AFIPS CONFERENCE PROCEEDINGS

VOLUME 26
PART II

1964

FALL JOINT COMPUTER CONFERENCE

VERY HIGH SPEED COMPUTER SYSTEMS

Sponsored by:

Association for Computing Machinery
Institute of Electrical and Electronics Engineers
Simulation Councils, Inc.
American Documentation Institute
Association for Machine Translation and Computational Linguistics

AFIPS CONFERENCE PROCEEDINGS

VOLUME 26
PART II

1964

FALL JOINT COMPUTER CONFERENCE

VERY HIGH SPEED COMPUTER SYSTEMS

Sponsored by:

Association for Computing Machinery
Institute of Electrical and Electronics Engineers
Simulation Councils, Inc.
American Documentation Institute
Association for Machine Translation and Computational Linguistics

1965
SPARTAN BOOKS, INC.
WASHINGTON, D.C.
MACMILLAN AND CO., LTD.
London

Library of Congress Catalog Card Number 55-44701
Spartan Books, Inc.
1106 Connecticut Avenue, N.W.
Washington, D. C.

Sole distributors in Great Britain, the British Commonwealth and the Continent of Europe:
Macmillan and Co., Ltd.
St. Martins Street
London W.C.2

CONTENTS

PREFACE

Volume 26 of the AFIPS Conference Proceedings is a permanent record of the papers presented at the 1964 Fall Joint Computer Conference. Part I, called AFIPS I, was distributed at the Conference; this book, which is Part II, hence AFIPS II, includes information which, because of its nature and timeliness, could not be made available for publication prior to the time of the Conference.

This departure from the usual pattern of a single volume for the Conference Proceedings reflects the unique character of the sessions whose papers comprise the major portion of this book. In recent Conferences, computer systems have been described in bits and pieces, and properly so. The technical programs were carefully organized to avoid inclusion of material which might have had other than technical interest or motivations. Furthermore, since technical sessions are generally organized around particular areas of specialization, only those specific features of new computer systems which were within the specialty area and which were intrinsically of interest could be presented. This approach, it is believed, is proper and should be maintained, but it has the effect of eliminating overall descriptions of computer systems, as often presented at earlier Joint Computer Conferences.

Consequently, as an experiment, the 1964 FJCC Program Committee under David R. Brown, Program Chairman, decided to sponsor, on an experimental basis, some sessions devoted to computer systems, permitting discussion of an entire system, regardless of whether or not each feature under discussion was unique or new. Attempts were made to present a complete description of each system—the underlying concepts, the hardware, the software, and the problems encountered in arriving at a final design.

The sessions were called Very High Speed Computers, 1964—The Manufacturers' Point of View. These sessions were organized and chaired by Dr. Sidney Fernbach. The major stipulation placed on each system to be described was that it should be new and that it must be capable of executing approximately one million instructions per second. While the manufacturers' point of view is represented, it is perhaps more relevant to note that this view is shaped predominantly by the designers of each system.

Publishing a post-Conference volume presented the unusual opportunity of including a record of some of the events which transpired there. Hence, the Keynote Address by David Sarnoff and the Luncheon Address by Gerard Piel are also in this book. The opinions advanced by these distinguished individuals should form the basis for further discussion and consideration and are representative of the broadening influence and impact of the information processing field.

Additionally, these pages contain a full account of the background of the Harry Goode Memorial Award and of its presentation to its first recipient, Howard H. Aiken. It is especially fitting that, at this stage in its growth, the information processing field should establish the means for honoring those who make notable contributions to it. In that vein, also, a notation of the awarding of the 1964 FJCC Prize Paper Award is included in this book.

In keeping with the recently completed effort to index the papers presented in earlier Joint Computer Conferences, an index of terms applicable to the papers in this book is included at the end.

As previously noted, AFIPS I is already in publication and has been widely distributed. It contains, as do the other volumes in the series, all of the papers presented at the usual technical sessions. These papers were carefully selected from the excellent material available and cover a wide variety of topics. The Session Chairmen, together with the Program Committee, performed an outstanding function in coordinating the choice and presentation of papers ranging through hardware, software and applications.

In retrospect, the 1964 FJCC was, in many respects, a stimulating experience. Its major permanent contribution, however, is represented by the information contained in the two parts of the Proceedings, and the contributors to these Proceedings are to be commended for their efforts and for the contributions they have made to the rapid advance and diversification of the information processing sciences.

[illegible]

General Chairman
1964 Fall Joint Computer Conference

KEYNOTE SPEAKER

Brig. Gen. David Sarnoff was the Keynote Speaker for the 26th AFIPS Joint Computer Conference. General Sarnoff is Chairman of the Board of the Radio Corporation of America. His life-long association with science, technology, and industry has resulted in a unique and unparalleled record of accomplishment, and the chronicle of his career is itself representative of many of the significant effects of science on society.

The Keynote Address

THE PROMISE AND CHALLENGE OF THE COMPUTER

David Sarnoff
Chairman of the Board
Radio Corporation of America
October 27, 1964

California has always been synonymous in my mind with innovation and progress, and the past year has given me fresh reason to think so.

Seven months ago, at the NBC studios in Burbank, I had the pleasure of presiding at the inauguration of a new dimension in business communications. By means of two-way, closed-circuit color television, two large assemblies of RCA's shareholders—one in California and the other in New York—were brought into instant visual and verbal communication at our 45th Annual Meeting. Despite their continent-wide separation, the two groups were as effectively unified as though they were meeting under one roof.

Three months later, the sense of participation was almost as direct and immediate with the successful performance of Ranger VII. Again, this was an accomplishment that spanned the continent. RCA cameras and transmitters built in New Jersey, aboard a spacecraft built in California, completed their historic mission at a third point—the moon.

Today, I am glad to be in California and to experience again this sense of participation in progress. It is expressed this time in a gathering of more than 4,000 scientists and engineers from a new and dynamic industry that traces its line of descent directly to the beginning of the art and science of electronics. The modern electronic computer is the precocious offspring of wireless telegraphy and radio telephony, and it is creating a new dimension of progress through the high-speed handling of electronic signals.

Since the birth of RCA in 1919, our principal efforts have been concentrated on transmitting, receiving, and recording information by electronic means. It was natural, therefore, that our scientists and engineers were among the first to begin the study of electronic computing techniques, and I believe their contributions to this new art have been significant.

Six years ago, when we entered the computer field commercially, it was the logical extension of everything we had been doing up to that point. We are making good progress—in technology, programming, service, sales, and revenue—and we will soon announce some significant new product developments which we believe will contribute to the industry's future growth. Last month, we crossed over into the promised land of computer profits. While the trip was rugged, we found the new terrain to our liking, and we expect to stake out a permanent profit claim.

THE LESSON OF STANDARDS IN ELECTRONICS

During my 58 years in electronics, I have seen several dynamic industries emerge from conceptual beginnings in the laboratory. The most memorable were radio communications, radio broadcasting, sound-movies, black-and-white television, and color television. While their origins differed in detail, all shared a common experience that has a distinct parallel today in the rise of computers as another major member of the electronics family.

All were intensely competitive from the beginning and have remained so. But they began fulfilling their potential only after agreement had been achieved for technical standards prescribing the kind and quality of service to the public. A pattern for progress was thus fashioned without sacrificing the vital stimulus of competition in developing newer, better, and more economical equipment, and in furnishing more efficient service to the user.

I am convinced that this same process must occur in the computer industry. Even now, the computer is stirring a revolution of the brain just as steam power stirred a revolution of the muscle. The potential effects are almost incalculable—but their full realization calls for the same definition of ground rules that permitted the growth of the older electronics industries.

When sight was added to sound with black-and-white television, the need for technical standards as the basis for orderly growth was clearly recognized. The receiver in the home and the transmitter in the studio had to be built to operate on the same standards. A committee representing all major elements of the industry obtained practical unanimity on such standards as a precondition to the establishment of a public television service. It was on this foundation that black-and-white television grew so phenomenally in the post-war era.

Again, in the early 1950s, the industry underwent the long and difficult process of reaching agreement on a workable service to the public. This time the issue was color television, and two sharply different systems and standards were in dispute. One was based on a mechanical "color wheel" which could produce color images but whose transmissions could not be received by the black-and-white receivers in the nation's homes. It was therefore incompatible with existing equipment. The other was an all-electronic compatible system which could be seen in black-and-white on any TV set in the home.

It was evident that, if the incompatible mechanical standards were to be adopted, the industry would be saddled with an inferior system and the public with an inferior, more costly product. To adapt the 10 million sets then in existence in order to receive a degraded picture in black-and-white would have cost the public approximately $500 million. Without an adaptor, the TV screen would simply go blank.

Clearly, there were many inherent advantages in adopting an electronic rather than a mechanical system of color television. For the industry, the basic issue came to this: should the millions of dollars already invested by television set owners be jeopardized by an incompatible color television system?

Once more, an industry group was formed to draft signal specifications and standards. The result, after 32 months of work, was a complete set of compatible color signal specifications closely following those that had been developed through long years of laboratory research and engineering. These ultimately became the basis for color television in the United States—a business that now stands with data processing in the forefront of the nation's industrial growth.

The industry committee did its work so thoroughly that every subsequent advance in the color television art has been put into service with no change whatever in the original standards.

THE NEED FOR STANDARDS IN COMPUTERS

The phenomenal rise of data processing bears certain resemblances to that of color television. It is confronted in similar fashion by a question of compatibility. The investment of the user is again a primary consideration. The issue becomes more acute as the growing computer industry intensifies its competitive drive for new and more ingenious ways to accommodate the user.

From the two-score or so machines in existence barely a dozen years ago, there are now some 17,000 general-purpose computers in the United States alone, and the number is increasing at a rate of more than 500 a month. Within the coming decade, the computer population can increase enormously.

Whether it realizes its full growth potential depends in very large degree, however, on the measures we undertake now to establish the basis for orderly development. The interests of the industry and the needs of the user demand a far greater measure of compatibility and standardization among the competing makes of computers and the means by which they receive and transmit information.

Neither the operators nor the machines we have built for the processing and transmission of information can yet speak to each other in a commonly understood and accepted language. The means of preparing data, of forwarding and entering data in the machine, and of instructing the machine in its use differ sufficiently from one make of equipment to the next so that none can readily accept the product of another.

We function today in a technological "Tower of Babel." There are, by conservative count, more than 1,000 programming languages. And there are languages within languages—in one instance, 26 dialects, and in another, 35 dialects. There are eight computer word-lengths in use. There are hundreds of character codes in being, at a ratio of one code for every two machines marketed. Four magnetic tape sizes are employed with at least 50 different tape tracks and codes.

Standards have not been accepted even for commonly used symbols, instruction vocabulary, or program development procedures. Words which have currency throughout the industry assume different meanings, depending on whether a man has trained in Pasadena, Poughkeepsie, or Camden. We have yet to produce a universally accepted computer glossary.

No means have yet been perfected for a program in one basic language to be run efficiently into computers of different makes. The result has been needless duplication, delay, and waste—both to the manufacturer and to the user—in cost, in equipment, in operating efficiency, and in manpower and skills.

Incompatibility has compelled the manufacturer to build optional choices into peripheral equipment for the input and output of data. It has required him to maintain various types of the same equipment, or to build to a customer's specifications on each order. It has diverted needed engineering and programming talent from the vital work of new product and systems development.

The burden of incompatibility has been even more onerous to the user. It has meant the extra cost of providing hardware and programs to handle the differences between incompatible systems, the cost of extra machine time to process data set for another computer, the cost of training people to do things differently, the cost of not being able to do the job immediately.

Last year, an estimated $2 billion was spent by American business and government for privately developed computer programs, representing thousands of man-years of effort. Yet, when a change to new equipment is made, portions of this effort must be thrown away because they have no validity to another make of machine, or they are retrievable only at further cost.

I have heard it said that even a degree of standardization and compatibility might inhibit the progress of the art. In my judgment, this argument is without substance. The nature of a computer is such that its operation is governed far less by its internal construction than by the program that is given to it.

The effort to bring order to the flow of computer intelligence need not affect competition either in creating programs or in seeking new generations of increasingly efficient machines. On the contrary, the result could be a greater concentration of effort toward this primary goal.

PRELIMINARY STEPS TOWARD INDUSTRY STANDARDS

During the past four years, certain essential preliminary steps have been taken toward industry standards and compatibility, largely under the aegis of the American Standards Association and the Business Equipment Manufacturers Association. Representatives of the industry, of users, and of technical groups have proposed industry-wide standards in such areas as data transmission, information exchange, and character recognition.

Working with a committee of the International Standards Organization representing the computer interests of 13 foreign countries, they have recommended world-wide standards which would make it possible for a credit card or invoice produced in any country to be read by equipment anywhere in the world. Another recommendation, for information interchange, would make it possible for computers in all countries to talk to each other in a common language, when it is adopted and implemented by the manufacturers.

FURTHER ACTION ON STANDARDS IS ESSENTIAL

That phrase—when adopted and implemented by manufacturers—is central to the resolution of the problem. For in our country, at least, the action is voluntary, and until these and other standards are put into general use they remain little more than statements of hope.

Today, Western Europe is energetically seeking to close the computer gap and is moving toward the establishment of standards. During the next five years, the use of computers in European industry and government is expected to develop at an accelerating rate. Ten years from now, the foreign market might well equal that of the United States.

Unless we achieve some coherence in our own ranks, we may find ourselves following instead of initiating standards.

All of us, in computer manufacturing, in communications, and among the user groups—at the technical as well as the managerial level—share a common interest in the free interchange of information, and the media and equipment through which it flows.

This demands that we give compatibility the urgent consideration which it merits but which it has not yet received. It requires the wholehearted support by all of us of the standardization work that is now going forward, and implementation of the results with all deliberate speed. It will require that we submerge our differences, through fair and equitable compromise, to achieve greater ends.

I do not suggest that existing systems be discarded. That would be unrealistic as well as costly. Even today's computer has reached maturity in one basic respect: its average time between failures, measured in minutes only a decade ago, is now measured in months. This is a level of operating reliability far beyond that of either the automobile or the airplane.

But new generations of systems are coming, and the time to bring order into progress is now, before they have fully arrived. Standards can be established which, if planned with thought and foresight, can guide us in the future, linking our separate efforts and facilitating the common evolution of our industry. Such standards are indispensable to continued progress.

THE COMPUTER'S IMPACT ON THE FUTURE

As the shape of tomorrow's technology takes form, the volume and accessibility of data stored in the computer will play a decisive role. All information as to what to do, how to do it, and what data to do it with, resides in the memory of the machine. With larger and faster memories there are few limits to the tasks that can be solved or the speed with which they are completed.

The time is soon coming when these memories will be capable of storing up to 100 million bits of information, retrievable in fractional millionths of a second. For external memories, the goal is a trillion bits, possibly advancing later to capacities that are many times greater. By these means we can hope to store all of the information that is presently contained in all the world's libraries.

Tomorrow's standard computers and their peripheral equipment will instantly recognize a handwritten note, a design or drawing which they will store and instantly retrieve in original form.

The computer of the future will respond to commands from human voices in different languages and with different vocal inflections.

Its vocabulary will extend to thousands of basic words in the language of its country of residence, and machines will automatically translate the speech of one country into the spoken words of another.

The computer itself will become the hub of a vast network of remote data stations and information banks feeding into the machine at transmission rates of a billion or more bits of information a second.

Laser channels will vastly increase both data capacity and the speeds with which it is transmitted.

Eventually, a global communications network handling voice, data, and facsimile will instantly link man to machine—or machine to machine—by land, air, underwater, and space circuits.

We will see computer switchboards in space, similar to those presently in operation on the ground, routing in milliseconds any communication to and from virtually any point in the world.

The interlocking world of information toward which our technology leads us is now coming closer to realization. It will be possible eventually for any individual sitting in his office, laboratory, or home to query a computer on any available subject and within seconds to receive an answer—by voice response, in hard copy or photographic reproduction, or on a large display screen.

We will see the emergence of national and global information processing utilities, serving tens of thousands of subscribers on a time-sharing basis. These utilities will accommodate the specialized needs of researchers and engineers, lawyers, medical men, sociologists, or the general needs of the public.

The ordinary citizen may well carry an individual credit card for use anywhere to charge his bank account electronically over a worldwide data communications network that would link up with the telephone systems of all nations.

Such an arrangement could employ simple input units located in all retail establishments—service stations, restaurants, hotels, and other public facilities. These would be in direct and instantaneous communication with a system of banking computers to permit the transfer of funds without the many duplicate bookkeeping and mailing steps that characterize the present credit card system.

A scientist will be able to discuss a problem by two-way television with a colleague anywhere on the globe, and both of them will be able to query a computer at another terminal point for assistance in finding the solution.

Private corporations, many of which will be international in ownership and operation, will have instant access to production and market information from data stations positioned around the globe.

Similar systems will operate on a vastly larger scale for government agencies—military, diplomatic, and economic.

The computer will evaluate and offer alternate courses of action, taking into account all the known and probable variables of a given situation.

This emerging pattern inevitably will set in motion forces of change within the social order, extending far beyond the present or presently predictable applications of the computer. It will affect man's ways of thinking, his means of education, his relationships to his physical and social environment, and it will alter his ways of living.

I believe, for example, that television in a vastly expanded form will become our major instrument for communicating general or specialized information. The same broadband channels that accommodate the TV picture signal can also transmit masses of computer data at ultrahigh speed for instant display.

One day, we will receive our newspapers and technical publications, photocomposed by a computer, by direct display on a wall screen in the home or office. If we wish to retain any part of them for further reading or reference, it can instantly be produced in electrophotographic copy.

As computers become amenable to simple commands, they will become as indispensable to education as the reference library. Indeed, they will become tomorrow's reference library, used by every student from the upper elementary levels through university.

Far from eliminating the need for intense intellectual effort, they will permit young people to undertake mental explorations far beyond the boundaries of the present classroom world.

The computer already is opening areas of knowledge long denied us by the sheer magnitude of the mathematics involved. The implications are no less fundamental for the social and life sciences than for the physical disciplines.

By correlating vast quantities of data and uncovering new relationships we can for the first time obtain significant information on social and human behavior—from the destructive tendencies of some to the learning power of all.

THE ULTIMATE CHALLENGE POSED BY THE COMPUTER

The ultimate implication of the computer is that it provides a means of releasing the productive powers of the human brain to an almost limitless degree. Yet the computer imposes as a precondition the sternest discipline to which the mind has yet been subjected.

Even to use the machine, we must apply clear and precise logic to situations which heretofore were assumed to be beyond analysis. We must state precisely what we know or do not know, and what we wish to know.

If we are to develop the computer to its full potential as a reference storehouse of human knowledge, we face the immense intellectual challenge of researching every major field of human activity, of assembling, analyzing, and identifying its documents, and reducing the information to acceptable machine form.

Before the end of the century, I believe that these codification efforts will coalesce into what unquestionably will become the greatest adventure of the human mind. We shall achieve a far more comprehensive understanding than we have today of man and his environment. We shall do so through the orderly compilation of accumulated knowledge and wisdom, beginning with the days of clay tablets and papyrus scrolls. The human horizon will then encompass all that man has ever known, and all that his science will enable him to know.

But how swiftly we scale these heights depends upon the steps we take today to bring order and compatibility to our art. It is an urgent task to which all of us who bear the responsibility for leading this industry into the future must turn our efforts.

It was Socrates who said: "Let him who would move the world move himself." His words have particular pertinence at this time and in this place. For we of the computer industry must surmount today's challenge before we can advance to tomorrow's promise.

Let me conclude on a personal note. Whether your individual role is large or small, the significance and scope of this new science and industry are such that in a genuine sense you are making history. The impact of your knowledge and talents will echo down the corridors of time. The quality and content of life on this planet will be profoundly affected—indeed are already being affected—by your labors.

I am grateful for the opportunity to have shared a few thoughts with you.

LUNCHEON SPEAKER

The Conference Luncheon Speaker was Gerard Piel, Publisher of **Scientific American.** As publisher of a magazine with broad appeal to the scientific community, Mr. Piel has contributed to the difficult task of describing and interpreting the sometimes bewildering developments in the fields of science and engineering. He has chosen for the title of his subject, "The Computer as Sorcerer's Apprentice." Mr. Piel is a graduate of Harvard University (magna cum laude) and holds honorary doctorates from Lawrence College, Colby College, Rutgers University, Columbia University, Tuskegee Institute, and the University of Bridgeport. Prior to building the new **Scientific American,** he was for six years Science Editor of **Life** magazine.

The Luncheon Speech

THE COMPUTER AS SORCERER'S APPRENTICE

Gerard Piel
Publisher, Scientific American
October 28, 1964

The computer is the engine of this latest phase in the acceleration of the industrial revolution. The role of the computer cannot be measured in the simple terms of the number of computers at work in the American economy or even in the extraordinary variety of functions in which the computer has found work to do—from accounting routines to industrial process control to creative enterprise in mathematics itself. More significantly, computer technology gathers in and brings to intense focus the most diverse discoveries on the frontiers of knowledge—from investigation into the nature of matter to speculations at the foundation of knowledge. It is the agency through which the advance of human understanding now finds its way to the control of natural forces in time intervals that grow shorter year by year and month by month.

Because the time lag between invention and application now diminishes so swiftly, it becomes possible—and necessary—to forecast the ethical, social, and economic implications of this development. Today in our country and in certain other industrial nations, men are compelled to recognize and give assent to a profound transformation in human values. Technological change has already largely eliminated people from production; it has sundered the hitherto socially essential connection of work to consumption. The citizens and the institutions of these nations must accommodate themselves to the law of material abundance: each individual can secure increase in his own well-being only through action that secures increase in the well-being of others.

This novel dispensation stands in contrast to the law of scarcity which, in the words of Alexander Herzen (1812-1870), declares: "Slavery is the first step toward civilization. In order to develop, it is necessary that things should be much better for some and much worse for others; then those who are better off can develop at the expense of the others."

The iron law of scarcity underlies the ethical dilemma of political economy which has sought for nearly three centuries to discover or to rationalize equity in social institutions long ago designed to secure the inequitable distribution of goods in scarce supply. Adam Smith, the first great systematizer of economic theory, was foremost a moral philosopher. In his *Theory of Moral Sentiments*, published in 1759 and the work which brought him his principal contemporary fame, he traced the roots of moral action to the "passion of sympathy"—"which leads us into the situations of other men and to partake with them in the passions which those situations have a tendency to excite." It was later, in the *Wealth of Nations* published in 1776, that he undertook to explore "those political regulations which are founded, not upon the principles of justice, but that of expediency, and which are calculated to increase the riches, the power and the property of the state." Against the princely mercantilism of the autocratic continental powers, Smith asserted the labor theory of value: "Labour is the real measure of the exchangeable value of all commodities. . . . Equal quantities of labour at all times and places are of equal value to the labourer. . . . Labour alone, therefore, never varying in its own value, is alone the ultimate and real standard by which the value of all commodities can at all times and places be estimated and compared." In the free play of supply and demand in the open market, the products of human labor found the just and equitable price at which they were to be exchanged. In the market, labor, itself a commodity in consequence of the division of labor, also found its fair price. Under the sure guidance of the "invisible hand" each man could seek his private interest, confident in the knowledge that he thereby secured the public weal.

For the generations that launched the industrial revolution in 18th century England, Adam Smith and his successors in political economy furnished not only the guidelines to practical action but the moral assurance necessary to the taking of action. Before the middle of the 19th century, however, it had become impossible to conceal—in the blight laid upon green England by the carboniferous phase of industrialization—the failure of their enterprise. Benjamin Jowett, Master of Balliol and translator of Plato, spoke for the alienation of the humanities from the sciences when he said: "I have always felt a certain horror of political economists since I heard one of them say that the famine in Ireland would not kill more than a million people, and that would scarcely be enough to do much good."

Even as Jowett wrote, the first phase of the industrial revolution had made such computations obsolete as well as patently immoral. In 1864, the year of the Emancipation Proclamation, mechanical horsepower generated by steam engines in the U.S. economy exceeded for the first time the output of biological horsepower by horses and men. As early as 1900, only 75% of the U.S. labor force was employed as "producers of goods"; more than half of these producers were engaged in farming and the next largest percentage in unskilled labor functions. By 1960, human muscle had been all but eliminated from the production process. The census for that year shows that less than half (46%) of the labor force was now employed as producers of goods; farmers (7%) and unskilled laborers (5%) were approaching statistically negligible percentages of the labor force. More than half of the producers were classified as "operatives," that is, human nervous systems still interposed in process control feedback loops not yet completely closed by electronics.

In the present phase of acceleration, as is well known, the industrial revolution is eliminating nervous systems from the production process. Robots—artificial sensory organs and mechanical controllers linked by feedback circuits—have been taking over from human workers in all of the fluid process industries. In at least 85 plants in the U.S., computers at the center of control networks have transformed the process streams into truly self-regulating systems. The computer and the feedback control loops have now begun a corresponding transformation of the discontinuous processes of the metalworking industry. The same revolution in technology—for example, transcontinental pipeline transportation of fluid commodities under computer and feedback-loop controls—is under way or impends in all of the production sectors of the economy.

During the past decade, blue collar employment in American manufacturing has actually declined, while the output of these industries has nearly doubled. The rate of increase in productivity in the production sectors of the economy, which has averaged 5.6% over the decade and has been accelerating, is grossly understated by productivity figures applied to the entire labor force. These, the figures given widest circulation, have shown an annual improvement of only about 2.5 to 3%.

Until recently, increase in employment in trade and distribution and in the services has compensated for disemployment from production. The computer, however, finds application even more readily in the functions that employ human beings in these sectors. The "white collar" computer, equipped with a typewriter on its input and output side, is conceptually a much simpler organism than the computer equipped with sensory organs and muscles that displaces the blue collar worker. A conservatively estimated million-fold increase in the data-processing capacity of organizations equipped with computers as compared to organizations manned by human beings and assigned to comparable tasks has already been demonstrated in military command and control systems. Although computer technology has just begun to find its way into trade and distribution and the services, increase in employment in these sectors has already begun to slacken. In the private sector of the economy it now barely offsets disemployment from the production industries. During the five-year period from 1957 through 1962, the private enterprise economy generated less than 300,000 additional new jobs.

The creation of new jobs in the economy as a whole has now lagged the growth of the labor force for more than a decade. This is a polite way of saying that the economy is afflicted with a constant and insidious increase in unemployment. Ever since 1952, the rate of unemployment has been larger at the peak of each ripple or boomlet in the economic cycle, and each recession has left a larger percentage of the labor force high on the beach.

Debate continues as to whether the country's rising unemployment is "cyclical" or "structural." Classical economists—and nowadays Keynesian economists are "classical"—assure us that the unemployment is cyclical. They point to the history of the past 60 years in stubborn support of the thesis that the labor-saving effect of technological progress merely frees labor from one task for employment in another. It is conceded that frictions make for unemployment in this turnover of the labor force, especially when progress goes forward rapidly. But sooner or later new jobs, generated by ever greater economic activity and an ever-expanding Gross National Product, soak up the unemployed. By tried and tested and now generally sanctioned counter-cyclical measures—for example, by the recent Federal tax cut—the fluctuations of the system can be damped and the peaks and valleys of unemployment smoothed out. When the Kennedy Administration took office, its official economists were arguing that unemployment at the rate of 4% could be regarded as normal. Despite the tax cut and the prolongation of the present boom, unemployment now ranges above 5%.

Increase in unemployment accompanying expansion of economic activity would seem to indicate that a rising percentage of the unemployment is indeed structural—that people, in other words, are being displaced from the economic system in ever larger numbers by mechanization, more specifically by the computer and its accessory and allied technologies. Consider, for example, the computer industry itself, thus broadly defined. If employment were to expand in any industry during this period of intensive mechanization, one would think first of the payroll of the industry that is doing the mechanizing. What is more, the technology of electronics that furnishes your hardware has been notably, if paradoxically, highly labor-intensive. Until a few years ago, labor would represent up to 60% of the production cost of a piece of electronic hardware. Engineering would constitute the major investment; materials would be a minor cost and capital equipment a negligible item on the balance sheet. In these respects electronics was like the garment industry: a business anybody could get into, providing he had a bright idea and could finance his payroll long enough to get his product on the market. Within the last 10 years, as I need not tell you, electronics has gone solid state. The transistor and the micromodule are even now yielding to the integrated circuit. With this development, acre after acre of work benches at which housewives and high school girls wield pliers and soldering irons has been disappearing. Labor cost is vanishing in the economics of electronics. Material costs have now become significant; engineering and plant costs, transcendent.

In other words, the prevailing relationships among the factors of production in electronics are being turned 180 degrees around. With people being exiled from the computer industry as rapidly as the computer itself is promoting the disengagement of people from jobs in other sectors of the economy, the expansion of this industry will not generate anything approaching a corresponding buildup of its payroll.

It cannot be said, any longer, that the industrial revolution is the same old story. The acceleration of technological change, driven by the accelerating advance of human understanding, reaches to the very heart of the institutions of our society; that is, to the value system upon which those institutions rest.

The unemployment figures present a profoundly misleading measure of the degree to which our capacity to produce material abundance has outrun the capacity of our institutions to secure the distribution of that abundance. It must be reckoned, in the first place, that some eight million persons are employed in the war economy and contribute nothing whatever to the flow of material abundance from the cornucopia of our non-military productive system. If the production workers in the war economy are subtracted from the productive work force, then the percentage employed as producers of goods falls below 40%. But this figure still overstates the truth because most of the goods circulating in commerce and consumed by American citizens are produced by the very much smaller percentage of the labor force that is employed by our most efficient production organizations.

Consider, for example, our farms. Some 85% of the food that moves from the farms to the markets comes from less than one million farms; and the same is true of industry. The few large and efficient corporations in each industry, with their relatively smaller payrolls, produce the overwhelming percentage of our industrial output. If a small minority of our working force is today doing most of the production, then, in the future, we can expect to see an even smaller minority of our working force account for all of the production of goods in our economy. The sorcerer's apprentice has thrown the switch. The great test of our democracy is to find ways to distribute or dispose of the mounting flood of abundance.

Actually, by the kind of improvisations that are so characteristic of democracy, we have had some success in coping with this task starting from the turn of the century. In 1900, 40% of the adults of our country were not employed; that is to say, they were either unemployed or they were not in the labor force. In those good old days, 57% of the adults of the country were employed in the private sector of the economy. Our country still approximated the description it gave of itself in the Declaration of Independence, as a people engaged in the pursuit of happiness—in the pursuit of private interest, either their own or that of their employers. Only 3% of the American people were on the public payroll. In 1960, the same 40% of our population was not employed, either unemployed or not in the labor force. But only 40% of the population was now employed in the private sector of the economy. A full 20% of the American people found their employment either directly on the public payroll or indirectly through the increasingly huge expenditures of governmental agencies for the product and services of private corpoartions—not only in the war economy but in the construction of highways and other major public works ventures. In other words, one-third of the working force now owes its employment to public expenditure.

Direct employment in the public sector has been increasing at five times the rate of increase in the private sector. During the past five years the public sector generated more than a million of the less-than-1.5 million new jobs in the economy. Since the Federal payroll remained constant during this period, this gain must be credited to state and local governments. It can be declared with pride, furthermore, that the biggest part of the gain was in the payroll of our public education system. This, in turn, may be taken as an indication of the responsiveness of our value system to the evolutionary pressure of abundance.

The national accounts also indicate, however, that the evolution of our social institutions is falling further and further behind the accelerating pace of technological change. It turns out that the magnificent industrial apparatus of America has been producing as much poverty as wealth. Poverty is now officially acknowledged to be the lot of at least 25% of our population. Contemporary American poverty is selective, as Michael Harrington has pointed out. It tends to settle in places where it disappears from sight—hidden away geographically, for example, in Appalachia and in the central cities from which more fortunate members of our society have fled to set up their new settlements in the suburbs. In New York and Chicago, the third generation of families on relief has already begun its blighted existence.

Poverty is selective also with respect to age. Unemployment rates, which for the labor force as a whole are officially acknowledged to exceed 5%, exceed that rate among the youth by at least twice, and among Negro youths it exceeds the rate among white youths by more than twice again. In fact, the prevailing rate of unemployment among Negro youths in the ghettos of our central cities runs from 40 to 50%. The high-school drop-out may spend five years or more in empty limbo between school and his first job. Out of such alienation of so many of our young people has come the rise in juvenile delinquency, and out of the rejection of our Negro youth came the riots in the streets of the north during the past summer. Poverty is equally selective with respect to age at the other end of life. The 40% of our adult population not counted in the labor force now includes several million men and women retired to live on the pittance of monthly social security checks, under contract not to seek gainful employment.

Such are the shameful facts that confront us in the midst of the most prolonged boom since the crash of 1929. Forecasters predict the boom will hold up well into the first quarter of the new year. Against the expectations of myself as well as a few other pessimists, the tax cut has had a strongly stimulating effect on the economic system. It has encouraged a remarkably high rate of investment by industry in new capital equipment—one-third of the investment going to modernization thereby also accelerating the rate of mechanization. Through the action of the familiar Keynesian multiplier, these expenditures on the capital investment side have helped to sustain consumer expenditures at new highs. The argument that fiscal measures may help to reduce unemployment, therefore, finds support in the current movement of the economy. Although these measures and the prolongation of the boom have not actually reduced unemployment below the 5% line, a catastrophic increase in unemployment has been forestalled.

The financial pages all agree, however, that this boom has a terminal date; most set it around the end of the first half of 1965. As the boom runs out, the application of mere counter-cyclical measures—a further cut in Federal taxes, for example—will be of no avail. At the same time, responsible citizens and public officials must face up to the question of the armaments budget. Even in advance of that distant date when we may see some substantial measure of disarmament, the military budget must be cut back. Our country long since acquired the capability of overkill, counting all the targets in China as well as in the U.S.S.R. Yet, with the business cycle turning downward, it will take brave men to cope with the fact that eight million jobs hang directly upon the size of the military budget.

Plainly, the termination of the present boom will require not a tax cut but, on the contrary, a considerable expansion in public expenditure. That expansion has got to come, moreover, in the Federal budget. It is perfectly plain that the payrolls of local governments are not equal to short run challenges; they cannot respond as flexibly and with the same massive effect as Federal expenditures can. The next Administration will be compelled to seek, therefore, a vigorous expansion in Federal expenditures on public works and public welfare.

I don't think I betray the security of the present Administration at this point in the national election campaign by telling you that task forces in every department in Washington are at work on the question of how to spend increased sums on non-military undertakings of the Federal Government. The house economists of the Kennedy Administration observed some time ago that the nation had accumulated a backlog of demand for public works and welfare equal in magnitude to the backlog of demand for consumer goods and capital goods at the end of World War II. According to the National Planning Association, such neglected tasks as urban redevelopment, mass transportation, control of environmental pollution, and restoration of natural resources could absorb additional public investment at a rate of $60 billion per year. The Arms Control and Disarmament Agency, which is principally responsible for analyzing the prospective impact of disarmament on the economy, predicts an easy transition from huge outlays for warfare to huge outlays for welfare—it points to this backlog of unmet public needs. Soviet economists join their American colleagues in assuring us that capitalism is equal to the task.

All of this is cheering to hear. And it is especially considerate of the Soviet economists to give us their encouragement. But, against a value system that stoutly resists every increase in Federal expenditure except those that carry the absolute sanction of the national defense, any effort to increase public expenditures for public welfare will encounter heavy political opposition.

The backward state of our value system is suggested by the following figures describing the condition of our society: America has, in fact, the highest rate of unemployment among all the industrial nations of the world. If the maintenance of adequate nutrition is taken to establish the poverty line, then Department of Agriculture studies show that not one-quarter but one-third of our fellow citizens remain not only ill-fed but ill-clothed and ill-housed as well. Our country has the lowest ratio of public to private expenditures, even with our gigantic war budget. In the public sector—in Federal, state and local budgets—our economy turns over 25% of its Gross National Product. The lowest figure you find in any other industrial society is 30%. America has the lowest rate of public expenditure on public welfare and public works; it comes to something less than 10% for the country as a whole. The lowest figure in any other industrial nation is nearly three times this percentage.

Last spring, the Johnson Administration took its first tentative steps to meet the impending short-run economic crisis. It assembled from already on-going and funded activities of the Federal Government an anti-poverty program. Meanwhile private institutions and individuals were attempting to draw the lines of long-run perspective. One committee of concerned citizens—the self-styled Ad Hoc Committee on the Triple Revolution, which included political economists, historians, former public officials, labor leaders, civil rights workers, and at least two men who have met payrolls—looked rather more deeply into the widening gap between the productive capacity of our industrial system and the effective demand of our consumer economy. In one conclusion to their analysis, they envisioned a day when "Society, through its appropriate legal and governmental institutions, must undertake an unqualified commitment to provide every individual and every family with an adequate income as a matter of right."

The idea of paying people incomes whether they work or not captured attention in newspaper city rooms all across the country. It seems scarcely necessary to add that the idea also won a great deal of unfavorable comment. Setting aside the ephemeral essays of the commentators and pundits who explain the news to the rest of us, the comments of two distinguished public figures are illuminating. The Secretary of Labor, Willard Wirtz, declared: "I think the analysis is right but the prognosis and the prescription is wrong." He added: "I don't believe the world owes me a living and I don't believe it owes anyone else a living."

The other comment comes from a man who was, at the time, candidate for the Republican Presidential nomination. You may recognize his voice. He said: "Our job as Republicans is to get rid of people who will even listen to people who say we should pay people whether they work or not!"

These two statements, taken together, speak faithfully for the austere premises of classical political economy and the tenacity of their grip upon the conscience of many members of our society. Yet the preposterous notion of a guaranteed annual income (or G.A.I., as it has come to be called) has found its way onto the agenda of public issues.

Upon deeper reflection over the summer, for example, *Life* magazine returned to the subject for the second time on its editorial page. This time, *Life* conceded that there is technological disemployment: "... experts can't agree whether technological unemployment is growing by 4,000 or 40,000 jobs a week. But it is growing fast enough to see that the seeming logic of the . . . plan for free incomes, or instant socialism, may grow too."

Having frightened itself with this prospect, *Life* goes on to say that there can be "more than one radical alternative" and puts forward one of its own: "It is private capitalism, after all, that has brought us to the brink of this daunting affluence, and there is an obvious capitalist solution to the problem that the success of capitalism is creating. It lies in the ownership of the machines and the processes that are destroying the old jobs and creating the new wealth." *Life's* proposal is that the ownership be spread—to everybody! Endorsing the analysis set forth in *The New Capitalists* by Mortimer Adler and Louis Kelso, *Life* would ". . . guarantee bank loans for new stock acquisitions through a Capital Diffusion Insurance Corporation modeled on FHA."

Let us tarry a moment, here, to contemplate the prospects of instant capitalism. The figures indicate that it would be much more difficult to achieve *Life's* worthy purpose by instant capitalism than by what it calls instant socialism. Consider these disparities in the wealth of our citizenry: As is well known, the bottom 20% of our population gets only 5% of the national income —at the summit of society these percentages are precisely reversed. The bottom 20% thus does poorly enough as income earners. But they show up even worse as capitalists: they hold no liquid assets whatever, except the cash they may happen to have in their jeans. The next 30% of the population above holds liquid assets not exceeding $500 per family. So the bottom 50% of our society holds less than 3% of our liquid assets. It scarcely need be added that these people have no equity or debt interest in the productive system of our land; for 87% of the people have no such stake in the system. As for real property, 50% of our population have an equity of less than $1,000 in the homes in which they live. You have to go to the top 10% of income earners before you find people whose assets begin to equal their annual income; you have to go to the top 1% before you find people whose equity and property holdings keep them in the style to which they are accustomed. Plainly the proposal that we seek a more equitable distribution of affluence through the redistribution of ownership presents a more radical alternative than the achievement of that end by the redistribution of incomes.

Life is not alone in its concern with the question of how American society might now go about the equitable distribution of the abundance that overburdens institutions framed for the opposite purpose. That soft-spoken quarterly, *The American Scholar,* the journal of Phi Beta Kappa, devotes most of this quarter's pages to a symposium on "the problems that unite us." Out of six articles in this symposium, four plainly reflect thoughtful consideration of the possibility of guaranteeing incomes to people.

I will quote from one author, August Heckscher, a perceptive and sensitive student of American life who served the Kennedy Administration as the President's special assistant in cultural affairs. He begins by saying: "The objections to this approach [that is, the guaranteed annual income] are obvious," and declares: "The very idea of large populations doing nothing but pleasing themselves goes against the American grain." Nonetheless, he comes forward with a radical alternative of his own: "Suppose the monetary rewards of society went, as now, to those who work —and also to those who study. Would this not be a means of assuring their being saved from a bored and barren existence?"

This author then goes on to suggest other ways in which the surplus of human capacity might be soaked up: "At the simplest level one can readily conceive efforts to organize personal or household services more imaginatively so that the work can be done more efficiently. Hours can be made regular and wages can be more nearly commensurate with those earned in other fields." A little later in his analysis, touching on the question of how these increased wages to domestics are to be financed, he comes up with a truly radical alternative: "The salaries . . . could be supplemented [from the public treasury] so as to keep the supply adequate and yet not put the wage out of reach of those who require such services. To supplement in a similar way the rate which people are ready to pay handymen or gardeners could substantially cut relief rolls."

This surely goes beyond either instant socialism or instant capitalism; you might call it instant feudalism. In fact the vision of instant feudalism comes clearer in this author's next, still "more far-reaching" suggestion: "It assumes retirement from the industrial work force at a considerably earlier age than now, together with pensions and social security which would be clearly conceived as 'deferred wages.' . . . From such a pool we could draw a host of talents and services which would make our common life more various, colorful, and pleasant. . . . We can indeed conceive a whole second economy—the economy of craftsmanship and service —growing up alongside the economy of the machine."

Probably, this vision could be more swiftly and effectively realized in certain of the underdeveloped countries where the economy of craftsmanship still exists and where it is threatened by destruction through the infectious spread of the industrial revolution. In America we would have to reconstruct the economy of craftsmanship from the ground up.

Before we start designing Utopias or building the Great Society, it seems to me, we ought to turn to a more searching and possibly painful re-examination of our inexplicit premises—our values. A good way to begin is to ask what we mean by work and what we mean by leisure. With these two words we precipitate the crisis. The proposal of a guaranteed annual income presses the underlying issue in its sharpest and most uncomfortable form.

The objection to the Heckscher vision of the dual craftsmanship-machine society rests upon its hierarchical character, implicit in the compulsion that relates the services of the handyman and the gardener to "us." This defect could, in fact, be cured by the guarantee of an annual income, paid as a matter of right and not in compensation for services rendered. There would then be no reason why the cultures of craftsmanship and machines could not flourish side by side in moral parity. And there could even be a third culture—of leisure, which would include, I hope, dry-fly fishing.

On the other hand, criticism of the G.A.I. notion from the left expresses the dark suspicion that this is a middle class stratagem to tranquilize the proletariat by putting the poor on the dole. Apparently, most people are deeply troubled by the thought of what *other* people might or might not do with their leisure time!

Except for the attention it has so recently won in public discussion, there is nothing very novel or profound about the idea of a guaranteed annual income. Nor is it so novel in practice. A substantial portion of our society is already living not on a guaranteed and not on a securely annual income but on an income from the public treasury. The people get these incomes on the most humiliating and degrading terms. They get their dole because they present themselves for certification by the appropriate authorities as indigents or paupers; or they get their monthly checks from Uncle Sam because they take an oath not to go back to work and earn more than a stated percentage of their Social Security income. In other words, the American society today offers an income without work to a large number of its members but makes the offer on terms that shame us all. The ugly transactions involved derive their ethical justification from the deep unconscious of society—from the institutional memory of the days when the lash drove 80% of the population to work in the fields and mines in order that the few might get on with the high occupations of making history and civilization. The cruelty and inhumanity that persists in our system from those days must be extirpated if we are to resolve successfully the issues that confront us in the tide of abundance set running in America by the present culmination of the industrial revolution.

In my opinion, the issues must be met under two major headings. First, we must recognize that economic and social institutions are man-made and so subject to human will. We can't see the invisible hand because, in truth, it isn't there! The enormous power conferred upon modern societies by industrial technology must be brought under the witting and rational control of democratic institutions still to be perfected.

Second, we must recognize that abundance sets the foundations of an entirely new ethical and moral order. The cultural deprivation that blights the life of a single child in the racial ghettos of our central cities ultimately exacts its cost in the lives of every other citizen. The prolongation of the agony of economic development threatens to destroy the frail parliamentary institutions of India and bring that poverty-stricken nation into the nuclear club under a military dictatorship.

At this turn in human affairs it is plain that each man's well being can increase only to the degree that the well being of all other men is increased. The work of the world still remains in large part to be done. But the instruments to accomplish it are now in our hands. The work that needs most to be done, especially here in America, is in tasks that enrich society as generously as the individuals who undertake them.

1964 HARRY GOODE MEMORIAL AWARD

Presentation to
Howard Hathaway Aiken
at the FJCC—October 28, 1964—Jack Tar Hotel
San Francisco

Synopsis

The presentation of the Harry Goode Memorial Award to Dr. Howard H. Aiken marked the first time the American Federation of Information Processing Societies signally honored an outstanding contributor to the field of computers and information processing. In conjunction with this first presentation, a Silver Medal was presented to Mrs. Harry H. Goode in recognition of the late Mr. Goode's invaluable contributions to the information processing sciences.

The Medalist

For more than 25 years Howard H. Aiken has given continuing momentum to the growth of computer technology. As a student, he proposed an automatic calculating machine and collaborated with IBM engineers in the design of Mark I, the first large-scale, automatic digital computer, completed in 1944. In the following years, Dr. Aiken was instrumental in perfecting computers, adapting them to nonnumerical applications, and in guiding students to productive careers in the computer sciences.

Howard Hathaway Aiken received the S.B. degree in electrical engineering from the University of Wisconsin in 1923. From 1923 to 1932 he was associated with Madison Gas and Electric, Westinghouse, and the Line Material Company, and he studied physics at the University of Chicago. He received the S.M. in physics in 1937 and the Ph.D. in 1939 from Harvard University. In 1941, after two years as instructor in physics, he became an associate professor in applied mathematics.

Following Mark I, Dr. Aiken built three other machines: Mark II, in 1947, largely an electromechanical machine, was operable not only as one machine solving one problem but as two machines solving two problems simultaneously; Mark III, completed in 1950, utilized magnetic tape drives and magnetic drums to store binary-coded decimal numbers and instructions; Mark IV, completed in 1952, relied heavily on solid-state components. It included magnetic core storage, selenium rectifiers to perform all arithmetic and control functions, index registers, and an encoding device which permitted the writing of programs in an algebraic notation quite similar to that normally used.

With the practicality of computers firmly established, Dr. Aiken concentrated on establishing sound scientific bases for the arts of computing and machine design, investigating nonnumerical applications and establishing an educational program in the computing field. Dr. Aiken gave his first course in the computer field at Harvard University in the academic year 1947-48. Largely through his leadership, Harvard became one of the first universities to have extensive graduate and undergraduate programs leading to degrees in the computer field.

Dr. Aiken encouraged his students to seek adequate mathematical methods for investigating the functional behavior of electronic control circuits and to venture into such fields as control systems engineering, data processing, mathematical linguistics, and applied discrete mathematics.

