GE
```

Summary of Command Language Syntax

The Multics Command Language contains the following syntactical elements.

command line

the character string representation of a command or sequence of commands.

command
: a sequence of zero or more elements separated by spaces (in the command line). The first element is taken as the command name and additional elements as arguments.

iteration set
: a sequence of zero or more elements, enclosed by parentheses, which are Inserted in turn in the command for evaluation.

command program
: either a defined macro to be recognized and expanded by a macro expander program, or executable machine instructions, whose name represents an entry point in a segment which conforms to Multics standards (as defined in BD.7.02).

element
: the basic component of a command; it may represent a command name, or argument.

active command
: a sequence of zero or more elements surrounded by brackets (it is not necessary to enclose the character string typed in at the user's console with brackets, in this case brackets are assumed). The character string within the brackets is treated as a command - it is evaluated, its value is inserted into the command line and its value is rescanned as part of the command line.

neutral command
: a sequence of zero or more elements surrounded by "|[" on the left and "]" on the right which is evaluated, as a command line, but is not rescanned.

empty command
: a sequence of zero or more elements surrounded by "||[" on the left and "]" on the right which is evaluated. Its value is thrown away.

literal string
: a character string surrounded by quotes or balanced left and right accents. Its value should be taken literally, i.e., reserved characters within the string should not be recognized for their special meaning.

semicolon
: denotes the end of a command and the beginning of another command of the same type.

new line
: new line characters within the command string passed to the command language interpreter are ignored if encountered in the scan of the command line. This should not be confused with the fact that the new line character may serve as a delimiter for whatever program called the interpreter.

Implementation

The command language interpreter, the Shell, is normally driven by the Listener. The shell provides the necessary parsing to process a character string as a command. The Listener can be conceptually described as

```
[[read_line]]; listener
```

Its function is to listen for requests in the form of command lines typed in at the user console. In the above command language description, the listener reads in a line from the console, evaluates the line as a command, and re-calls itself to repeat the function. In actuality this is usually accomplished by a Multics procedure which calls the shell which accepts as its single argument a character string (fixed length or varying) to be evaluated as a command.

Formal Description of the Multics Command Language

The following Backus-Naur description formally defines the syntactic components of the Command Language. For simplicity we have not provided definitions for the ASCII characters, based on the assumption that the alphabet is not open to design changes. If a construct is enclosed in parentheses it is to be interpreted as zero or more occurrences of that construct.

```
<command sequence> ::= <command> (<semicolon> <command>)

<command> ::= <element list>

<element list> ::= <element> (<spaces> <element list>)

<element> ::= <element component> (<element>)

<element component> ::= <function> | <iteration set>
                        <literal string> | <unreserved character>

<function> ::= <active function> | <neutral function>
               <empty function>

<active function>  ::= <left bracket> <command> <right bracket>

<neutral function> ::= <vertical bar> <left bracket>
                       <command> <right bracket>

<empty function> ::= <vertical bar> <vertical bar> <left
                     bracket> <command> <right bracket>

<iteration set> ::= <left paren> <element list> <right paren>

<literal string> ::= <quoted string> | <balanced quoted string>

<quoted string> ::= <quote> <balanced quoted string> <quote> |
                    <quote> <unquoted character string> <quote>

<balanced quoted string> ::= <left accent> <balanced quoted
                             substring> <right accent>

<balanced quoted substring> ::= <character string not containing ' or `>
                                | <balanced quoted string>
```

---

thvv@best.com


Multics home page
Features
History
Sites
Glossary
Stories
Biblio-graphy
Multics general