Dr. Aiken was Director of Harvard University's Computational Laboratory for twelve years. He is presently Professor Emeritus of Harvard University and Director of the University of Miami's Institute of Informational Science. Dr. Aiken's achievements have received international recognition. He was one of the first members of the National Research Council's Committee on High-Speed Calculating Machines. He is a Fellow of the American Academy of Arts and Sciences, the Association for the Advancement of Science, and of the Institute of Radio Engineers, and he is a member of the Swedish Ingeniors Vetenskaps Academien. In France, he is a Chevalier de la Legion d'Honneur, and in Spain, he is a Consejero de Honor del Consejo Superior de Investigaciones Cientificas. He received the Testimonial of Eminent Professional Services from the University of Wisconsin, his alma mater, and he holds an honorary doctorate from Germany's Technische Hochschule at Darmstadt. He was awarded Belgium's Officer's Cross of the Order of the Crown, the U.S. Navy's Distinguished Public Service Award, and the U.S. Air Force's Decoration for Exceptional Civilian Service.

The Harry Goode Memorial Award

Harry H. Goode, born in New York City on July 1, 1909, was a pioneer and leader in the field of system engineering. One of the first scientists to fully comprehend the powers and abilities of computers, he formulated many principles of system engineering and developed techniques for the design, analysis and evaluation of large-scale systems. He was instrumental in initiating early systems projects, including the Typhoon and Whirlwind computers at MIT. Among other activities, he participated in the study which led to the Bomarc missile, and he conceived and developed the Air Defense Integrated System Project.

In addition to his scientific contributions, Mr. Goode advanced the information processing sciences through his teaching at the University of Michigan and his many publications on statistics, stimulation and modeling, vehicular traffic control, and system design. One of his most important achievements was coauthoring the first book on *System Engineering* which classified and regularized systems and their design processes.

Mr. Goode, statistician, mathematician, electrical and chemical engineer and professor, was a member of the IRE (now IEEE) professional group on electronic computers and of the computer advisory committee of the Society of Automotive Engineers. As chairman of the National Joint Computer Committee, he led this group in creating an expanded and strengthened organization, AFIPS, to help meet the needs arising from the rapidly growing information processing technology. Mr. Goode died in an automobile accident before AFIPS was formally chartered.

The Harry Goode Memorial Award has been established in recognition and appreciation of Mr. Goode's invaluable contributions to the information processing sciences. Its purposes are to encourage further development of the field and to acknowledge and honor outstanding contributions to the information processing sciences.

The Scribe Accroupi, the famous Egyptian statue in the Paris Louvre, inspired the noted European sculptor, Andras Beck, in the conception of the Harry Goode Memorial Medal. In the sculptor's own words, *The Scribe* allegorizes man's intellectual effort, while the AFIPS emblem symbolizes the machine that aids and supports human effort in this field. The arrow is the signature of the Hungarian-born sculptor, former Professor of Sculpture at the Budapest Academy of Fine Arts, who has resided in France since 1957.

The Award Presentation to Howard Aiken

E. L. Harder
Chairman, American Federation of Information Processing Societies

Shortly after the formation of AFIPS from the Joint Computer Committee, work started on the creation of an award which could be bestowed on the very few individuals who had made the greatest contribution to this profession. This award, which now exists, and the first presentation of which will be made today, has been named the Harry Goode Memorial Award because of the unique part played by the late Harry Goode in the organizing and teaching of the computer and system engineering science and because of his outstanding leadership in creating this American Federation of Information Processing Societies itself.

Very fittingly, the Committee has selected as the first recipient the one man in the world who earliest, and with great vision and foresight and enormous strength of purpose and perseverance, led the way straight to the goal of the computers that we have today. This man is Howard Aiken and he is here today, although his fame is so great that you might expect he only existed in fairy tales. Also, this is a very special day in that Mrs. Harry Goode, the wife of the man in whose name this medal stands, is here to witness this first occasion on which it will be bestowed—on this illustrious pioneer of today's computers.

The place that Harry Goode held in the estimation of his fellows, his impact on the science of system engineering and of information processing, can hardly be more forcibly conveyed to you than by the fact that his fellows have chosen to engrave, with his image, this medal signifying the highest honor that the computer societies of America are able to give to any man.

Now, let me tell you about the man who has been selected. As with all legendary characters, you all know something about him, but let me put some of the pieces in perspective—something which is getting a little easier to do as the years roll on. For now we can associate his work not only with an important scientific venture but with a great profession and a great industry which he played a very leading part in creating. This very association, AFIPS, exists for the purpose of exchanging technical information upon which the progress of our profession and of our industries and scientific institutions depends. It exists for the purposes to which Howard Aiken devoted a good part of his life and in which he led the way. He set a pattern in visualizing what needed to be done and what was possible, in persevering in doing it, and in seeing to it, in a very real way, that this technology became available to humanity. This he did through organizing the teaching of it in the universities and by gathering together groups from all over the world at Harvard to work personally with him in the carrying through of the Mark I, Mark II, Mark III, and Mark IV programs. As you travel the world over today, you encounter leaders in the computer profession and in the industries who got their early start through an invitation to come to Harvard and work with Dr. Aiken on the development of these machines. This will give you some idea of the breadth of the vision and of the force and vigor with which it was carried out.

Dr. Howard Aiken, Professor Emeritus of Harvard University, you have been honored by many scientific institutions and by many countries, including your own. It is now the computer and information processing profession itself, which you helped so greatly to create, that wishes to do you honor. It is the very great privilege of the American Federation of Information Processing Societies to bestow upon you, as first recipient, the Harry Goode Memorial Award—the citation for which I would now like to read in the presence of these several hundred members of this new profession—of *your* profession.

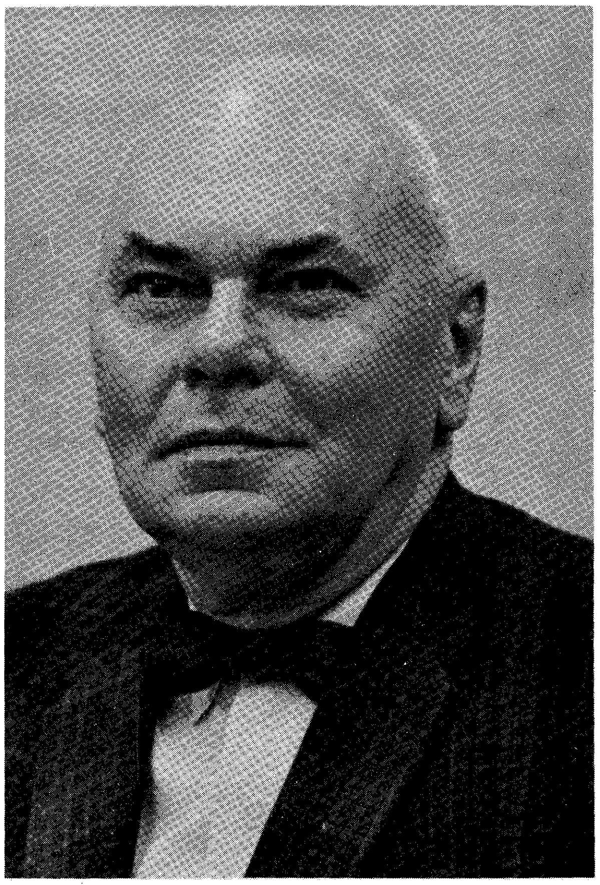
The Citation for the award follows:

To HOWARD HATHAWAY AIKEN

for his original contribution to the development of automatic computers that led to the first large-scale, general-purpose, automatic digital computer ever to be put into operation;

for his continuous work in the field of digital computers as an Engineer;

and for the knowledge and inspiration imparted to many as a Teacher.



DR. HOWARD AIKEN

Recipient of the Harry Goode Memorial Award to be made at the 1964 Fall Joint Computer Conference in San Francisco.

The Reply of the Medalist

Dr. Howard Aiken

It is difficult for me to convey to you how deeply honored I feel at this moment. I would like to say a few words about the significance of this event to me. First of all, it is an honor that comes to me because of my association in the field which is so dear to each and every one of us and for which I feel so great an affection. Second, it has been a great pleasure to me on this occasion to find so many of my friends, former associates, and colleagues here and have the opportunity to see them and think of old times again. There is, however, still another reason on which I wish to spend a few more words.

Years ago, when the old Air Materiel Command of the United States Air Force first discovered the fact that computers might be useful in the area of logistics, they called in a number of consultants to assist in programming and choosing machines, debating the various techniques which were to be employed and applying computational equipment in the supply end of our Air Force. At that time, I was a consultant to the United States Air Force and spent a great deal of time—perhaps I should say too much time—traveling back and forth from Cambridge up to Dayton to the Wright-Patterson Air Force Base. It was, however, a period which is very pleasant in my life, and it was at this time that I was associated with Harry Goode who was also an associate consultant of the Air Force in this problem of making early applications of machines to logistic problems. It was here that I met Harry, worked with him, came to respect him, his judgment, and his influence on the problems in which we were both concerned. It is, therefore, especially gratifying to me that the award which I have just received bears his name. Thank you.

Presentation of a Replica Medal to Mrs. Harry Goode

Isaac L. Auerbach
Chairman, Harry Goode Award Committee

The Harry Goode Memorial Award is the highest recognition that can be given to anyone within the computer field. It is an award given by the profession itself to one of its own contributors for outstanding accomplishments. By our very nature, as scientists and engineers, we tend to depreciate our achievements that benefit society and mankind. Following the rules for scientific behavior, we tend to strip the products of our efforts of all glamour in our constant search for the ultimate. Therefore, an award by this hypercritical, intensely analytical profession to one of its own members is of great significance. There has been no profession that has influenced civilization's forward movement more than that of science and engineering, and, within our lifetime, the advent of the computer is the single most important technical development, one that will invade every aspect of our lives. It is, therefore, timely that AFIPS, the foremost national society of societies, recognize the oustanding contributions made by our colleagues.

Credit should be given to two men who did most within AFIPS in the preparatory work that led up to this award: Mr. Claude R. Kagan, first Chairman of the AFIPS Award Committee, and Samuel Levine, present Chairman of the Committee. It was under their guidance that the medal itself was prepared, through the assistance of Mortan Astrahan, and they are responsible for the folders that are at your seats.

The Harry Goode Memorial Award is a tribute to the memory of a man of rare versatility, talent, vigor and vision. He was a "regular guy," never elevating himself above his peers or flaunting his accomplishment. He was always concerned about the feelings of his associates and went out of his way to recognize and promote the contributions and suggestions of his co-workers. As a professor at the University of Michigan, he was respected and revered by his students, and his advice was constantly sought by his colleagues at the University. His contributions to systems engineering through his writings and teachings are well known; he was the senior author of the first book on systems engineering. His participation in many vital defense and government projects provided the springboard to some of the most forward thinking of those projects. He was frequently consulted by the largest corporations in the United States. His leadership provided the guiding spirit that brought AFIPS into existence, and his example has provided the encouragement necessary to accomplish its fulfillment and growth.

This award, named in honor of the man who forged the framework of AFIPS, is to recognize outstanding contributions to the field of information sciences. The beneficial impact of information sciences on mankind will be greater than any other technological development in this half-century.

It is therefore fitting and proper that this silver replica of the Harry Goode Memorial Award, just given to one of the world's foremost computer scientists, be presented to Mrs. Harry Goode, the worthy partner of this man we all loved so much. Elsie provided the enthusiastic support and encouragement which made her husband's achievements possible. She accepted without complaint the many sacrifices necessitated by his dedication to his profession because she was aware of the importance of his contributions. We are indeed happy to honor her today.

Harry Goode Memorial Award Committee

Isaac L. Auerbach, *Chairman*

Samuel N. Alexander	John C. McPherson
Alston Householder	Jerre D. Noe

AFIPS PRIZE PAPER AWARD

for the

1964 FALL JOINT COMPUTER CONFERENCE

The recipient of the Award for the best paper presented at the 1964 Fall Joint Computer Conference was:

ERROR CORRECTION IN CORC

by David N. Freeman
IBM, General Products Division
Development Laboratory

PARALLEL OPERATION IN THE CONTROL DATA 6600

James E. Thornton
Control Data Corporation
Minneapolis, Minnesota

HISTORY

About four years ago, in the summer of 1960, Control Data began a project which culminated last month in the delivery of the first 6600 Computer. In 1960 it was apparent that brute force circuit performance and parallel operation were the two main approaches to any advanced computer.

This paper presents some of the considerations having to do with the parallel operations in the 6600. A most important and fortunate event coincided with the beginning of the 6600 project. This was the appearance of the high-speed silicon transistor, which survived early difficulties to become the basis for a nice jump in circuit performance.

SYSTEM ORGANIZATION

The computing system envisioned in that project, and now called the 6600, paid special attention to two kinds of use, the very large scientific problem and the time sharing of smaller problems. For the large problem, a high-speed floating point central processor with access to a large central memory was obvious. Not so obvious, but important to the 6600 system idea, was the isolation of this central arithmetic from any peripheral activity.

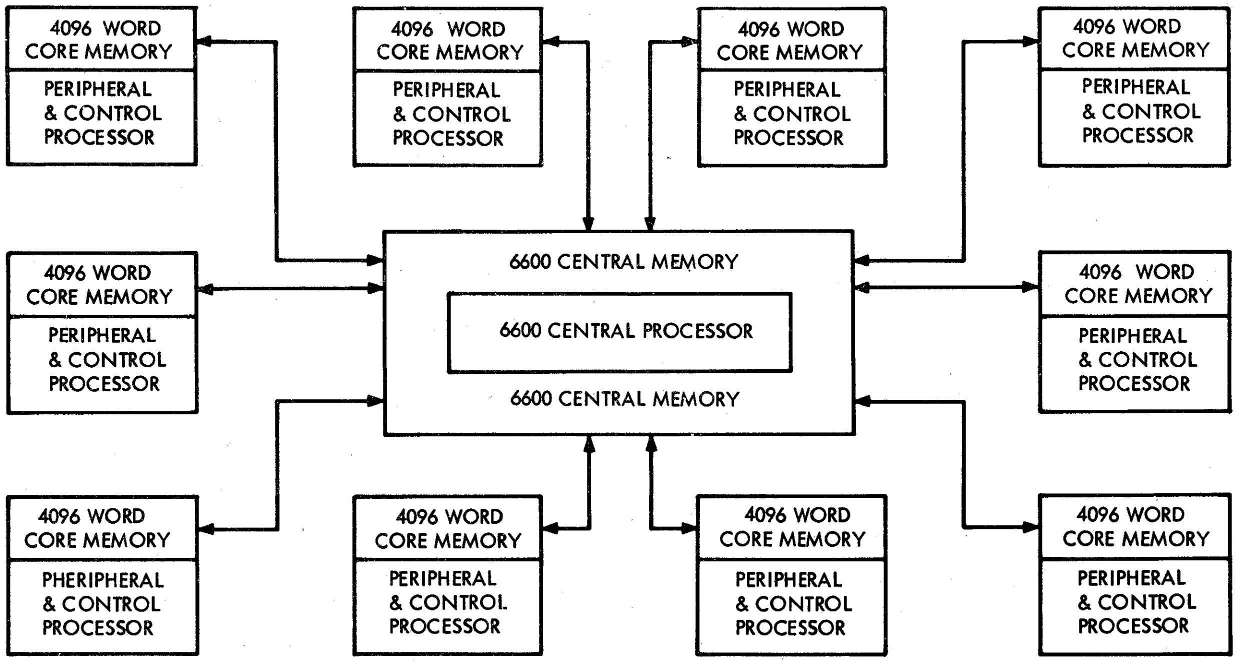


Figure 1. Control Data 6600.

It was from this general line of reasoning that the idea of a multiplicity of peripheral processors was formed (Fig. 1). Ten such peripheral processors have access to the central memory on one side and the peripheral channels on the other. The executive control of the system is always in one of these peripheral processors, with the others operating on assigned peripheral or control tasks. All ten processors have access to twelve input-output channels and may "change hands," monitor channel activity, and perform other related jobs. These processors have access to central memory, and may pursue independent transfers to and from this memory.

Each of the ten peripheral processors contains its own memory for program and buffer areas, thereby isolating and protecting the more critical system control operations in the separate processors. The central processor operates from the central memory with relocating register and file protection for each program in central memory.

PERIPHERAL AND CONTROL PROCESSORS

The peripheral and control processors are housed in one chassis of the main frame. Each processor contains 4096 memory words of 12 bits length. There are 12- and 24-bit instruction formats to provide for direct, indirect, and relative addressing. Instructions provide logical, addition, subtraction, shift, and conditional branching. Instructions also provide single word or block transfers to and from any of twelve peripheral channels, and single word or block transfers to and from central memory. Central memory words of 60 bits length are assembled from five consecutive peripheral words. Each processor has instructions to interrupt the central processor and to monitor the central program address.

To get this much processing power with reasonable economy and space, a time-sharing design was adopted (Fig. 2). This design contains a register "barrel" around which is moving the dynamic information for all ten processors. Such things as program address, accumulator contents, and other pieces of information totalling 52 bits are shifted around the barrel. Each complete trip around requires one major cycle or one thousand nanoseconds. A "slot" in the barrel contains adders, assembly networks, distribution network, and interconnections to perform one step of any peripheral instruction. The time to perform this step or, in other words, the time through the slot, is one minor cycle or one hundred nanoseconds. Each of the ten processors, therefore, is allowed

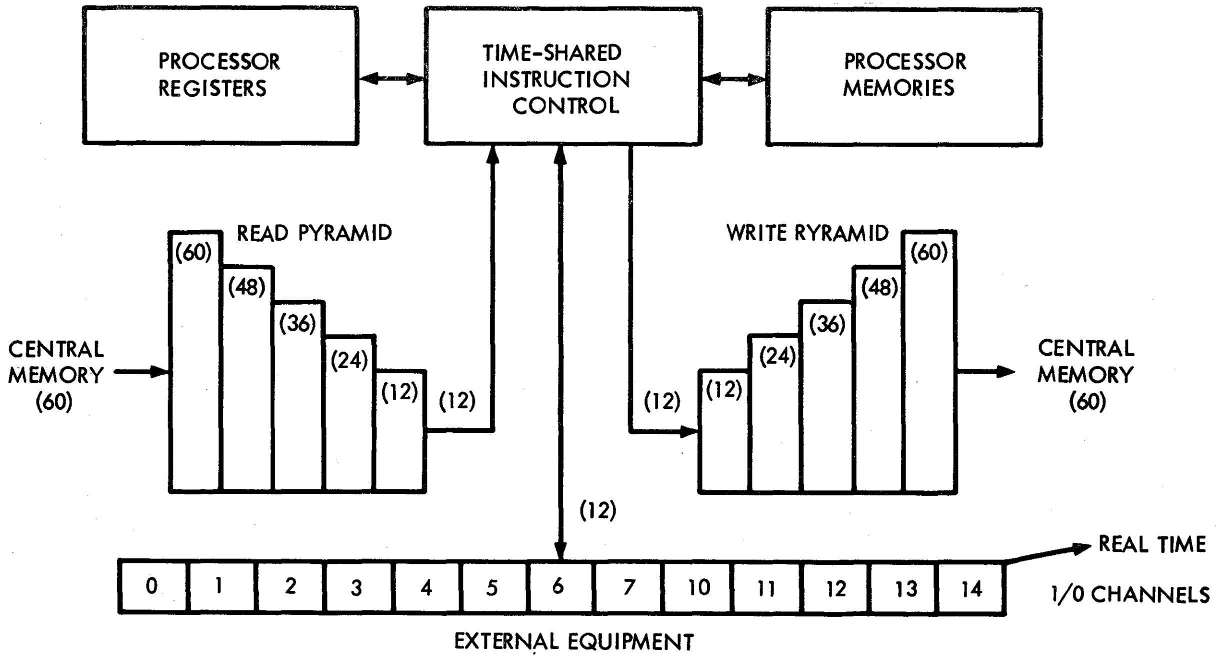


Figure 2. 6600 Peripheral and Control Processors.

one minor cycle of every ten to perform one of its steps. A peripheral instruction may require one or more of these steps, depending on the kind of instruction.

In effect, the single arithmetic and the single distribution and assembly network are made to appear as ten. Only the memories are kept truly independent. Incidentally, the memory read-write cycle time is equal to one complete trip around the barrel, or one thousand nanoseconds.

Input-output channels are bi-directional, 12-bit paths. One 12-bit word may move in one direction every major cycle, or 1000 nanoseconds, on each channel. Therefore, a maximum burst rate of 120 million bits per second is possible using all ten peripheral processors. A sustained rate of about 50 million bits per second can be maintained in a practical operating system. Each channel may service several peripheral devices and may interface to other systems, such as satellite computers.

Peripheral and control processors access central memory through an assembly network and a dis-assembly network. Since five peripheral memory references are required to make up one central memory word, a natural assembly network of five levels is used. This allows five references to be "nested" in each network during any major cycle. The central memory is organized in independent banks with the ability to transfer central words every minor cycle. The peripheral processors, therefore, introduce at most about 2% interference at the central memory address control.

A single real time clock, continuously running, is available to all peripheral processors.

CENTRAL PROCESSOR

The 6600 central processor may be considered the high-speed arithmetic unit of the system (Fig. 3). Its program, operands, and results are held in the central memory. It has no connection to the peripheral processors except through memory and except for two single controls. These are the exchange jump, which starts or interrupts the central processor from a peripheral processor, and the central program address which can be monitored by a peripheral processor.

A key description of the 6600 central processor, as you will see in later discussion, is "parallel by function." This means that a number of arithmetic functions may be performed

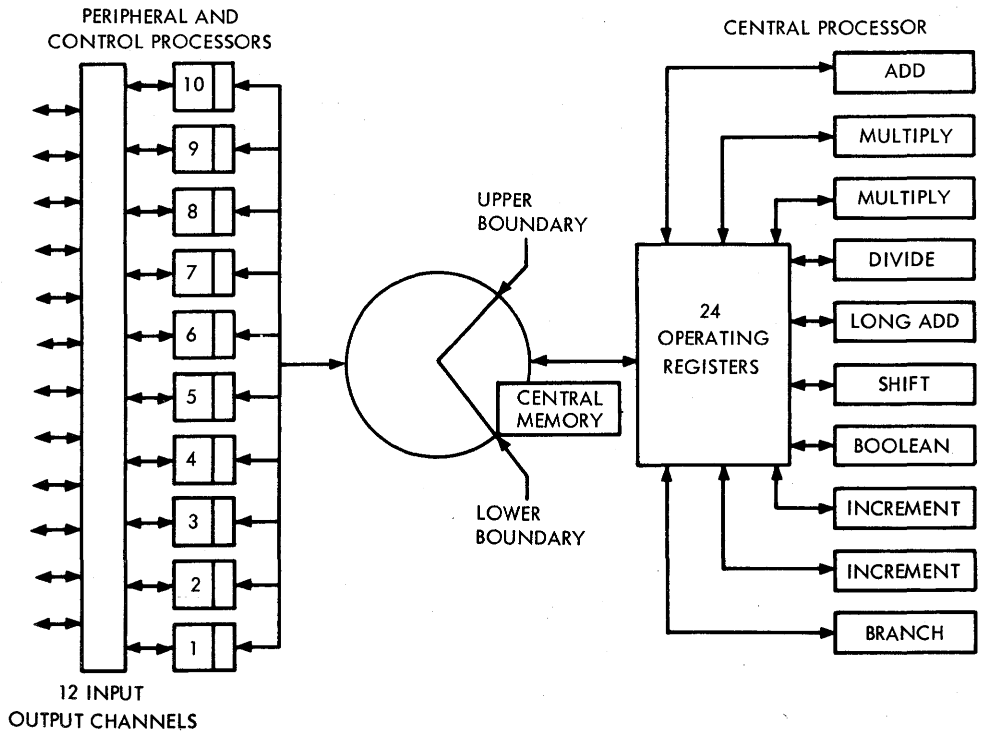


Figure 3. Block Diagram of 6600.

concurrently. To this end, there are ten functional units within the central processor. These are the two increment units, floating add unit, fixed add unit, shift unit, two multiply units, divide unit, boolean unit, and branch unit. In a general way, each of these units is a three address unit. As an example, the floating add unit obtains two 60-bit operands from the central registers and produces a 60-bit result which is returned to a register. Information to and from these units is held in the central registers, of which there are twenty-four. Eight of these are considered index registers, are of 18 bits length, and one of which always contains zero. Eight are considered address registers, are of 18 bits length, and serve to address the five read central memory trunks and the two store central memory trunks. Eight are considered floating point registers, are of 60 bits length, and are the only central registers to access central memory during a central program.

In a sense, just as the whole central processor is hidden behind central memory from the peripheral processors, so, too, the ten functional units are hidden behind the central registers from central memory. As a consequence, a considerable instruction efficiency is obtained and an interesting form of concurrency is feasible and practical. The fact that a small number of bits can give meaningful definition to any function makes it possible to develop forms of operand and unit reservations needed for a general scheme of concurrent arithmetic.

Instructions are organized in two formats, a 15-bit format and a 30-bit format, and may be mixed in an instruction word (Fig. 4).

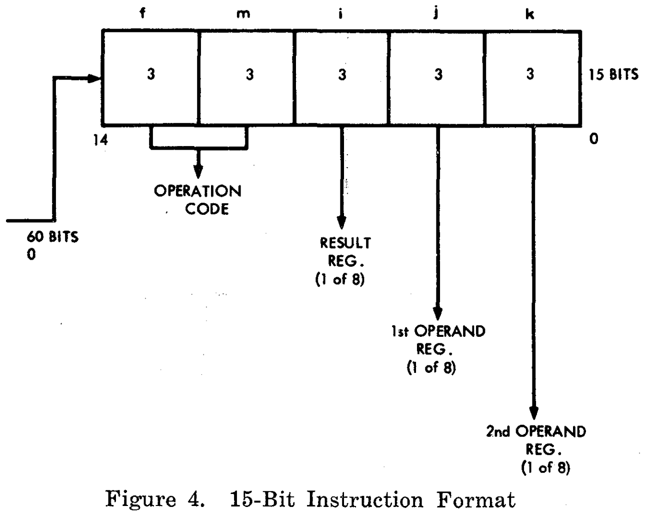


Figure 4. 15-Bit Instruction Format

As an example, a 15-bit instruction may call for an ADD, designated by the *f* and *m* octal digits, from registers designated by the *j* and *k* octal digits, the result going to the register designated by the *i* octal digit. In this example, the addresses of the three-address, floating add unit are only three bits in length, each address referring to one of the eight floating point registers. The 30-bit format follows this same form but substitutes for the *k* octal digit an 18-bit constant K which serves as one of the input operands. These two formats provide a highly efficient control of concurrent operations.

As a background, consider the essential difference between a general purpose device and a special device in which high speeds are required. The designer of the special device can generally improve on the traditional general purpose device by introducing some form of concurrency. For example, some activities of a housekeeping nature may be performed separate from the main sequence of operations in separate hardware. The total time to complete a job is then optimized to the main sequence and excludes the housekeeping. The two categories operate concurrently.

It would be, of course, most attractive to provide in a general purpose device some generalized scheme to do the same kind of thing. The organization of the 6600 central processor provides just this kind of scheme. With a multiplicity of functional units, and of operand registers and with a simple and highly efficient addressing system, a generalized queue and reservation scheme is practical. This is called the *scoreboard.*

The scoreboard maintains a running file of each central register, of each functional unit, and of each of the three operand trunks to and from each unit. Typically, the scoreboard file is made up of two-, three-, and four-bit quantities identifying the nature of register and unit usage. As each new instruction is brought up, the conditions at the instant of issuance are set into the scoreboard. A snapshot is taken, so to speak, of the pertinent conditions. If no waiting is required, the execution of the instruction is begun immediately under control of the unit itself. If waiting is required (for example, an input operand may not yet be available in the central registers), the scoreboard controls the delay, and when released, allows the unit to begin its execution. Most important, this activity is accomplished in the scoreboard and the functional unit, and does not necessarily limit later instructions from being brought up and issued.

In this manner, it is possible to issue a series of instructions, some related, some not, until no functional units are left free or until a specific register is to be assigned more than one result. With just those two restrictions on issuing (unit free and no double result), several independent chains of instructions may proceed concurrently. Instructions may issue every minor cycle in the absence of the two restraints. The instruction executions, in comparison, range from three minor cycles for fixed add, 10 minor cycles for floating multiply, to 29 minor cycles for floating divide.

To provide a relatively continuous source of instructions, one buffer register of 60 bits is located at the bottom of an instruction stack capable of holding 32 instructions (Fig. 5). Instruction words from memory enter the bottom register of the stack pushing up the old instruction words. In straight line programs, only the bottom two registers are in use, the bottom being refilled as quickly as memory conflicts allow. In programs which branch back to an instruction in the upper stack registers, no refills are allowed after the branch, thereby holding the program loop completely in the stack. As a result, memory access or memory conflicts are no longer involved, and a considerable speed increase can be had.

Five memory trunks are provided from memory into the central processor to five of the floating point registers (Fig. 6). One address register is assigned to each trunk (and therefore to the floating point register). Any instruction calling for address register result implicitly initiates a memory reference on that trunk. These instructions are handled through the scoreboard and therefore tend to overlap memory access with arithmetic. For example, a new memory word to be loaded in a floating point register can be brought in from memory but may not enter the register until all previous uses of that register are completed. The central registers, therefore, provide all of the data to the ten functional units, and receive all of the unit results. No storage is maintained in any unit.

Central memory is organized in 32 banks of 4096 words. Consecutive addresses call for a different bank; therefore, adjacent addresses in one bank are in reality separated by 32. Addresses may be issued every 100 nanoseconds. A typical central memory information transfer rate is about 250 million bits per second.

As mentioned before, the functional units are hidden behind the registers. Although the units might appear to increase hardware duplication, a pleasant fact emerges from this design. Each unit may be trimmed to perform its func-

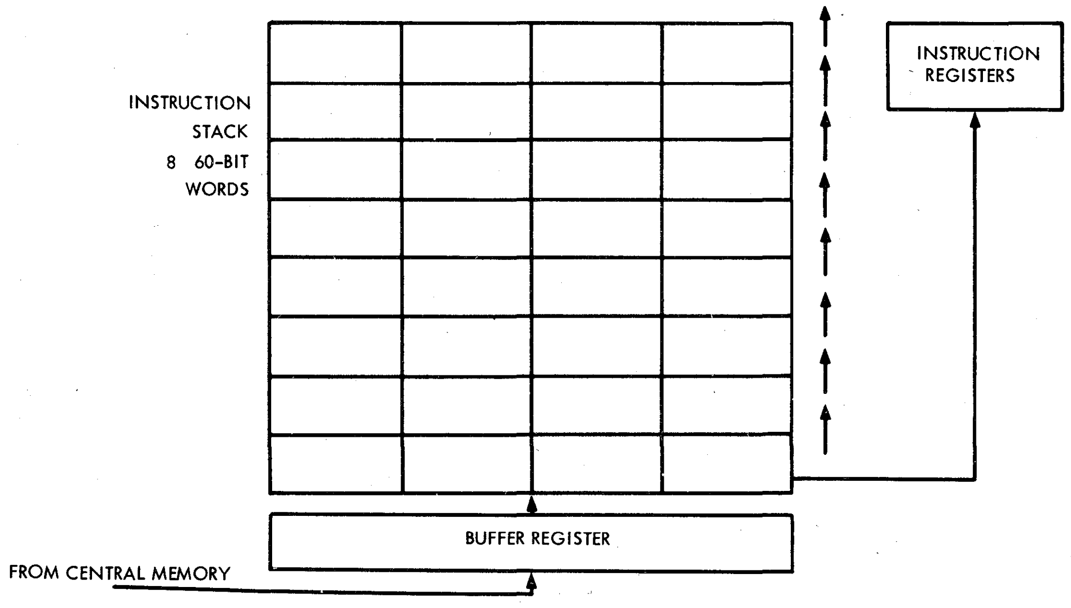


Figure 5. 6600 Instruction Stack Operation.

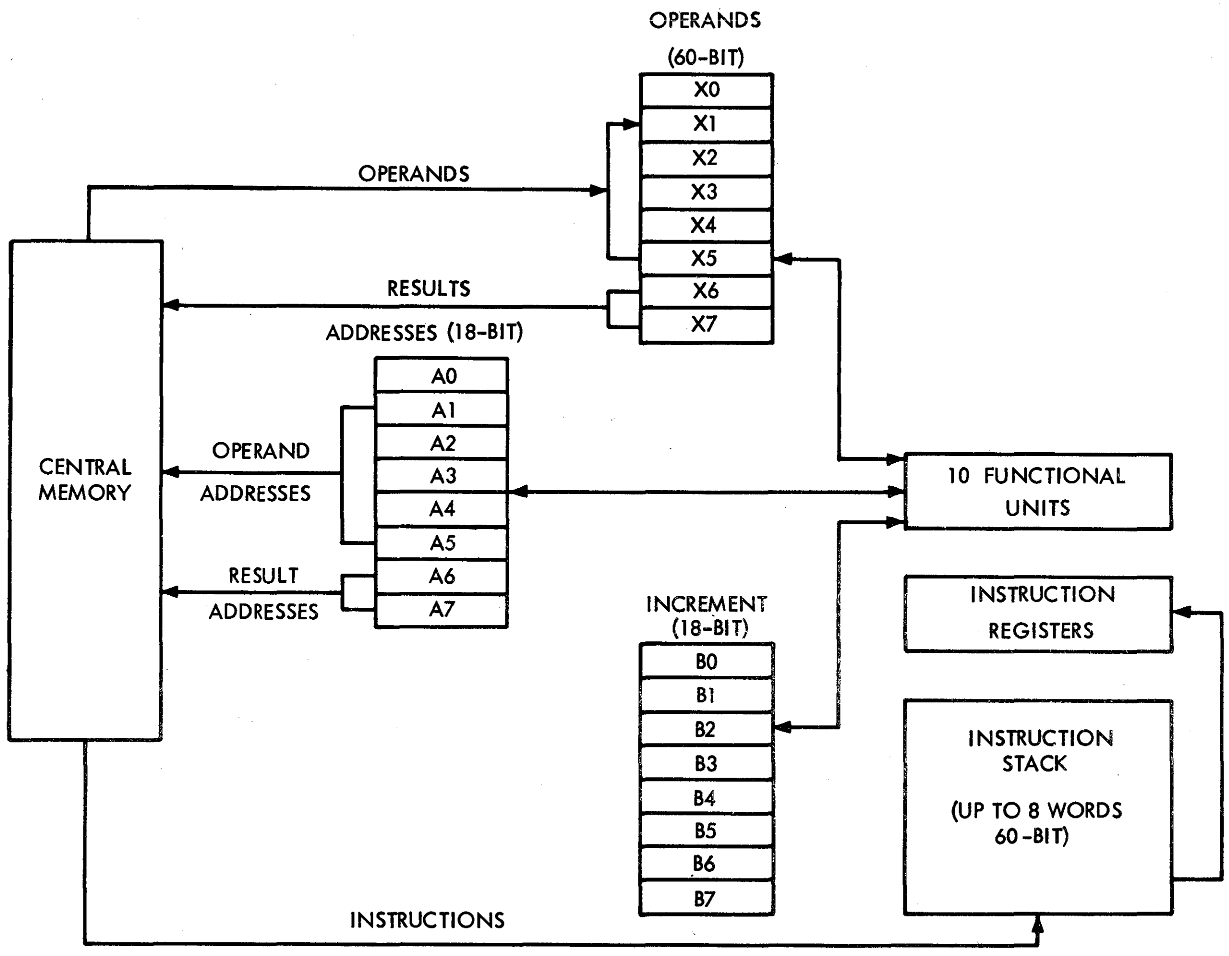


Figure 6. Central Processor Operating Registers.

tion without regard to others. Speed increases are had from this simplified design.

As an example of special functional unit design, the floating multiply accomplishes the coefficient multiplication in nine minor cycles plus one minor cycle to put away the result for a total of 10 minor cycles, or 1000 nanoseconds. The multiply uses layers of carry save adders grouped in two halves. Each half concurrently forms a partial product, and the two partial products finally merge while the long carries propagate. Although this is a fairly large complex of circuits, the resulting device was sufficiently smaller than originally planned to allow two multiply units to be included in the final design.

To sum up the characteristics of the central processor, remember that the broad-brush description is "concurrent operation." In other words, any program operating within the central processor utilizes some of the available concurrency. The program need not be written in a particular way, although certainly some optimization can be done. The specific method of accomplishing this concurrency involves *issuing* as many instructions as possible while handling most of the conflicts during *execution*. Some of the essential requirements for such a scheme include:

1. Many functional units
2. Units with three address properties
3. Many transient registers with many trunks to and from the units
4. A simple and efficient instruction set

CONSTRUCTION

Circuits in the 6600 computing system use all-transistor logic (Fig. 7). The silicon transistor operates in saturation when switched "on" and averages about five nanoseconds of stage delay. Logic circuits are constructed in

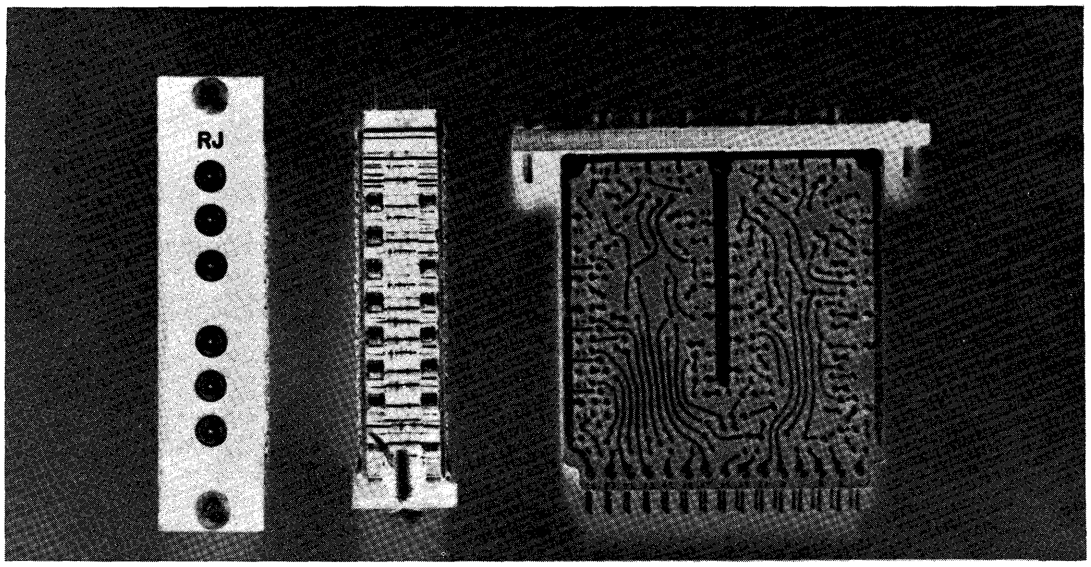


Figure 7. 6600 Printed Circuit Module.

a cordwood plug-in module of about 2½ inches by 2½ inches by 0.8 inch. An average of about 50 transistors are contained in these modules.

Memory circuits are constructed in a plug-in module of about six inches by six inches by 2½ inches (Fig. 8). Each memory module contains a coincident current memory of 4096 12-bit words. All read-write drive circuits and bit drive circuits plus address translation are contained in the module. One such module is used for each peripheral processor, and five modules make up one bank of central memory.

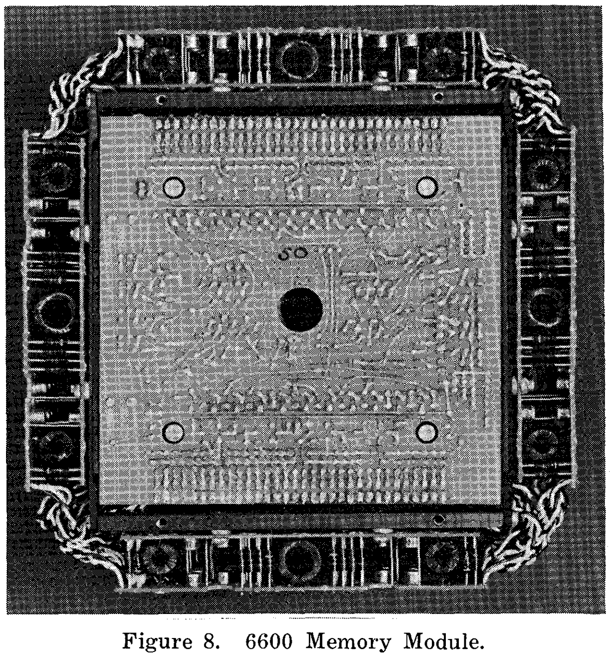


Figure 8. 6600 Memory Module.

Logic modules and memory modules are held in upright hinged chassis in an X shaped cabinet (Fig. 9). Interconnections between modules on the chassis are made with twisted pair transmission lines. Interconnections between chassis are made with coaxial cables.

Both maintenance and operation are accomplished at a programmed display console (Fig. 10). More than one of these consoles may be included in a system if desired. Dead start facilities bring the ten peripheral processors to a condition which allows information to enter from any chosen peripheral device. Such loads normally bring in an operating system which provides a highly sophisticated capability for multiple users, maintenance, and so on.

The 6600 Computer has taken advantage of certain technology advances, but more particularly, logic organization advances which now appear to be quite successful. Control Data is exploring advances in technology upward within the same compatible structure, and identical technology downward, also within the same compatible structure.

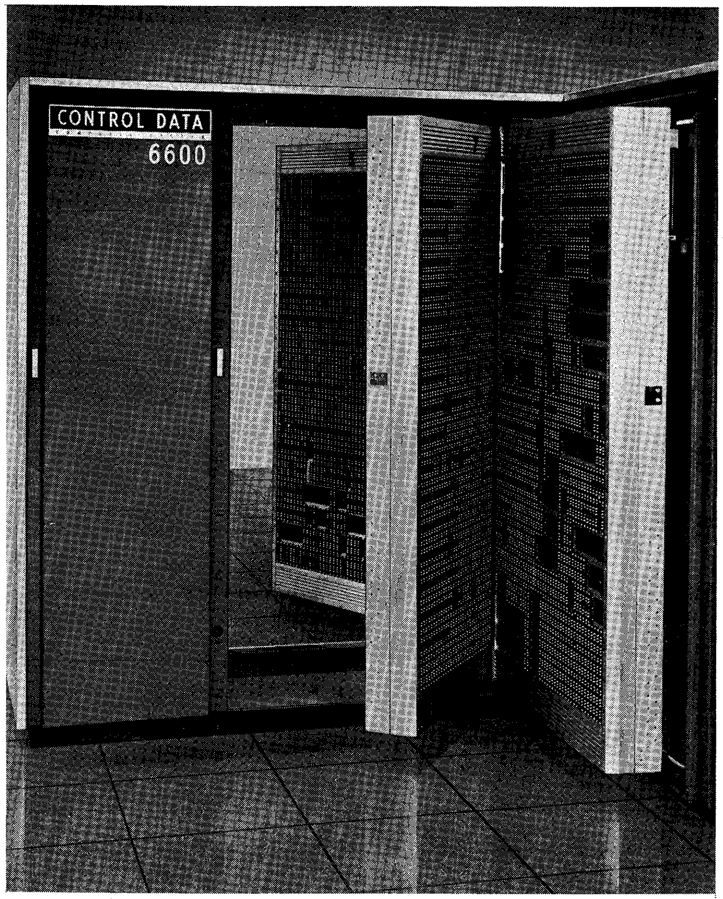


Figure 9. 6600 Main Frame Section.

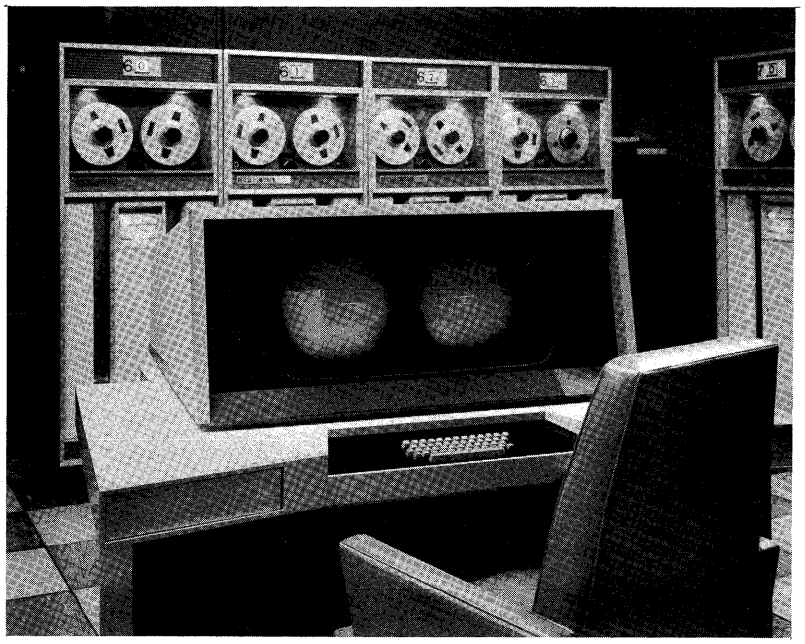


Figure 10. 6600 Display Console.

AN OPERATING SYSTEM AND PROGRAMMING SYSTEMS FOR THE 6600

B. B. Clayton, E. K. Dorff, and R. E. Fagen
Control Data Corporation
Minneapolis, Minnesota

1. INTRODUCTION

As has been seen from the discussion of the 6600 organization, the hardware design leaves a flexible arrangement, and communication between the 10 peripheral processors (PP's) with the 12 I/O channels and with central memory has few built-in hardware restrictions. Similarly, the rules for the central processor (CP) are also simple but relatively unrestricted, with the CP operation and communication with central memory subject to control of any of the 10 PP's at any time. Thus a complete discussion of operation of the 6600 system is possible only in the context of an operating system. There are many ways in which an operating system may be organized; each different way leads in effect to a different overall system when considered in the light of the processing of a given work load or problem mix.

This paper describes SIPROS, Control

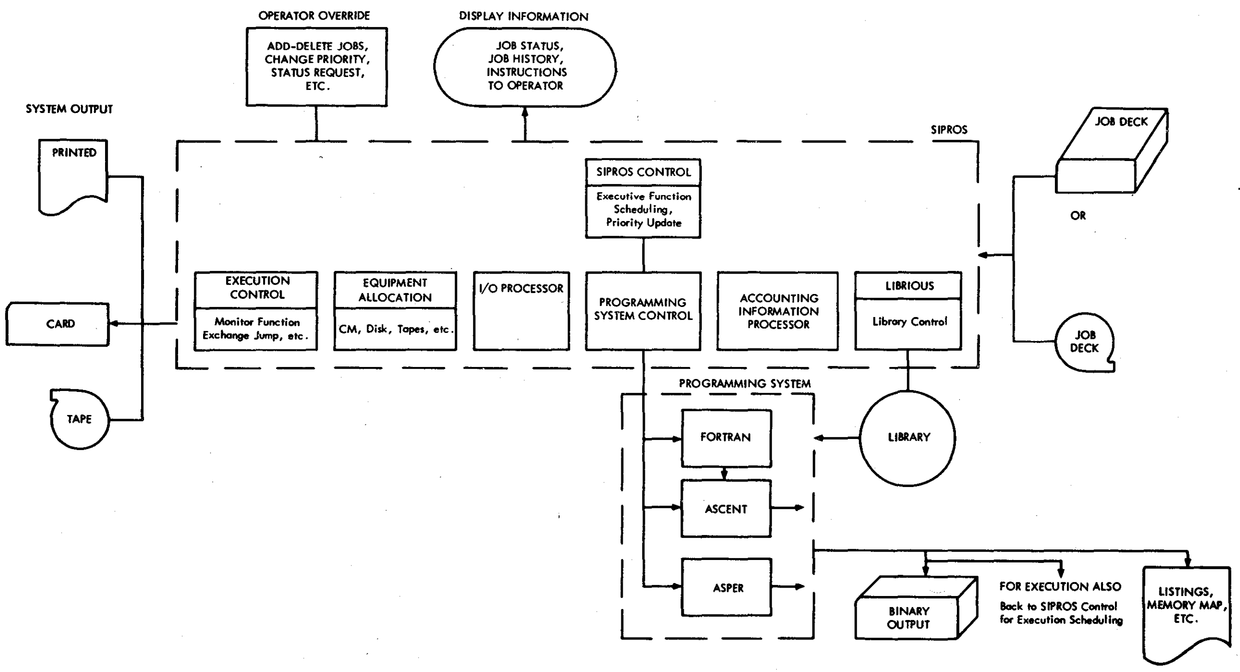

OPERATOR OVERRIDE
ADD-DELETE JOBS, CHANGE PRIORITY, STATUS REQUEST, ETC.
DISPLAY INFORMATION
JOB STATUS, JOB HISTORY, INSTRUCTIONS TO OPERATOR
SYSTEM OUTPUT
PRINTED
CARD
TAPE
SIPROS
SIPROS CONTROL
Executive Function Scheduling, Priority Update
EXECUTION CONTROL
Monitor Function Exchange Jump, etc.
EQUIPMENT ALLOCATION
CM, Disk, Tapes, etc.
I/O PROCESSOR
PROGRAMMING SYSTEM CONTROL
ACCOUNTING INFORMATION PROCESSOR
LIBRIOUS
Library Control
JOB DECK
OR
JOB DECK
PROGRAMMING SYSTEM
FORTRAN
ASCENT
ASPER
LIBRARY
BINARY OUTPUT
FOR EXECUTION ALSO
Back to SIPROS Control for Execution Scheduling
LISTINGS, MEMORY MAP, ETC.

Figure 1.

Data Corporation's standard operating system for the 6600. Since a complete description would fill a number of detailed manuals, this discussion is limited to a broad outline of those features that influenced the design of the system from various points of view. These viewpoints are loosely categorized as "what the 6600 hardware sees," "what the programmer sees," "what the using installation and operator sees," and "what a job (or sequence of jobs) sees." While these four points of view are interrelated, a clear picture of the total system is best presented as though these are four different topics, and this description of SIPROS, is organized in that order.

Control Data's standard programming system is a single package under SIPROS control consisting of a FORTRAN 66 compiler, ASCENT (assembly system for the CP), ASPER (assembly system for the PP's) and a library system of mathematical, utility, and I/O routines used by all of the systems. A brief description of these packages is also included. Figure 1 gives a block diagram of the relationship of the parts of SIPROS and the programming systems.

Finally, a short discussion of system speed and throughput, and a typical example is given. These are intended more as an indication of what must be considered in compiling or measuring these quantities than as definite answers in themselves. The use of super computer systems and "multiple processing" are now about to pass from paper to hardware; existence of running standard hardware in actual operational environments will provide the necessary "laboratories" in which quantities such as throughput can be empirically determined. This in itself will soon lead to more meaningful criteria and "problem mixes" than now exist for measuring and evaluating super computer systems.

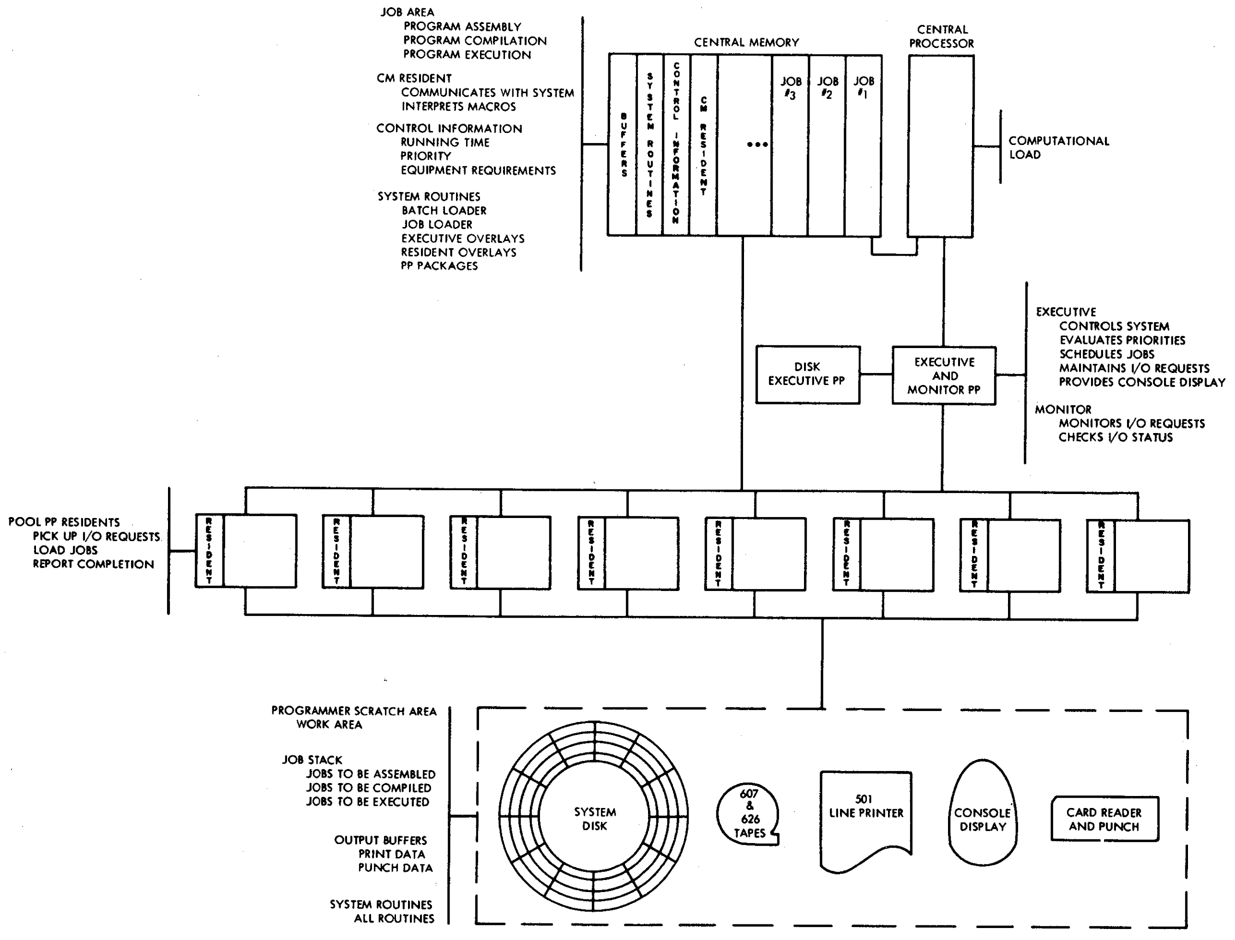


Figure 2.

2. SIPROS—OPERATING SYSTEM FOR THE 6600

Control Data's objectives in its standard operating system (SIPROS) have been to provide an efficient and yet widely applicable operating system. In order to do this, and still provide the possibility for individual installations to add to and tailor the system to their unique problem mixes, considerable attention has been given to making the system open-ended, and to leaving as parameters as many of the system "trade-offs" as possible. To describe these, we have chosen to present the system from several points of view.

From the Hardware Point of View

During operation, parts of SIPROS actually reside in different portions of the 6600 system. Figure 2 illustrates where some of these parts are normally located and what their functions are. The executive and monitoring role of SIPROS is actually contained in one of the PP's. It is responsible for the control and management of all the other parts of the system, including allocation of central memory, tasks assigned to the other peripheral processors, and allocation of and communication with peripheral equipment in the system. It should be understood at the outset that the 10 peripheral and control processors play exactly that role during normal operations; that is to say, although they appear to the programmer as independent computers in their own right, their normal assignments are for control of the system and for I/O buffering and processing. Exceptions can be made to this, however, in that any of the peripheral processors and peripheral equipment can actually be removed from direct control of SIPROS and assigned to individual jobs. Thus, if desired, the peripheral processors can share portions of the computational load, and this is an easily obtainable programmer's option. However, it is not intended as the normal mode to be described here.

The PP running the system performs all of the executive and monitoring functions. It watches the status of the job currently in execution in central memory and every 200 μ sec checks for changes in status. In addition, it keeps track of the status of central memory, all the jobs currently in central memory or in the job stack on the disk, and of the status and availability of the other peripheral processors and peripheral equipment. When specific system or I/O operations must be performed such as job loading, reading or writing of the disk, input or output from the tapes or card equipment, etc., it directs one or more of the peripheral processors to perform this operation and re-assumes control of the PP so assigned once the task is complete.

Typical of the tasks performed by the peripheral processors are the following:

1. disk executive and console display driver
2. card reader to disk
3. job loader
4. disk to printer
5. disk to tape, tape to disk
6. disk to printer, punch
7. card to tape, tape to card

Although any of the PP's may be assigned to any of the tasks described, normal practice is for the executive to watch the work load by category of operation and keep individual PP's on the same type of task as long as possible to avoid loading and unloading of the necessary PP routines. For example, a tape handling package may be loaded in an individual PP and this PP kept on tape handling tasks for some time. Should the number of demands for tape operation increase, several PP's may be semi-permanently allocated to this task. A very important feature of the system is the use of the disk which, in normal operation, will be kept busy almost all the time. To optimize use of the disk, a disk handling system consisting of routines for a PP disk executive and two disk slaves will normally be loaded into three PP's. Finally, such functions as card reading and job loading and driving the printer or printers from the disk may also be semi-permanently loaded in appropriate PP's. The point to remember, however, is that these assignments are flexible and may be altered dynamically by the system as fluctuations in the types of demands occur.

The disk is used for different functions as shown in Fig. 3a. These are:

1. storage of the library and programming systems
2. buffer area for all output to printer, etc.
3. scratch area for programmer and system use
4. storage of the job stack

Considerable attention has been paid to optimizing the use of the disk. A disk executive routine, permanently assigned to the same PP used for the console display driving package, serves to process all requests involving reading or writing the disk. Two disk "slave" PP's under control of the disk executive PP then take turns, in cooperative communication with each other, in writing (or reading) information out consecutive sectors of the disk and reading (or writing) information into central memory. Also, records are organized on the disk in a pre-

SYSTEM DISK FUNCTIONS

PROGRAMMER SCRATCH AREA
  WORK AREA FOR JOBS AND SYSTEM

JOB STACK
  JOBS TO BE ASSEMBLED
  JOBS TO BE COMPILED
  JOBS TO BE EXECUTED

OUTPUT BUFFERS
  PRINT DATA
  PUNCH DATA

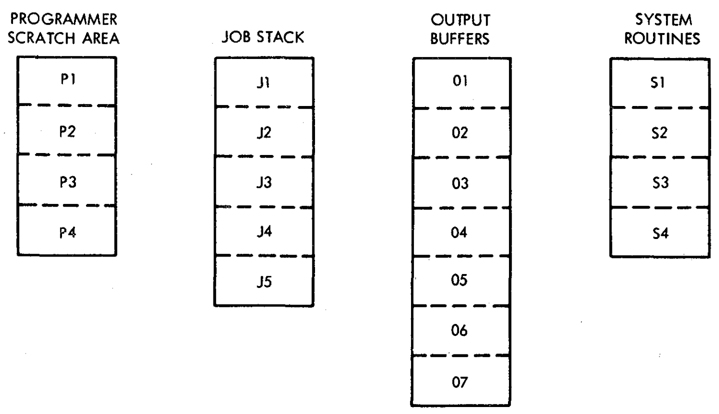
SYSTEM ROUTINES
  ALL ROUTINES

VARIABLE LENGTH LOGICAL RECORDS

EACH OF THE FOUR AREAS IS BROKEN DOWN
  INTO VARIABLE LENGTH LOGICAL RECORDS

A LOGICAL RECORD CONSISTS OF ONE OR MORE
  PHYSICAL RECORDS

PHYSICAL RECORD LENGTH IS AN INSTALLATION
  PARAMETER AND MAY BE 512 TO 4096 60-BIT WORDS



VARIABLE LENGTH LOGICAL RECORDS ARE BROKEN
  DOWN INTO PHYSICAL RECORDS ON DISK

PHYSICAL RECORDS FOR THE FOUR DIFFERENT AREAS
  ARE INTERMIXED ON DISK

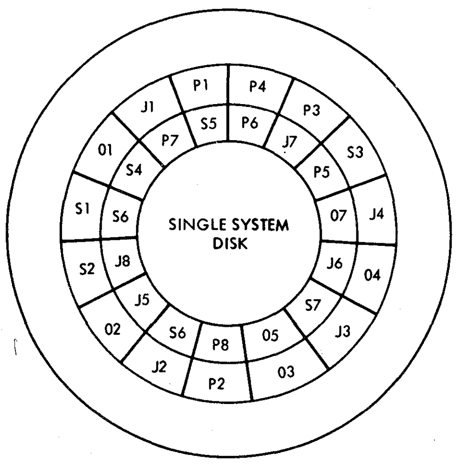
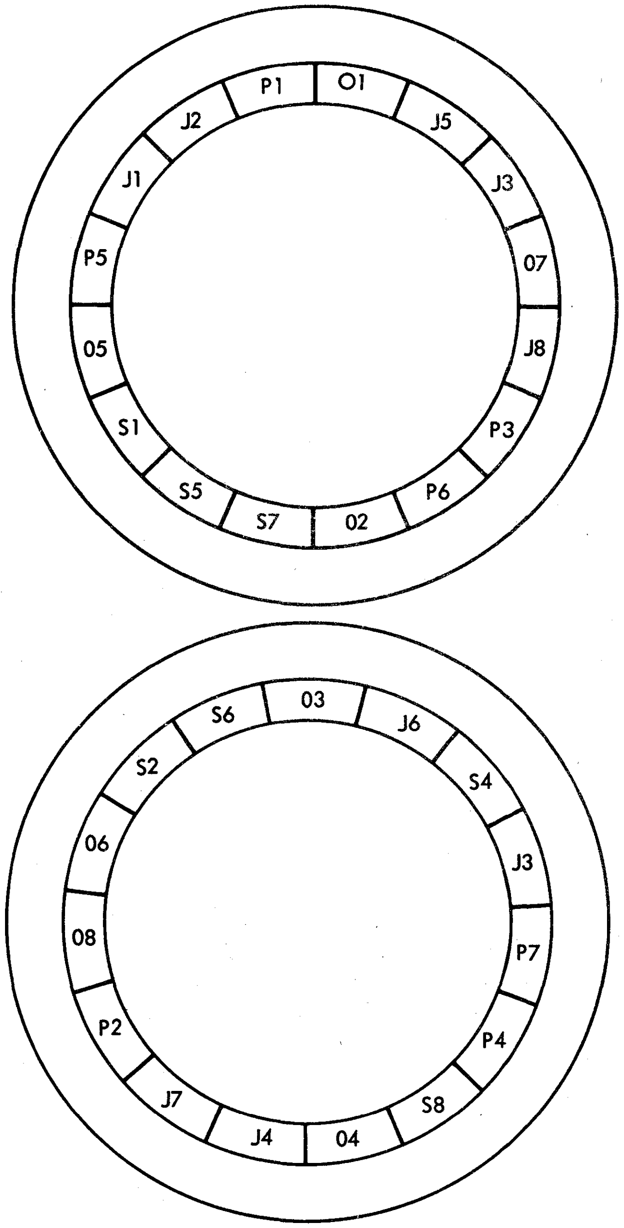


Figure 3a.



VARIABLE LENGTH LOGICAL RECORDS ARE BROKEN DOWN INTO PHYSICAL RECORDS

PHYSICAL RECORDS FOR EACH LOGICAL RECORD ARE WRITTEN ON BOTH DISKS

PHYSICAL RECORDS FOR THE FOUR DIFFERENT AREAS ARE INTERMIXED ON DISK

Figure 3b.

scribed manner, and a set of record tables and chaining information is stored in directory form in central memory and also in certain disk positions. The disk executive uses these tables to minimize arm repositioning, and "look ahead" through the request lists to allow as much information, per revolution, to be transferred as possible. The disk executive is written in such a way as to allow for a second disk and automatically increase efficiency by overlapping arm repositioning and reading of tables from one disk with I/O operations on the other. A more detailed description of these operations is beyond the scope of this paper. Figures 3a and 3b illustrate the type of organization of physical records used.

In normal operation the system maintains control of and assigns appropriate peripheral equipment as required. In addition, it notifies the operator of special requests such as mounting or dismounting of tapes and assignment of physical units to jobs. The system is written in a parametric form so that the number of peripheral items is unimportant so long as certain minimum requirements are met. For example, one disk is necessary, but no assumption is needed on the number of tapes present. (A minimum of one tape unit is assumed for disk I/O overflow.) Having two disks requires no system change, but results in a throughput gain. The system can run equally well on the 131K or 65K central memory versions of the 6600. Certain nominal values assumed by the system are actually parameters left to the discretion of the using installation. For example, the allocation of disk space to the four types of data mentioned is a parameter. The nominal choice of 512 central memory words for disk physical records and the size of I/O buffers attached to individual jobs in central memory is another parameter, as is the amount of space in central memory reserved for system tables and other status information used by the system.

From Programmer's Point of View

Problem Programming. While the 6600 is a multi-processing computer system, the intention in the design of SIPROS and the programming systems is to make the machine appear to the programmer as a traditional serial machine. Normally a programmer who is programming in either ASCENT (central processor assembly language) or FORTRAN is completely unaware of the fact that there are peripheral processors performing his I/O, and he does not explicitly program them; neither is he aware of the specific use of disks or tapes in his scratch work, and he requests scratch operations in terms of macros defining logical records and logical tape units. Programming of I/O operations is done through system macros which are handled by the operating system and these, of course, call the appropriate peripheral processor routines into play and are of no concern to the programmer. Similarly, checking channel status and

CONTROL CARDS

(*REQUIRED CONTROL CARDS)

JOB IDENTIFICATION
- * JOB NAME AND ACCOUNT NUMBER
- PRIORITY
- CENTRAL PROCESSOR RUNNING TIME LIMIT

EQUIPMENT
- SCRATCH TAPE
- INPUT TAPE
- OUTPUT TAPE
- PRINTER
- DISK
- CARD READER
- CARD PUNCH
- PERIPHERAL PROCESSOR
- VARIATIONS
  - VARIABLE vs FIXED REQUIREMENTS
  - EQUIPMENT EXCHANGE
  - SPECIFIC ASSIGNMENT

MEMORY ESTIMATE
- CENTRAL MEMORY
  - FIXED
  - VARIABLE
- DISK MEMORY
  - FIXED
  - VARIABLE

DEBUGGING
- MEMORY DUMP
- MEMORY MAP
- CONSOLE DEBUGGING
- ERROR HALT CONDITIONS

OTHER
- IGNORE EXPONENT OVERFLOW
- IGNORE INDEFINITE RESULT
- IGNORE EXPONENT OVERFLOW AND INDEFINITE RESULT
- COMPILE PROGRAM
- * FINIS

CARD DECK LAYOUT

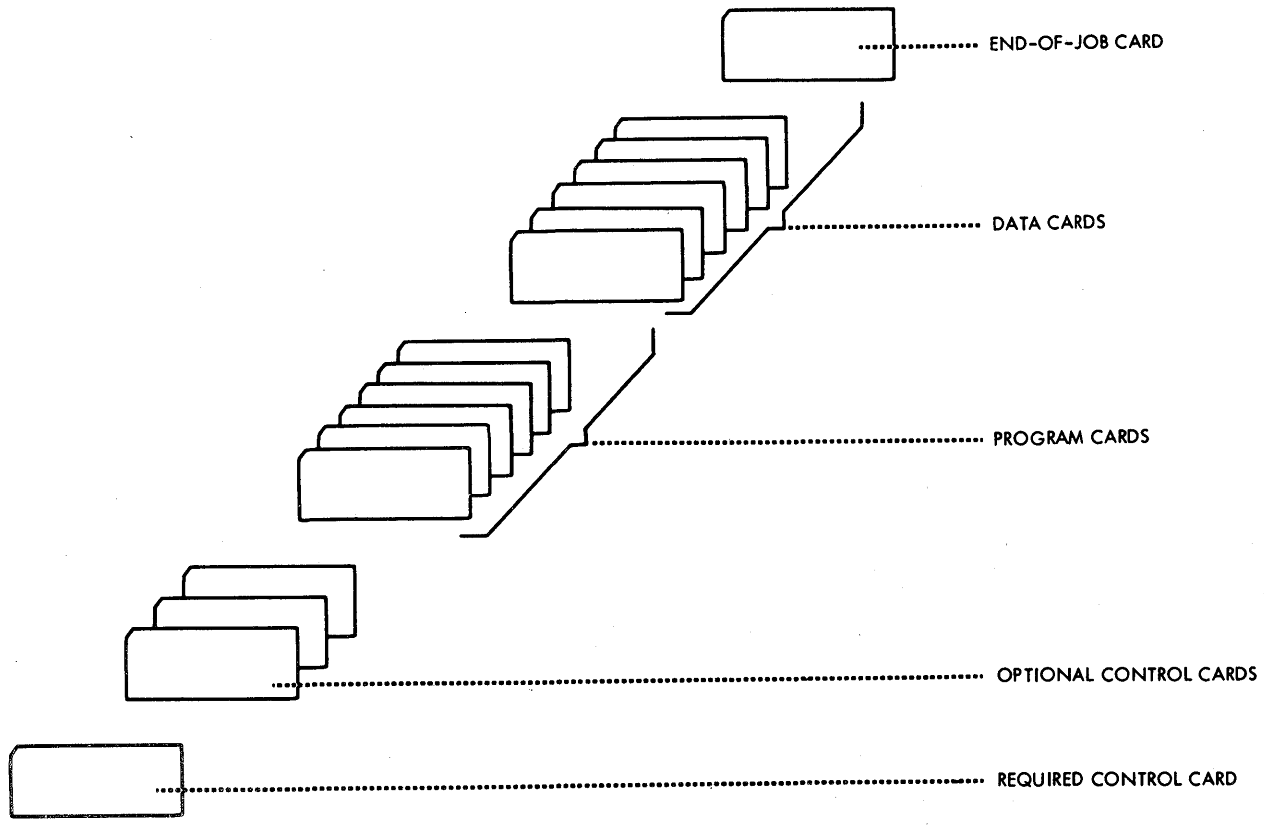


Figure 4a.

peripheral equipment status is done through the use of system macros.

While a great number of other programs will occupy the 6600 system during the execution of a programmer's problem, his own programming is never concerned with this nor the possible interference with or from other programs. The system automatically takes care of questions of scheduling, loading, allocation of memory space and peripheral equipment, relocation of programs within central memory, and of protection of the various parts of the program from conflicts with other programs. This is done through operating system use of such features as the exchange jump package and field length restrictions on central memory references.

The programmer is also presented with a very simple control card format. Although he can specify a large number of conditions, estimates, and options for his own convenience (as shown in Fig. 4a), he is actually required to provide only a job identification card and an end-of-job card. The other features he can call for are of convenience either to himself (for debugging or diagnostic procedures) or for system convenience (such as level of priority, estimates of memory required, estimates of peripheral equipment requirements, etc.), but are not required. A simple system for control card provision for program segmentation is also provided. Figures 4a and 4b summarize the various possibilities for job deck organization.

Special Systems Programming. In programming certain jobs, it is necessary to write pro-

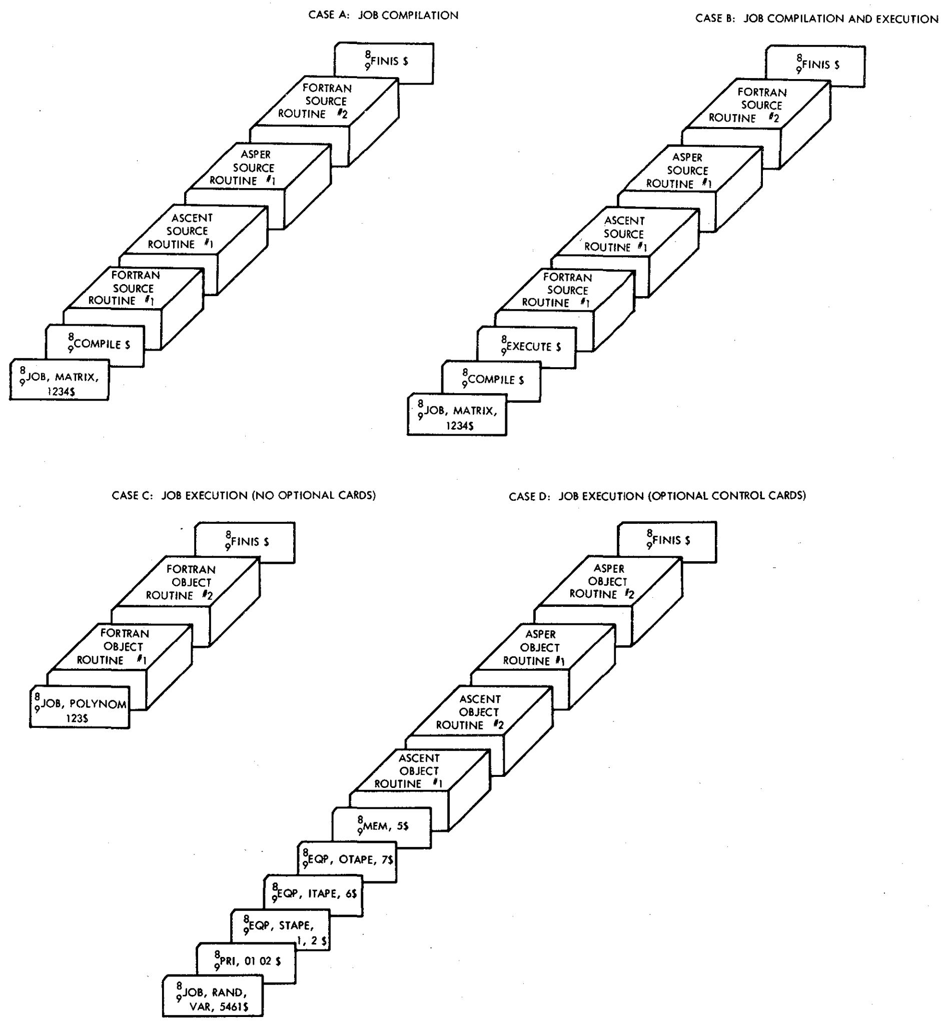


Figure 4b.

grams using peripheral processors in conjunction with the central processor. Examples are:

- real time programming, where on-line I/O must be handled.
- additions to the operating system due to novel peripheral additions (multiple remote consoles, new peripherals, etc.).
- programs where timing restrictions cause direct coordination between CP execution and I/O from a peripheral device.

SIPROS permits such special programming by allowing a programmer's job to consist of a mixture of central processor and peripheral processor routines. In addition, assembly languages for both machines are provided as one package. The programmer organizes his deck into CP routines (ASCENT) and PP routines (ASPER) through appropriate control cards. In assembly, the programming system creates the necessary communication links between these (since the programmer can use cross-referenced symbols and COMMON areas between the programs) and the SIPROS loader will automatically take care of loading the object program into allocated central memory space, and allocated PP's. The PP's allocated are removed from normal SIPROS control, and "belong" to the job during execution. The "job" itself remains under SIPROS control, and the monitor watches its status and treats the entire job as any other in the system. In this way, SIPROS is "open-ended," and a programmer using ASPER programs can actually be "adding" to the system. In many cases, this amounts to no more than writing a trivial control or driver routine for the PP with respect to a peripheral or I/O device. The mechanism for writing programs in ASPER is similar to that for writing ASCENT routines and, except for the cross-referencing of symbols, a programmer is in effect writing traditional programs for two separate and traditional machines, and the usual programmer and system macros are provided in both systems. Figure 5 shows the steps involved in real-time job multi-processing.

From the Operational Point of View

Operator. During normal operation, the operator communicates with the system through his console with keyboard and two CRT's. SIPROS (through the PP display driver) provides status information on the flow of jobs through the system, and also gives certain messages concerning necessary operator actions. The operator is provided considerable override capability in that he can introduce or delete jobs, change their priorities, cause equipment reservation, or interrogate the system for status information not routinely provided.

The operating system also runs at intervals or as an idle routine a high-level diagnostic

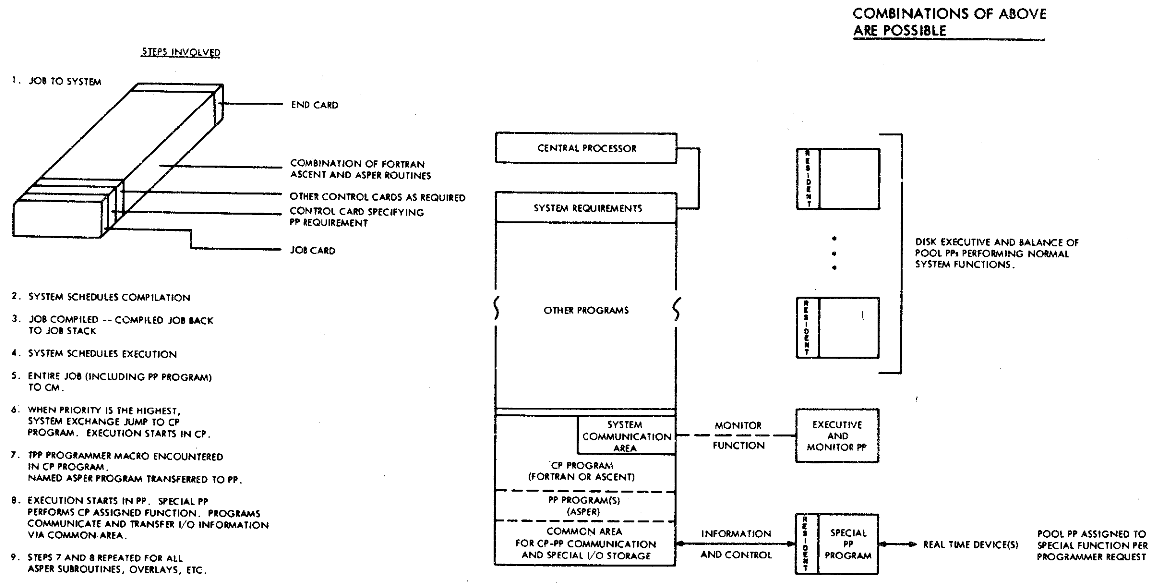

STEPS INVOLVED
1. JOB TO SYSTEM
END CARD
COMBINATION OF FORTRAN ASCENT AND ASPER ROUTINES
OTHER CONTROL CARDS AS REQUIRED
CONTROL CARD SPECIFYING PP REQUIREMENT
JOB CARD
2. SYSTEM SCHEDULES COMPILATION
3. JOB COMPILED -- COMPILED JOB BACK TO JOB STACK
4. SYSTEM SCHEDULES EXECUTION
5. ENTIRE JOB (INCLUDING PP PROGRAM) TO CM.
6. WHEN PRIORITY IS THE HIGHEST, SYSTEM EXCHANGE JUMP TO CP PROGRAM. EXECUTION STARTS IN CP.
7. TPP PROGRAMMER MACRO ENCOUNTERED IN CP PROGRAM. NAMED ASPER PROGRAM TRANSFERRED TO PP.
8. EXECUTION STARTS IN PP. SPECIAL PP PERFORMS CP ASSIGNED FUNCTION. PROGRAMS COMMUNICATE AND TRANSFER I/O INFORMATION VIA COMMON AREA.
9. STEPS 7 AND 8 REPEATED FOR ALL ASPER SUBROUTINES, OVERLAYS, ETC.
COMBINATIONS OF ABOVE ARE POSSIBLE
CENTRAL PROCESSOR
SYSTEM REQUIREMENTS
OTHER PROGRAMS
SYSTEM COMMUNICATION AREA
CP PROGRAM (FORTRAN OR ASCENT)
PP PROGRAM(S) (ASPER)
COMMON AREA FOR CP-PP COMMUNICATION AND SPECIAL I/O STORAGE
RESIDENT
RESIDENT
DISK EXECUTIVE AND BALANCE OF POOL PPs PERFORMING NORMAL SYSTEM FUNCTIONS.
MONITOR FUNCTION
EXECUTIVE AND MONITOR PP
INFORMATION AND CONTROL
RESIDENT
SPECIAL PP PROGRAM
REAL TIME DEVICE(S)
POOL PP ASSIGNED TO SPECIAL FUNCTION PER PROGRAMMER REQUEST

Figure 5.

check of the hardware operation, and reports results to the operator; the operator can call this job at any time. Features allowing him to examine selected portions of memory are also included. Figures 6a and 6b give diagrams of typical communications. Since most status information is dynamic, the system keeps a permanent record of job history, which the operator can call at any time.

Installation. A thorough set of use information, broken down by job, is provided for each of the parts of the 6600 system used by the job, including maximum central memory use, CP

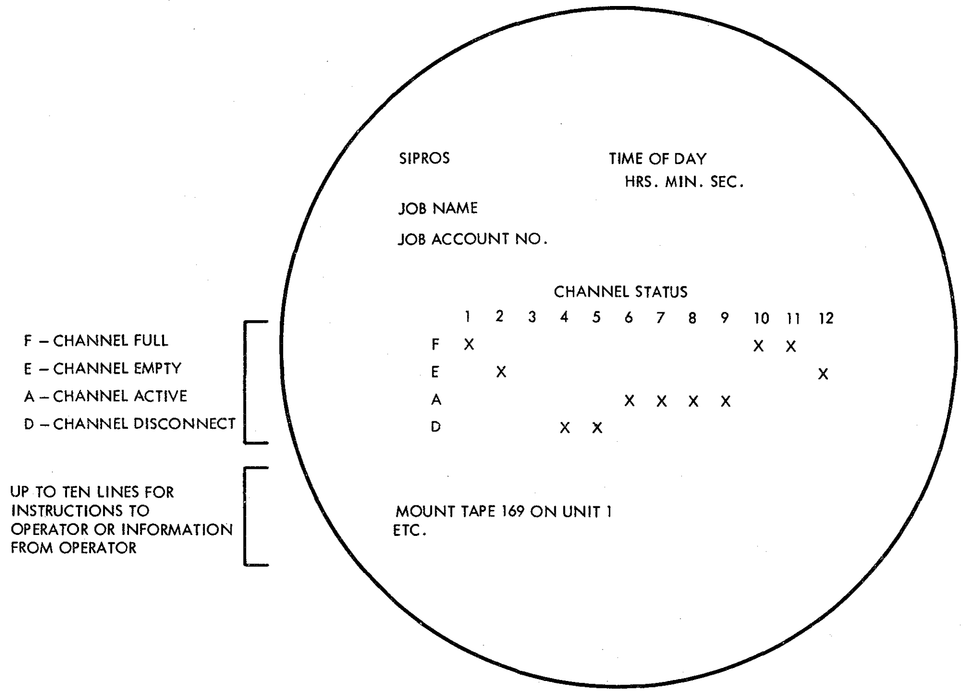


Figure 6a.

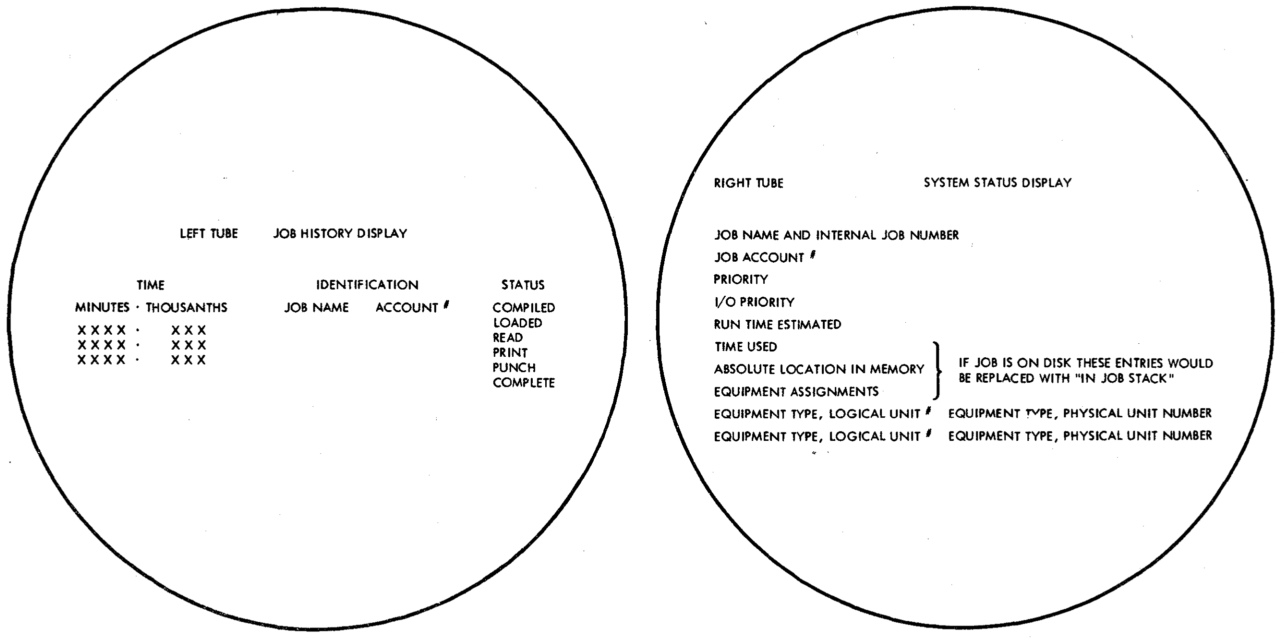


Figure 6b.

elapsed time, and time of peripheral equipment usage.

From installation to installation, the actual mix of work or job load will vary considerably as could the system throughput. To facilitate adjustment, SIPROS allows certain of its parameters to be changed by the installation. These parameters relate mostly to the trade-offs between time and memory space. Figure 7 gives a list of these options and parameters.

Another important installation choice is the use and regulation of the priority system. This is of particular importance in a multi-input system, where jobs entering the system could come in off-line, on-line, or from a number of sources with varying priority requirements. To provide each installation with a wide range of choice, keep the priorities simple, and still avoid certain bottlenecks and logical "lockouts" possible in any queuing system, SIPROS provides a priority system which itself is parametric, and which is divided into three distinct priority "classes." The first class has a number of priority levels; one of these is fixed with the job, is part of its loading criteria, and does not change so long as the job is in the system. The second or intermediate class has an equal number of priority levels. However, these are further partitioned by the system internally once the job enters the system, and the levels are increased incrementally at time intervals. The time interval is an installation parameter. Finally, there is a third class of priority which consists of a single priority level higher than any possible in the other two classes. This priority is included to handle the case of a job with real time requirements; when a job of this priority is in execution, no other job can gain central processor control through internal build-up of its own initial priority. Figures 8a, 8b, and 8c show the relationship between these classes.

LIST OF INSTALLATION PARAMETERS

1. <u>CENTRAL MEMORY</u>
   - A. MEMORY SIZE 131 OR 65K
   - B. TOTAL SYSTEM REQUIREMENTS
   - C. I/O BUFFER SIZE (512 CM WORD MINIMUM)
   - D. SPACE ALLOTTED TO SYSTEM ROUTINES
   - E. SPACE ALLOTTED TO TRIAL LOAD OF JOB IF MEMORY ESTIMATE NOT SPECIFIED
   - F. SIZE OF JOB TABLE AREA
2. <u>DISK</u>
   - A. SPACE ALLOTTED TO LIBRARY FUNCTIONS
   - B. SPACE ALLOTTED TO OUTPUT AREA
   - C. SPACE ALLOTTED TO JOB STACK
   - D. SPACE ALLOTTED TO PROGRAMMER "SCRATCH" AREA
3. <u>PROGRAMMING SYSTEM</u>
   - A. SYMBOL TABLE SIZE
   - B. TEMPORARY STORAGE REGION
   - C. PROGRAMMER MACRO STORAGE
4. <u>OPERATION</u>
   - A. INSTALLATION PRIORITY STANDARD
   - B. CENTRAL PROCESSOR EXECUTION TIME LIMIT
   - C. PRINT OUTPUT LIMIT. JOB EXCEEDING THIS LIMIT WILL HAVE ALL PRINT OUTPUT ON TAPE
   - D. PROGRAMMER DISPLAY TIME LIMIT
   - E. SYSTEM BALANCE PARAMETERS. TAILORS PP USAGE TO INSTALLATION NEEDS.

Figure 7.

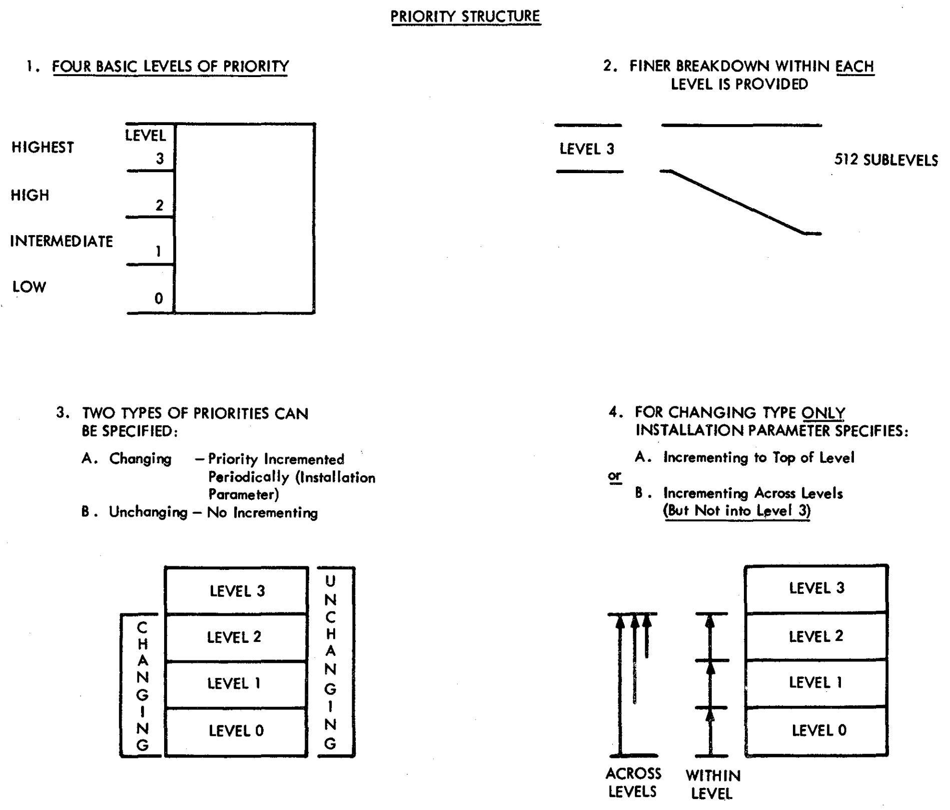

PRIORITY STRUCTURE
1. FOUR BASIC LEVELS OF PRIORITY
HIGHEST
HIGH
INTERMEDIATE
LOW
LEVEL 3
2
1
0
2. FINER BREAKDOWN WITHIN EACH LEVEL IS PROVIDED
LEVEL 3
512 SUBLEVELS
3. TWO TYPES OF PRIORITIES CAN BE SPECIFIED:
A. Changing – Priority Incremented Periodically (Installation Parameter)
B. Unchanging – No Incrementing
4. FOR CHANGING TYPE ONLY INSTALLATION PARAMETER SPECIFIES:
A. Incrementing to Top of Level
or
B. Incrementing Across Levels (But Not into Level 3)
CHANGING
LEVEL 3
LEVEL 2
LEVEL 1
LEVEL 0
UNCHANGING
ACROSS LEVELS
WITHIN LEVEL

Figure 8a.

Installations are free to use any or all three classes of priority, and as many of the levels within each class as they choose. Each of the classes, and the idea of incrementing priorities, was included to allow solution to certain possible problems. The second class guarantees that a job in the job stack will eventually get loaded into central memory and once there will eventually get executed. Thus a top level priority job with all computation and no I/O could not prevent lower priority jobs with little computation but considerable I/O to get into execution and thereby to contribute to overall system utilization. Existence of the lowest class allows entry of low priority jobs that at worst occupy disk space, but where there is genuinely no concern over when they get run. Finally, as mentioned, the "real-time" class allows a real-time requirement to fit with other high priority but non-real time jobs in the system. Figures 8a, 8b, and 8c show in more detail the relations between priority levels and classes.

Flow of Jobs Through the System

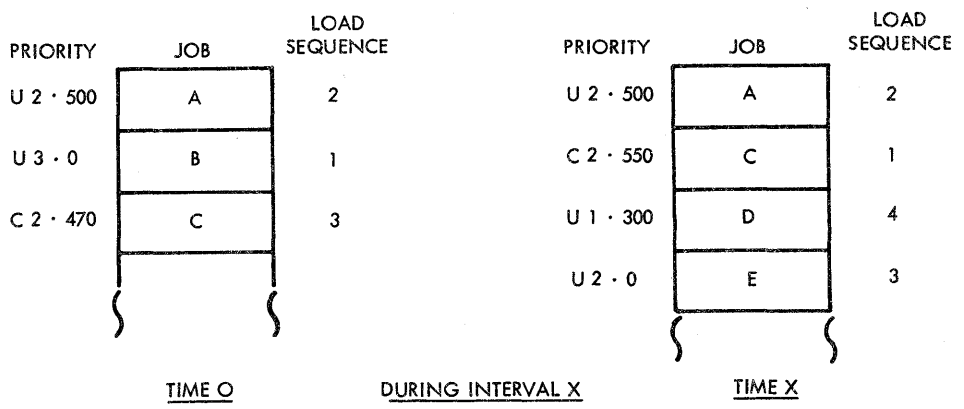
Normal Computer Center. During normal operations, jobs enter the system either through the card reader or from magnetic tape with information stored in card image format. As soon as space becomes available in the section of the disk allocated to storage of the job stack, the system assigns a peripheral processor to read the cards and enter the job onto the disk. During this operation the system extracts all the necessary information from the control cards, enters these pieces of information into its own records of job status, and assigns an internal job identification number to the job. Items logged at this time include such things as job priority, special equipment requirements, estimates of memory space requirements, whether the job is an execution or requires assembly or compilation, etc. The job stack as kept on the disk is open-ended, and whenever the system notices that disk space is available, it adds to the job stack from the card reader.

As soon as a job in the job stack meets certain criteria, the system, through the job loader, causes the job to be loaded into central memory. A diagram of the route from the job stack to central memory and a list of job loading criteria is given in Fig. 9 (parts 1 and 2). Jobs with priorities of the second class are assured to be loaded eventually since their priorities continually build up while in the stack, and once they reach the highest priority, the system will automatically reach a point where it actually reserves sufficient memory space and peripheral equipment to satisfy the job's requirement. This reservation feature is included to prohibit mutual lock-out situations where several high priority jobs could be loaded, but due to equipment or memory space requirements, none meet the criteria.

The system also keeps as many jobs as possible in central memory and dynamically reallocates memory space as jobs terminate and release space and/or peripheral equipment. The system assigns the central processor to execution on the highest priority job in central memory. As soon as another job in central memory reaches a higher priority, control—through exchange jump—is transferred to this job. When the job in execution reaches a status change such as a request for an input/output operation, appropriate flags are set in a status word in that program's area and the system monitor during its normal cycle will notice this status change. I/O requests will be of one or two kinds as designated by the programmer in his system macro. Either the

SIPROS PRIORITY HANDLING

1. IN JOB STACK



DURING INTERVAL X

A. ONLY JOB B LOADED TO CM
B. JOB C PRIORITY INCREMENTED SUCH THAT AT TIME X IT REACHED C 2 · 550

2. IN CENTRAL MEMORY

A. JOB RUNNING IS NOT INCREMENTED
B. CHANGING PRIORITY JOBS NOT RUNNING ARE INCREMENTED PERIODICALLY. THIS COULD EFFECT SEQUENCE OF JOBS WAITING TO RUN

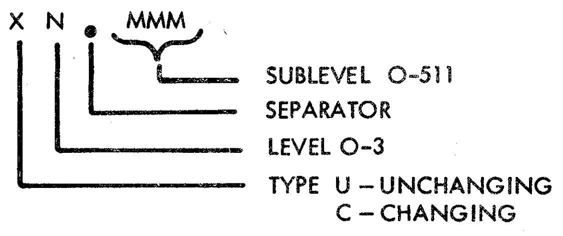
3. IN I/O STACKS

A. I/O TASKS PUT IN STACK ACCORDING TO FUNCTION
B. TASKS NOT IN OPERATION, WITH CHANGING PRIORITY, ARE INCREMENTED PERIODICALLY. THIS COULD EFFECT SEQUENCE OF WAITING TASKS.

Figure 8b.

FROM PROGRAMMER POINT OF VIEW

1. PRIORITY SPECIFICATION:



Examples: C 2 · 0, U 1 · 200, etc.

2. PROGRAMMER OPTIONS:

SPECIFY A. Job Priority and Different I/O Priority

OR B. Job Priority Only
(System Sets I/O Priority = Job Priority)

OR C. No Priority
(Systems Sets I/O = Job = Installation Standard Priority)

Figure 8c.

programmer has indicated that the I/O operation must be completed before computation can resume (non-buffered mode) or he indicates that computation can proceed while the I/O operation is taking place (buffered mode). In the first case the system will enter the I/O request on an I/O request list and will turn control, through an exchange jump, to the job of next highest priority in the system. In the second case the system will enter the I/O request in its request list and allow the program to retain control of the CP. Thus, a programmer is not really concerned with buffering his I/O operations; the system automatically provides for this. All jobs in central memory, with the exception of the one currently in execution, have their priorities periodically incremented. This assures that jobs which are mostly computation will not prevent jobs which have considerable I/O from getting into execution and will assure a more uniform use of the different parts of the hardware. As a job in execution requests additional memory space, disk space or tape units, the system automati-

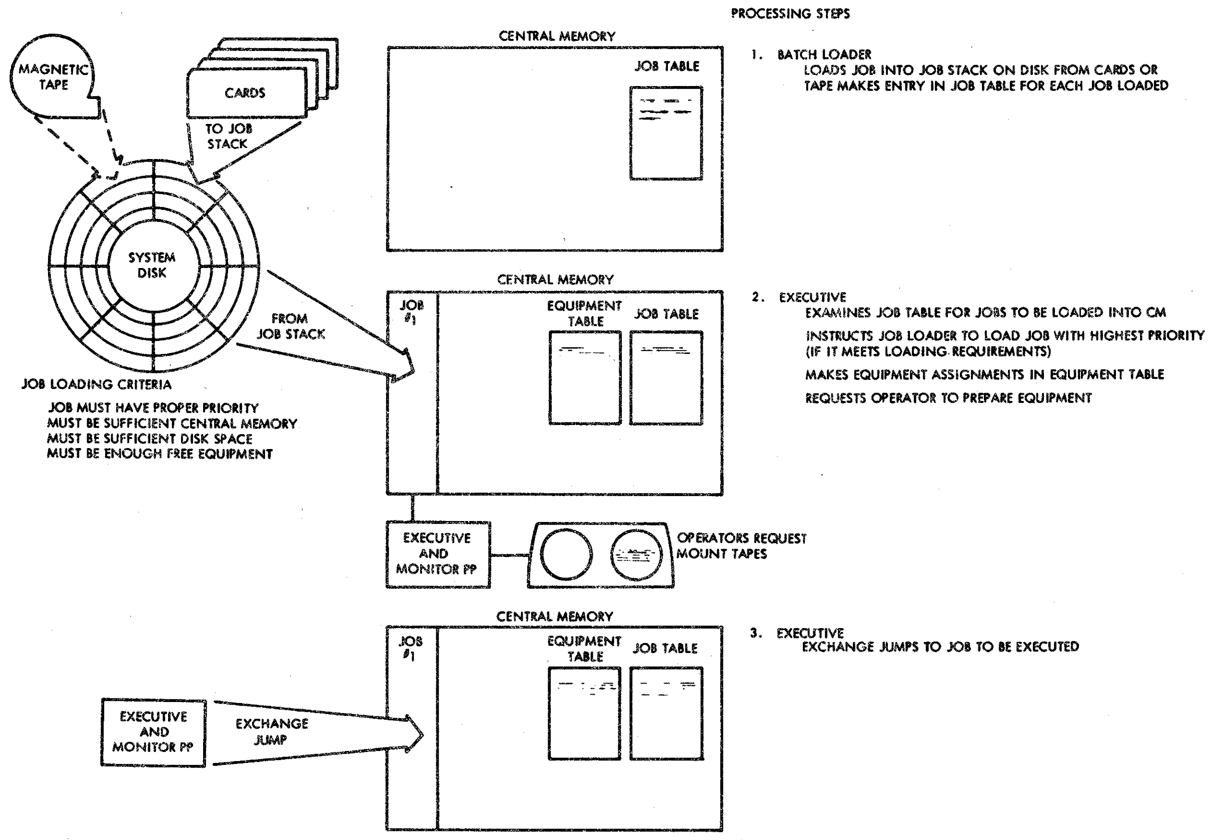

PROCESSING STEPS
MAGNETIC TAPE
CARDS
TO JOB STACK
SYSTEM DISK
CENTRAL MEMORY
JOB TABLE
1. BATCH LOADER
LOADS JOB INTO JOB STACK ON DISK FROM CARDS OR TAPE MAKES ENTRY IN JOB TABLE FOR EACH JOB LOADED
FROM JOB STACK
CENTRAL MEMORY
JOB #1
EQUIPMENT TABLE
JOB TABLE
2. EXECUTIVE
EXAMINES JOB TABLE FOR JOBS TO BE LOADED INTO CM
INSTRUCTS JOB LOADER TO LOAD JOB WITH HIGHEST PRIORITY (IF IT MEETS LOADING REQUIREMENTS)
MAKES EQUIPMENT ASSIGNMENTS IN EQUIPMENT TABLE
REQUESTS OPERATOR TO PREPARE EQUIPMENT
JOB LOADING CRITERIA
JOB MUST HAVE PROPER PRIORITY
MUST BE SUFFICIENT CENTRAL MEMORY
MUST BE SUFFICIENT DISK SPACE
MUST BE ENOUGH FREE EQUIPMENT
EXECUTIVE AND MONITOR PP
OPERATORS REQUEST MOUNT TAPES
CENTRAL MEMORY
JOB #1
EQUIPMENT TABLE
JOB TABLE
3. EXECUTIVE
EXCHANGE JUMPS TO JOB TO BE EXECUTED
EXECUTIVE AND MONITOR PP
EXCHANGE JUMP

Figure 9, Part 1.

cally assigns these. Provision is made, through macros, to allow the programmer to specify dynamic release of any of these items once no longer necessary to execution.

As a job proceeds in execution, its output is collected on the disk; once the job completes in central memory, the task of publishing the completed output is commenced as a separate "off-line" operation; this final operation, of course, is automatic and is treated, as an I/O request, as one part of the job.

Since input/output requests can build up from a variety of jobs in various states of completion in central memory, the handling of the input/output request queue is performed in a completely separate manner, and independent priorities may be assigned to these requests through control cards by the programmer. The system takes care of allocation of the required peripheral processors and peripheral equipment to satisfy these requests, and attempts to balance assignments made to the PP's by class of operation as was mentioned previously.

Multiple Inputs or Controls. In the above, the operation at a normal computer installation is described, with the operator's console in communication with the one allocated PP display driver, and with normal job entry to the job stack being the card reader or magnetic tape. As will be described in the paper on 6600 applications, SIPROS is designed to allow more than one station or network of input stations to feed jobs into the system. Figure 10 illustrates this type of situation.

The normal way of handling another input of any sort is to introduce a special job into central memory, which has communication with and is under control of SIPROS, but which has its own PP or PP's assigned and which has the appropriate ASPER routines in the PP's. These PP's then are assigned to handle the in-

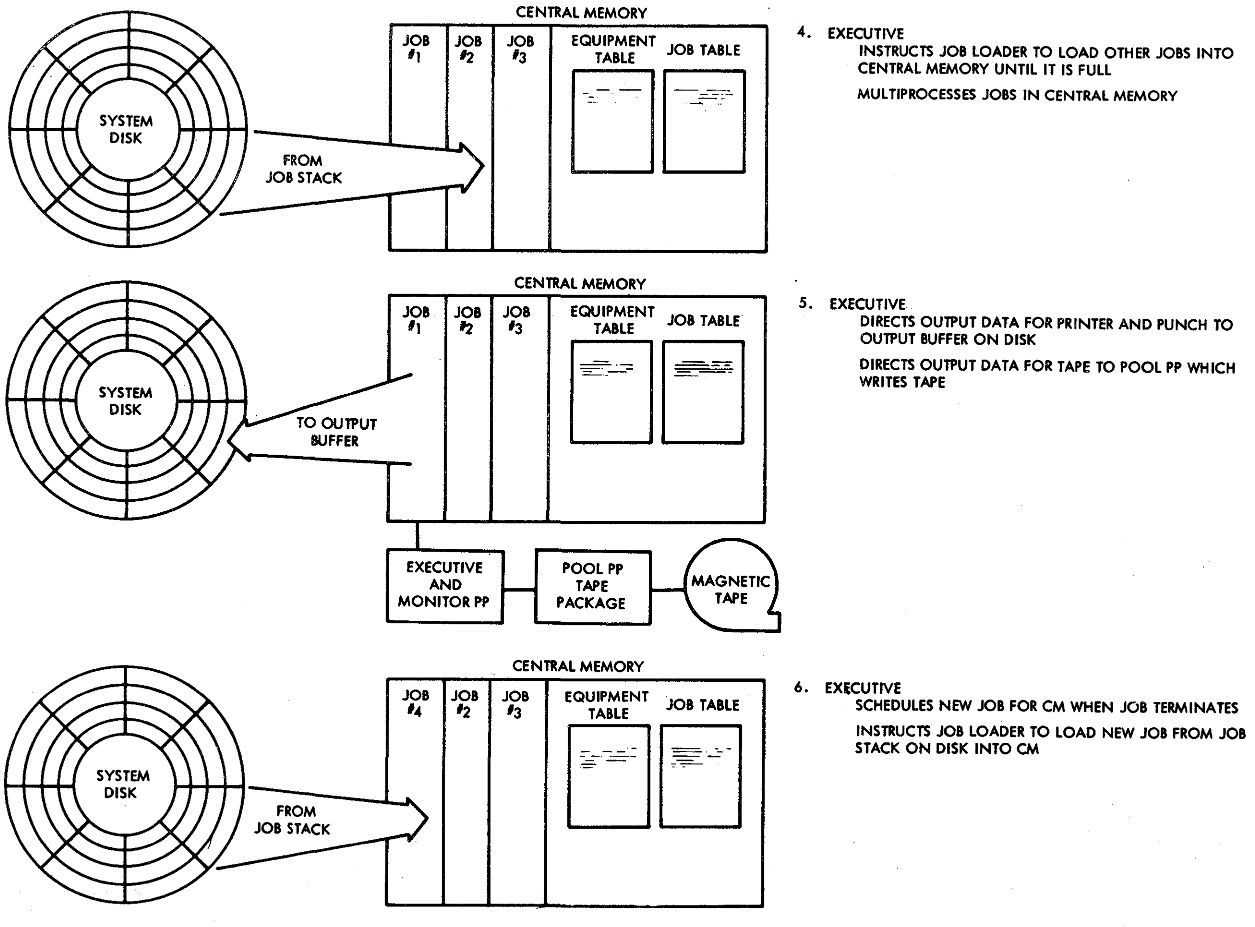

CENTRAL MEMORY
JOB #1
JOB #2
JOB #3
EQUIPMENT TABLE
JOB TABLE
SYSTEM DISK
FROM JOB STACK
4. EXECUTIVE
INSTRUCTS JOB LOADER TO LOAD OTHER JOBS INTO CENTRAL MEMORY UNTIL IT IS FULL
MULTIPROCESSES JOBS IN CENTRAL MEMORY
CENTRAL MEMORY
JOB #1
JOB #2
JOB #3
EQUIPMENT TABLE
JOB TABLE
SYSTEM DISK
TO OUTPUT BUFFER
EXECUTIVE AND MONITOR PP
POOL PP TAPE PACKAGE
MAGNETIC TAPE
5. EXECUTIVE
DIRECTS OUTPUT DATA FOR PRINTER AND PUNCH TO OUTPUT BUFFER ON DISK
DIRECTS OUTPUT DATA FOR TAPE TO POOL PP WHICH WRITES TAPE
CENTRAL MEMORY
JOB #4
JOB #2
JOB #3
EQUIPMENT TABLE
JOB TABLE
SYSTEM DISK
FROM JOB STACK
6. EXECUTIVE
SCHEDULES NEW JOB FOR CM WHEN JOB TERMINATES
INSTRUCTS JOB LOADER TO LOAD NEW JOB FROM JOB STACK ON DISK INTO CM

Figure 9, Part 2.

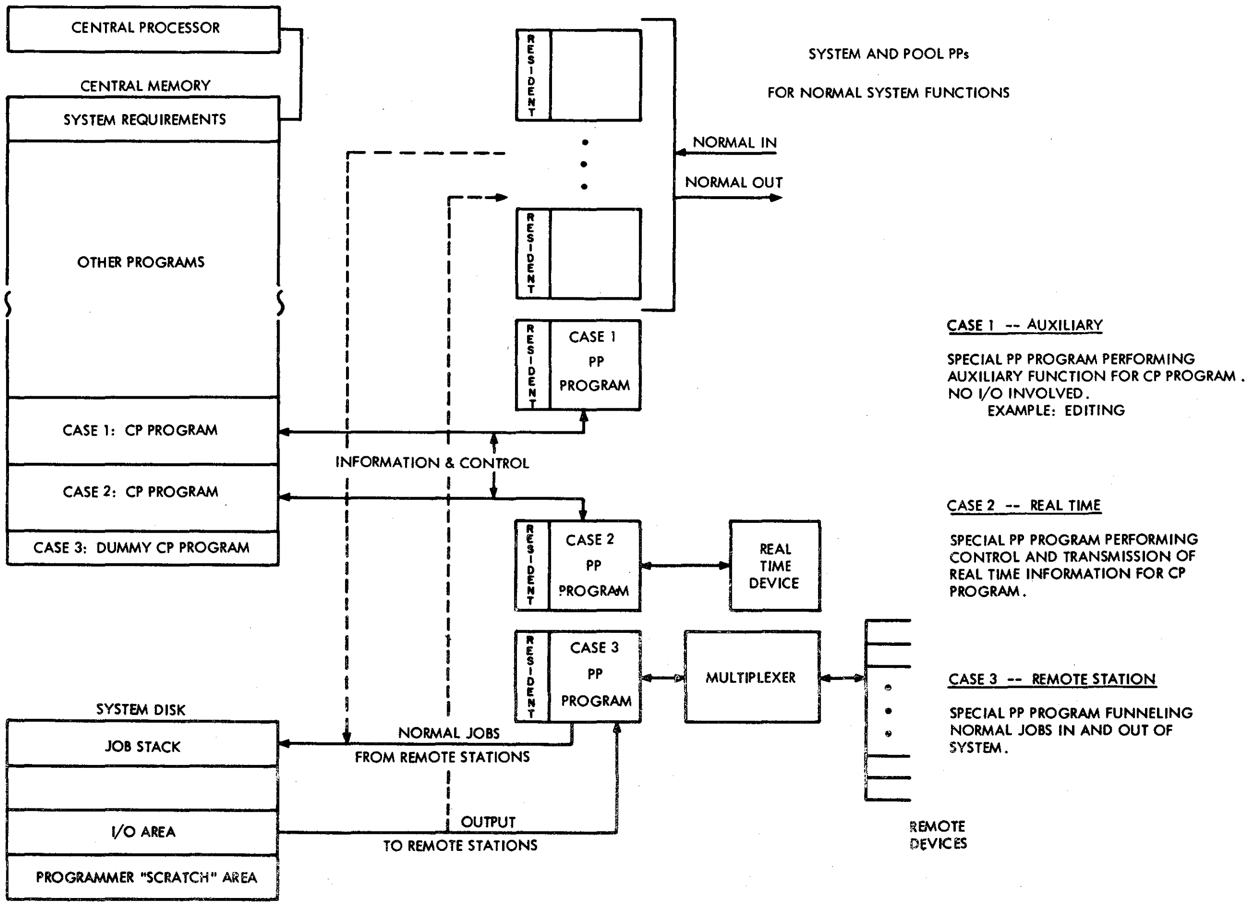


Figure 10.

put, and perform the necessary operations. For example, if the input is a set of jobs from a remote site, the PP routine would perform the necessary housekeeping to extract the required system information and enter the jobs into the job stack along with those originating "on site." If the job is a real-time computation, the central memory portion of the program will control the central processor during the necessary time periods. It should be clear from the description above that the freedom of allocation of the 10 PP's allows SIPROS to accommodate a mix of several real-time or remote input networks on a time-sharing basis.

3. PROGRAMMING SYSTEMS — FORTRAN 66, ASCENT, ASPER

The standard programming system, under SIPROS control, is a single integrated package consisting of a FORTRAN 66 compiler (FORTRAN 63 language, essentially upward compatible with FORTRAN IV) and assembly systems for the central processor and peripheral processors. Figure 11 is a hierarchy diagram of these systems. Since a program for compilation may contain a mixture of FORTRAN, ASCENT, or ASPER, the system treats the package as one single entity normally residing on the system disk. When a job requiring compilation is to be loaded in central memory, the loader loads the programming system and treats the source program as data to the programming system for the first pass of compilation. The output of the second pass of compilation is a relocatable binary version of the object program, which is routed to the job stack on the disk for later execution if this is specified on the control cards. A typical sequence for compilation and execution is shown on Fig. 11, including the inputs, outputs, and processes within the two passes.

Special FORTRAN Features

In implementing FORTRAN 66, two features aiming at flexibility and efficient object programs were provided. The first is the ability to intermix FORTRAN and ASCENT on a line-for-line basis, to reference register names in ASCENT statements, and to reference FORTRAN statement numbers in ASCENT. As a matter of fact, coding in ASCENT with intermixed FORTRAN statements and coding in FORTRAN with intermixed ASCENT statements are indistinguishable.

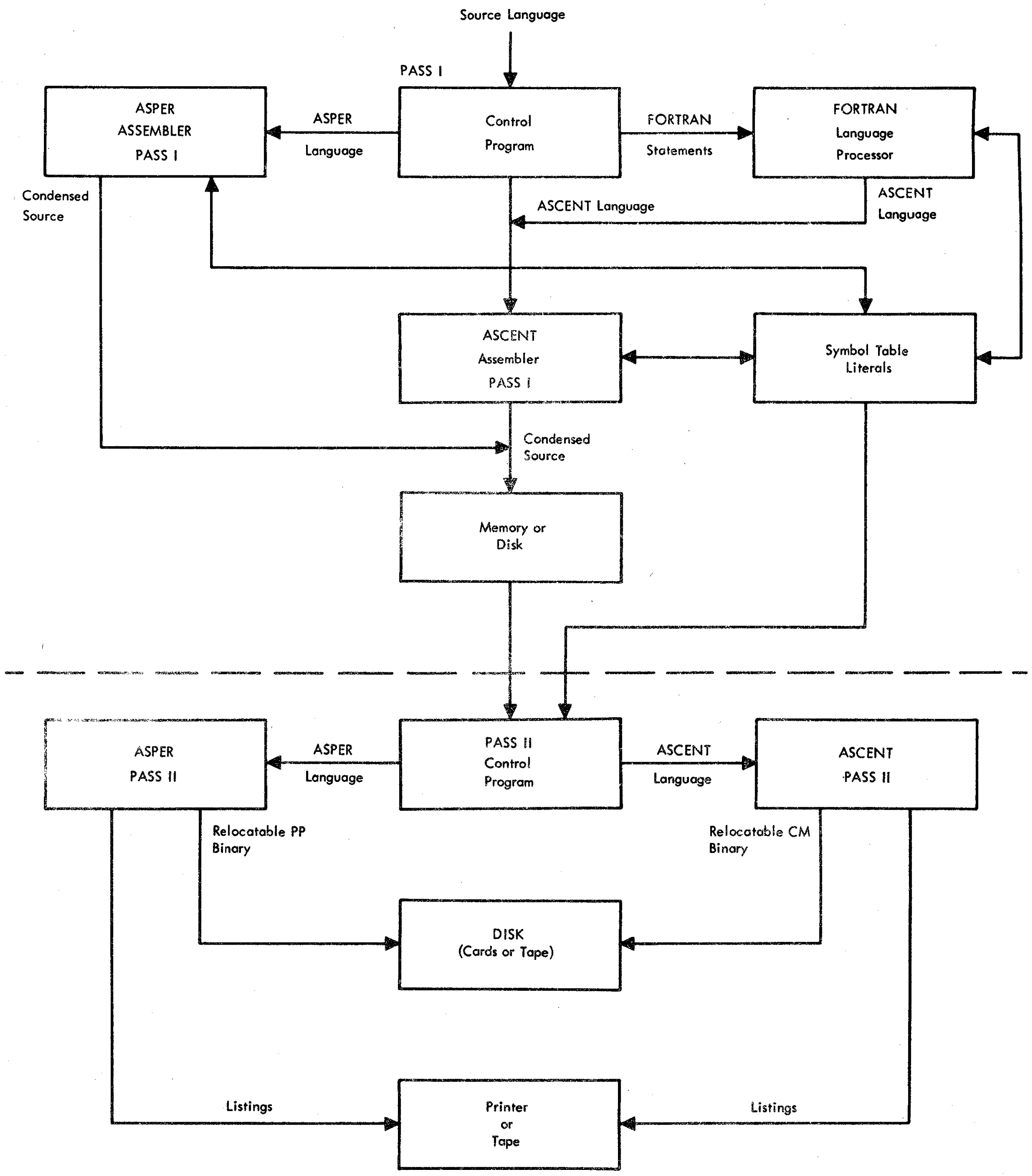


Figure 11.

The second feature in FORTRAN 66 that helps exploit the hardware design is the code generation optimization algorithm. This algorithm scans the sequence of instructions to be generated, and through simulation of the multiple function boxes and operating registers makes an attempt to optimize the code sequence by minimizing conflicts and delays that would occur due to registers busy, functional units busy, etc. Since this optimization will result in slower compilation but faster execution of object programs, a switch is provided that allows bypassing the optimization when compilation only (as in debug) is desired.

ASCENT-ASPER Communication

As has been mentioned, coding special systems that must bring peripheral processors into explicit cooperative operation with the central processor is facilitated by the communication between ASCENT and ASPER. The format for a card deck having programs with both ASCENT and ASPER programs is given in Figures 4 and 4a.

During the second pass of assembly, the programming system produces relocatable binary CM or PP programs representing the ASCENT program (for CP) and the ASPER routines (for PP). The ASPER programs are separated by control cards which represent information to the loader for use at execute time. System macros, inserted by the programmer, request the assignment of a PP at execute time, and cause the system to assign a PP (removing it from the SIPROS pool) and cause the ASPER routine to be loaded. At this time, the cooperative operation between the routines in PP and central memory can proceed, and the common reference to symbols is automatically achieved by PP routine reference to the appropriate central memory locations during execution.

4. ESTIMATES AND EXAMPLES

The throughput of a system depends on job mix, and is also affected by memory requirements of the operating system and programming system themselves. The latter are actually quite complex, since pieces of these reside in different parts of the system (as shown in Fig. 2), and also since the actual memory requirements are affected by installation options, estimates provided by job control cards, etc. However, for some rough comparison purposes, it can be estimated that the minimum central memory requirements for SIPROS itself is about 10,000 words, and that the programming system (when required for compilation or assembly) requires roughly that amount also.

Although the disk space allocation also is a variable, some typical figures might be 100,000 60-bit words for system libraries, 1,000,000 60-bit words for the job stack, 3,000,000 words for job and system output area, and the remainder (approximately 3,000,000 60-bit words) for programmer scratch area.

As an illustration of throughput gain, consider a problem requiring 10 seconds of CP time for compilation, 300 seconds for execution, and which has an aggregate total of 100 seconds of input-output interspersed with computation, during which the program in execution must wait for the completion of the I/O. Suppose further that the average central memory requirement for the job is 20,000 CM words. In such a case, the throughput time for a single run would be 310 seconds. However, if the exact same job is continuously fed, as though on an infinite "conveyor," through the 6600 and SIPROS, an average of five copies of the job could be in "execution" in central memory at once, and it can easily be seen that the average rate of throughput, per case, is 210 seconds. Although this example obviously represents a hypothetical problem and even more hypothetical mix, it is typical of the type of situation that would lead to overall throughput gain due to the 6600 multi-processing capability.

REMOTE TIME-SHARING OF A CENTRALIZED 6600

B. B. Clayton, E. K. Dorff, R. E. Fagen, and J. D. Johnson
Control Data Corporation
Minneapolis, Minnesota

The Control Data 6600, with its immense capability, lends itself readily to several new application or problem areas in which computers have not previously been fully utilized due to either technical or economic considerations. Foremost among these is the area of extremely large problems. In these problems, the size either prohibited use of existing computers in all generality or in all detail or else could be solved only at the expense of a large amount of computer time. Examples of situations of this type can be found among some nuclear and linear programming problems. In particular, when problems of this type involve real-time inputs or outputs, an extremely large computer is an absolute necessity. Another variation of this problem occurs when a special purpose black box or simulation model has to be constructed to work in conjunction with presently existing computers.

As computers become larger and faster, they are also becoming somewhat more expensive. However, the rate of increase in capability is much greater than the rate of increase in cost. For example, let us consider Fig. 1. In this figure we consider four machines along with instructions per second, cost per second, cost per 10,000 executed instructions, cost-performance ratio, and finally the cost to compile 100 FORTRAN statements. We observe that while the cost per second generally increases as the capability of the machine increases, the instructions per second increase at a much greater rate. This is particularly evident when we look at the cost per 10,000 instructions and is further reflected in the cost to compile 100 FORTRAN statements, although the first entry in that column might be tempered somewhat by inefficiencies in the compiler. The striking point to be brought out here is that computing costs are actually going down. If the 6600 can be distributed among a number of users, whose work loads are themselves insufficient to justify the high cost of the 6600, the overwhelming economic advantages of the 6600 could be fully exploited. Each user would have at his disposal

MACHINE	INSTRUCTIONS/ SECOND	COST/ SECOND	COST/ 10,000 INSTRUCTIONS	PERFORMANCE-COST RATIO	COST TO COMPILE 100 FORTRAN STATEMENTS
704	40,000	$.04	$.01	1	$16.80
7090	150,000	.09	.006	1.6	1.40
3600	400,000	.08	.002	5	.50
6600	3,000,000	.16	.0005	20	.10

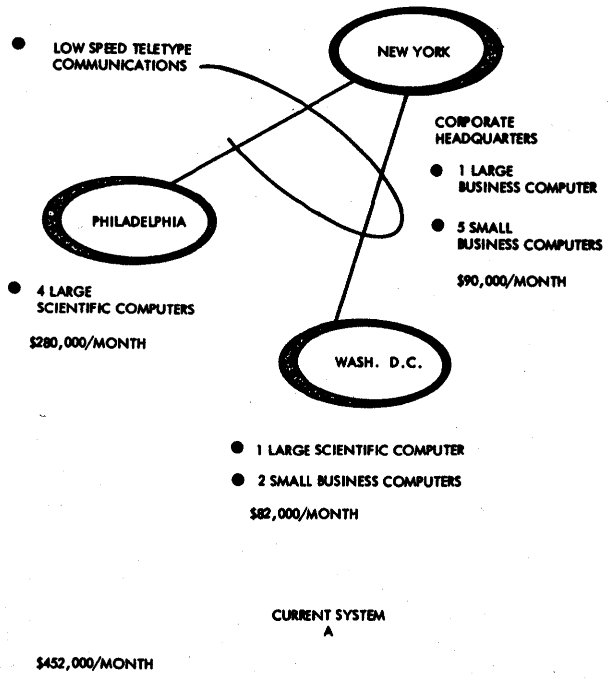
PERFORMANCE-COST TABLE

Figure 1.

large machine at a cost approaching that of present-day medium-sized systems.

The purpose of this paper is to describe various systems for distributing the computing the capabilities and powers of an extremely power of the 6600 to a number of remote users. The communications equipment necessary for this purpose, and to be described later, is presently available standard equipment. All techniques and technologies described are within the current state of the art. Also, as mentioned in the previous paper, the operating system for the 6600 (SIPROS) is organized in such a way as to allow a system with distributed inputs to operate as though the remote inputs were simple extensions of the peripheral facilities in the central facility.

To illustrate the economies inherent in a centralized system, let us look at two systems. Figure 2 shows three de-centralized facilities



LOW SPEED TELETYPE COMMUNICATIONS
NEW YORK
CORPORATE HEADQUARTERS
● 1 LARGE BUSINESS COMPUTER
● 5 SMALL BUSINESS COMPUTERS
$90,000/MONTH
PHILADELPHIA
● 4 LARGE SCIENTIFIC COMPUTERS
$280,000/MONTH
WASH. D.C.
● 1 LARGE SCIENTIFIC COMPUTER
● 2 SMALL BUSINESS COMPUTERS
$82,000/MONTH
CURRENT SYSTEM
A
$452,000/MONTH

Figure 2.

along with monthly lease costs. This might be typical of any one of a number of large users of present day electronic data processing equipment. Figure 3 indicates a different approach to solve the user's needs and provide the same or greater capabilities. The several computers have now been replaced by a single large-scale centralized computer with remote stations. These remote stations can be within the same building, same area, or up to several hundred miles away. The economic advantages are evi-

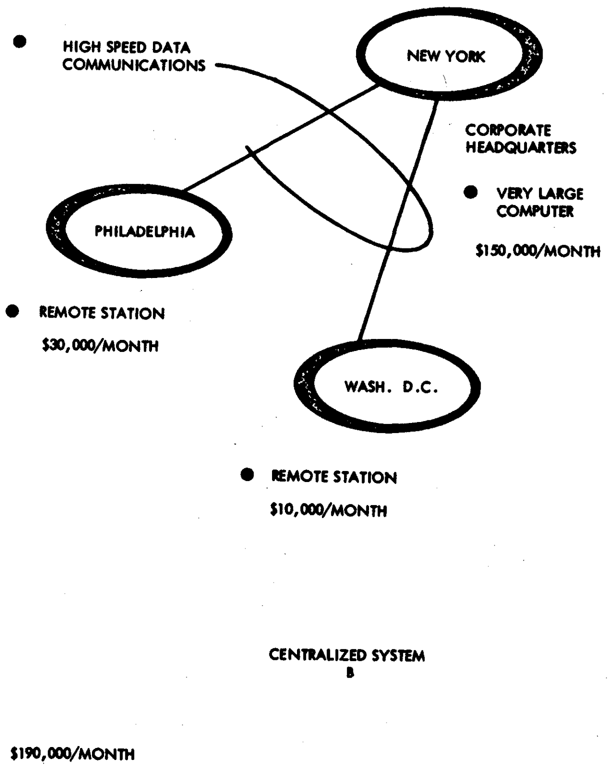


Figure 3.

dent. In this example the communications costs are included as a part of the corporate headquarters facility.

Let us now be a little more specific and look in particular at a 6600 configuration such as the one shown in Fig. 4. We will show how this computing power can be economically distributed at only comparatively slight increments of cost. Since the prime difference in the systems is one of replacing a large computer at a remote station, let us begin by looking at remote stations. Figure 5 depicts a typical remote system having a small-scale computer with punched cards as the primary input and line printers providing the primary output. The punched card inputs could be FORTRAN decks, COBOL decks, data, information retrieval statements, or other commands or requests. The capability of this remote station is such that the 1200-card-per-minute card reader and at least two 1,000-line-per-minute line printers could be operating simultaneously at full speed.

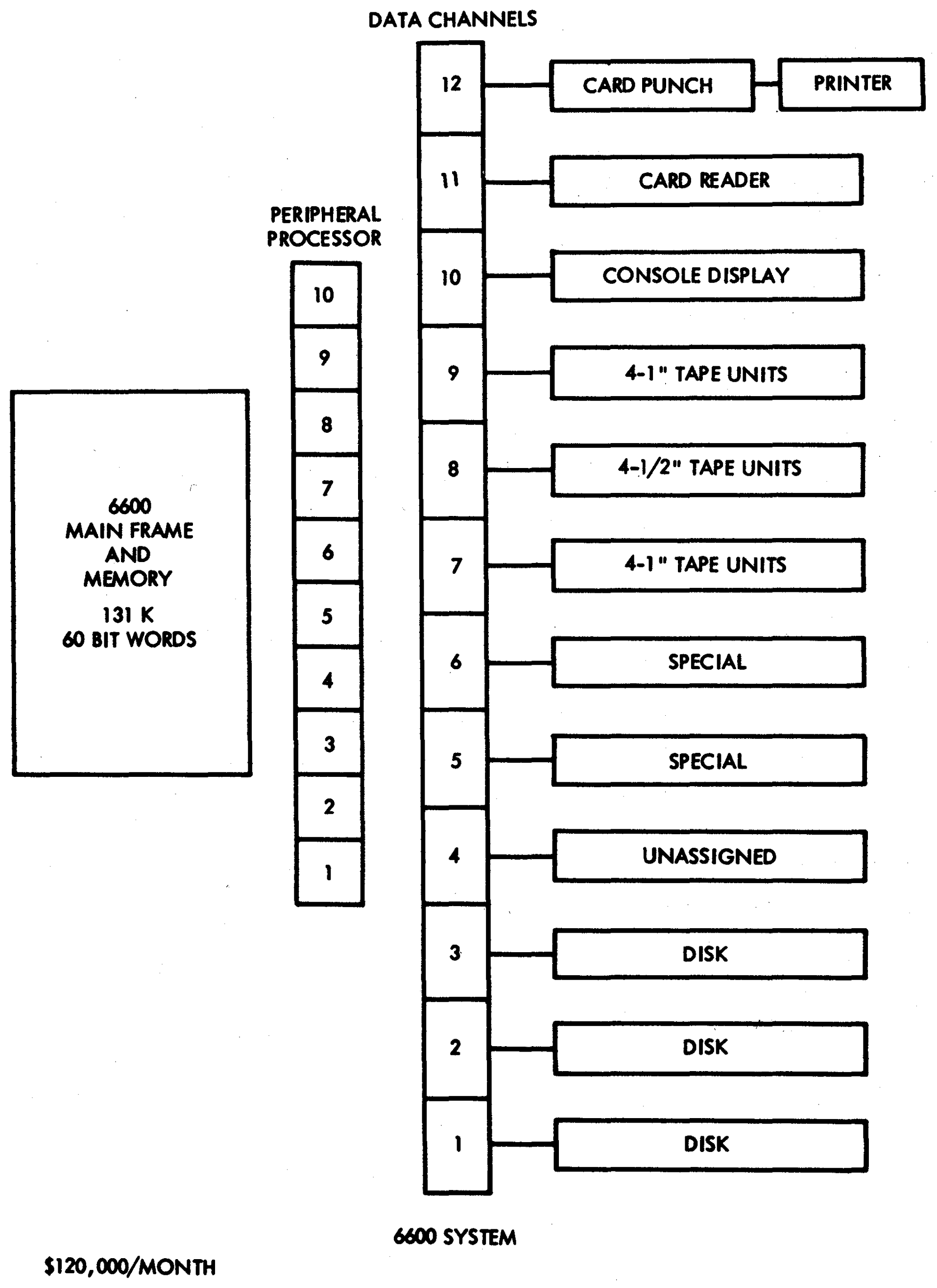


Figure 4.

Perhaps line printers could be printing out results from previous runs while the card reader would be reading in a new job for the next run. If, for example, the remote station were to be an output type only, then three 1,000-line-per-minute printers could be kept running at full speed.

If additional input-output capability is desired, it can be achieved in several different ways. Increasing the memory of the computer will allow additional buffer areas and thereby allow more peripheral equipment to operate concurrently. Of course, a point is reached where the bandwidth of the communications network will have to be increased from Telpak A to wider bandwidths. The typewriter is included in the system to allow communication with the operator. For problems of a repetitive nature or involving larger amounts of data, magnetic tape units might be added. The card to tape function could be performed at the remote facility itself without using the central computer.

Other types of remote stations are also available (Fig. 6). A much less expensive one would be the standard teletype unit such as used in Project MAC at MIT. Since these devices are very slow by nature, a multiplexor could be used to service up to 64 teletype units on a single computer input-output channel. The teletype could be used for low volume information retrieval, business reports, and for entering and running short programs. An increase in capability from this teletype station could be achieved by having a low speed card reader and a low speed line printer at the remote facility. At remote stations of this type, it would not be necessary to use a computer at all. A peripheral adaptor, which in essence extends the input-output channel of a computer to remote locations via communications lines, is all that would be necessary. This peripheral adaptor could not operate as many peripheral devices simultaneously, and requires a buffer memory in each of the devices. This adaptor or channel extender, however, does not make as efficient use of the communications channel as a computer would; nevertheless, as you observe, the costs are reduced.

As can be seen, the number of possible

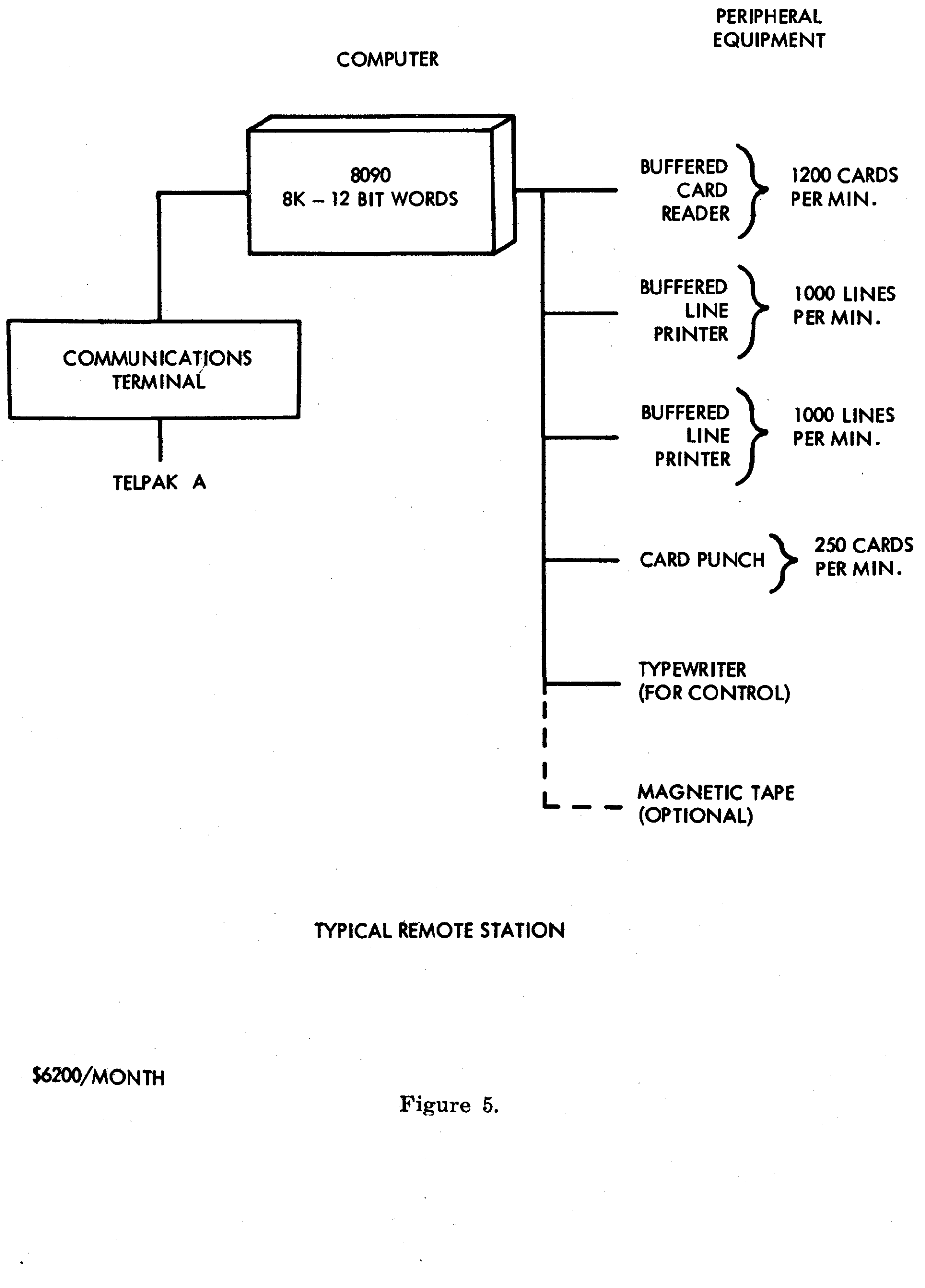
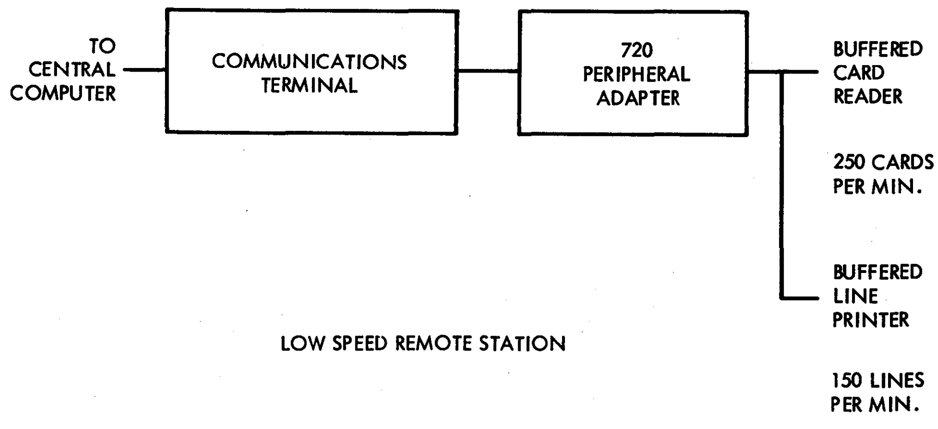
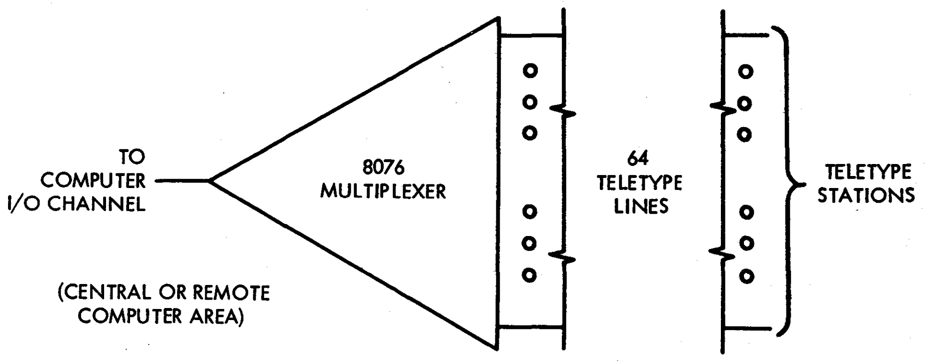


Figure 5.



LOW SPEED REMOTE STATION

$2000/MONTH



REMOTE TELETYPE STATIONS

Figure 6.

configurations at the remote sites is almost endless. The remote computers may be of a small 8090 type as illustrated here, or small to medium 3100 or 3200 type computers could be used to advantage as remote stations in certain types of applications. The communications or data transfer facilities to handle these remote stations are presently available. As has been mentioned, Telpak A may be used for communication with several types of remote facilities. In addition to this and other common carrier facilities, several other methods are used for transmitting data. For distances up to several miles, a coax cable may be used; for further distances, a microwave transmission is available. Both the hard line and microwave transmission are extremely fast with rates of up to 5 million bits per second.

Now that the remote stations as well as communications have been discussed, let us look at the other end of the line. Depending upon the number and the nature of remote stations to be utilized in a centralized computing facility, a number of different methods may be used to get into the 6600. For example, if the number of remote stations is small, a connection may be made directly into an available data channel, as shown in Fig. 7. Here we show both microwave facilities directly connected into a data channel and 64 multiplexed teletype terminals into another data channel on the 6600.

As the number of remote stations increases, other considerations must enter the picture. One problem is that of the duty cycle of the peripheral processors. If the type of problems are of a nature that require little input-output activity, then many of the peripheral processors are available to monitor the remote stations. However, if problems to be run involve a substantial amount of input-output activity, then a suitable solution for servicing the communication lines must be found. If the data rates are slow, a time-sharing method can be used such as the multi-

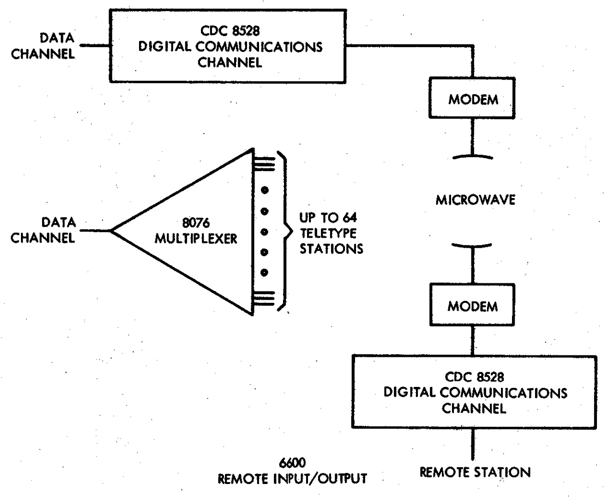

DATA CHANNEL
CDC 8528 DIGITAL COMMUNICATIONS CHANNEL
MODEM
MICROWAVE
DATA CHANNEL
8076 MULTIPLEXER
UP TO 64 TELETYPE STATIONS
MODEM
CDC 8528 DIGITAL COMMUNICATIONS CHANNEL
6600 REMOTE INPUT/OUTPUT
REMOTE STATION

Figure 7.

plexor that handled the 64 teletypewriters. If data rates are fast, such as when using Telpak or microwave units, and a large number of these must be handled simultaneously, as normally will be the case, additional input-output channels must be used. One way of solving this problem is to add an additional, but much less expensive computer, to the facility to handle the increased simultaneous input and output. This is illustrated in Fig. 8, which

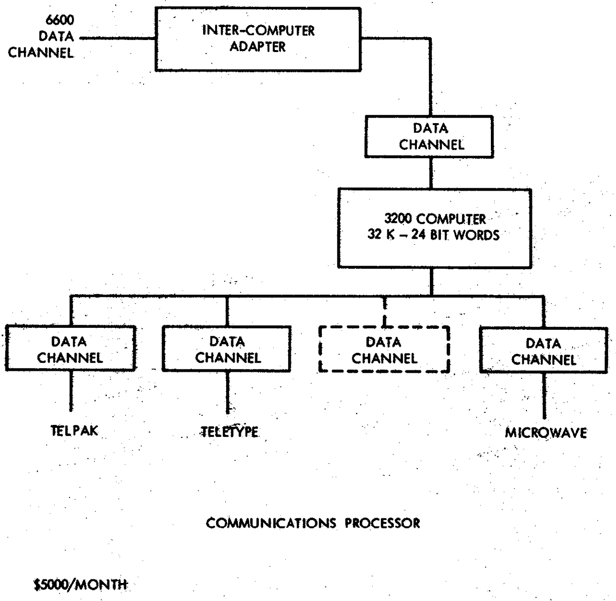


Figure 8.

shows a 3200 computer serving as a communications processor. An additional advantage of this system is that most of the ordinary housekeeping tasks which are necessary for dealing with communications equipment can now be handled by the 3200. The additional expense of this system is very small considering the increased capability.

Since the 6600 SIPROS operating system places all inputs on a stack on the system disks, one further consideration is necessary; that is, access time to the disk. This 6600 disk access time is approximately 200 ms in the worst case. The flexibility of the SIPROS operating system allows additional disks to be added for stacking inputs and outputs to the 6600 system, can allow different allocations of storage stacking to the multiple disks, and can allow the inputs to the job stack to be from a variety of sources, both at the center and from remote sites. However, if the number of remote stations is large or if the volume of input-output data is substantial, then we must consider the problem of accessing the disks for all system input and output. The disk access time can be buffered out by transferring long blocks of data to the disk, thereby decreasing proportionally the number of disk accesses necessary (Fig. 9). The buffering occurs in a mass core storage placed between the communications processing 3200 and the centralized 6600 system. All system input-output is stored in blocks of 16,000 to 32,000 characters per remote device in this mass core store. By the use of this buffering technique, effective system transfer rates can be achieved. We have now worked our way up to a large-scale centralized system, completely illustrated in Fig. 10.

In the typical large-scale system shown here we have 12 remote facilities, a dual 3200 communications processor and the 6600 central computer. The peripheral equipment at a typical remote facility consists of two 1,000-line-per-minute printers, a 1200-card-per-minute reader, and a 250-card-per-minute punch. The number and type of input-output equipment, however, varies at the different remote facilities depending on the load at each individual station. A small-scale 8090 computer is located at each remote site to service the input-output equipment. The memory size, however, of the 8090 varies at the remote stations as according to the number of external equipments.

The communications terminals shown in Fig. 10 consist of a Control Data 8529 and a 301-B Data Terminal. Besides interfacing the

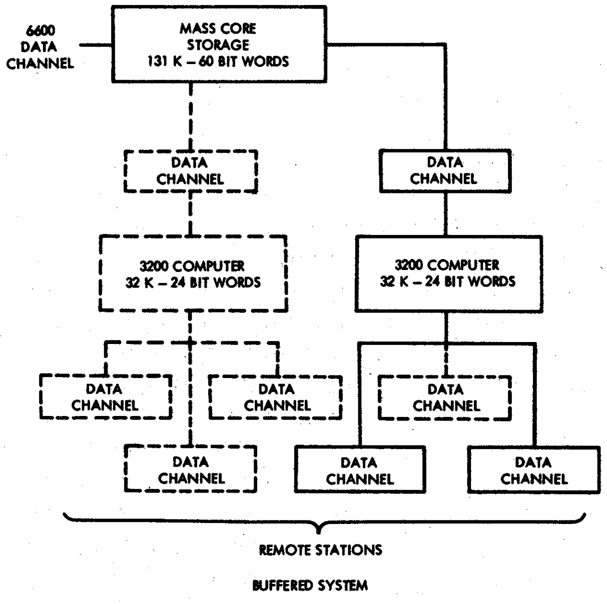


Figure 9.

parallel computer interface with the serial 301-B interface, the 8529 units provide for error checking and detection. A scheme for encoding and decoding blocks of data by generating a 12-bit cycle code is used. This provides the following error detection capabilities:

1. Any odd number of errors.
2. All error burst of length 12 or less.
3. 99.95% of all error bursts of length 13.
4. 99.98% of all error bursts of length 14 or greater.

A burst is defined as any pattern of errors for which the number of bits between the first and last errors, including these errors, is the burst length. The cyclic code is one of the most effective and economical error detection techniques for serial transmission presently known.

Let us now study the data flow in the system in greater detail. First let us examine the data transfer between the remote stations and the communications processor. All such transfers are made in blocks of eight-unit records. This block length is an effective trade-off between maintaining high line efficiency and low core requirements in the remote computer. The 8090 at the remote site initiates all data transfers. This is done by generating an interrupt to the communications processor, and is followed by a transfer of a triple redundant 72-bit control word block. The required eight-unit records are now transferred and are followed by an acknowledgement from the receiving computer (illustrated in Fig. 11 along with

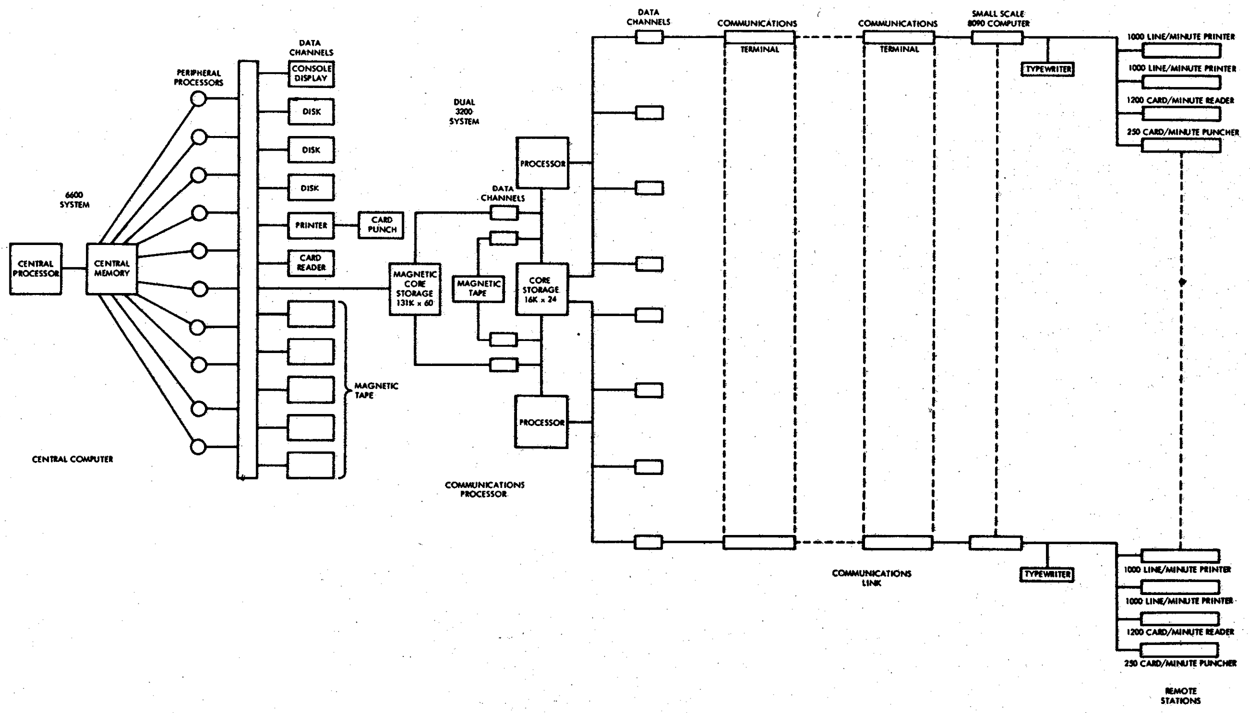


Figure 10.

the corresponding times). Here we are assuming a propagation delay of 4 ms and data rates of Telpak A, 40,800 bits per second. You will note that the complete transfer time required for eight cards is 115 ms, and for eight lines on the printer, is 183 ms. Since it takes a 1200-card-per-minute reader 400 ms to read the eight cards, then this communications link could easily handle three 1200-card-per-minute card readers simultaneously. Further, since a 1,000-line-per-minute printer requires 480 ms to output eight lines of print, data rates are sufficiently fast in order to keep two 1,000-line-per-minute printers and one 1200-card-per-minute card reader fully or simultaneously operating. From the 3200 communications processor, the input data is transferred to the appropriate area in the mass core storage. Each input-output device in the system is dynamically assigned a certain area of this bulk core. Input areas are assigned blocks of 16,000 characters while output devices are assigned areas of

	TIME	
ACTION	CARD EQUIPMENT	PRINTER
1. Remote Station to Communications Processor Interrupt and Processing	5 ms	5 ms
2. Transfer of 72-bit Control Block	7 ms	7 ms
3. Transfer of 8 Unit Records	96 ms	164 ms
4. Acknowledgement	7 ms	7 ms
	115 ms	183 ms

COMMUNICATIONS LINK TRANSFER TIME

Figure 11.

Typical Input:	600 cards = 48,000 characters
Bulk Core Storage Assignment:	16K characters/card reader (divided into double buffer areas)
Time to Transfer from Core to Input Disk·	6 x (220 + 6) ms = 1356 ms
Time to Transfer Input Disk to Central Memory:	220 ms + 36 ms = 256 ms
Total Disk Time for Transfer from Core to Central Memory:	1356 ms + 256 ms = 1612 ms
600 Cards Require 30 seconds on 1200 Card/Minute Reader	30 seconds/1.612 seconds = 18 – number of card readers
18 Card Readers at 1.6K Words each:	28.8K – 60 Bit Words

INPUT DISK UTILIZATION TIME

Figure 12.

Typical Output:	1200 lines = 160,000 characters
Bulk Core Storage Assignment:	32K characters/line printer (divided into double buffer areas)
Time to Transfer Central Memory to Output Disk:	220 ms + 122 ms = 342 ms
Time to Transfer Output Disk to Core:	10 x (220 + 12) ms = 2320 ms
Total Disk Time for Transfer from Central Memory to Core:	342 ms + 2320 ms = 2662 ms
1200 Lines Require 72 seconds on 1000 Line/Minute Printer	72 seconds/2.662 seconds = 27 – number of line printers
27 Line Printers at 3.2K words each:	86.4K – 60 bit words

OUTPUT DISK UTILIZATION TIME

Figure 13.

25 – 1000 LINE/MINUTE PRINTERS AT 3.2 K	=	80K
18 – 1200 CARD/MINUTE CARD READERS AT 1.6K	=	28.8K
6 – 250 CARD/MINUTE CARD PUNCHERS AT 3.2K	=	19.2K
TOTAL STORAGE REQUIREMENTS		128 K

BULK CORE STORAGE UTILIZATION

Figure 14.

32,000 characters. Each of these is divided into a double buffer area.

Let us examine a typical input of, say, 600 cards or 48,000 characters (Fig. 12). Since the purpose of the bulk core store is to buffer out the access time to the system input disk, we will assume in all further discussions worst case timing; that is, 220 ms for access and latency time on the disk. Generally speaking, however, it will be substantially less than this. Now, since a typical input consists of, for this example, 48,000 characters and each input area is divided into a double buffer area of 8,000 characters, it is necessary to make six transfers between the bulk core store and the input disk. Allowing 220 ms for access and 6 ms for writing the disk, then the six transfers as shown will not exceed 1356 ms. The time to transfer information from the input disk to the central memory is again 220 ms for access, plus 36 ms for transferring the 48,000 characters. Thus, full disk time required for transfer from core to central memory is not greater than 1612 ms; in fact, generally it will be considerably less. Now, since 600 cards require 30 seconds on a 1200-card-per-minute reader, 30 seconds divided by the input disk time necessary to service one reader yields a figure of 18 for the effective number of card readers which can be kept operating simultaneously by using this technique. If these 18 card readers are operating simultaneously and since each requires 1.6 thousand 60-bit words, a total of 28.8 thousand 60-bit words in the bulk core store is required to handle the system input.

A similar technique may be employed in the output case, assuming in this case typical output consisting of 12,000 lines or 160,000 characters, and bulk core assignment of 32,000 characters per printer (Fig. 13). The time required to transfer from central memory to the output disk is 220 ms (worse case access) plus 122 ms for the data transfer, yielding a total of 342 ms. Further, since 16,000 characters are transferred between the output disk and the bulk core each time, it is necessary to transfer 10 times for the total of 160,000 characters. These 10 transfers each requiring 220 + 12 ms yield a total of 2320 ms for the time to transfer from the output disk to the bulk core. The total disk time thus necessary for transferring a typical output from central memory to bulk core is 2662 ms. Since this typical output of 1200 lines requires 72 seconds on a 1,000-line-per-minute printer, dividing 72 seconds by 2.662 seconds yields 27 as the number of line printers which can be simultaneously operated by this system. Further, these 27 line printers at 3200 60-bit words each require a total of 86,400 60-bit words in the bulk core store whenever they are all operating simultaneously.

Similar consideration for a 250-card-per-minute punch yields a conservative 3 to 1 trade-off between printers and punches; that is, with respect to the system, three punches require as much activity as one line printer. Thus, trading two printers for six punches, as shown in Fig. 14, we have total storage requirements necessary for simultaneously operating 25 1,000-line-per-minute printers, 18 1200-card-per-minute-readers, and 6 250-card-per-minute punches. Further trade-offs between printers and readers can be accomplished by increasing and decreasing corresponding storage areas in the bulk core.

What has been demonstrated here is the way in which the large capability of a centralized computer such as the 6600 can be effectively and economically distributed to a number of users at a number of remote sites. Further, this distribution of computing power is economically accomplished and uses only existing equipment and existing communications facilities. Finally, the capability for such distribution and time-sharing is provided by the organization of the 6600 and the flexibilities in the design of SIPROS.

THE MODEL 92 AS A MEMBER OF THE SYSTEM/360 FAMILY

G. M. Amdahl
IBM Corporation
Los Gatos, California

The Model 92 is a full-fledged member of the System/360 line of processors. At the beginning of the design of System/360, the goal was to provide a system capable of satisfying all of the varied computing applications to which IBM machines were applied in the past. It was also desired to apply the new system to a number of applications for which those older machines were somewhat inadequately prepared to serve. At that time, the previous lines of machines were evaluated to see which ones might possibly serve as a base for the start of the design. The older commercial processors did not form a suitable base for the scientific computational capability desired. The scientific line was also not elected for use as the base.

In the particular instance of the 7090, it was felt that this was an improper base for several reasons. First of all, the word size was really somewhat inadequate for a majority of the problems that were actually being run in that word size. Secondly, the address size was insufficient and improperly constructed to handle the variable field length processing which was important, not only from the standpoint of doing commercial work, but from the standpoint of general data processing in the scientific area in addition to such things as compilation.

The Model 92 is completely compatible with System/360 in every respect except that decimal arithmetic is not offered. All of the variable field length handling capabilities for compilation and data processing are provided, as well as conversion between decimal and binary and back again. Compatibility also means that the same systems configuration flexibility is provided in the Model 92 as in any other members of the System/360 line (Fig. 1).

The Model 92 can efficiently utilize the programming support for all of the System/360. This requires that the codes which are written for the slower members of System/360 will run in very nearly optimal fashion on the Model 92. This can be done because of the internal logical organization of the Model 92 in which the optimalization of the execution of code is essentially carried out by hardware algorithm rather than by programming algorithm. This point will be well worth bearing in mind while reading Dr. T. C. Chen's description of the internal organization of the Model 92.

The same checking techniques are employed in the Model 92 as in the rest of the System/360 processors. The checking not only checks the residence of data in memory, but checks the actual execution of the arithmetic and logical operations on this data. Further, the determination of the existence of an error provides an interrupt which allows transfer of control to a diagnostic routine which can determine whether or not the computation can be re-entered as well as just determine the state of the CPU with this error.

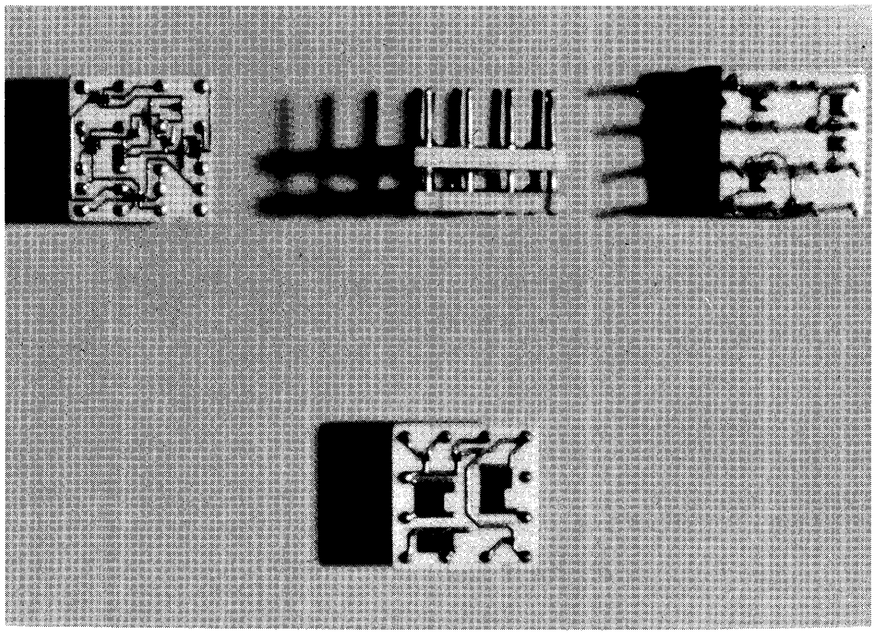


Figure 1. Model 92 Circuitry.

The performance gain of the Model 92 above its nearest neighbor, the Model 70, is markedly greater than the performance gap that exists between any other pair of the System/360 processors. In part, this is because in the search through the possible ways in which higher performance machines could be realized, a design was found which was believed to be the most optimum point on the design curve existing above Model 70. There are two ways in which this gain is achieved. First of all, the use of faster componentry. The memory used is two times as fast as the memory in the Model 70. The higher performance circuitry used is between 2½ and 3 times as fast (Fig. 2). It is a little hard to get an actual

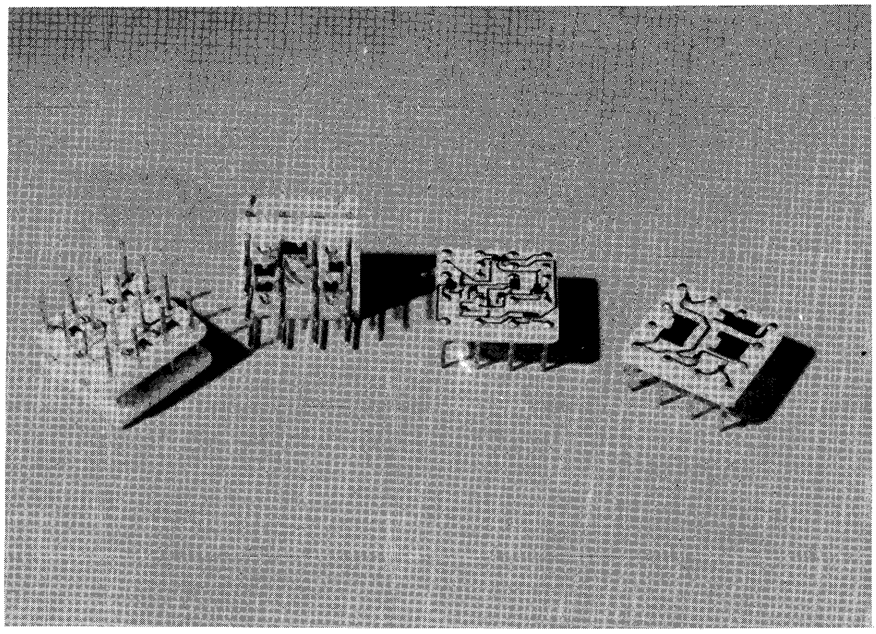


Figure 2. Model 92 vs. Model 70 Circuitry.

figure for the circuitry itself, since the logic circuitry employed in the Model 92, has properties which yield a more efficient realization of many of the hardware algorithms.

However, this factor of 2 to 3 gain from memory and circuits only inadequately describes the means by which the performance is achieved. There is much greater sophistication within the CPU, such as better algorithms for the arithmetic operations and also non-sequential look-ahead and multiple concurrency, which makes by far the greatest difference in the performance gain. This performance gain is on the order of 15 times that of the Model 70. The inclusion of the Model 92 in the System/360 line provides then a family of compatible processors which cover approximately a range of three orders of magnitude in performance.

In the Model 92 memory, one may have either eight or sixteen boxes of 8,192 64-bit words in each box, with the boxes interleaved. Interleaving implies that successive word addresses appear in successive physical boxes.

In addition to the ½ μsec memory on the Model 92, one may also have as addressable memory the large capacity storage; this is an 8 μsec full cycle memory in which up to eight boxes can be interleaved. This is somewhat in excess of two million 64-bit words. The impact of such a large memory array will be discussed in Mr. Carl Conti's paper.

The circuitry employed in the Model 92 is an off-shoot of the SLT circuitry employed in the rest of the System/360 line. Essentially, the same ceramic substrate is employed, on which the circuit components are mounted. In ACPX technology, there is a multiplicity of transistors appearing in a single silicon chip; in the SLT technology, there were a multiplicity of diodes appearing in a single silicon chip. In a sense, they are a kind of hybrid between individual components and integrated components. It was indicated earlier that the performance of the ACPX circuitry was 2½ to 3 times that of the SLT. The circuit delays experienced in the ACPX circuitry ranges from 1.2 nanoseconds to 4 nanoseconds, depending on the loading seen by the logic circuits. The average in the machine environment appears to be about 1.7 nanoseconds per stage.

The performance of the circuit in the machine environment is, of course, very dependent upon the density achieved in the package. In ACPX, densities of something like 6 times that of the SLT packaging are achieved. This is attained by use of both sides of the ceramic substrate for placement of the circuit components and with two ceramic substrates placed one on top of the other to form a module.

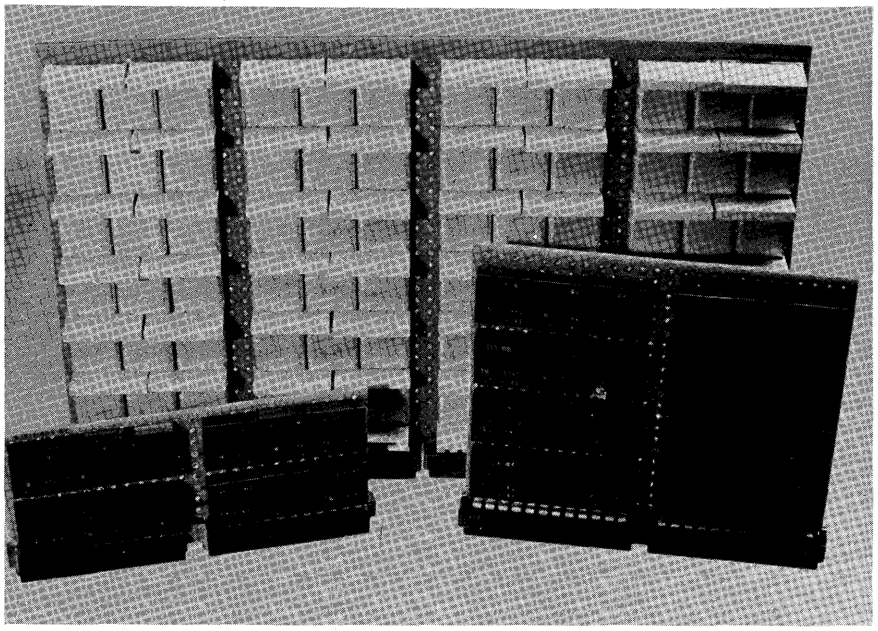


Figure 3. Comparative Card Packaging.

The reduced use of area consuming printed resistors improves the effective area for active elements by about 50%. The number of contacts on this module is increased from the 12 which appeared on the SLT to 16 contacts in the ACPX. This is done by merely filling in the inner parts of the grid, which are on tenths of an inch spacing.

For packaging, a pluggable card is used which has three times the capacity of the largest SLT card (Fig. 3). These cards plug into a board which is about the size of an 8½ x 11 sheet of paper, and which is equivalent to the back panel wiring (Fig. 4). The density achieved by this packaging is about 5,000 circuits per board.

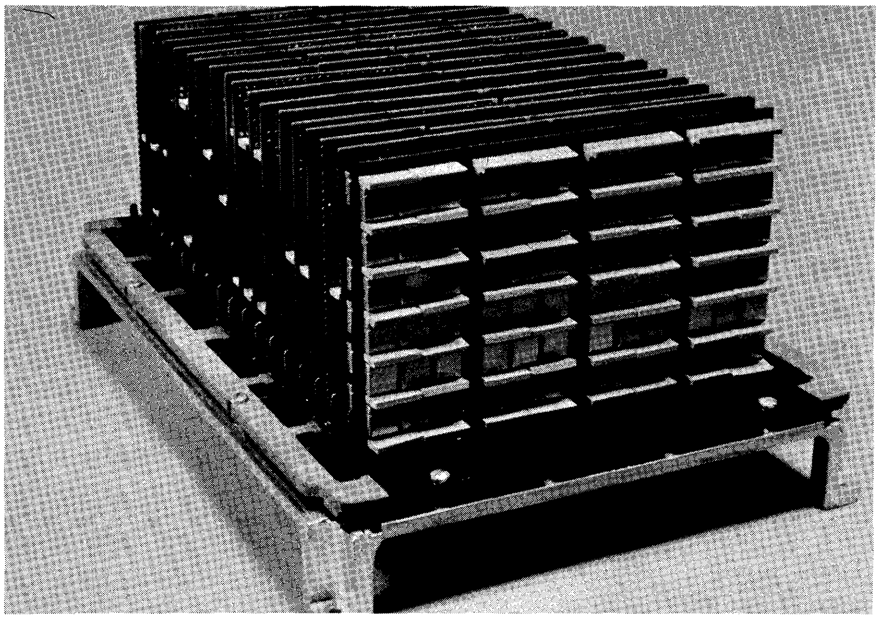


Figure 4. Fully Populated Board.

THE OVERLAP DESIGN OF THE IBM SYSTEM/360 MODEL 92 CENTRAL PROCESSING UNIT

Tien Chi Chen
IBM Corporation
Poughkeepsie, New York

1. INTRODUCTION

The design of the IBM System/360 Model 92 Central Processing Unit features a very high degree of overlap and concurrency in processing, and full compatibility with other members of the System/360 machine family.* †

Pitfalls in overlap machine designs are many. In order to speed up some portions of the code, the efficiency of other instructions—indeed sometimes the logical coherence of the program itself—may have to be compromised. Optimizing techniques frequently have to be applied externally to avoid pitfalls and enhance efficiency. On the other hand, the machine gains speed by internal asynchronism, and the resources within are rapidly changing functions of time and instruction context. We have, therefore, the paradox: the more asynchronism, the greater the need to optimize, yet the harder it is to perform the optimization externally.

The Model 92 design is based on a local autonomy principle, allowing each of the individual units to optimally allocate its resources within its jurisdiction.

This local autonomy allows the Model 92 CPU to handle the full System/360 instruction repertoire (except the decimal instructions) with no loss of logical coherence, in an optimal fashion. The need for external optimization, by hand-honing or by compiler action, is thereby sharply reduced, and debugging effort is correspondingly minimized.

2. MEMORY HANDLING AND INPUT-OUTPUT

High Speed Memory

The high speed memory of the Model 92 consists of ferrite core arrays of 0.5 μsec cycle time, 8K long words (each of 64 information bits plus eight parity bits) per module, 8 or 16 modules (64K or 128K long words) per computer. The addresses are fully interleaved: consecutive long word addresses refer to different modules.

Large Capacity Storage

To supplement this, we have the large capacity storage (LCS) consisting of 8 μsec cycle time core arrays, 128K or 256K per module, 2, 4, 8 or 16 modules (that is, up to more than 2 million fully addressable long words) per computer. The LCS modules are also interleaved up to eight-fold, leading to a maximum effective rate of one long word per μsec. The LCS contents are fully addressable down to the eight-bit "byte" level, just like the high speed memory.

* For details of IBM System/360 architecture see: G. M. Amdahl, G. A. Blaauw and F. P. Brooks, Jr., *Architecture of the IBM System/360*, IBM J. Research and Development, 8 (April 1964), pp. 87-92; also *IBM System/360 Principles of Operation Manual*, Form A22-6821-1 (1964).

† For a comprehensive discussion of overlap designs the reader is referred to W. Buchholz, *Planning a Computer System* (McGraw-Hill, N. Y., 1962).

Memory Protection

The System/360 memory protect mechanism applies to the entire storage—LCS as well as the high speed memory—as administered by the memory bus control element. For any request to store, the latter supervises the matching between the storage lock and the key of the requestor (CPU or I/O channel); attempts at violation are disabled, with an alarm signal sent to the CPU.

Treatment of Memory Conflicts

Because the decoding time (one cycle, no more than 90 nanoseconds) and the effective execution time (close to one cycle per operation) are small relative to the high speed memory cycle time of 0.5 μsec extensive overlap is required not only for the CPU, but the memory bus as well. Memory interleaving reduces the probability of conflicts, but does not eliminate them. When an attempt to access more than one long word from the same memory module is detected, the bus must put the conflicting requests in a waiting queue, and install the proper delays.

Further, since each delay can be equal to several instruction times, the occurrence of a conflict should not put the entire bus in an idling state. In the Model 92 there can be eight independent concurrent memory operations; several of them could be in conflict without affecting the remainder.

I/O Handling

Input-output operations are fully overlapped with CPU action, and are handled via the memory bus. External devices can communicate either with the high speed memory, or LCS.

Storage Transfer Channel

In addition, there is a storage transfer channel which handles memory-to-memory transmittal in a way similar to I/O channel operation, with the following major difference: a programmable spacing parameter δ can be specified such that either

1. consecutive source long words can be transmitted to the sink area, each separated by δ long words; or
2. long words from the source area separated by δ long words can be transmitted to the sink area as consecutive long words.

3. INSTRUCTION SEQUENCING CONTROL

The Model 92 accepts standard System/360 instructions. These come in three sizes: 16 bits, 32 bits, and 48 bits. In a program some of these instructions may overstep the word boundaries.

The design of the instruction sequencing control mechanism is to access, decode, and perform the indexing functions of the instructions at the rate close to one instruction per cycle. Toward this end an instruction buffer is provided, consisting of a primary buffer of eight long words and an alternate buffer of two long words.

Primary Buffer

In the normal mode of operations, the primary buffer is kept half-full of new instructions, and half-full of past instructions. The decoding mechanism extracts instructions from the buffer, and an instruction-word fetch is initiated when a long-word boundary is crossed.

The "new instruction" area houses roughly 10 instructions. When a branch is decoded and executed, the new instructions in the primary buffer may have to be rejected. "Overfetch" of instructions beyond four long words may mean creation of unneeded memory conflicts, and is not permitted in the normal mode.

The Gulf of Ignorance

Because of the slowness of memory access, the execution of an instruction may lag the decoding by 0.5 μsec or more (Fig. 1). For a given instruction, there is a gulf of ignorance between the termination of decoding and the termination of execution. Within the gulf of ignorance decoding and execution can still proceed normally, but not conditional branch instructions based on the result of the instruction creating the gulf. Such conditional branches are undecidable until the gulf is passed.

Conditional Branches

A branch conditioned upon the outcome of a long-passed instruction is outside the gulf

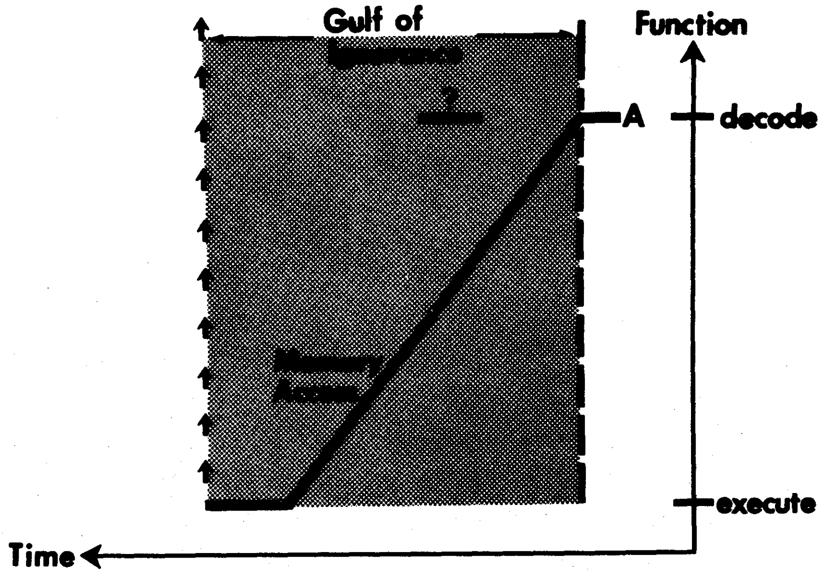


Figure 1. Symbolization of an instruction in an overlap machine. Condition branch instruction based on outcome of execution is marked by question mark.

of ignorance, and is *decidable.* It is handled like a simple branch or a NO-OPERATION. Most conditional branches in routine System/360 programming tend to fall into the decidable category.

On the other hand, if a conditional branch instruction occurs immediately after a condition-generating floating point instruction, it is almost always *undecidable.*

Undecidable Branches

For undecidable conditional branches the machine makes the assumption that the condition probably will not be met. The new instructions in the primary buffer are thus not rejected out of hand. However, the assumption could prove to be wrong, therefore all instructions handled within the gulf of ignorance are issued conditionally, not being allowed to alter addressable registers. Also, the alternate buffer is made to accept two long words of the alternate instruction stream.

Upon emergence from the gulf of ignorance, the condition becomes fully known. If the branch is not successful, a "proceed normally" signal is sent to all areas. Otherwise the tentatively issued instructions are cancelled, and the critical instruction-access problem is bypassed by having available the contents of the alternate buffer.

Loop Mode

The Model 92 has a provision for effective handling of short loops. Whenever a successful branch to a numerically smaller target address occurs, and if the address of branch instruction and that of the target both fall into an eight-long word block (and if the machine is not already in loop mode), the machine will enter loop mode.

During loop mode the machine fills the primary instruction buffer to full capacity (eight long words, or about 20 instructions), starting from the word containing the target down through the loop-creating branch instruction and beyond. The machine will

1. utilize the contents of the buffer without further instruction fetch;
2. guard against violations over the eight long words in the buffer;
3. speed up the loop-creating branch. A simple branch will be faster by one cycle; a conditional branch will be treated as "probably successful."

A subset of the primary buffer contents during loop mode could be a loop also; the latter already benefits from the loop mode and will not require additional treatment.

The machine reverts to the normal mode if the loop-creating branch becomes unsuccessful, or if a branch outside the primary buffer is executed.

4. STEP-BY-STEP BUILDUP OF CPU

The design and purpose of the remainder of the CPU can best be shown via a step-by-step procedure.

Thus far we have discussed the memory, the memory bus, and the instruction buffers. In conformity with System/360 specifications, there are 16 general purpose registers (GPR's), and four floating-point registers. Because the GPR's serve as index registers as well as base registers, aside from being fixed-point accumulators, they are put in close proximity of the decording portion of the machine, forming the Instruction Unit, or the I-Box.

The floating-point accumulators and the floating-point arithmetic units are the basis for the Execution Unit, or the E-Box.

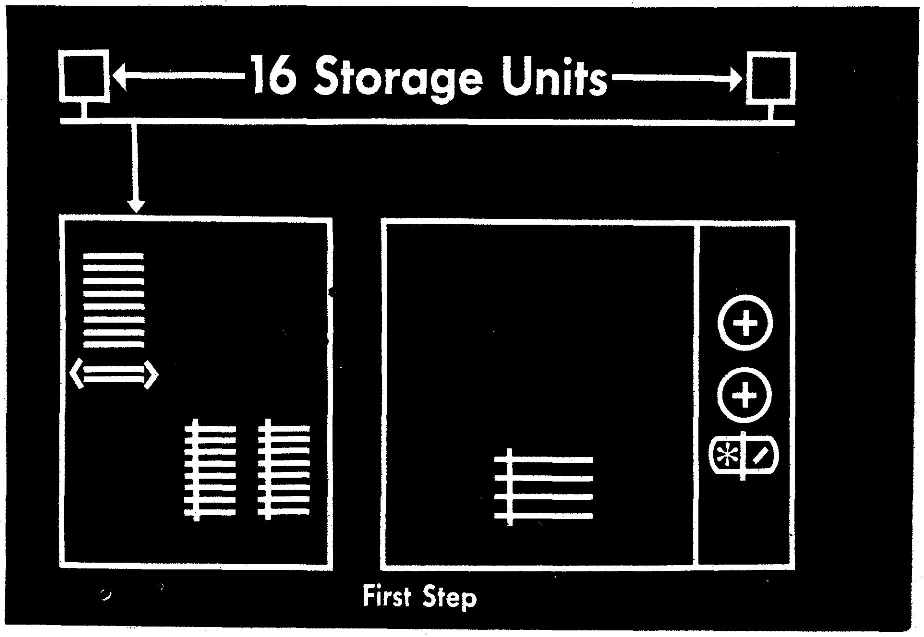


Figure 2a. Step-by-step CPU buildup.

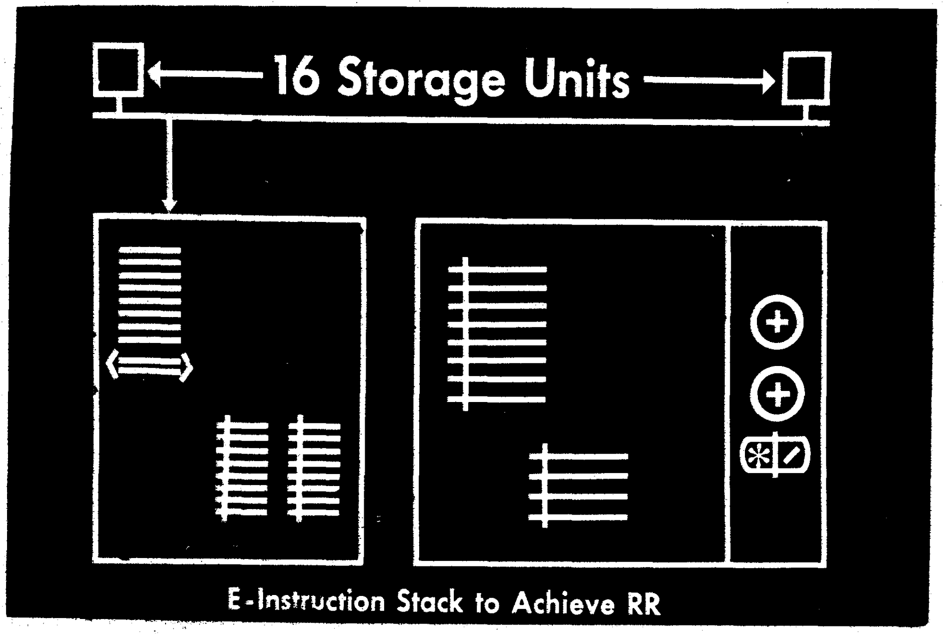


Figure 2b. Floating Point Instruction Stack added.

A sketch of the design thus far is seen in Fig. 2a. The floating-point arithmetic organs include two floating-point adders and a multiplier-divider. The speeds are as follows:

• normalized floating add:	2 cycles
• normalized floating multiply:	3 cycles
• normalized floating divide:	10 cycles

Floating-Point RR Instructions

A typical floating-point RR instruction is the following:

ADR 2,4 'ADD DOUBLE NORMALIZED RR

which means

$$C(R2) + C(R4) \qquad C(R2)$$

where the + symbolizes a floating-point normalized add involving 56-bit hexadecimal fractions with corresponding exponent adjustment.

It is noticed that the capability for the execution of such instructions resides in the E-Box. It is, therefore, desirable to put the E-Box RR instructions in the E-Box as well.

This is accomplished by the installment of an eight-level E-Box instruction stack, which converts the E-Box into a stored program computer (Fig. 2b).

The decoding mechanism, upon encountering a floating-point RR instruction, merely puts the partially decoded instruction in the next available position in the E-instruction stack. The E-Box from this point on takes care of the optimized execution.

Floating-Point RX Instructions

The other class of floating-point instructions are the RX instructions, typified by the following:

AD 2, DOG (14, 15) 'ADD DOUBLE NORMALIZED RX

which has the following meaning:

$$C(X14) + C(B15) + DOG]_{24} = m$$
$$C(R2) + C(m) \qquad C(R2)$$

(X 14 is GPR 14, being used as an index register; B 15 is GPR 15, being used as a base register; $]_{24}$ means truncation to 24 bits to yield the effective address m.)

Upon careful study it becomes evident that "mathematician's reasoning" can be gainfully employed to reduce the new problem to a previously solved one.

We note the first equation refers to the I-box. To handle it properly we need rapid accessing of the contents of GPR 14, GPR 15, which should combine with DOG to form the 24-bit effective address in one cycle. A three-input adder is used towards this end.

The second equation can be further broken into two parts:

$$C(m) \qquad C(R13)$$
$$C(R2) + C(R13) \qquad C(R2)$$

The first part belongs to the memory bus, and the second part is a standard (in appearance, at least) RR instruction. To accomplish this it is necessary to have an E-Box operand stack, with an addressing scheme similar to that of the floating-point accumulators (Fig. 2c).

The floating-point RX ADD instruction is handled this way:

a) Upon decoding, the instruction is discovered to be a floating RX fetch-type instruction.
b) A triple-add is performed to yield m, the effective address. R13 in the 6-level E-operand stack is known to be vacant.
c) m is presented to the memory bus with R13 as return address.
d) The RX instruction is disguised as an RR instruction involving R13, and sent to the E-instruction stack.
e) When R13 becomes full, the treatment of the pseudo-RR instruction is indistinguishable from the "true" RR instruction.

Fixed-Point Arithmetic

The scheme adopted for the E-Box is quite applicable for the fixed-point instructions, and there is an I-Box instruction stack (6 levels), an I-Box operand stack (6 levels), and a separate fixed-point arithmetic unit. The protection against logical coherence violation is, however, different, since the GPR's serve also as index registers and base registers.

There is a two-level store operand buffer to handle to-memory instructions from either I-Box or E-Box, and eight address registers to guard against logical conflicts involving fetch-

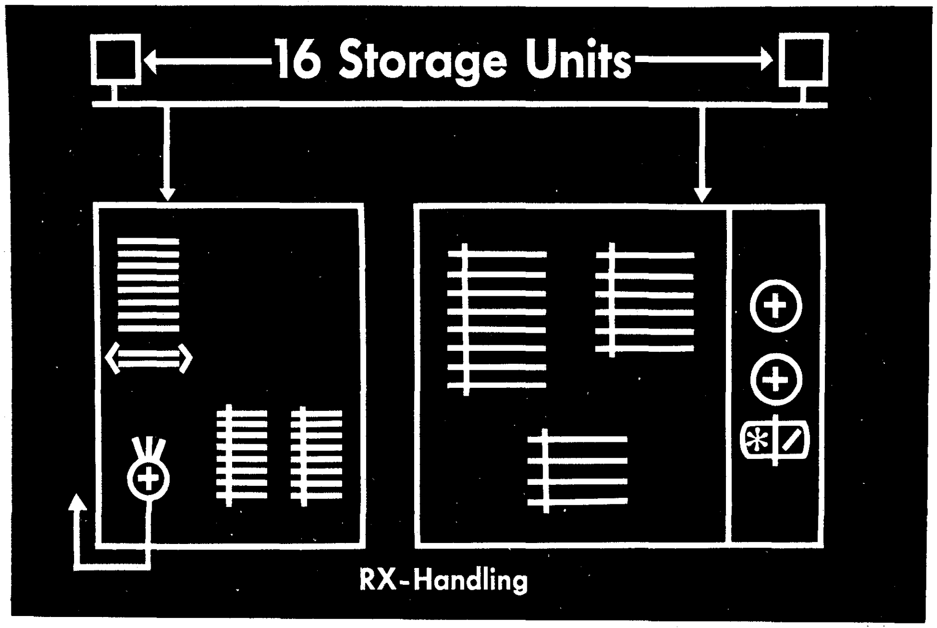


Figure 2c. Floating Point R-X handling. 3-Input Adder and Floating Point Operand Stack added.

store sequencing. The design now has the appearance of Fig. 2d.

In the handling of System/360 variable-field length instructions, the I-Box and the E-Box pool their buffer resources together to speed up the processing.

5. INTERNAL FORWARDING

An interesting feature in the System/360 Model 92 CPU is adapted from the IBM STRETCH design, namely, internal data forwarding. Consider the following floating-point instructions:

A	LD	0,P
B	DD	0,Q
C	ADR	0,2

meaning

A	C(P)	C(R0)

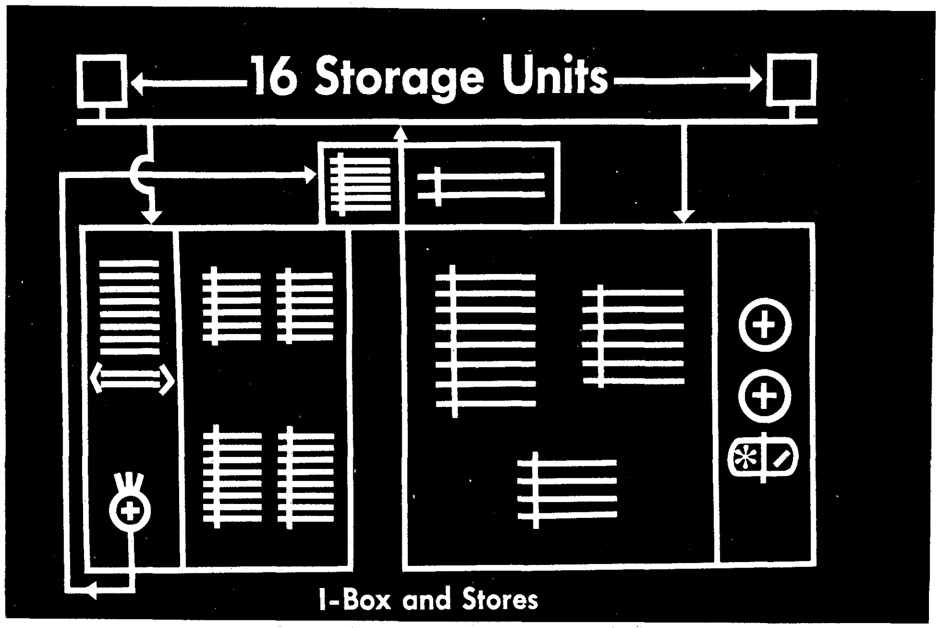


Figure 2d. Complete CPU.

B	C(R0)/C(Q)	C(R0)
C	C(R0) + C(R2)	C(R0)

Since the Model 92 has four floating-point accumulators, the fixation on one accumulator could be construed as bad programming. However, we adopt the point of view that such programming is not necessarily bad; there are many situations in which it would be unnatural to do otherwise. An attempt is therefore made to run these sequences quite efficiently.

The three instructions lead to the linear graphs in Fig. 3a. It is assumed that adder A2 is employed for C. The combination has a double loop. We note that the act of putting an operand into R0 and subsequently removing it has a high cost for the Model 92.

However, the three instructions can be synthesized into one macro-instruction, using cancellation techniques reminiscent of freshman chemistry:

A	C(P)	C(R0)
B	C(R0)/C(Q)	C(R0)
C	C(R0) + C(R2)	C(R0),

yielding

ABC	C(P)/C(Q) + C(R2)	C(R0)

and the greatly simplified linear graph in Fig. 3b results. The simpler graph is handled vastly more efficiently than the complicated one, and the E-Box has the ability to perform the "topological" path distortion, yielding paths equal in outcome but enhanced in efficiency.

The situation is even more striking if two more instructions are added:

D	STD	0,R
E	LD	0,S

The store will be gated directly from the output of the adder, and instruction E signals to the E-Box that R0 need never be referenced in handling instructions ABCD at all. Thus, E begins an independent string of instructions, capable of being handled concurrently with the ABCD string, and may even finish before ABCD.

6. HARDWARE OPTIMIZATION

The System/360 Model 92 decodes instructions in sequence; the decoding action distributes the work load to various autonomous units. From that point on, strict sequence is no longer adhered to, but is frequently upset.

To the uninformed bystander, the machine seems to replace the orderliness of the given code by chaos.

This apparent chaos actually results from the local reorganization of the available resources. The memory bus concentrates on the optimum handling of memory requests. The fixed-point, floating-point units concentrate on optimum execution of corresponding instruc-

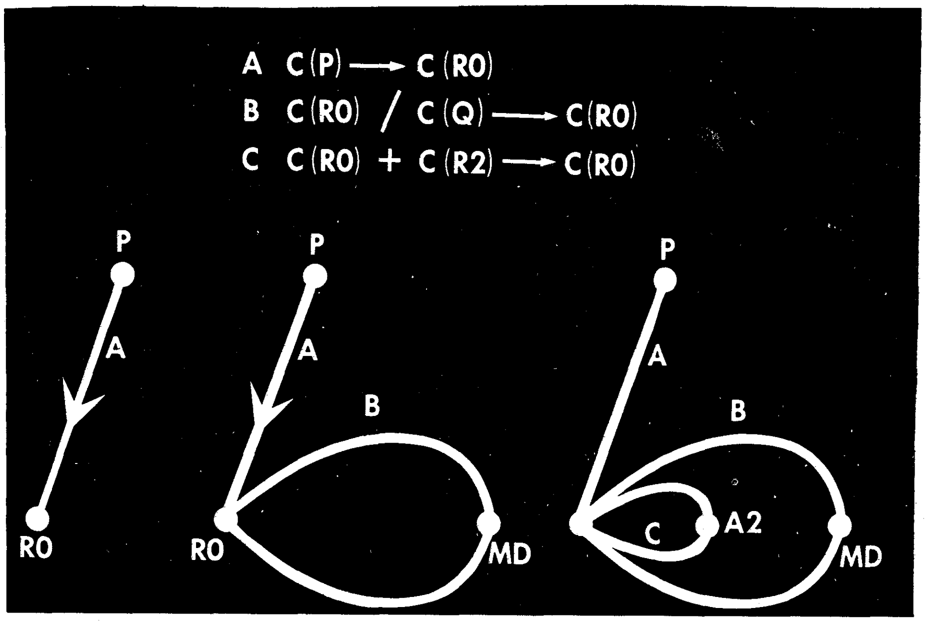


Figure 3a. Linear graph for three instructions A, B and C.

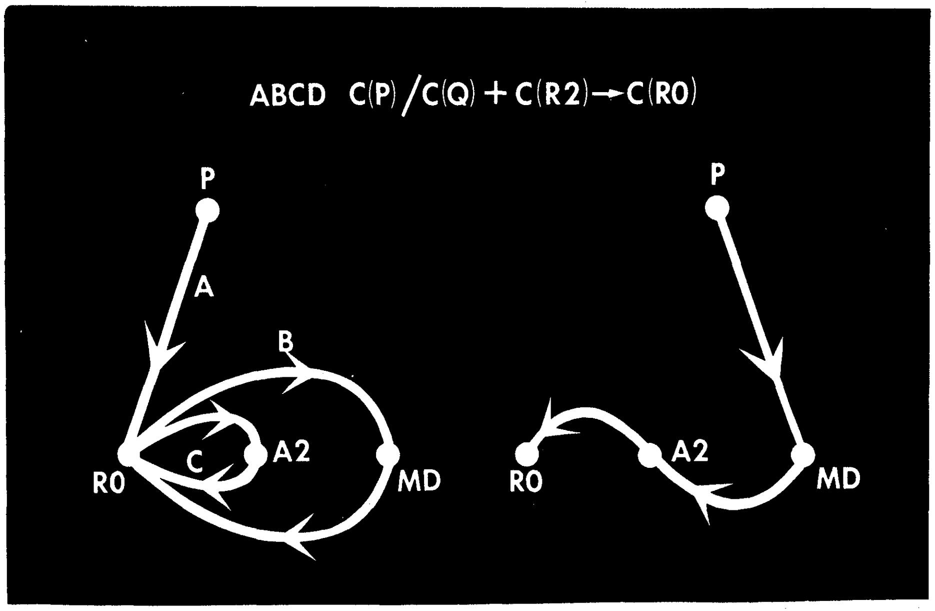


Figure 3b. Optimization through internal forwarding.

tions. The instruction sequencing mechanism meanwhile optimizes instruction-fetch and branching. Within each unit, sub-units frequently also possess autonomous behavior. The degree of parallelism is extremely high within the CPU.

The asynchronism makes it impossible strictly to predict the detailed behavior within the CPU, short of a full-fledged simulator. It is not possible, for instance, to predict conflicts between an instruction-fetch and a floating-point store. Consequently, it is extremely difficult to optimize a program by external means.

However, this external optimizing is not really necessary. No one is in a better position to examine the available resources and detect bottlenecks in data flow within a given unit than the very unit itself. It is, therefore, also provided with hardware tools to seek and adopt optimizing alternatives.

SYSTEM ASPECT: SYSTEM/360 MODEL 92

Carl Conti
IBM Corporation
Poughkeepsie, New York

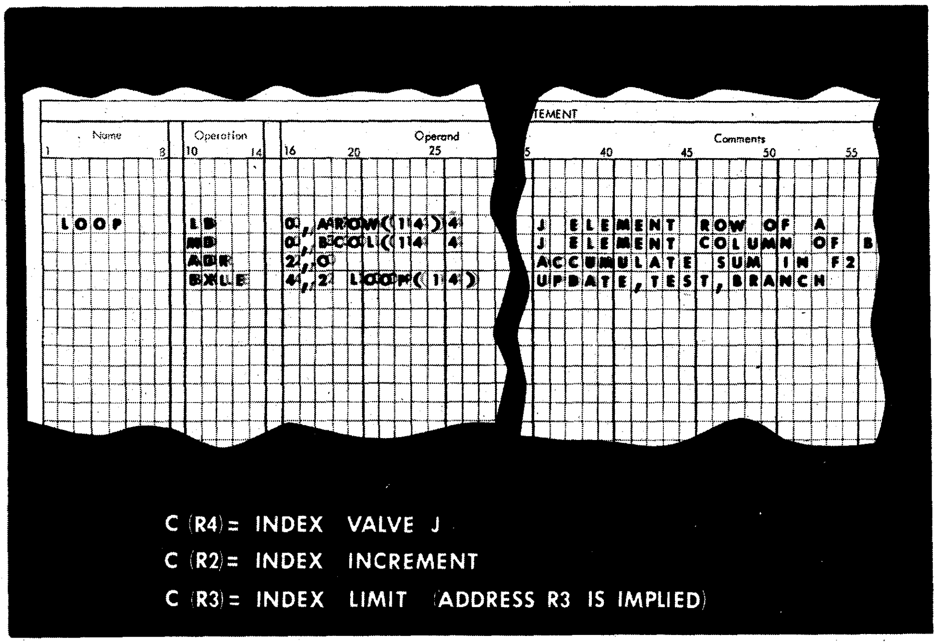
In the papers by Dr. Amdahl* and Dr. Chen† an extremely powerful CPU was described. I would like to illustrate on some problems just how powerful it is, in order to better pinpoint its actual performance. With a CPU of this sort, we had a few problems in the actual design: to be able to get problems or applications in and out of the CPU, and to put the processor into an environment where it is useful.

Unfortunately, when considering high performance, it is not possible to support the CPU with conventional I/O used in a conventional way. Several unique pieces of equipment are available with the 92 to provide the capability to process problems at the rates we can achieve in internal CPU performance. Before these are described, let us look at internal performance in a little more detail.

Figure 1 is an extremely simple example

* *The Model 92 as a Member of the System/360 Family.*

† *The Overlap Design of the IBM System/360 Model 92 Central Processing Unit.*



```
LOOP    LD    0,AROW(14)4        J ELEMENT ROW OF A
        MD    0,BCOL(14 4        J ELEMENT COLUMN OF B
        ADR   2,0                ACCUMULATE SUM IN F2
        BXLE  4,2 LOOP(14)       UPDATE,TEST,BRANCH
```

C (R4) = INDEX VALVE J
C (R2) = INDEX INCREMENT
C (R3) = INDEX LIMIT (ADDRESS R3 IS IMPLIED)

Name
Operation
Operand
TEMENT
Comments

Figure 1.

of coding. This is, in fact, the inner loop of a matrix multiply, where one matrix is stored in its transpose. We have an RX load, an RX multiply—that is from memory to register, followed by a register to register add to accumulate the sum; finally, we have an indexing branch (branch on index low or equal) which is analogous to a DO statement in FORTRAN.

We can look at how this particular program will proceed through the CPU by first observing a single iteration of those four instructions. This is illustrated in Fig. 2.

On the left, we have the instruction decode and memory address generation time. The instruction decode is overlapped in the I box with the memory address generation of each previous instruction. There is overlap here on a purely functional basis. Next is the memory fetch time for operands. At the right is the execution time showing the load above followed by the multiply, then by the Add Register to Register.

Now, clearly in a code of that sort, looking at a single iteration, you have a completely dependent series of operations in the load multiply add. They have to proceed in sequence so that if you looked at this as one iteration of the loop, it looks like a fairly long time to process. Because of the internal forwarding structure in the 92, however, when we get into the second iteration some interesting things happen. In Fig. 3, we have instructions 1, 2, 3, and 4 repeated again, showing the second iteration of that loop. We have a completely smooth flow in this case, and for this reason I have chosen this example. This is the kind of problem where things go very well. It is executing completely in loop mode; therefore, there are no instruction fetches shown here. The indexing branch, the BXLE, requires only three cycles, at which point the first instruction for the second iteration loop is fetched out of the instruction buffer down to the decoder and is begun in execution immediately. You will notice these three operations for load, multiply, and add for iteration 1 are somewhat overlapped with a load, multiply, add for iteration 2, even though, because this is a simple routine, both sets of sequential operations require the same floating register zero. So again, this is not poor code even though it does use only one register in that particular portion.

There are, however, other cases of code that do not run quite that well. Figure 4 represents a general range of CPU performance.

We have found in the problems we have looked at that we range from approximately 15 to 120 times an IBM 7090. We rose from as low as 15 times 7090, where the 92, and for that matter any other parallel machine, cannot take

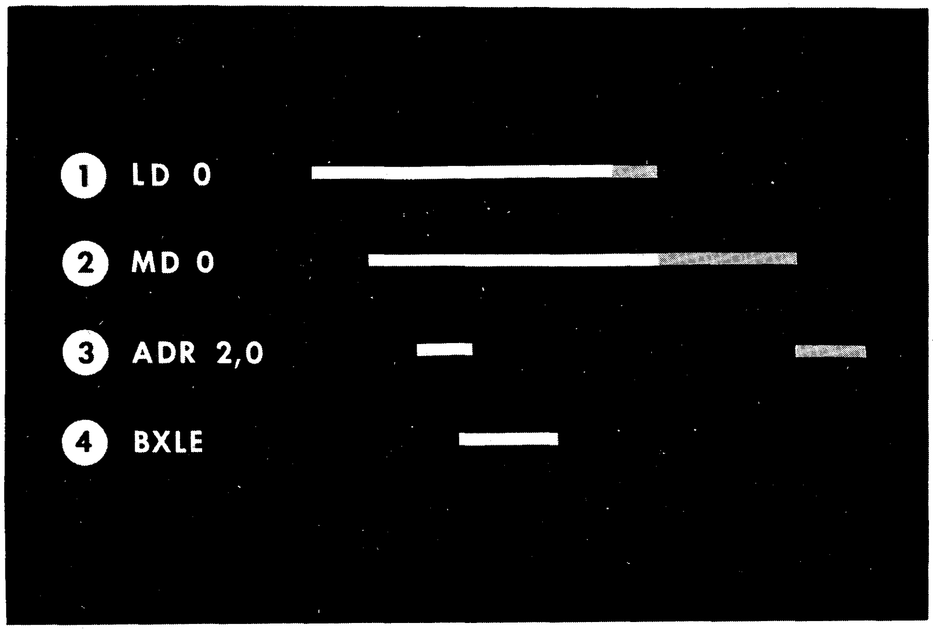


Figure 2.

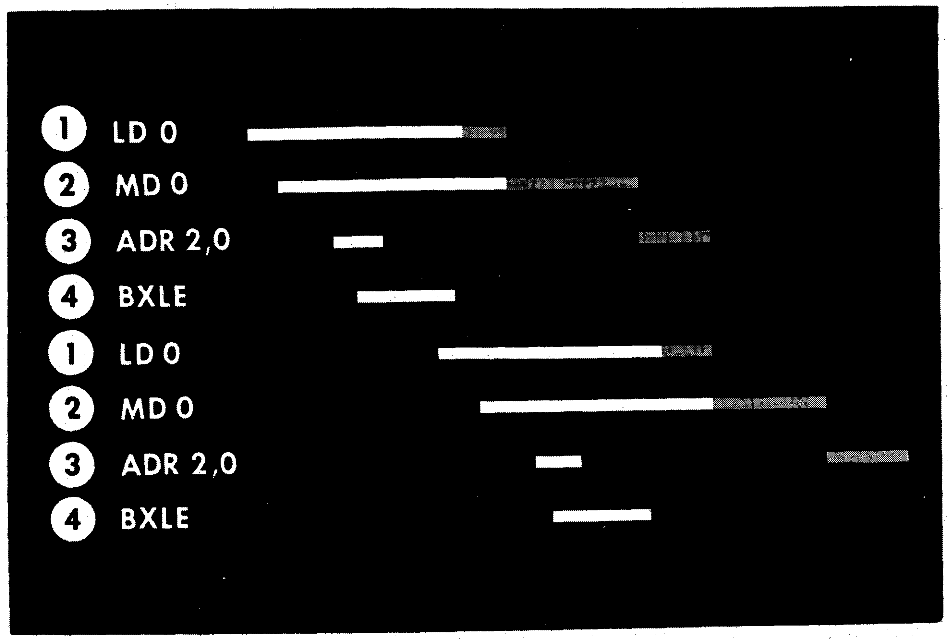


Figure 3.

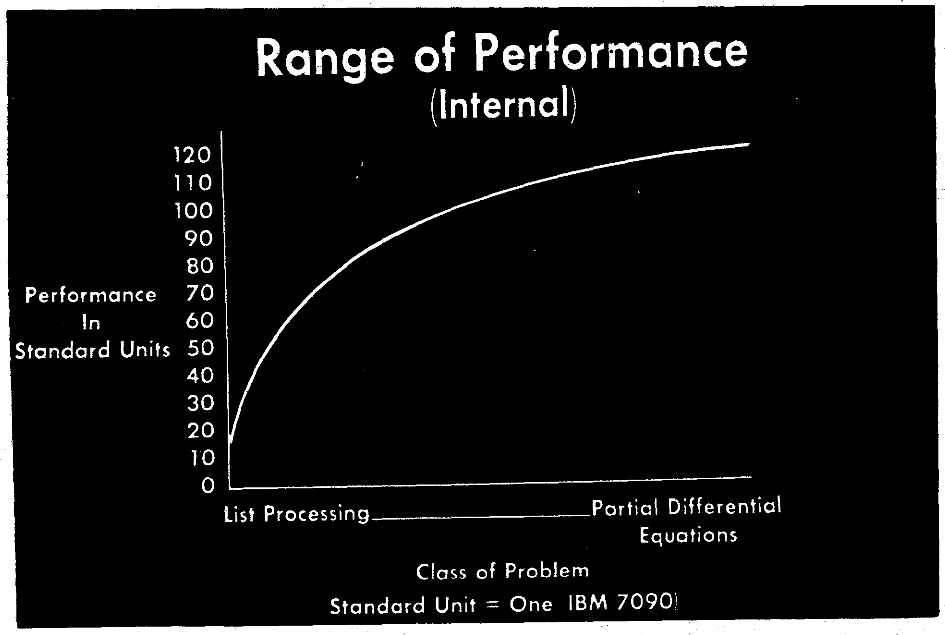


Figure 4.

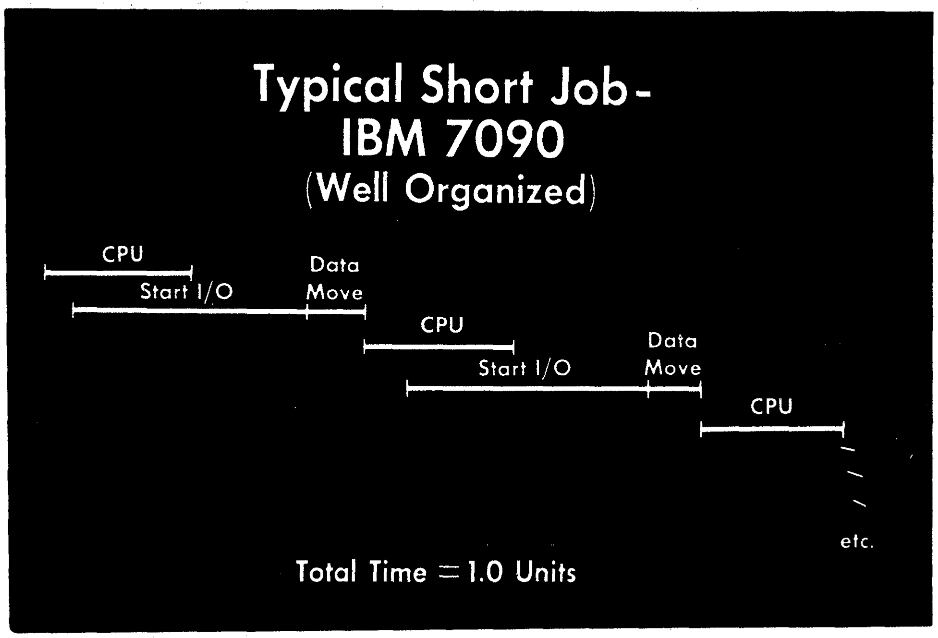
advantage of the parallelism in the processor, up to approximately 120 times 7090 for the partial differential equations type of jobs, the matrix problems, etc., where things proceed in a very orderly manner.

The list processing class here is characterized by a dependence on memory access time, and as one moves through a threaded list, it is impossible to get the data for the next piece of information until a previous piece of information has been obtained. All of the parallelism we can possibly design into a machine does not do much good when programming is done in that way. Compile falls at about the 80 mark, which is around the middle. Compilation takes advantage of the VFL and other instructions which aid scanning.

So far, we might say we are looking at

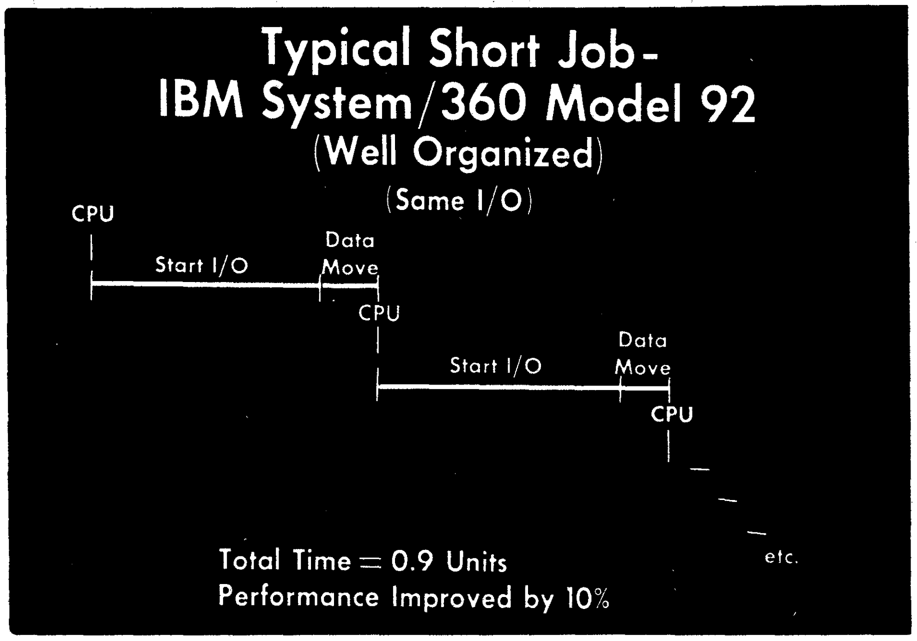
the extremely large problems that are found in a fairly limited number of installations. We have to worry about applying an extremely huge capability, or large piece of hardware, to perform small jobs as well as large ones. When we begin to consider this class of jobs which might be characterized by more I/O relative to compute than one might like for the high performance machine, we can go back to the 7090 and look at a typical—and I wince a little at my choice of the word typical, because I do not really think there is one—IBM 7090 small job.

In Fig. 5, in fact, there is a fairly well coded example of such a problem. Observe the overlap of the computer with the CPU time. This job probably came into the system complete with a program followed by its data, prob-


Typical Short Job-
IBM 7090
(Well Organized)

Total Time = 1.0 Units

Figure 5.


Typical Short Job-
IBM System/360 Model 92
(Well Organized)
(Same I/O)

Total Time = 0.9 Units
Performance Improved by 10%

Figure 6.

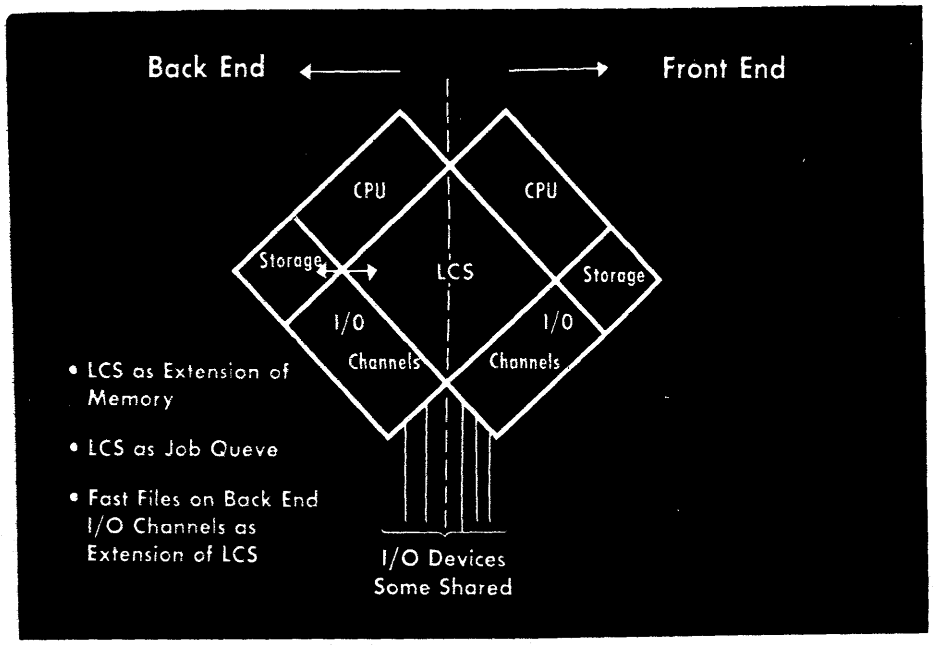


Figure 7.

ably originating in cards. It was converted off-line to tape, placed on an IBM 7090 on a tape containing program and the data with about 5 to 10 words per record in the data portion of the tape. We essentially have a situation where we do a little computing, and before finishing, begin the request for I/O for the next bit of computing we want to do, thus overlapping. We have here a minimum start time of about seven milliseconds followed by about a millisecond of data move before we can begin the next piece of computation, and we continue in this particular fashion.

Let us assume the user who has this program and this hardware buys a Model 92 and uses it exactly the same way. Then we have the picture in Fig. 6, where the CPU time is shrunk so that it's not even visible, but the I/O time has stayed the same. In that particular problem, we have shown one could run at about 10% faster than the 7090.

This is one way of increasing the amount of hardware in the field, but not, apparently, a very good way of making many people happy. It is necessary to solve these problems differently, and with much different I/O devices. In fact, we have proposed a number of alternate methods.

Let's look at a system organization as shown in Fig. 7. The key to support on a 92 is drawn in the center of this picture, which is LCS (large capacity storage.) This is a basic eight microsecond box of storage in which one can provide up to 2 million 64-bit words, on the Model 92. The storage shown is surrounded by other elements of the computer system, including CPU, I/O channels, and main working storage. The broken vertical line separates functionally the things one is planning to do when solving a problem in the system. To the right, we have the front end which we will assign at this point to basic dealings with the outside world—referring to the basic input of that little program that we were worried about earlier, and such. So, we have I/O devices on the front end; to the left, or back end, we will assign the functions of problem solving.

I am not necessarily implying a two computer or an (n) computer organization. Both of the CPU's indicated here may be the same computer; there is no necessary reason that they be separate. In fact, we allow both kinds of system configuration. The actual advantage of one over the other depends on application, so there isn't a single system which is right. There are a number of them, since there are a variety of uses they will be put to.

The characteristics of the CPU relative to LCS are identical to the characteristics of main working storage and the CPU. That is, the large capacity storage (all 2 million words of it) are directly addressable, so that one could execute programs out of large capacity storage or one could fetch data in the inner loop of the matrix multiply out of large capacity storage. Clearly there are cases—even with the interleaving of large capacity storage since it is a basic eight microsecond cycle—where one must pay very dearly for using it that way for the access time. We have found, however, especially in the list processing end of that performance chart, that the availability of extremely large storage is as important to system performance as is the speed at which things can be put in and out of that storage.

In addition, the channels also have interface with LCS identical to the interface of the channels to main working storage. Thus, you

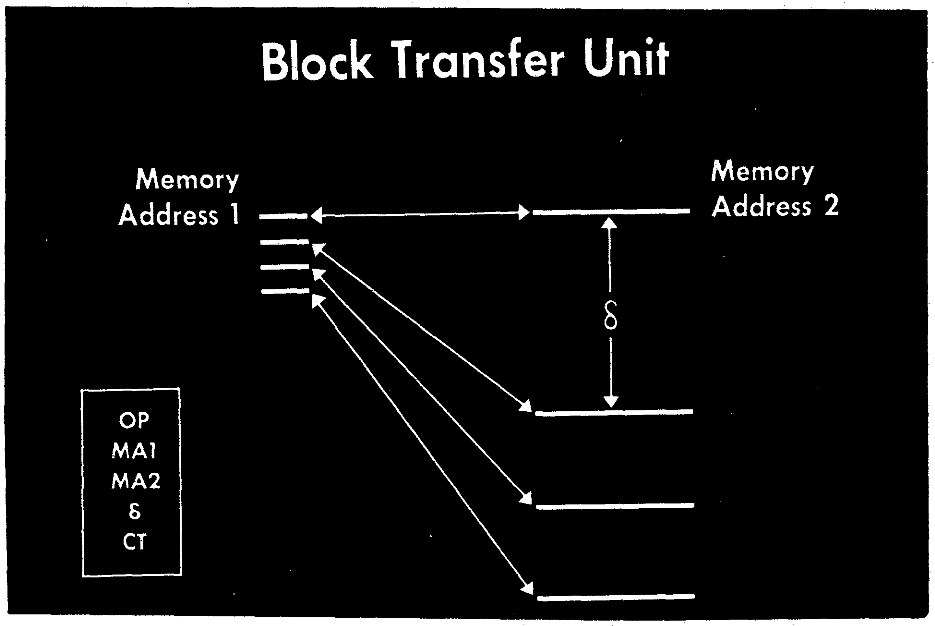


Figure 8.

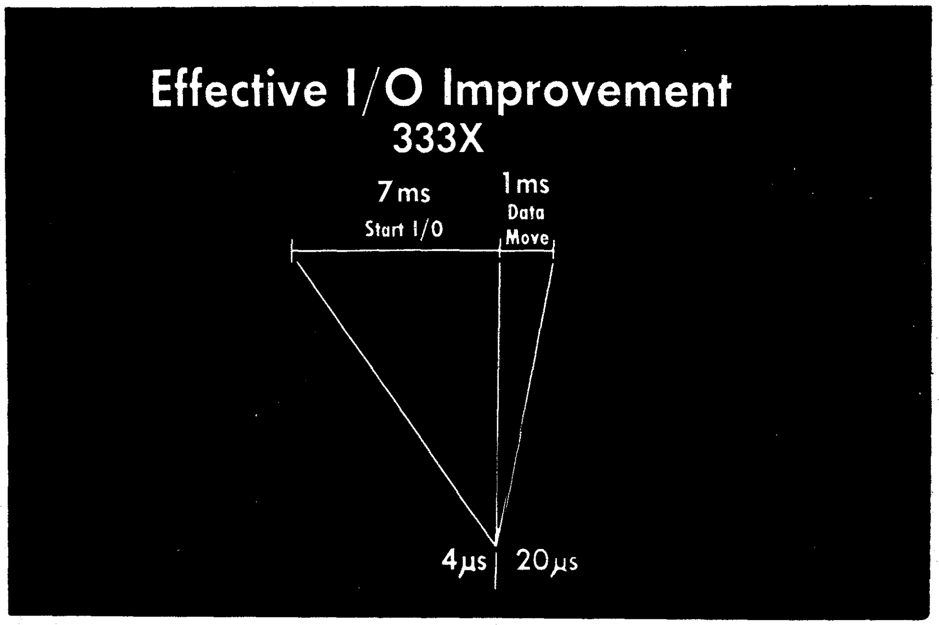
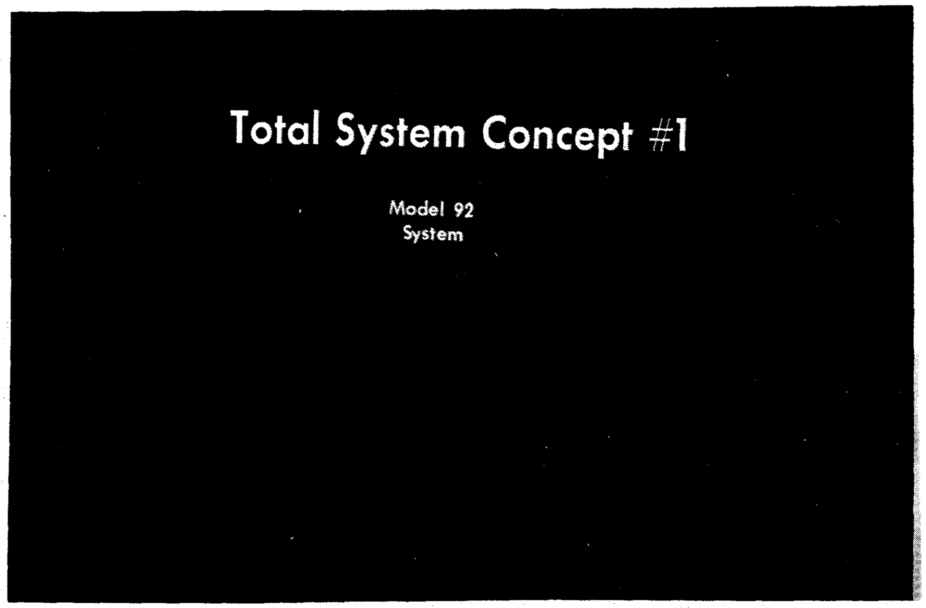


Figure 9.



Total System Concept #1

Model 92
System

Figure 10.

can move from I/O devices directly into LCS or directly into the main working memory and from memory back to I/O devices through the channels.

If we get back to that short job, let's envision that through this functional part of the system we place that job, its program, and all of the data in an extremely small corner of LCS. In solving the job, we then treat LCS by the back end CPU just like the tape device that the 7090 programmer was moving when he wrote his code. With all of the programming data here, we first move the program to main working storage, then operate identically to the mode we had for the IBM 7090, pulling the 5- or 10-word records of LCS into the main storage.

In the same operation, but using the storage channel which was announced with the Model 92, we can have rapid sequential transfers and somewhat non-sequential transfers of information from one area of storage to another. Transfers can be made on the storage channel for LCS to main storage and the reverse, and information can be moved between areas of the same storage.

A study of the characteristics of that storage channel reveal that it has some other interesting aspects. First, it operates like an I/O channel in that it is put into action with a start I/O operation. It proceeds to the conclusion of its particular operation independent of the CPU, save for the very small amount of memory interference, and causes an interrupt at the completion of the transfer. This particular transfer at a maximum rate of one microsecond per 64-bit word is accomplished just like an I/O transfer. Further, as illustrated in Fig. 8, there is a scatter-gather capability such that in the scatter mode one can go from sequential words in one area of storage to non-sequential words in another area of storage separated by a constant delta specified in the operation by the programmer. The opposite or gather case would be the transfer in this direction in which the delta quantity is applied to the source rather than the sync. An interesting aspect of this is that the matrix transpose can, in effect, be accomplished on the fly while bringing the matrix in from large capacity storage.

We can compare I/O moved this way with the tape device we had on the 7090. Note in Fig. 9 the seven millisecond start time, which is the access time of the tape, and one millisecond data move compared, for LCS, with a four microsecond analogous start time or access time and, with only four-way interleaving, a 20 microsecond transfer time, or a ratio of 333 times that on the 7090. In order to accomplish

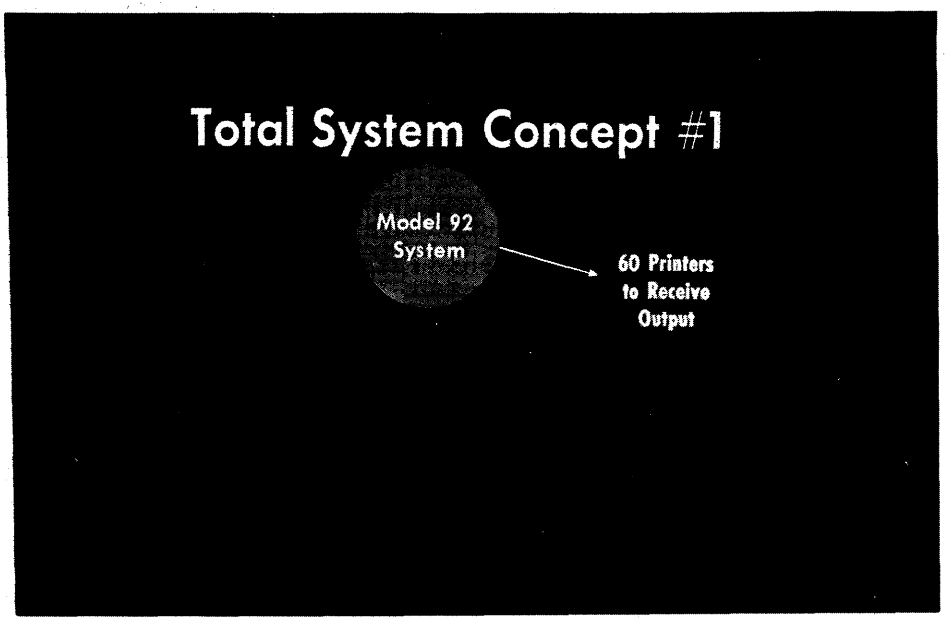


Figure 11.

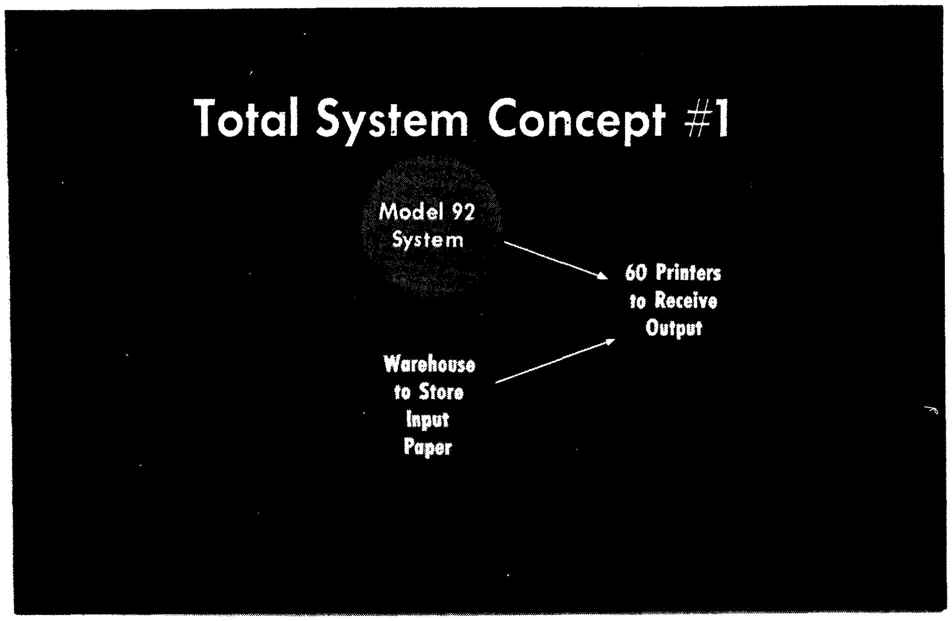


Figure 12.

this, the information must be gotten into LCS initially and, alternately, back out. Hence, the system must be expanded from that point, and there are several interesting phenomena that occur when this is done.

First, if we take a very simple approach, the original source of information has to be expanded considerably, i.e., in terms of number of card readers and such, over what the 7090 needs for a basic support. Next, to avoid making the numbers in this expansion become ridiculously high, new approaches to techniques of solving problems are also required. Let us just look at a few possible further system organizations. If we start in a circle (Fig. 10) with what has been described in the kite diagram as the Model 92 main system, then we can begin to add some of the things we need in support of this system.

As illustrated in Fig. 11, approximately 60 printers are required in order to handle the output information. Assuming for the moment that this is acceptable to everyone and proceeding with the design of the system, one then adds from here. Of course a warehouse will be needed to store the paper that is going to input to the printer (Fig. 12). Since warehouses are easy to build, this step is acceptable and the process can go on. Then the paper itself is needed (Fig. 13)—and this step might result in a lumberjack population increase of about 200%. As it has already been decided to build one warehouse, it will not be particularly difficult to build the second warehouse for the output

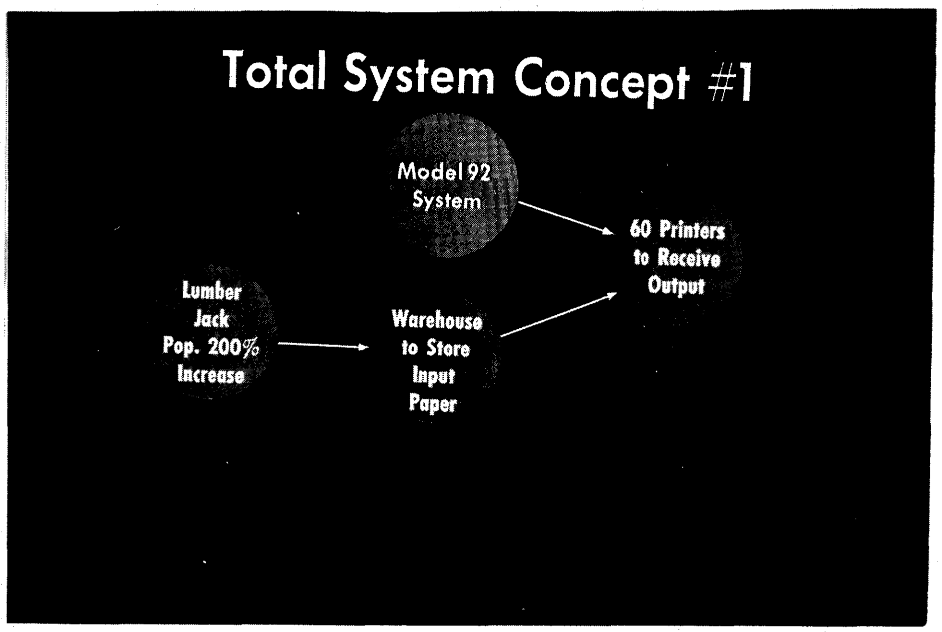

Total System Concept #1
Model 92 System
60 Printers to Receive Output
Lumber Jack Pop. 200% Increase
Warehouse to Store Input Paper

Figure 13.

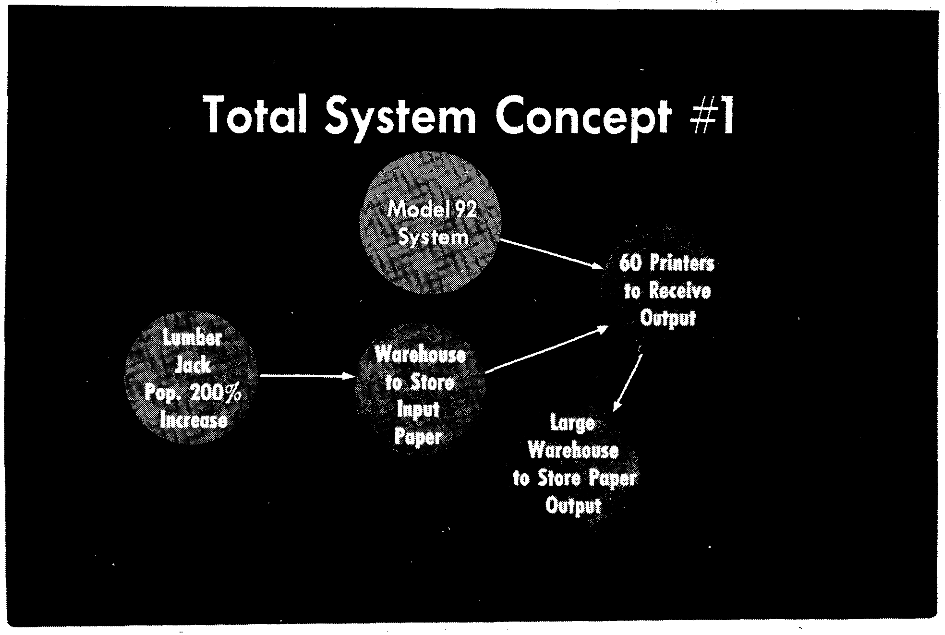

Total System Concept #1
Model 92 System
60 Printers to Receive Output
Lumber Jack Pop. 200% Increase
Warehouse to Store Input Paper
Large Warehouse to Store Paper Output

Figure 14.

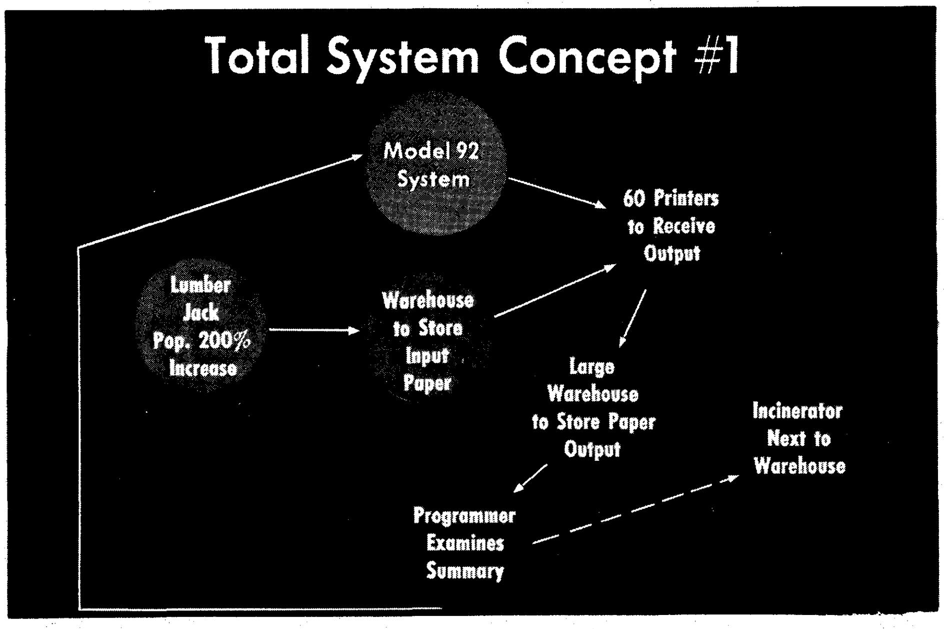

Total System Concept #1
Model 92 System
60 Printers to Receive Output
Lumber Jack Pop. 200% Increase
Warehouse to Store Input Paper
Large Warehouse to Store Paper Output
Incinerator Next to Warehouse
Programmer Examines Summary

Figure 15.

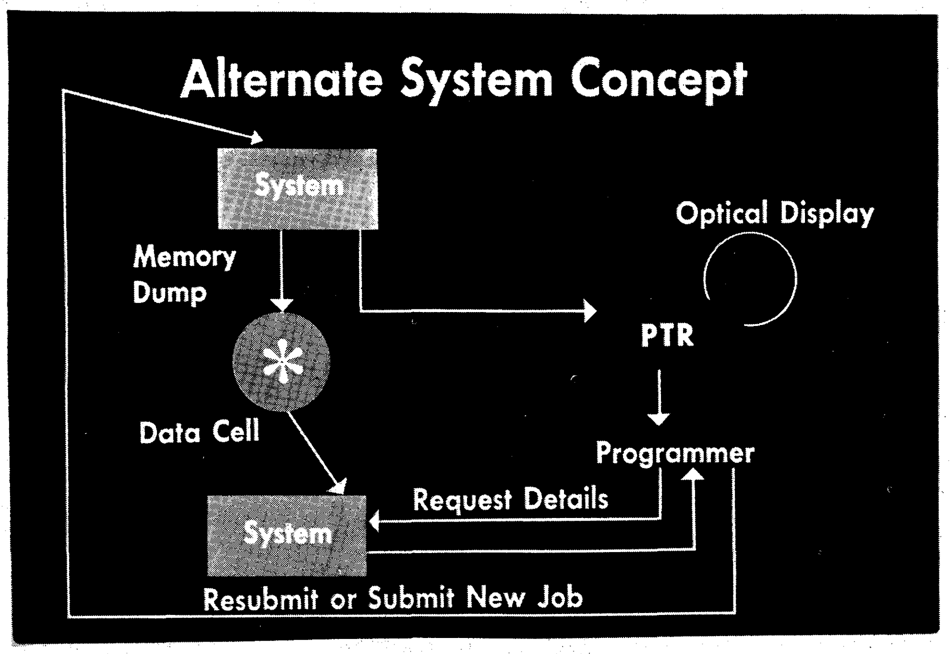


Figure 16.

paper (Fig. 14.). But, what is most disturbing in this situation is what happens when the programmer enters the picture (Fig. 15.). He first examines this paper. An incinerator, for him to burn it, is added, since he doesn't use it anyway, and he further makes the situation worse by being very nasty and re-inputting another job or that particular job which hasn't been debugged yet anyway. Things get even worse, because the net effect of that programmer—and only one is being considered here—continues to build up the cycle.

We are providing the equipment and a possible solution to this problem. The main piece of hardware that can help here is the IBM Datacell which will store 400 million characters of information in a tape strip fashion with individual portions, actually 40 million per portion, removable and storable on a shelf. A reasonable mode of operation for a system of this sort to avoid the massive print requirements is to store dumps on the data cell (Fig. 16). Rather than dumping memory dumps to a printer for the programmer, which usually represents most of the print load, we place it in the datacell. We then send to the programmer the summary information that he specifically requires with either very conventional devices such as on line printers or tapes that will be printed off line, or more sophisticated devices such as optical displays and very fancy terminals. This is to know whether the run is go or no go.

He then has at his disposal in the data cell the detailed information which he can request, let us say through the front end portion of the system. Again, I want to emphasize that the front end is not necessarily a separate processor; this concept is not particularly new, nor is it complete. There is a great deal of work to be done before we can solve all of the associated programming problems as well as the language problems. We have the hardware and a very good start on the programming, but as hardware designers we recognize that we are completely at the mercy of the programmers, both systems and users.

As we proceed down our design path, our next question is how good is the system or any other that we might build in a real systems sense. This is the subject of system evaluation. When starting out to evaluate systems, I think, historically we have looked simply perhaps at the floating multiply time or the floating add time, which if we did here, would look very fine for the Model 92, but it is not a true measure of system performance. As we have shown, it is not even a true measure of CPU performance.

To go further, I think we have passed a phase in which we have looked at instruction mixes. To calculate average instruction times, we have reams of data on 7090, etc., as to what instructions we execute and how often. The theory then is that one can somehow relate that instruction mix to another machine, perhaps get some alternate mix number which is the millions of instructions per second and relates the power of one machine to another.

Not all instructions in all machines accomplish the same actual quantity of useful work, however. The pure use of instruction mixes without very sophisticated translations from one machine to another is completely useless. The mere quoting of instruction rates is not very exciting because we can get factors of three or four differences in performance between machines that have the same identical

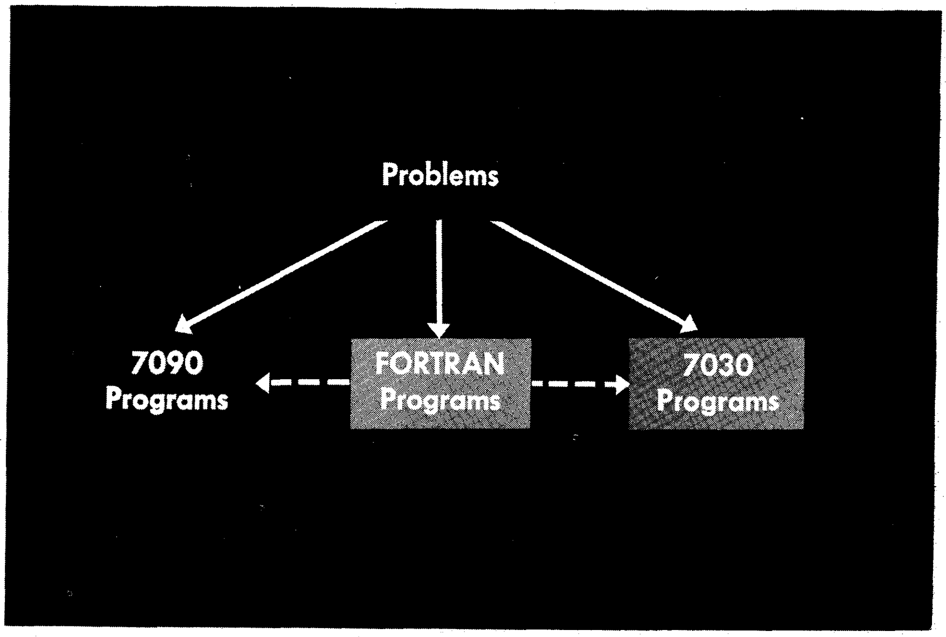


Figure 17.

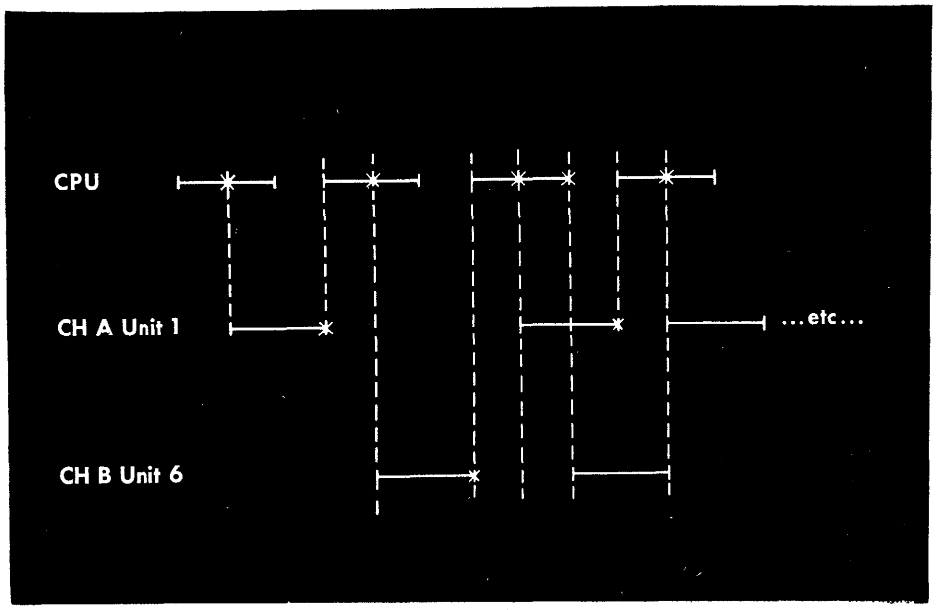


Figure 18.

mix rates. Therefore, we do not intend to continue on this particular path. An example here was the matrix multiply loop which took only four instructions on the Model 92. With the 7090 this would be perhaps twice as many instructions. There is no simple instruction ratio that can tell you anything about performance. Instead, there is the rate at which you can actually solve problems in a system environment. This is a meaningful number.

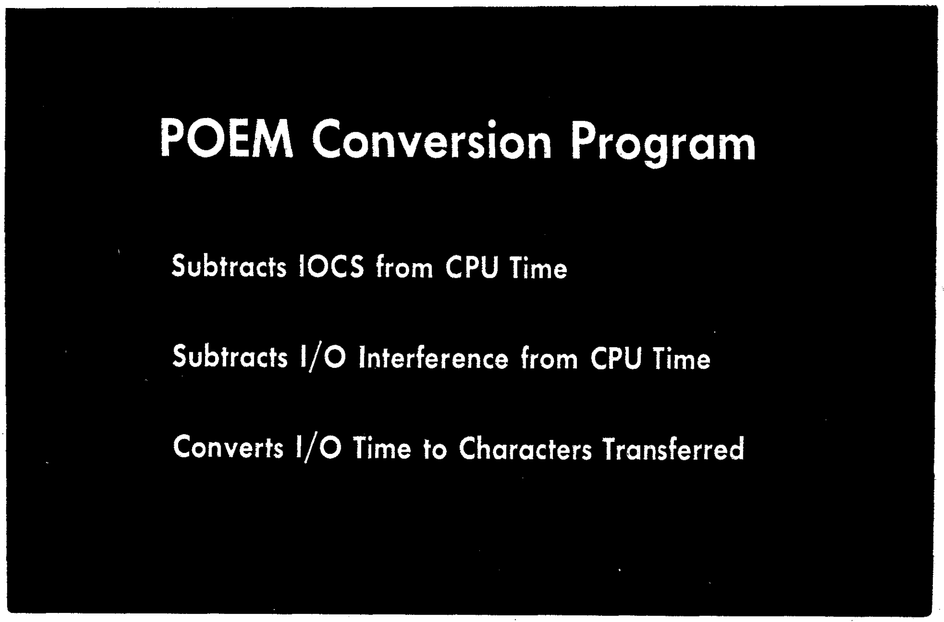
This is what we have a fairly elaborate scheme for doing. As it happens here, of course, there are many ways of looking at a problem. One way that we propose is again through the experience with a 7090 class of system. We can look at problems already solved—although this is not an ideal procedure because we are talking about problems of the past, not the future—and the way they were solved on a 7090. In Fig. 17, FORTRAN is separated from the machine language, since it is somewhat machine independent.

Looking mainly at problems run on a 7090 now and in the past, we have built a hardware device, called POEM, which we attached to the 7090 giving us a dynamic trace of the actual system performance on a problem for the 7090. Let us look at some of the characteristics of POEM in Fig. 18. It gives us a dynamic 7090 profile; it attaches into the hardware monitor interface on IBM 7090/94 type equipment and produces a self-contained tape record of what went on in the IBM 7090 during the solution of the problem. For each individual portion of CPU time and each individual I/O time there is a specification on the tape. There are further dependencies shown here as evidenced by the dotted lines that demonstrate that a particular I/O action did not proceed on the 7090 until the CPU had gotten so far. I have shown two channels here, since this is fairly normal.

We then get a very long tape with information of this sort on it related purely to the 7090. That information as such is not directly usable to analyze the performance of any other system other than the 7090, for which we have many numbers, anyway.

We have developed a set of conversion programs called POEM conversion programs that simplify and purify the picture that POEM produces. It does not end up in a completely pure picture but a less impure picture than it started with. Let us look at Fig. 19 and see what the POEM conversion program does. It simply makes an attempt to subtract from the CPU portion of the 7090 time the IOCS overhead, since we are not interested in that IOCS. It subtracts from the compute time that amount which was added to the pure compute time because of interference by I/O devices in memory with the computation. Again, this tends to shrink the 90 CPU time and convert the I/O time as indicated on the POEM tape to a transfer of (n) characters, for instance, rather than time to transfer the characters. This makes this particular line device independent so that when we look at that particular tape in respect to a 360 system such as a 92, we can plug in lots of different devices and experiment as to just how things go.

The net intent here is to provide through the POEM conversion program an input to what we call a COMPASS simulator. We can then put the information on the work that was done on a 7090 through this path to a systems simulator for System/360 including the 92 (Fig. 20). We set as parameters to the simu-



POEM Conversion Program

Subtracts IOCS from CPU Time

Subtracts I/O Interference from CPU Time

Converts I/O Time to Characters Transferred

Figure 19.

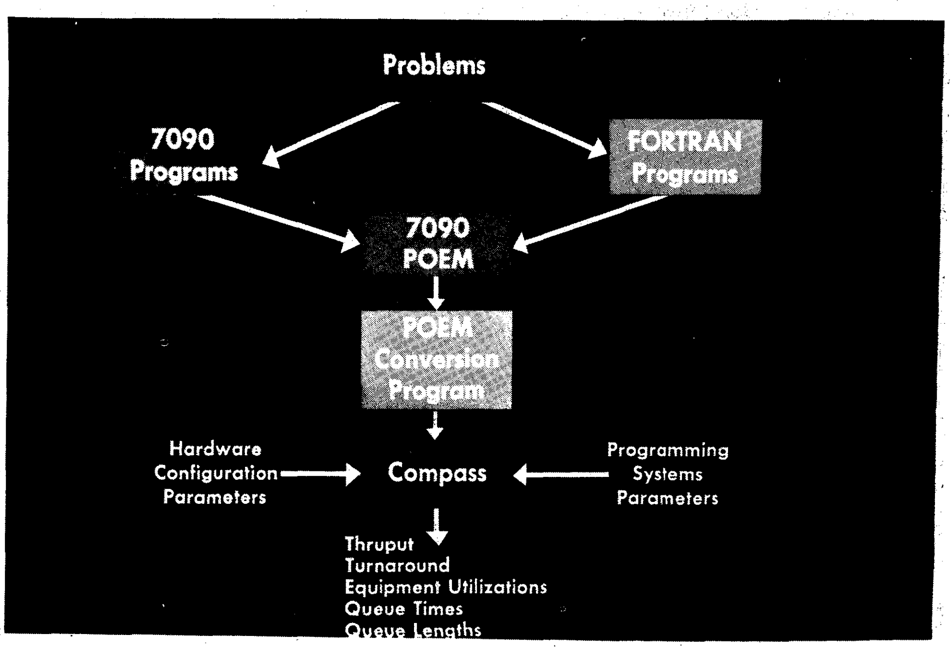


Figure 20.

lator the specific hardware configuration that we are currently looking at in the simulation run. We allow all of the flexibility in the simulation that is allowed in the hardware connections themselves, i.e., in all of the shared memories between CPU's, shared I/O devices on channels, pooled devices, etc.

We also specify—and there is another simulator involved here—the programming systems parameters such as compile speeds, algorithms employed in the control section, etc. And we get out hopefully accurate information on thruput which is a little bit of an undefinable number, but represents essentially the amount of work done in a given time, turn around time from input of the job to output conclusion, then of course both of these in, statistical form as well as individual form. It will also indicate equipment utilizations and queue lengths: how much of the time of the CPU will be actually used, how much of the time with each I/O device, and so on. It will indicate queue lengths and queue times on each device.

What we have done with this program to calibrate it is to take that complete path from the 7090 and to plug in hardware configuration parameters such that we make the hardware we are looking at look exactly like a 7090. Now, we have run problems on the 7090, then passed them through our set of programs with the hope that we will get a number simulated which is identical to what we measured with a stop watch on the 7090. In fact, on the jobs we have done in this way we've been within 8% of being identical in simulated time to 7090.

The basic use of this is to explore various hardware and software configurations for machines like the 92 so that we can hope to optimize them. There are other ways of getting information into the simulator and Fig. 21 essentially represents a fairly complete picture of some of the other things we are doing to evaluate systems. Among them are counters, one of which is a modification of the direct couple system that we have in Poughkeepsie available to us which allows a front end IBM 7044 (instead of operating as a direct couple system) simply to monitor the back end IBM 7094 operating in stand alone fashion. Then we just use the 7044 as a big counter and get very elaborate statistics on instructions processed in the 7094. We have a program called the MAP timer which can take a 7090 instruction mix and convert it through a comparison to 20 other programs we hand-coded to a System/360.

There is some meaning to instruction mixes if you have a reasonable conversion procedure. In MAP, a conversion procedure ac-

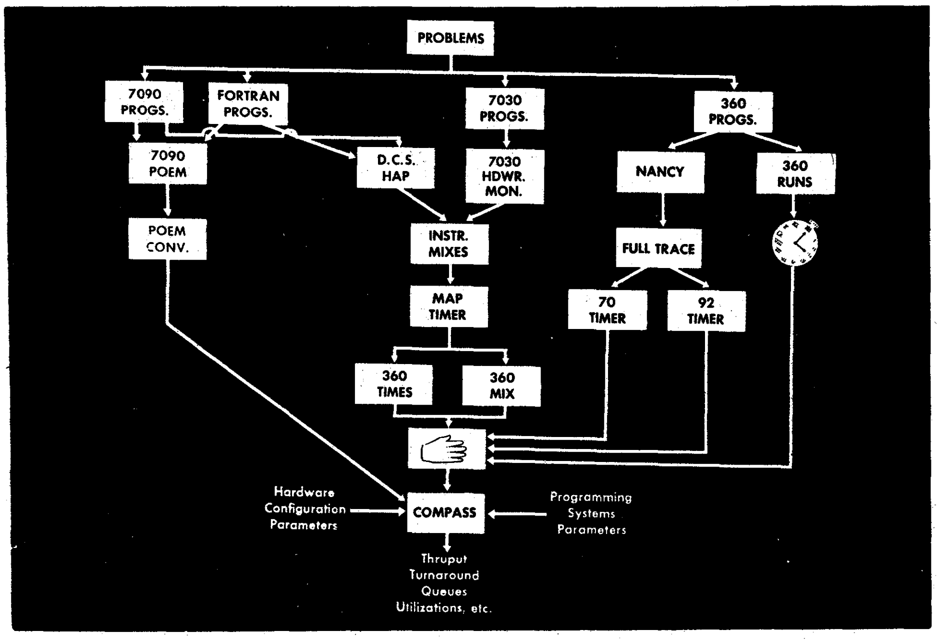


Figure 21.

tually works on a linear combination principle of the instruction mixes and the performances, etc. From this program we can get for things run on the 7090 actual CPU times for 360 machines and instruction mixes which we can input after some hand massaging to the system simulator compass.

We also can go through a similar pass on 7030 where we have a fairly elaborate instruction counter device attached to give us these instruction counts. The original scheme described has in it a number of constraints on the 7090 (Fig. 21) that we are measuring; but they may not necessarily be true constraints on the system under evaluation.

For instance, the fact that you've got two channels means you have only two channel actions going on at once, so there are artificial dependencies there. What we will get out of the final information here is a lower limit on the 360 performance, although we're not too concerned with this fact since we feel that the relative value of different systems is still quite accurate and not necessarily lower limit.

As we move to the right and do a little more work we are getting more accurate information for the 360 system, such as the Model 92. Now as we go clear over to where the problem is coded and perhaps run it on some models of 360, or simulate it, then you have the completely accurate time. However, you have also done a lot more work since you must code the problem. The answer, then, to what the system performance is, is that there is no single number; it is going to depend on what is done with the system, but it is our design objective to handle large problems. We also intend to handle lots of small problems, and it is our design principle to handle a combination of the two.

A PHILCO MULTIPROCESSING SYSTEM

Herbert S. Bright
Philco Corporation
Willow Grove, Pennsylvania

INTRODUCTION

Although the purpose of this paper is to describe the Philco Advanced System and the design and application considerations that gave rise to it, it will be necessary to review in some detail the Philco 213 System. The two systems are quite similar in concept and organization, and differ only in three respects:

1. *Speed*—The 213 System operates in the vicinity of one million single-address instructions per second for each of one to four processors; the objective for the Advanced System will be at least four times that speed.
2. *Technology*—The 213 uses present-production discrete-component circuits, and ferrite cores for both main memory and bulk memory; the Advanced System will use integrated circuits, thin-film main memory, and ferrite core bulk memory.
3. *Vocabulary and Unit Organization*—The Advanced System will be totally upward-program-compatible from the 213, but will contain additional facilities both in machine language and in internal organization of system units.

In order to explain the reasoning that has led to many of the new characteristics of these systems, this paper will review, at an elementary level, the characteristics of, and several problems associated with, multiprogramming and multiprocessing.

1. OBJECTIVES

This section discusses, not in any order of importance or urgency, the objectives Philco seeks in defining the Advanced System to follow the recently announced 213.

Definitions: Two terms that will be extensively used in this paper are defined as follows:

- *Multiprogramming* is the time-sharing of processor(s) by a number of not necessarily related programs simultaneously present in main memory; the number of programs may be larger than the number of processors.
- *Multiprocessing* is the use of two or more self-sufficient processor units with a single, logically continuous and jointly addressable main memory.

1.1 Numeric Computation Power Growth

The demand continues for greater and greater speed and flexibility in performance of numerical calculations. Experience with lower-powered computers has given rise to descriptions of large physical problems that justify more detailed solutions of more realistic numerical models, which are potentially capable of saving substantial calendar time and of producing more accurate physical analyses and better engineering designs.

With the advent of general-purpose "computational colloquy" time-sharing systems for large machines, it seems clear that the great efficiency, flexibility, and generality of even higher-powered hardware and software can be applied to small problems conveniently and economically. The awesome reserve power of the multimillion-operations-per-second numeric engine is on reserve as required.

1.2 Non-Numeric Computation Power Growth

Even in scientific laboratories devoted to large numerical calculations, the workload of processing non-numeric tasks (program compilation, natural language analysis, complex problems in mathematical logic) grows steadily and in many installations has already preempted several hours' machine time per day. This work is characterized by a relatively small number of macro-operations that are compiled into many programs, are composed of a relatively large amount of character moving and testing, and are executed with high frequency.

The 213 adds to the 212's vocabulary a number of character-string testing and manipulating hardware macro-commands, which provide modest improvement in program space and dramatic improvement (in significant cases, up to 40:1)* in speed of execution of some important functions. The Advanced System will extend the 213's vocabulary as indicated by analysis of actual 213 program workloads.

1.3 Computation Economy Improvement

The 210[1] offered a few-hundred-percent increase in computation capability, as compared with the highest-powered and most economical vacuum-tube computers of the preceding generation, at an operating cost increase on the order of only 10%. The 211 doubled the 210's capability at another 10% cost increase. The 212 offered a fivefold increase in power over the 211, at only another 20% cost. With each of these machines, problems that had been handled on earlier computers could be solved at substantial cost reduction per unit workload. In addition, the effective buffering of data flow between main magnetic core memory and magnetic tape used as auxiliary memory permitted efficient solution of large problems; in the earlier machines, such problems would have been unreasonably inefficient unless limited in size by core memory.

The 213 provides new kinds of problem-solution power and flexibility, in addition to further reduction in the cost per unit of computation workload.

The Advanced System will continue the trend, with a capacity increase of at least a decade, at a much smaller increase in total system operating cost.

Highly generalized multiprogramming capability has, within the past two years, come to be recognized as a potentially valuable tool for solving a frustrating problem, viz., the low effective speeds of large computers on existing workloads as compared to the instantaneous or internal speeds of which the machines are capable. Even the most highly refined types of programs seldom utilize more than 70% of system capability; i.e., the central processor is computing effectively less than 70% of its observed productive time. Careful observations on large working systems by sophisticated users have disclosed effective utilization of CP time, averaged over the computing day, between 30% and 50%; hence, a throughput improvement by a factor of 2 or more may be available through highly efficient utilization techniques *without increasing processor internal speed.*

For time-sharing users, the continuing improvement in unit economy of computation service from super-powered computing systems is coupled with another kind of computation service economy: while each user may be committed only to input/output facilities as needed and to average workload cost, he has available *upon demand* the huge power of the high-powered system to satisfy his peak requirements. In many cases, this could avoid the need for a user with sharply peaked workload to acquire peak-capacity equipment; in others, it could permit the economy of specifying an occasional large problem in optimal fashion rather than as constrained by small- or medium-scale on-site equipment.

1.4 Turnaround Time Improvement

The values and the problems associated with improvement of service delay for conventional computation jobs in a batch-monitor en-

* As an example, the 213's Move-Break instruction performs, in about 15 microseconds for the first 8 characters and 2.4 microseconds per 8 additional characters, logical operations that require about 380 microseconds per 8 characters on the 212. This instruction will be explained in Section 3.2.

vironment have been discussed at length in a previous paper.[3]

A detailed report on an actual installation workload, and its service delays before and after a tape-to-tape monitor system (with off-line device operation) was replaced by an all-on-line dynamic-rescheduling system, was given by Kory and Berning.[4] This system created queues that were either at least fifteen minutes duration or comprised at least five jobs, selecting from among queues not yet executed by consideration of priority class, arrival sequence, estimated running time, and estimated output volume, qualifying run sequence by checking available tape units and tape requirements.

Kory and Berning observed an average reduction in turnaround time from 4.1 hours to 1.6 hours. (It may be noted that this service improvement in a short queue rescheduling system was not, contrary to the prediction by Patrick,[6] achieved at the cost of lower system efficiency.) Kory and Berning observed an average reduction of setup time from 1.3 to 0.5 minutes per job, resulting in an overall reduction in machine time required to handle the same volume of workload.

With regard to I/O tanking space, a simulation of their proposed system prior to its installation, using previously-recorded dual-7090 workload data, predicted that ". . . the maximum backlog (cards plus print) on disc would not exceed 2.4 million (six-character) words." This conclusion supports reasonably well the original design decision by Lavine[5] (discussed in Section 5) for the Philco monitor 212SYS, to allow a minimum I/O tanking space of 1.5 million (eight-character) words.

We found the Kory-Berning paper to be particularly significant in that it seems to be the first occurrence in generally available published form of a before-and-after study of a dynamic rescheduling system installation, supported by meaningful empirical data taken from comprehensive records of an actual workload.

It demonstrated neatly that significant improvement in both turnaround time and system efficiency can be achieved at the same time merely by optimizing over a few workload parameters the sequence in which short job queues are executed. We feel that the results are conservative, since none of the compilers were rewritten to utilize disc files.

The proposed system will extend the method by optimizing run sequence over more parameters and by subdividing unit runs two levels deeper, to segments of individual jobs. By means of multiprogramming, the system will switch a processor to another task when one job sequence encounters a delay such as an I/O request or in response to a priority change. Not only will system efficiency be considerably higher, but the effectiveness of job segment run sequence optimization should be improved substantially. Thus this system should attain an even greater improvement in turnaround time, as compared with a conventional modern tape-to-tape monitor, than have the first-generation dynamic reschedulers.

1.5 Computational Colloquy by Time-Sharing from Multiple Consoles

Bauer proposed a concept that has received wide acceptance: computer service should be available as a public utility, in the image of electric and telephone service, with each user able to command almost arbitrary amounts of service. Bauer perceived that it would be necessary to provide for automatic assignment, by a powerful executive system, of memory space and of a multiplicity of central processors. The central theme of the present paper seems almost to be a specific answer to Bauer's challenge of six years ago.

It has been demonstrated at several pioneer installations, including Massachusetts Institute of Technology, Carnegie Institute of Technology, RAND Corporation, and System Development Corporation, that prompt-response dialogue between several concurrent users and a single computing system can result in sharp increases in speed of program checkout and in progress rate on research projects. Such man-machine communication also makes practicable concurrent access to large files by many requestors for very small data manipulation tasks, which may be coordinated with generalized file maintenance. The status of the MIT and SDC projects as of mid-1964 has been summarized by Fano[7] and Schwartz[8].

This concept of computer utilization is different in principle from the now-conventional automatic monitoring of "jobs" as submitted to a modern computing center. Instead of an entire computer request-for-service being prepared in advance and submitted as a unit, a colloquial user communicates with the system through a two-way console, entering elements of his request in serial fashion and with system response possible at many points during the communication. For instance, a small program could be keyed into a console in source language, one character at a time; compilation could be requested from the console; compilation diagnostic comments may be presented to the requestor via the console; execution may be requested; results may be presented; another program may be requested to be brought into active status from quiescent status in on-line mass memory; conventional debugging operations may be requested and results observed either in whole or in selected parts; all by user activation of a keyboard or other input means and by system response to the user through typeout or other visual display.

The distinguishing difference between the two modes of machine use is that in "job" mode, a machine run is prepared and submitted as a single entity, and results are returned in a single printed report after run completion; in "colloquy" mode, preparation, request, and return of results occur as a continuing chain of events with elements of each interspersed among the others without any predetermined sequence.[2]

This type of operation becomes particularly effective if a high-powered hardware-software complex is accessible through a user's console. Since much of the user's time is spent in manual or mental operations when accessing a fast machine for short bursts of computation service, it is desirable to serialize or "time-share" the central system.

Early time-sharing experiments were performed using multiprogramming techniques to permit a few users to memory-space-share as well as to time-share the system. Hardware then available was inadequate to permit accommodation of a large number of users with reasonable efficiency and response speed. The first relatively large-scale experiment[2] made use of a brute-force scheme of memory swapping from on-line mass memory.[3] It achieved reasonable system efficiency, with fractional second keyboard response and fractional minute service request response, for up to a few dozen concurrent "foreground" colloquy users, with any available central machine time made available to service a "background" job-mode workload.

At the present time, several time-sharing systems are being planned, to accommodate from a few dozen to a few hundred concurrent on-line users. Different approaches are being taken to several aspects of system planning: some provide console multiplexing under direct control of central processor(s), while others use auxiliary communication processors. Some use dialed-up connections from a large number of remote locations, permitting only a fraction of the stations to be on line at a time; others have all stations connected full-time. Some retain active information in core memory and attempt to minimize communication with user files in disc or drum; others process all information from the rotating machinery, using core memory primarily as a buffer, as regards processing of user files.

The 213 and Advanced Systems have adequate flexibility and power to handle conventional I/O devices as well as multiple remote keyboards directly from the central processor(s), although present plans call for use of one or two Philco Type 170 Communications Processors to handle the remotes.

1.6 Full Generality

Of central importance is that the new system must be applicable to all classes of computation work for which conventional serial single-processor computers have become useful. Its utility must not be dependent upon development of new mathematical or programming methods.

One way to meet these requirements, and the way chosen for these machines, is to have a processor execute a program segment *seriatim* in the manner of a classical Von Neumann computer; i.e., instructions are executed serially and in an order that is completely dependent upon each program and the data supplied to it.

1.7 Efficiency with Procedural Languages

For predictable system economy it is important that the machines be capable of performing calculations by executing object language programs compiled from source language programs written in existing procedural languages at "full efficiency," i.e., it must be feasible to construct fully automatic compilers that produce object coding comparable in compactness and in execution speed to hand-optimized machine language coding.

We have rejected, on the basis of inadequate generality, schemes that achieve full speedup only for short instruction loops, requiring that those loops be executed from a finite-length string of special registers.

1.8 Graceful Degradation

A large system requirement that has been given close attention in the past by military users is now assuming significance in the plans of advanced engineering-scientific and commercial computer users; the system should not go out of service in the event of failure of any unit, even a central processor. A fault indication should evoke automatic test procedures. If found defective, a unit should be withdrawn from service and the system should continue in operation with reduced capacity. The executive function should not assign further tasks to the defective unit until that unit has been returned to the system by maintenance personnel and has been found, by tests performed under executive control, to be fully operational.

Such testing, withdrawal of equipment from assignment to system tasks, proof-testing after repair, and re-integration of equipment into the workload pattern, ideally should be performed without manual intervention.

This resilient performance under equipment fault conditions, sometimes called "crippled mode capability," requires that the system be highly modular in organization and operation and that more than the operational minimum number of each type of unit be present. There must be at least two central processor units that must operate cooperatively and anonymously. Memory must be organized so that at least one major unit of main memory and one of mass memory may be withdrawn from service while the operation continues. Similarly, the system must be capable of continuing operation with at least one of each type of input/output device out of service. Finally, at once the most difficult and the most important consideration is that switching redundancy must be built into the system so that no single fault, however severe, can prevent a successful reconfiguring under program control into an operational crippled-mode system.

In view of its potential importance in large-scale computing systems of all types, the concept of crippled-mode operation has received surprisingly little attention in recent literature. The concept was considered briefly by Shafritz and others[10] in the context of a discussion of stored - program - computer - controlled communication switching systems.

A related but separate consideration, which becomes significant in the case of a system fault (such as loss of an area of memory containing active-status-list information) that cripples the executive function itself, is that of system rollback and restart.

In order to guarantee restart capability after an unscheduled and perhaps disastrous interruption of operation, it is desirable to perform periodically a total dump of main fast memory, critical tables in main slow memory, and all processor registers. It is also necessary to enforce system conventions that avoid the possibility of irrecoverable faults; thus, for example, a previous-state copy of a file being updated must be preserved, together with source data for the updating, until a later reserve-master file has been confirmed as correct and has been placed in protected storage.

In the past, system dumps to conventional types of tape or disc equipment have been inordinately expensive because of the excessive time required. The Advanced System will use the High-Speed Drum for system dumps, requiring only one drum revolution (just under 35 milliseconds) per 32,768 words of fast memory; thus, a typical 131K system with up to four processors could be dumped in less than 1/7 second, and dumping once per minute would consume less than 1/4% of system time. (The Executive will minimize collateral time losses by taking such steps as choosing, if possible, dump times to occur when no I/O is waiting.)

In both the 213 and the Advanced System, Memory Control is decentralized for two reasons:

a) Sections of Memory Control are placed in close proximity to processors in order to minimize cable delays, especially for those signals requiring response and hence subject to round-trip delays.

b) Decentralization limits the memory area that can be disabled by any one fault, even in memory switching; in a two-processor 213 system with total memory size an integral power of 2, for example, no more than half of memory can be disabled by any one fault, including loss of memory switching or loss of power supply to any unit.

One of the most comprehensive systems of the resilient type that has been specified to date is the second phase of the AUTODIN (AUTOmatic DIgital Network) System. Philco is now designing this system[15] and is studying the applicability of the concept to a broad range of computer systems.

The Advanced System is intended to be capable of operating as outlined above, with hands-off system response to unit-disabling faults by conversion under program control to crippled-mode operation and with limited loss of information even under loss of current executive data.

1.9 No Reprogramming

A frequent goal in the development of computers has been that programs operational on one type of machine shall operate, without any conversion effort, on a later type of machine.

This absolute upward compatibility was achieved literally with the Philco 211 following the 210, and again with the 212 following the 211. 212 programs will be operational without change on the 213, and 213 programs on the Advanced System.

In each case where a program is to be operated in a machine other than that for which written, not only the hardware environment but the software environment must be compatible. Thus either the same executive-monitor system must be used in both machines, or appropriate steps must be taken to adapt to the second executive the program under consideration and its system and library routine calls.

In order for upward compatibility for programs to be realized, it is necessary that a new machine not only logically contain the previous machine, but that all I/O operations be absolutely *program-compatible.* (It is not necessary, except for file-handling convenience, that they be *device-compatible.* Thus Philco 90K ch/s and 240K ch/s magnetic tape transports use different physical tape, and Philco 120K ch/s IBM-compatible magnetic tape transports use not only a different type of tape, but a different reel of a different width; yet *the same program* will operate without any change on Philco 212 or 213 computers using any of the three types of transports with appropriate tape, provided the programs use the fixed block size (1024 characters) that is required for the 90K ch/s transports, or on either the 240K ch/s or the 120K ch/s transports with any chosen block size.)

A plan now under study in this organization would obviate or greatly reduce the magnitude of program adaptation to a new executive. This plan would place the original executive system under control of the new executive, which would utilize the original library and system routines. If successful, this will permit 212SYS and programs running under it to operate as a single job under control of 213SYS, a new executive with multiprogramming and multiprocessing capability.

1.10 "Common-Access" to Routines and Data

This objective, and our means for attaining it, are discussed in Sections 4.1, 4.2, and 4.3 below.

2. SYSTEM PLANNING FOR MULTIPROGRAMMING AND MULTIPROCESSING

2.1 Introductory Remarks on Multiprogramming

In the context of the present paper, the primary purpose of multiprogramming is to keep the central processor or processors occupied with useful work as nearly continuously as possible.

Thoughtful users realize that a typical computer job includes substantial central processor wait time. If input/output is not well-bufferred, as is surprisingly often the case, lost mainframe time may even be visually noticeable (at a maintenance console); such a circumstance in a high-powered system is, and rightfully so, abhorrent to the cost-conscious user. The magnitude of central processor time wasted through I/O waiting alone may, in even a well-managed installation, exceed 50% for significant periods of time. Hardware micro-timing analysis of operational environments by several computer manufacturers on behalf of individual customers during the past two years has provided (privately) considerable evidence on this topic.

Even programs of the kinds that typically are tuned to a high order of performance encounter significant unavoidable delays. Examples of such programs are sorts and matrix manipulations.

After an initial startup delay to load memory with data to get the first string generation process started, internal sorting typically uses all available processor time. The merge phase, however, which takes most of the total time, is typically severely limited by input/output speed. If there are only a limited number of channels; if the system has a fixed relationship between particular I/O devices such as tape units and particular channel numbers; or if the tape units cannot read backward, the processor idles a substantial part of the time. Thirty per cent or more lost time averaged over an entire sort-merge would not be unusual.

In matrix operations, startup delays occur as in sorts. For large problems additional inefficiencies may exist in that manipulations are too complex to be handled with real efficiency on the number of tape units available, or it may not be appropriate to adjust the balance between compute time and tape time to use the CP at full efficiency. Again, 30% or more lost time is not unusual.

When effective processor utilization is considered as averaged over entire workloads, the situation looks considerably worse. As remarked in Section 1.3 above, during the past we have learned of careful timing results showing that indicated net operational time of central processors lower than 30% of total "productive" time was not unusual; net CP time was less than 50% in all of the test results we have seen.

Use of random-block-addressable mass memory instead of tape, and addition of large-capacity Slow Core memory, can lessen those difficulties associated with the serial-access nature of magnetic tape.

If system net CP time of 30% can be raised to, say 75%, by multiprogramming at an added system time cost of no more than 15% of total processor time, system throughput *for a given processor speed* can be doubled.

Of course this improvement in duty factor of the CP does not come free of dollar cost. The most obvious additional hardware cost is for more memory; in the past large programs tended to be planned around the size of usable main memory space, and programs that leave little available memory space for other programs will vitiate a multiprogramming system.

There are several conflicting factors at work affecting the memory size that is optimal for a given system in its workload environment:

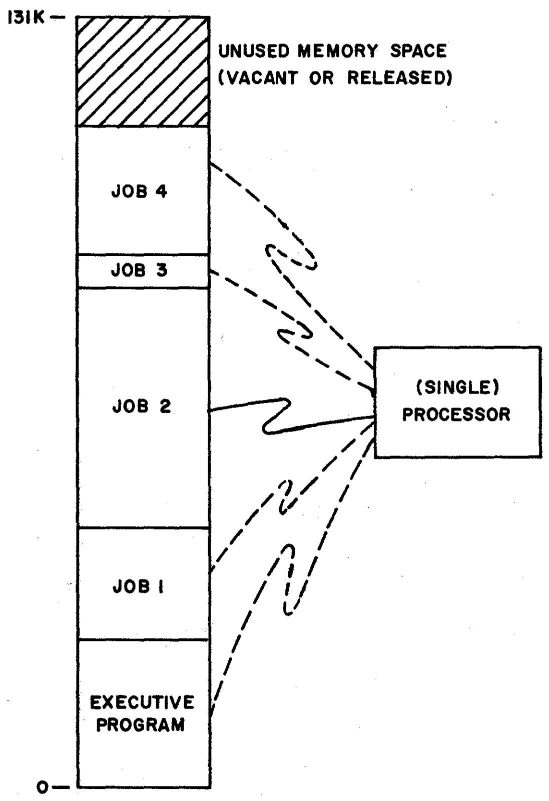
a) Several techniques discussed in this paper increase the effectiveness of main memory utilization in a system.
b) The growing power of mass memory facilities; their effective use instead of tape for system service functions not to be kept in main memory; and the fact that multiprogramming permits reloading of service routines to be overlapped effectively with other work, now permit some functions that have typically been retained in "hard core"* to be brought in as required from mass memory.
c) The increasingly complex nature of executive systems and the many additional buffer spaces needed for I/O in all-devices-on-line systems are now, despite factors a) and b), inexorably driving upward the amount of memory space needed for executive functions.
d) The increasing use of mathematical and data processing techniques that use (very effectively) large memory space, and the solution of more and more complex (i.e., large) problems, are tending to increase the memory space demands of important programs.

* Memory space permanently assigned to executive functions.

e) Multiprogramming systems, in order to perform effectively their task of improving processor utilization, need concurrent access to many cohabiting jobs in main memory.
f) The cost of high-performance memory is decreasing relative to the cost of processors and other parts of large systems.

Factors a) and b) tend to reduce main memory space demands; factors c), d), e), and f) tend to make larger main memories desirable and practicable. We feel that the net result of these several factors will be substantial increase in the typical size of memory in large multiprogrammed systems, whether single or multiple processors are used. While this will increase the total typical cost per system, the net effect will be to decrease the unit cost of computation.

So much for the motivation for multiprogramming; it is largely economic in nature, as what significant considerations in heavy-computing-machinery operations are not?



MULTIPROGRAMMING SYSTEM

Figure 1

Let us now examine some aspects of multiprogramming operation of a system.

Figure 1 shows a memory map of an elementary multiprogramming system shortly after startup. Initial core loads for the first 4 jobs have been loaded and Job 2 is active; i.e., the processor is at the instant of the picture executing Job 2.

As in conventional one-job-at-a-time processing, the processor spends some of its time as needed in the executive program that provides for system control and job sequencing.

When Job 2 encounters its first delay, or when another consideration such as priority causes another job to get precedence, the processor switches to a job selected by the executive.

A frequently-used signal for the processor to check job segment sequence is the initiation of any input/output order.

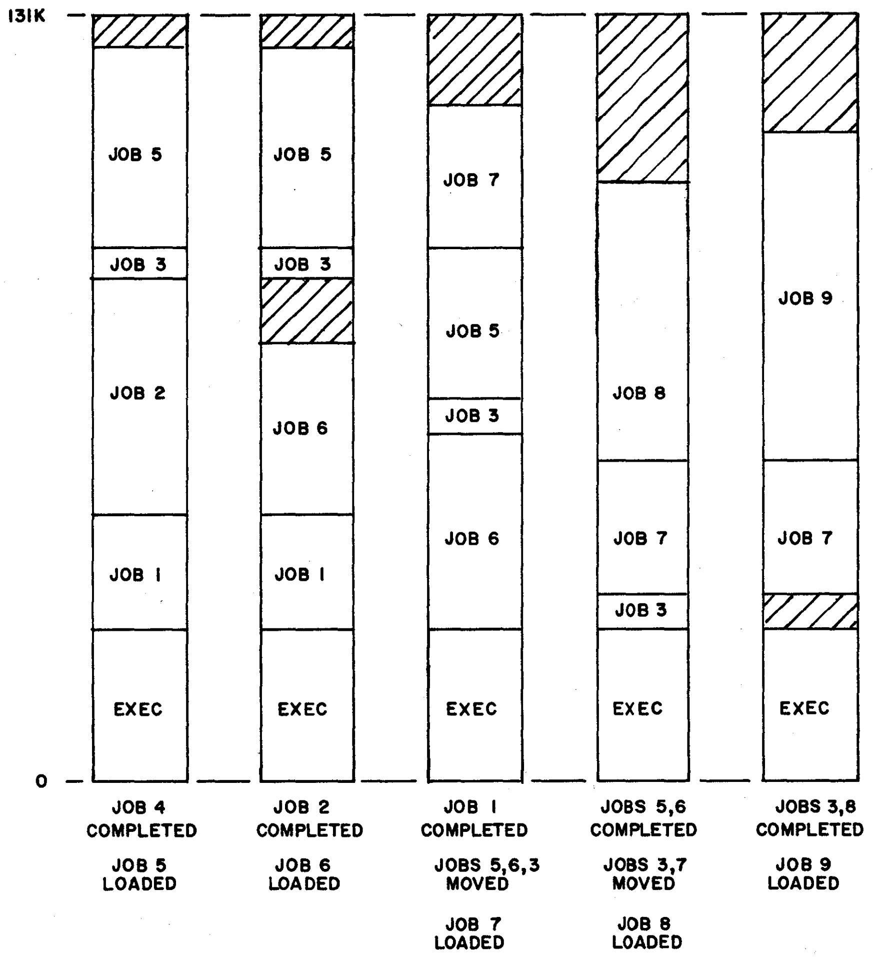
Figure 2 shows five subsequent memory maps of this system as some jobs complete, others come in, and some get moved as needed to coalesce space for a waiting job. Concern with the time-cost of the moving process has given rise to considerable interest in hardware means for making the moving unnecessary.

We are convinced that there are two serious concerns associated with program and data moving at job-load time in high-powered multiprogramming systems:

a) Program modification required prefatory to the actual move can be unacceptably time consuming.
b) In general, programmed relocation of significant areas of memory content in a multiprogramming system requires that all currently active and outstanding input/output orders be permitted to terminate. In a large system this can be a prohibitive requirement, again because of the large potential time loss associated with each relocation.

In machines of modern type in the million-instructions-per-second range, the moving of memory contents does not appear to be a significantly costly process. As an example, let us consider the time to perform a typical move of memory contents, assuming no special hardware for relocation.

Assume main fast memory is 131K words; half of all jobs entering the system



MULTIPROGRAMMING SYSTEM MEMORY MAPS
(SYSTEM WITHOUT PAGING)

Figure 2

cause moving of one fourth the contents of main memory; the time to move one word (memory read-write under some kind of repeat control) averages 1.8 microseconds. Then the moving time per job is:

$$0.131 \times 10^6 \text{ words} \times 0.5 \times 0.25 \times 1.8 \times 10^{-6} \frac{\text{second}}{\text{word}} = 0.029 \text{ second.}$$

With these assumptions, which we feel are conservative, average total job time would have to go down to 3 seconds before moving time consumed 1% of machine time. Under such circumstances we do not foresee an immediate justifiability for hardware for removing the need to move the contents of memory.

For some advanced applications, however, in "computational colloquy" operation, as mentioned in Section 1.5 above, and as an Advanced System approaches full-scale expansion with four processors, we conclude that it will be necessary to provide for reasonably efficient execution of job segments as short as 1 millisecond.

Paging hardware (not in the basic 213), in addition to drastically reducing moving time, provides a convenient way to apply hardware memory protection to scattered areas of memory by association with particular users in several different ways. Some kind of applications inherently require greater inter-job security than is required in a typical closed- or open-shop job-oriented facility or even in a time-sharing system; in such cases, redundancy of protection (in addition to that supplied through job area assignment by the Executive) is mandatory. Paging can provide such redundant protection, establishing positive hardware identification of small memory blocks with particular jobs or even with particular users.

For use in the circumstances described in the preceding paragraph, we have concluded that "paging" or "block symbolic" addressing capability should be planned as a standard hardware option for the Advanced System and perhaps as a special hardware modification of the 213, but it will not be used by the software presently under construction. The paging hardware plans will be described in Section 7.

For the basic 213 and Advanced System we have defined Implicit Base and Limit Register Selection hardware that permits dynamic relocation to be performed without interference with currently-active input/output orders. This hardware and its application will be described in Section 3.6.

An ingenious method of displaying the dynamic memory map of a time-sharing system in actual operation, showing information that would be of value in monitoring a multiprogramming system, was used by H. A. Kinslow[21] in his oral presentation. This display represented the contents of each logical block of memory addresses (in Kinslow's case, 256 words per block) as a character position in a line of print, with each of several classes of activity indicated by the identity of the character representing that memory block at that instant; e.g., X for program now in Execution, + for program moving from disc to core, = for program in core, not being executed, (blank) for core space available, • for program going to drum, — for program going to disc, S for core space assigned but not yet filled. Each line also carried other coded information about system status at that instant of time.

By use of the Kinslow-Johnson dynamic memory map, the kind of information that was shown in Fig. 2 merely as an abstraction can be displayed for a running system on a live workload.

A cautionary remark must be made here about the extent to which multiprogramming can provide calking, as it were, for chinks in system workload. A suitable supply of subdividable tasks must, in fact, exist in order to be handled in this way.

Some classes of engineering calculations, notably "nuclear design codes," traditionally have taxed the capacities of the most powerful machines available. These programs have been planned to use available hardware to the utmost, although this utmost, as mentioned in 1.3, may be constrained to half-time usage of central processor time. Such users may have only incidental amounts of small-problem workload.

For this type workload, three avenues to greater system throughput seem to be open:

1. Better system balance, with use of faster mass memory devices, can make possible higher internal efficiency of the large programs running alone.
2. Versions of each of several of the largest programs might be produced that would operate with less than full-machine facilities, and that complement, to a significant extent, each other's limiting demands upon the total system; two or more such programs, not related to each other, could then be run concurrently in a multiprogramming-multiprocessing environment.
3. For a multiprocessor system, some large tasks will justify efforts to apply more than one processor in some parts of large programs, i.e., to solve the single-task multiprocessing problem.

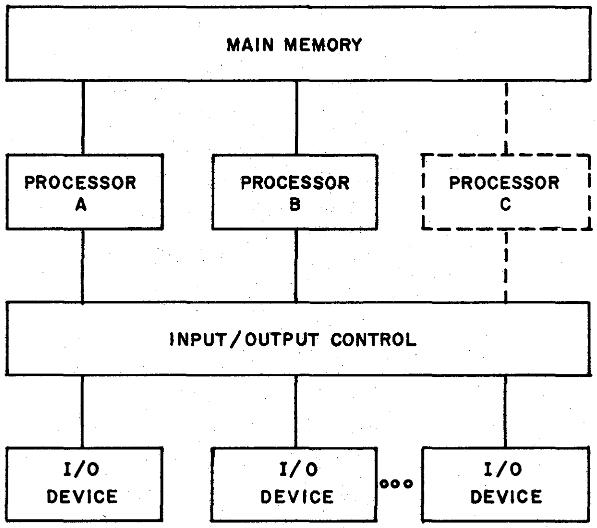
2.2 The Anonymous-Processor-Pool Concept

In this type of system, a generalization of the single-processor system, any number of fully independent processors up to a design limit can operate within a single, jointly-addressable memory.

Each processor may assume the executive function, and must do so in order to obtain system service, to write in memory areas reserved for executive purposes, or to obtain reserved information. Thus, each processor must be able to operate in two somewhat different modes, which we call *executive mode* (E) and *job mode* (J).*

An important executive function is that of *sequencing* or determining what computation job or job segment is to be performed next,

* A third mode, Common Routine (CR), will be described later.



ANONYMOUS PROCESSOR POOL SYSTEM

Figure 3

following a change such as completion of one job.

In this system all input/output is initiated by an E processor. Any I/O request from a program in operational status is used as a signal that the basic sequencing algorithm is to be executed.

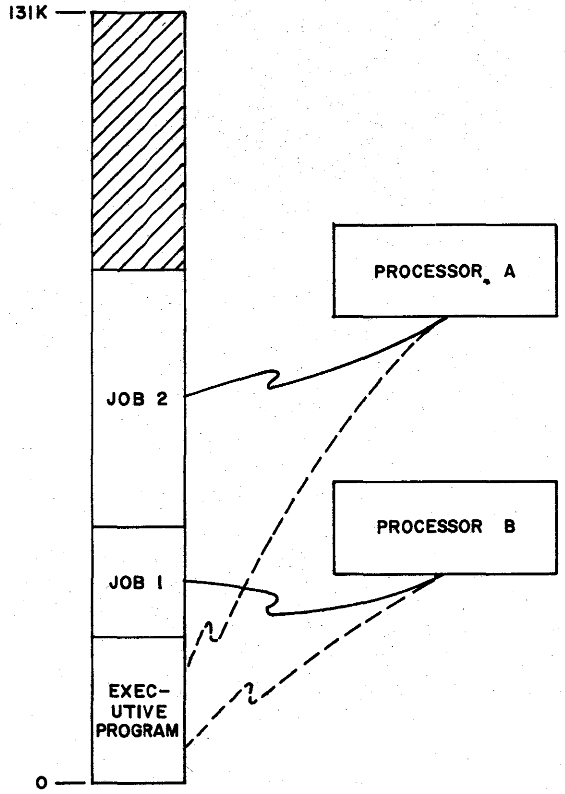
Processors enter and leave E mode frequently; hence, little time can be used in executing the standard interrupt procedure. A processor in requesting executive service always makes itself available to be assigned to it; likewise, in assigning a job, the processor performing the assignment subsequently becomes the one most available for assignment. Transfers of control to and from the executive program are always carried out by the processor making the request; thus, some transient status information can be, and in fact is, passed back and forth in a processor's own registers. For security purposes as well as for operational efficiency, status information on all programs currently in the system is maintained in lists in the executive area of memory, and complete information on each job is carried in lists in that job's assigned memory area. In J mode, a processor cannot initiate input/output and has read and write access only to the memory areas assigned to the particular job on which the processor is currently working.

Figure 4 illustrates the basic operating situation in main memory with only one active assigned job for each processor, i.e., without multiprogramming. Local job functions are performed by Processor A for its currently assigned job, which is Job 2, and by Processor B, for Job 1.

In general more than one, and in the extreme all, processors can operate within the executive program at the same time, although some critical executive functions such as sequencing must be performed by only one processor at a time.

Thus, in Fig. 4 it is possible that the physical processor that had been performing Job 2, upon exiting from E mode, may become from the executive program's point of view Processor B and be assigned to (that is, assign itself to) Job 1, provided Processor B has meanwhile been assigned an Executive function. Thus, the processors themselves are truly anonymous, assuming identities only for executive job assignment purposes and retaining them only until returning to E mode.

Figure 5 is a generalization of Fig. 4 to include more than one job per processor, i.e.,



ELEMENTARY MULTIPROCESSING (WITHOUT MULTIPROGRAMMING)

Figure 4

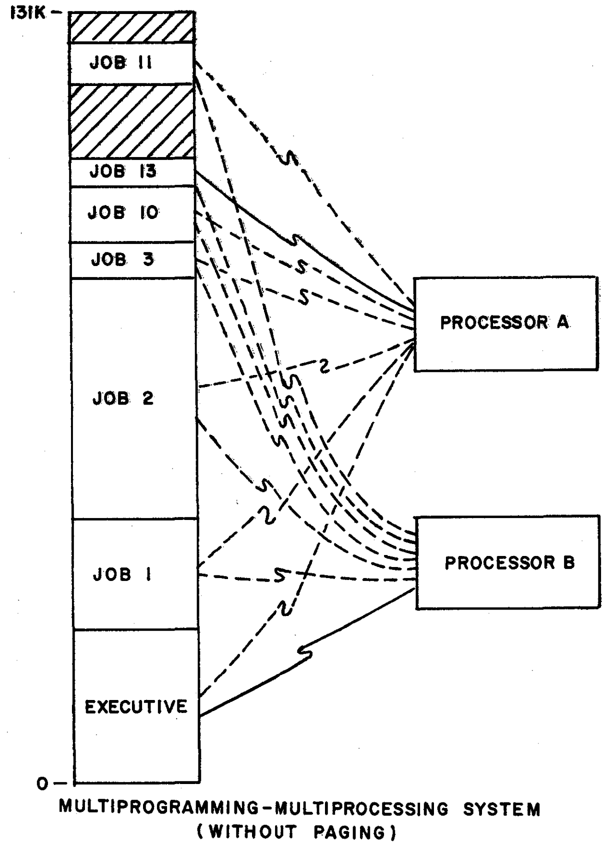


Figure 5

multiprogramming. At the instant portrayed, Processor A is in J mode executing Job 13, while Processor B is in E mode, performing an executive function. With two sizeable released areas of memory, the system will presumably load at least one more job (perhaps moving Job 11 to coalesce the available space if necessary) if a job in the input tanking area (in this system, on drum) has been found by the Executive to need little enough memory space and otherwise to qualify for active status at this time.

For clarity, this map of *physical memory addresses* has been drawn for a traditional-type main memory system without paging. In a memory control scheme having hardware paging capability, as described Section 7, available space as well as assigned single job areas are scattered throughout the (physical or actual) memory addresses in sections most of which are one page in size. *Logical* address allocations, as they appear to processors, more closely resemble the simple map shown in Fig. 5.

2.3 Planning Considerations for Memory Control Hardware

In the 213, which was intended from the outset to be applied in a multiprogramming-multiprocessing environment, several new hardware requirements had to be met. Some of these (a-f below) were evidently mandatory if good efficiency was to be achieved, and were included in the basic 213 System. Others (g and h), while they appeared to be realistic for typical Advanced System applications, were considered to be required only in specialized installations of the 213; hence, g and h are not in the basic 213 hardware and are not used in the basic 213 software.

a) Direct addressing of large memory (by character in the case of character-string-manipulation instructions).
b) Processor status information transferred by hardware.
c) Positive memory protect for each job with respect to its currently-assigned memory area.
d) Dynamic relocation of programs and data without significant loss of processor speed.
e) Automatic access to (Read-Only) Common Routines (one copy of each) and to Common Data from any number of calling programs.
f) Ability to utilize conveniently a hierarchy of main memories.
g) Memory remapping (paging) with individual-job number memory protect assigned to individual "pages" of memory.
h) Three different modes of memory protect, each different in behavior with respect to each processor and individual memory pages in accordance with each processor's currently-assigned mode (Executive, Job, Common-Routine).

Our decision has been to specify the hardware for dynamic-remapping and page-oriented memory-protect and the manner in which software will use them in order to assure the upward-compatible growth potential of the 213 hardware and software, but to make these memory control features optional pending detailed hardware monitor studies of operational 213's in customer environments. The proposed paging hardware, and its use, are discussed in Section 7.

One concern, hitherto unmentioned in the present paper, has seemed to us from the outset of the 213 system planning work to be a serious one: There has been growing evidence of the potentially high value-to-cost ratio of slow ferrite core memory in large capacities (hundreds of thousands of words), and hence of its potentially great importance in high-powered machines. This has brought us to the conclusion that positive control should be provided over access to a hierarchy of main (directly-addressable) memories, in flexible and convenient, preferably automatic, fashion.

By "automatic," we mean the following: A particular information area, once designated to the Executive System by a user as being suitable (most likely because of low frequency of access) for assignment to what we will call "Slow Core," should have symbolic addresses so assigned by the Executive. It should be accessible, thenceforth, by a program merely through use of appropriate locally relative-to-zero address references initiated by the assigned processor. It should not require that special control information (such as a bit in the instruction or in the address) be supplied by the program to designate the reference as being made to Slow Core. Although all Slow Core assignment to a given job may be restricted to a continuous string of physical addresses, this string should be assignable unrelated to Fast Core assignment to the same job.

It will be explained in Section 3.6 how this is accomplished by means of Implicit Selection of Base and Limit Registers.

3. BASIC 213 HARDWARE EXTENSIONS BEYOND THE 212

3.1 Large Memory Addressing (Fig. 6)

The basic address field length in the 213 and in the Advanced System is 24 bits. We shall refer to this field as bits numbered 0 through 23 although the actual location of the 24-bit string may be in Bits 24-47 of a computer word. Bits 0-20 specify word number over an absolute range of zero through 2, 097, 151.

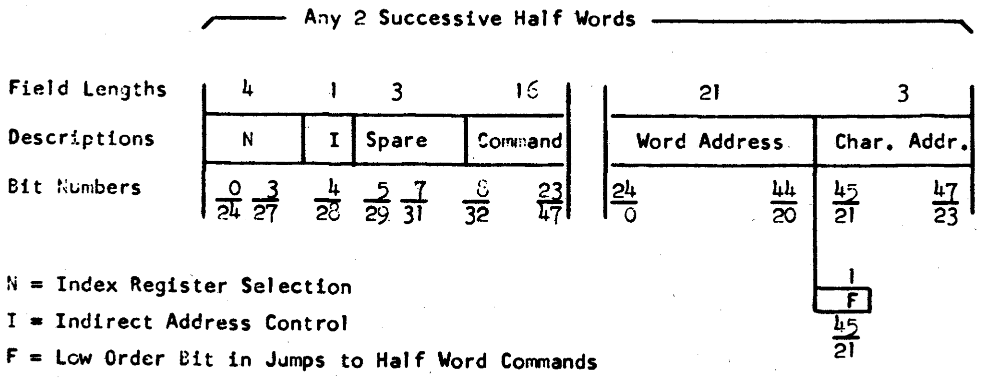
Only the word address is relevant for full-word-operand-manipulation instructions.

Bit 21, the upper bit of the character field, is relevant for some half-word-referencing instructions such as jumps and compound instructions that use (variable-length) arrays of half-words to contain jump vectors for many-way branching functions.

Bits 21-23 specify, for character-string-manipulation instructions only, character number within a word. For the initial 213 machines only 6-bit characters are handled explicity. The basic 210-211-212-213 64-character alphabet*

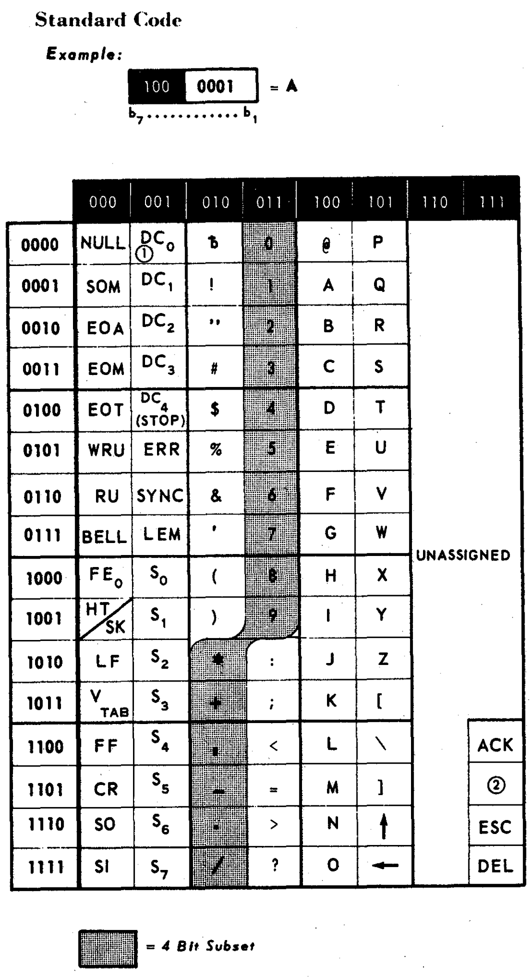
ABCDEFGHIJKLMNOPQRSTUVWXYZ
1234567890+—*/=.,$()'blank
><#&% ⌴ ⅂ ≡ ;: ?null"@end[18]

* The subset consisting of the first 48 characters listed above ("A" through "blank") are identical, both in graphics and in coded representation, with the SHARE 709-7090 alphabet.



FULL-WORD INSTRUCTION FORMAT

Figure 6


Standard Code
Example:
100 0001 = A
b_7 .... b_1
▨ = 4 Bit Subset

	000	001	010	011	100	101	110	111
0000	NULL	DC_0 ①	ƀ	0	@	P	UNASSIGNED	
0001	SOM	DC_1	!	1	A	Q		
0010	EOA	DC_2	"	2	B	R		
0011	EOM	DC_3	#	3	C	S		
0100	EOT	DC_4 (STOP)	$	4	D	T		
0101	WRU	ERR	%	5	E	U		
0110	RU	SYNC	&	6	F	V		
0111	BELL	LEM	'	7	G	W		
1000	FE_0	S_0	(	8	H	X		
1001	HT/SK	S_1	)	9	I	Y		
1010	LF	S_2	*	:	J	Z		
1011	V_{TAB}	S_3	+	;	K	[		
1100	FF	S_4	,	<	L	\		ACK
1101	CR	S_5	–	=	M	]		②
1110	SO	S_6	.	>	N	↑		ESC
1111	SI	S_7	/	?	O	←		DEL

American Standard Code for Information Interchange

Figure 7

is capable of representing explicitly all currently active procedural compiler source languages [including FORTRAN IV, NPL, COBOL, and ALGOL (with shifting instead of separate lower-case letters)], and also the information-content subset of the 8-bit ASCII character code,[16] Fig. 7. For these characters and in instructions that manipulate them, word content is 8 characters per word and Bits 21-23 represent an octal number 0-7.

In the future, as other characters of the ASCII set become defined and as explicit handling within general-purpose machines of the full 8-bit-specified character set becomes necessary, as already implemented within modern communication-processors such as the large-scale Philco 170 AUTODIN computer, an 8-bit-character-set-handling instruction group will be defined and added to the 213; it is now planned to be included in the initial Advanced Systems. For the alphabet of 8-bit characters stored 6 to a word, Bits 21-23 will represent a base-6 number; and control counters used by the 8-bit-character-string-manipulation instructions will carry from Bit 21 into Bit 20 on count of 6, rather than on count of 8 as in the 6-bit present 213 instructions.

Consideration is also being given to including as an option in the 213, and as standard in the Advanced System, a set of 4-bit (binary-coded decimal) character string manipulation commands, primarily to meet the needs of very-large-scale commerical applications. The ASCII 4-bit subset is the one considered.

Each of the 8 index registers in the 213 (each of the 15 in the Advanced System)* contains 24 numeric bits numbered 0-23. For word-oriented instructions, only Bits 0-20 are relevant. There are also two control bits designated C and Y. As in the 210 and 211, a "one" value of the C bit causes designated index register contents to be counted (increased by one word after each reference to it) when C is set to 1, and to remain unchanged when C is zero. As in the 212, the C and Y bits taken together provide decrementing as well as incrementing from the address field. Counting and incrementing/decrementing operate according to this scheme:

C	Y	Contents of Selected Index Register After Reference to It
0	0	Unchanged
1	0	Increased by unity in word address value
0	1	Previous contents plus value of instruction word-address field
1	1	Previous contents less value of instruction word-address field

The word-oriented index register stor-

* In the 210, 211, 212, and 213, eight index registers are selected by a 4-bit field with binary numbers 1000-1111 inclusive. In the Advanced System, the additional bit combinations 0001-0111 inclusive will be activated by the new format half-word instructions and for all full-word instructions, to designate seven more index registers, with 0000 designating a pseudo-register containing the constant zero.

age and loading instructions of the basic 213 System treat the index register contents of a 24-bit string, with Bits 0-20 representing their functions as described above; Bit 21 containing the C bit; Bit 22, the Y bit; and Bit 23, at present irrelevant. These instructions thus pack index register contents in computer half-words, two to a word.

3.2 Character Manipulation Instructions

The guiding principle in the choice of character-oriented hardware macros was that program functions must be frequently written and executed to justify inclusion as explicit hardware. Furthermore, although the hardware macros need not save memory space, they must save substational processor time if they are to justify their existence.

An example of the type of function considered is the scan of an input character string for break characters as is commonly performed in compiler source language analysis. Each character in sequence is checked to determine whether it is one of a set that causes action to be taken; if it is one of these, control transfer occurs and the scan pauses or terminates.

Using the word-oriented vocabulary of the 210-211-212, a typical scan involves a program sequence of 33 half-words and from 4 to 12 full-word constants. On the 212, such a scan requires 400 microseconds for the first eight characters in a string and 370 microseconds for each additional eight characters.

As Move-Break is implemented in the 213, a mask in the A and Q registers defines which of the 64 character configurations are members of the set of break characters. A single hardware macro scans (and moves) a string until a break character is encountered; a jump is then executed to the address associated with the specified break character. An instruction option, Move-Break-Squeeze, squeezes out blanks en route.

Total space for using this instruction (including pre-setting of registers) is six full words plus one half-word for each break character. Total execute time is about 15 microseconds for the first 8 characters plus 2.4 microseconds for each succeeding 8 characters. For a typical scan of up to 62 characters, averaging 22 characters, the maximum and average times are 31 and 22 microseconds, respectively, as compared with 2.99 and 1.14 *milliseconds* for the programmed equivalent sequence on the 212.* The speedup factor for the average case in this example is 52:1.

In the Advanced System, because of the facility and economy with which the integrated circuitry can be used to implement complicated logic functions, this particular instruction will lend itself well to parallel instead of serial processing. Thus, as in the case of other frequently executed commands, the speedup over the 213 will be greater than the "minimum 10:1."

Because of rapidly developing interest in non-numeric data processing, the basic 213 character manipulation instructions, which constitute the core of the set that will be built into the Advanced System, will now be described more formally.

3.2.1 Format[19] Character-oriented instructions that reference character strings do so via index registers. The conventions are:

X_1—specifies the character address in memory from which the first of a string of characters is obtained.

X_2—specifies the character address in memory into which the first of a string of characters is stored.

X_3—specifies the (maximum) length of the character string.

Note that the character count in X_3 is positioned in the low-order character position of the register.

Character-oriented instructions that reference a single character specify the address in the address field, with the usual indexing and indirection options.

3.2.2 Notation[19] The following notation is introduced to simplify further discussion:

* In order to give those unfamiliar with the 212 a feeling for its speed on typical tasks, we offer this yardstick: The 212 executes about 0.75 million instructions per second on typical object coding, including floating arithmetic instructions, compiled from FORTRAN source language. The fastest 212 operation we have seen in any program is one sequence of instructions executed from memory on operands in registers, for which the 212's speed is 1.62 million instructions executed per second.

X_s—the content of X_1 at the start of a character-oriented instruction (starting address).

X_r—the content of X_2 at the start of a character-oriented instruction (receiving address).

K—the content of X_3 at the start of a character-oriented instruction (maximum string length).

N—the number of characters stored at the conclusion of a character-oriented instruction.

P—the number of space characters (60_8) encountered during a transmission.

rA—register A, whose bits are numbered 0-47 from left to right.

rQ—register Q, whose bits are numbered 0-47 from left to right.

3.2.3 Instruction Catalog[19]

MOVE —a) copies a character string of length K from X_s to X_r.

b) if any character in the string is numeric (0-9), bit 45 of rQ is set to one; if any character is alphabetic, bit 46 of rQ is set to one; if any character is special, bit 47 of rQ is set to one. Note that this determination is made as a character is stored.

c) the final register contents are as follows:

$X_1 = X_s + K$
$X_2 = X_r + K$
$X_3 = 0$

MVBRK —a) performs the functions described under a) and b) for MOVE.

b) each character leads to examination of that bit of rA, rQ corresponding to the numeric value of the character. If the bit is one, transmission stops; otherwise, the character is transmitted.

c) if no one bits are found in rA, rQ (i.e., no break characters are encountered), the instruction terminates after K characters have been transmitted. The processor then performs a jump to the address specified in the half-word following the MVBRK instruction. The final register contents are the same as c) under MOVE.

c) if no one bits are found rA, rQ (i.e., a break character is encountered), the instruction terminates. The processor counts the number of one bits, b, to the left of the bit on which a "match" occurred. The processor performs a jump to the address specified in the (b + 2)nd half-word following the MVBRK instruction. The final register contents are:

$X_1 = X_s + N$
$X_2 = X_r + N$
$X_3 = K - N$

MVBRKSQ—a) performs all the functions listed under MVBRK.

b) all characters equal to 60_8 (space) are not transmitted.

c) if no break characters are encountered, the final register contents are:

$X_1 = X_s + K$
$X_2 = X_r + K - P$
$X_3 = 0$

d) if a break character is encountered, the final register contents are:

$X_1 = X_s + N + P$
$X_2 = X_r + N$
$X_3 = K - N - P$

FILL —a) copies the character in bits 42-47 of rQ into K consecutive character locations starting at X_r.

b) the final register contents are:

$X_1 = X_s$ (unaltered)
$X_2 = X_r + K$
$X_3 = 0$

TCHQR —transfer the character specified in the address field to bits 42-47 of rQ.

TCHQL —transfer the character specified in the address field to bits 0-5 of rQ.

TQRCH —transfer the character in bits 42-47 of rQ to the specified character address.

TQLCH —transfer the character in bits 0-5 of rQ to the specified character address.

COMP —a) alphanumeric - compare the character strings of length K starting at X_s and X_r.

b) perform jump to the address contained at one of three half-words following the COMPA instruction according to the following:

string $X_s <$ string X_r
string $X_s =$ string X_r
string $X_s >$ string X_r

SCANOT —a) scan the character field of length K starting at X_s.
Compare each character with the character at bits 42-47 of rQ. Stop scanning on the first occurrence of a failure to match.

b) if the first M characters did match, then the final register contents are:

$X_1 = X_s + M$
$X_2 = X_r$ (unaltered)
$X_3 = K - M$

3.3 Processor Status Information Automatically Transferred By Hardware

All reassignments of processors, input-output requests, and calls on Common Routines are performed under Executive control. Since considerable generality is required in control transfer and in interrupt capability in order that interrupt response time be short, complete information on processor status is stored relative to the base of the program segment from which a particular processor jumps. If one or more processors were in a Common Routine when one or more of them were interrupted, status information is stored relative to the base of the original calling program(s). Automatically transferred status information occupies a string of three words in a fixed position relative to the base of each program, and includes:

FUNCTION	STATUS BITS
Control Registers	
V field	24
Repeat Mode	1
Repeat Counter	16
Overflow	1
Inhibit Clear Overflow	1
Double Pecision Mode	4
Processor Number	2
Processor Mode (Executive, Common Routine, Job)	2
Trap Reasons	
Machine Errors	
Operand Parity Error	1
Input/Output Parity Error	1
Indirect Address Parity Error	1
Store Parity Error	1
Instruction Parity Error	1
Program Errors	
Executive Command (in CR or J Mode)	1
Address Limit Violation	1
Command Fault	1
Trapped Instructions	
Halt	1
Breakpoint Jump	1
Toggles to D Register	1
Typewriter to Memory	1

D Register to Typewriter	1
Input/Output	1
Intentional Trap	
Jump to Executive	1
Interval Timer	
Time is Up	1
Interrupts	
External	1
Executive	1

All other processor registers are addressable and are handled under program control.*

3.4 Inter-Processor Control Communication

In order to perform processor assignment, the Executive program must keep current in its own active lists information on each processor's status at the time of most recent execution of the assignment algorithm. In order to permit selective interruption of processors in consideration of their current status between points of communication with the Executive, however, it is necessary that a processor currently in Executive Mode be able to query and, if necessary, interrupt all other processors.

Nine basic inter-processor communication commands have been specified. The first seven of these are present in the 213 and are used by its basic software; the other two relate more particularly to the single-task multiprocessor problem and are to be included only in the Advanced System.

It was explained in Section 2.2 how all processors are anonymous in that any of them can and does perform Executive functions upon demand, usually in response to a request issued by that same processor in a task previously assigned; and assignments made by a processor while in Executive Mode typically are carried out by that same processor as soon as it has changed back to Job or Common Routine Mode. This anonymity is limited only on one respect: each processor contains a 2-bit Processor Number (assigned by manual switch setting while out of service) so that it can be identified explicitly in connection with task assignment. The parameter N in the first three commands below is this number.

STAY Processor N
RESUME Processor N
TRAP Processor N
ENTER EXEC
ENTER JOB
ENTER COMMON ROUTINE
RDCLR
TRAP Processor(s) in Mode M } In Advanced
TRAP Processor(s) not in Mode M } System only

The commands ENTER EXEC, ENTER JOB, and ENTER COMMON ROUTINE provide for automatic transfer of a significant amount of processor information, as listed in Section 3.3, to fixed memory areas related to the base locations of programs being entered and left, as well as expediting storage and retrieval of the programmable-access registers. These commands provide for efficient control transfer between processors to be performed by the Executive.

The READCLEAR command, application of which is discussed in Section 4, is apparently trivial in function but is profound in significance. Consisting of a memory-read subcycle only, it transfers the contents of a specified memory location to the A Register and stores zeroes in that memory location, requiring only the first part of a memory cycle for total execution. Use of this command provides for resolution of race conditions when more than one processor simultaneously attempt to access the same data or program; for enforcing the convention, necessary in parts of the Executive Program, that only one processor can be in a particular program segment at a time; and for providing information, through the use of "read-tally" and "write-indicator" words, to make possible Common Data access as explained in Section 4.3. This informs any processor that may inquire what

* At an earlier date, it was planned to include one pair of hardware macros, SAVE and UNSAVE, for saving and restoring all index registers and a second pair, SNAP and UNSNAP, for saving all processor registers (including index registers). While each of these macros would have required only a few memory cycles for execution, subsequent program analysis disclosed that frequency of usage probably would not justify their inclusion. The SAVE/UNSAVE and SNAP/UNSNAP instructions have consequently been deleted from basic machine specifications, will be made available only as machine modifications on 213 and Advanced Systems and, at least initially, will not be used in standard software.

other activity may be under way in a particular commonly-accessible data string.

3.5 Basic Single-Segment Dynamic Relocation and Memory-Protect

In both the 213 and the Advanced System, provision is made for all programs to be loaded relative to local-address-zero; *all* memory accesses by a processor are offset from System Zero (i.e., physical address zero) by being added to a relocation register associated with that processor.

All memory references are checked, in the course of being issued, by the memory control hardware in the processor to determine that they fall between the relocation base and a limit value (absolute address, not relative to the base) stored in a *limit register* also associated with that same processor.

The base and limit registers are automatically loaded under Executive control with address values designating appropriate memory areas at the time a processor is assigned to execute a particular job.

Any attempt to reference addresses outside the assigned range for a particular job causes a trap to the Executive.

Note two nuances of wording in the third and second paragraphs back: address references are checked *in the course of,* not *prior to,* being issued; and memory *areas,* not *area,* are assigned to a processor in connection with a task.

Checking address references before issue would slow down the machine; hence, they are checked while being issued. *Read* references may actually complete before an out-of-range trap is initiated by the hardware; in that event, the trap will be executed before the user can use the forbidden information.

Write references do not cause modification of memory contents until completion of the memory-read subcycle; by that time range-checking has been completed; and the previous contents of a particular location is written back into it in the event of an out-of-range Write trap.

The fact that several memory areas, rather than merely a single string of memory locations, can be assigned to a processor along with a job to be executed, will be explained in the next section. Briefly, it is because several base-limit register pairs exist in each processor and can be assigned separately by the Executive.

Dynamic memory allocation is performed as follows: If a memory area assigned to a processor for a job is to be changed, the Executive loads into the appropriate base and limit registers new address data commensurate with the new assignment. Subsequent accesses are relocated and checked in accordance with the new assignment.

All input/output, as well as Common Routine calls, are performed via the Executive.

3.6 Multiple-Segment Dynamic Relocation and Memory-Protect by Implicitly Selected Base and Limit Registers[19]

There are several purposes for which it is desirable to have a multiplicity of relocation (address base) and address-range-limiting registers, as opposed to single relocation and range registers, in each processor. These include facility for:

a) Partitioning of a single program into non-contiguous areas of memory.
b) Designation that certain infrequently referenced arrays may be stored in "slow" main memory when a hierarchy of directly-addressable memory exists.
c) Access to Common Routines (a single copy of), which may be executed concurrently by more than one program, and/or simultaneously by more than one processor.
d) Common Data access by multiple programs to an arbitrary number of separate tables, with no restriction on simultaneous read-access (by several processors) unless one processor is writing into a particular table, in which case all others must be automatically prevented from either writing into or reading *that table only.*
e) Ability to handle the single task multiprocessor problem.

To this end, each 213 processor contains two base registers (designated B_0, B_1) and two associated limit registers (designated as L_0, L_1); the basic Advanced System will have eight each B and L registers. The programmer, however, has no need to know of the existence of these registers nor can he address or alter them.

The base and limit register operation may be defined formally: let b_i and l_i designate the contents of B_i and L_i, respectively, and S_i, an address. Define the sequence of addresses S_i by

$$S_i = S_{i-1} - l_{i-1} + b_i, i = 1, 2, \ldots, n$$

with

$$S_0 = A + b_0,$$

where A is an effective address.* At each stage, compare S_i with l_i. If $S_i < l_i$, then a legitimate memory request exists for physical location S_i. Otherwise, form S_{i+1} and repeat the process. Either $S_i < l_i$, for some i, or a request has been made outside the jurisdiction of this program.

The effect of the Implicitly Selected Base Register hardware is to permit execution, dynamic relocation, memory protect, Common Routine calls, and Common Data access, as though every job (program and data, both shared with other jobs as well as private) had been loaded into a single, continuously-addressed string of memory locations, addressed via a single relocation register and protected by a single range register. In fact, the memory allocation may be split into two separated strings in the 213 and into up to eight, in the Advanced System. Allocation by the Executive will take into account that:

a) Several scattered strings of locations may be available, which together are sufficient for this job on this run. (Moving will not occur unless an adequate total allocation cannot be patched together by the Executive for the job that the sequencing algorithm has scheduled next.)
b) Some of the space requirements (e.g., program, scratch, live data) call for Slow Core memory. The Executive will assign strings of fast and slow memory as specified by run control information.
c) Some references will be to Common Routine or to Common File areas; for a processor in Job Mode, such references should be Read-Only protected, although a processor in Executive Mode should be able to write into those areas.

* An effective address is the result of indexing and/or indirect addressing, if used; if not, the effective address is the instruction address field.

The time penalty paid for the automatic-by-hardware selection of base and limit registers is surprisingly small: In the 213, no delay occurs if a reference is via B_0, L_0, because multiple-input address arithmetic hardware handles the entire sequence during normal indexing. If the second pair of registers is selected, effective instruction execute time, on the average, increases about 80 nanoseconds. In the Advanced System these delays will be more nearly completely covered in address preprocessing, and the maximum delay, for the eighth pair of registers, will be less than 15 nanoseconds.

To illustrate the manner in which the registers operate, let us consider a program (that, of course, is compiled relative to zero and) is loaded into memory.

PHYSICAL MEMORY ADDRESS	COMPILED ADDRESS
1000	0
•	•
•	•
•	•
1999	999
5000	1000
•	•
•	•
•	•
5499	1499

SPLIT RANGE PROGRAM EXAMPLE
Figure 8

Figure 8 represents a program requiring a total of 1500 words of memory that has been loaded, at the convenience of the Executive's memory allocator, into distinct, non-contiguous areas: physical locations 1000-1999 and 5000-

5499. Note that the program contains legitimate references to locations 0-1499.

Before entering the program, the Executive sets the base and limit registers as follows:

$$B_0 = 1000 \qquad L_0 = 2000$$
$$B_1 = 5000 \qquad L_1 = 5500$$

When the program references location 1200, the computer forms

$$S_0 = 1200 + (B_0) = 1200 + 1000 = 2200$$

and compares with $(L_0) = 2000$. Since $S_0 >$ 2000, the computer forms

$$S_1 = S_0 - (L_0) + (B_1) =$$
$$2200 - 2000 + 5000 = 5200$$

and compares with $(L_1) = 5500$. Since $S_1 < L_1$, a legitimate memory request exists for physical location 5200.

In the basic 213 the following restriction exists: I/O buffer cannot be split between the two non-contiguous regions B_0, L_0 and B_1, L_1. A later version of the machine, and the Advanced System, will provide facilities in the I/O control hardware for handling split buffers. Since large buffers will usually be assigned (entirely) in Slow Core, we do not feel that this restriction is a serious one.

4. COMMON-ACCESS PROBLEMS AND A PROPOSED SOLUTION

4.1 Common Routines

4.1.1 Introduction. We shall discuss in this Section, and present our first version of a hardware-software answer to, a multiprogramming problem that we believe will assume crucial importance as systems get more powerful and with increase in the number of users—especially computational-colloquy-by-time-sharing users—concurrently active in a general-purpose facility.

Common Routines as here described have also been called Pure Procedures,[11] Common Procedures,[9] Re-Entrant Routines,[12] Single-Copy Routines,[13] and various less-polite names.

The objective is that subroutines, to arbitrary nesting depth, and large routines such as compilers and executive routines that are called upon to perform service for many users, should be accessible to any number of calling routines in memory, whether or not they are currently being executed by other processors in a multiprocessing system. Execution of a Common Routine may be interrupted before normal exit, and the calling program may request proper continuation of the calculation even though both calling program and Common Routine may have moved meanwhile.

We accept as a present restriction the requirement that subroutine nests must always be retraced in the reverse order in which they were called, although entry may be made at any level in the nest.

Figure 9 shows a much-oversimplified linkage path from two routines through a three-level subroutine nest. The subroutines are shown fixed in location, while each of the two calling programs moves from time to time. Program B accesses Subroutine 3 through Subroutine 2; Program A accesses Subroutine 3 through both Subroutine 1 and Subroutine 2, in that order.

If the linkages are spread out in time, they might occur as shown in numerical sequence. The link 1-2 from Program B in its Location 1 to Subroutine 2 and return is shown dotted to denote that B's execution of SR2 was interrupted before completion.

4.1.2 Common Routine Access.[19] Common Routines may be executed simultaneously by more than one processor. Such routines:

a) Cannot alter themselves or be altered by calling programs; i.e., must be in a memory area that is protected as Read-Only (except by the Executive).
b) Must have access to parameters, arrays, and scratch areas within the memory areas assigned to each calling routine.
c) Must be compiled relative to zero.
d) Require base and limit registers to be set before entrance.

The design considerations of the 213 and the Advanced System satisfy the above requirements of Common Routines. Condition c) is trivial, since all 213 programs are normally compiled relative to zero. Condition a) must be satisfied by the programmer; however, certain processor hardware (discussed below) de-

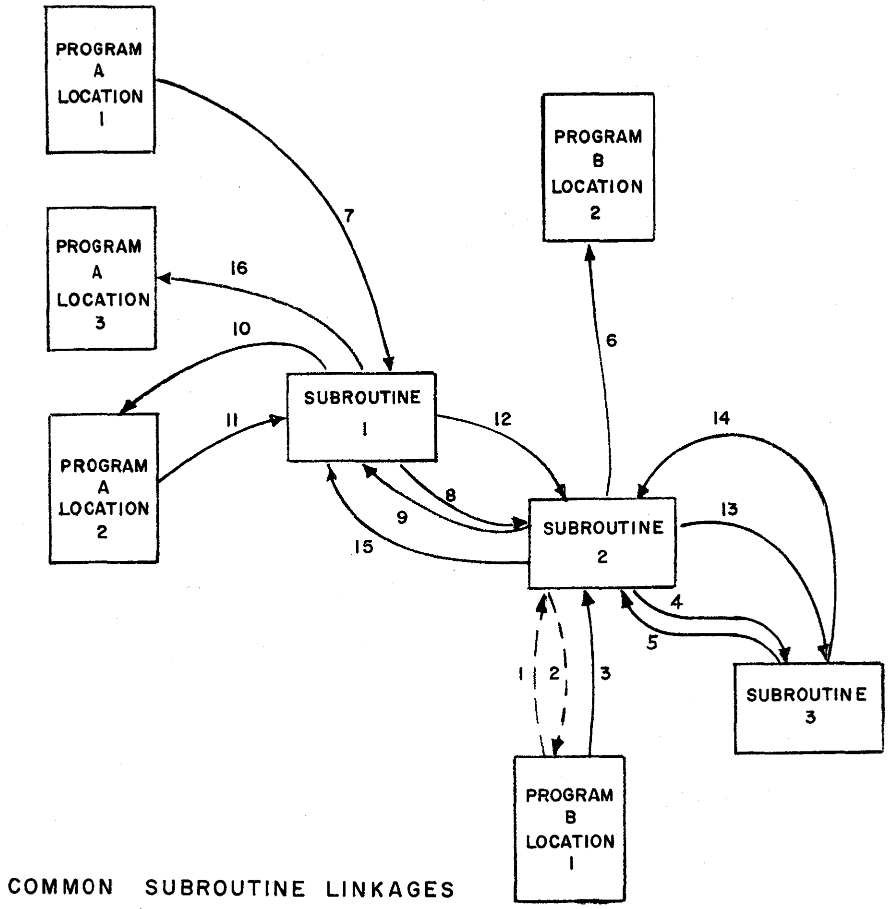


Figure 9

tects and protects against attempts to write into Common Routines.

Conditions b) and d) require elaboration, since they lead to another kind of utilization of base registers. We shall assume that a subrountine is called by a jump instruction followed by a list of parameters consisting of the addresses of variables and arrays.* In the case of a call on a Common Routine, the jump will be directed to the Executive System. The Executive will then perform the following changes in the contents of the base registers (notation as in Section 3.6):

b_0	$\rightarrow B_1$
l_0	$\rightarrow L_1$
Start of Common Routine	$\rightarrow B_0$
Upper limit of Common Routine	$\rightarrow L_0$

* This is consistent with the form used in Philco FORTRAN IV.

Further, a processor is set to Common Routine Mode, denoting that memory references relative to B_0 are "Read-Only," providing the extra protection promised in the discussion of condition (a) above.

The Executive now enters the Common Routine and supplies it with the address of the first parameter (relative to the base of the calling program). The Common Routine may now reference itself (via B_0, in "Read-Only" mode) and may reference any parameters and arrays in the calling routine.** To reference location X in the calling routine, the Common Routine simply refers to its own upper limit +X.

** The reader will have noted that the Common Routine cannot reference any portion of memory specified by (B_1 L_1) in the 213 ((B_7, L_7) in the Advanced System) prior to the subroutine call.

Note that this scheme permits nesting of Common Routines. For this case, the Executive at the time of each call alters B_0 and L_0 to contain the starting and ending addresses of the nested subroutine. Further, the Executive makes space provision in the calling program for the required parameter tables to be used and generated by the Common Subroutines, as well as scratch areas, at all levels of nesting.

4.2 Exclusive-Occupancy Routines[19]

To complete our discussion of routine access control, let us consider the antithesis of the Common Routine: the routine that cannot be allowed to be executed simultaneously by more than one processor, or even concurrently by more than one program. We shall describe a scheme by which any program can determine whether another program has entered an Exclusive-Occupancy Routine and either is still in it or has been interrupted and has hence exited prior to normal completion.

The RDCLR (Read/Clear) instruction was introduced to provide a means by which race conditions involving subroutines and Common Data can be avoided. This instruction transfers the contents of a specified memory location to register A and stores zeros in that memory location. It consists merely of a memory-read subcycle.

RDCLR can also be used to protect Exclusive-Occupancy Routines. Consider a subroutine in the Executive that must not be executed by more than one caller at a time. By convention, the first word of the subroutine is used as an "entry word." Before entering the subroutine, any program must check the entry word by issuing a RDCLR. If the content of register A is then non-zero, entry is permissible; further, all other programs are now alerted to avoid entry since the entry word is zero. The convention for normal exit from the subroutine must include as a final gesture the setting to non-zero of the entry word.

4.3 Common Data Access[19]

It is desirable for an arbitrary number of calling programs to have access to a commonly-available memory area while each program is located in its own job area. Further, if one program is altering a table in Common Data, it should be possible to lock out access to that particular table by all other programs.

Access to Common Data is provided through use of the base registers. B_0, L_0 are used to reference the program proper; the pair B_1, L_1 are assigned to the Common Data area.

The RDCLR instruction (defined in 4.2 above), in conjunction with a "write-indicator" and a "read-tally" word is used to provide for protected reading and updating of Common Data tables. A zero write-indicator word connotes that some program is currently altering a table entry or is altering the read-tally word; a non-zero read-tally word connotes that one or more programs are accessing for "Read-Only" purposes. A program desiring access to alter the table must ensure that the write-indicator word is non-zero and the read-tally word is zero. A program desiring to read the table must perform the following in the indicated sequence: ensure that the write-indicator word is currently non-zero and force its content to be zero (all by a RDCLR instruction); increase the tally in the read-tally word by unity; reset the write-indicator word to non-zero; decrement the tally in the read-tally word by unity when no further access to the table is desired.

The method outlined above permits an arbitrary number of programs to read a Common Data table and provides a tally of the number of such programs currently reading the table. A program desiring to alter the table first prevents access by any additional programs and then alters the table only after ensuring that all programs have relinquished access to the table.

5. JOB SEGMENT SEQUENCING

Lavine[5] discussed dynamic rescheduling by macro-segmentation; i.e., the treatment of input processing as one segment, execution (including scratch I/O) as a second segment, and output preparation as a third segment, optimum sequence of all such job segments to be determined by a set of algorithms. Input reading from cards (not processing) would occur whenever any unread cards were in any reader, unless halted by exhaustion of available input tanking space on drum. Sequencing considered quantitative factors including time since submission, estimated execute time, estimated printing volume, priority rating, and time of day. The sequence determination also considered such imperative factors as number of tape units needed and available and amount of drum space assignable for I/O tanking, and the optional factor of which form is on which printer at which time.

He pointed out that the nature of the sequencing algorithm needed was heavily dependent upon user installation management policies, and suggested that a manufacturer should supply a simple, fast one that is easily modified as required.

The multiprogramming task as discussed in the present paper may be seen to be a generalization of the dynamic rescheduling concept discussed by Lavine. We feel that all of his observations are valid and that the sequencing considerations he outlined must be taken into account in a realistic multiprogramming system planned for general application in a job-shop environment.

6. MULTIPLE-PROCESSOR ASSIGNMENT

The three levels in assignment of multiple processors will be listed in ascending order of difficulty.

a) Independent Tasks
The basic sequencing algorithm is insensitive to whether there is more than one processor in the system; hence, the handling of independent tasks changes little as the system configuration changes. Truly independent jobs are merely stacked and sequenced.

b) Interdependent Tasks
Some jobs depend upon prior completion of others for input data. In these cases the required prior run(s) are specified to the executive system by control card, and the executive checks for their completion and for the on-line availability of earlier results before permitting the follow-on run.

c) Single Task
We have looked at several aspects of the problem having several processors work simultaneously on a single task, and feel intuitively that this must be planned in detail by a user who understands a particular application. For this reason we are supplying some basic tools for the use of application programmers, but feel that the serious problems of single-task multiprocessing are theirs. Three types of problems, two of which are similar in nature, appear to be clearly practicable for single-task multiprocessing:

1. In relaxation schemes for manipulation of large matrices, mesh sweeps can be started by two processors at opposite boundaries and can proceed independently toward the center until two adjoining rows must be relaxed; the processors then must check each other at each mesh point. This amount of inter-processor communication is relatively trivial.
2. In Monte Carlo calculations, a large number of completely independent histories of problem elements are developed. Statistics can be accumulated by any number of processors working essentially independently, although final results must be analyzed (presumably by one processor).
3. We have seen a number of examples of large data processing and data reduction problems in which individual items were essentially independent and could be examined in parallel. As in the case of the random-shot technique mentioned in the previous paragraph, only the final summing-up of results need be handled as a one-machine task.

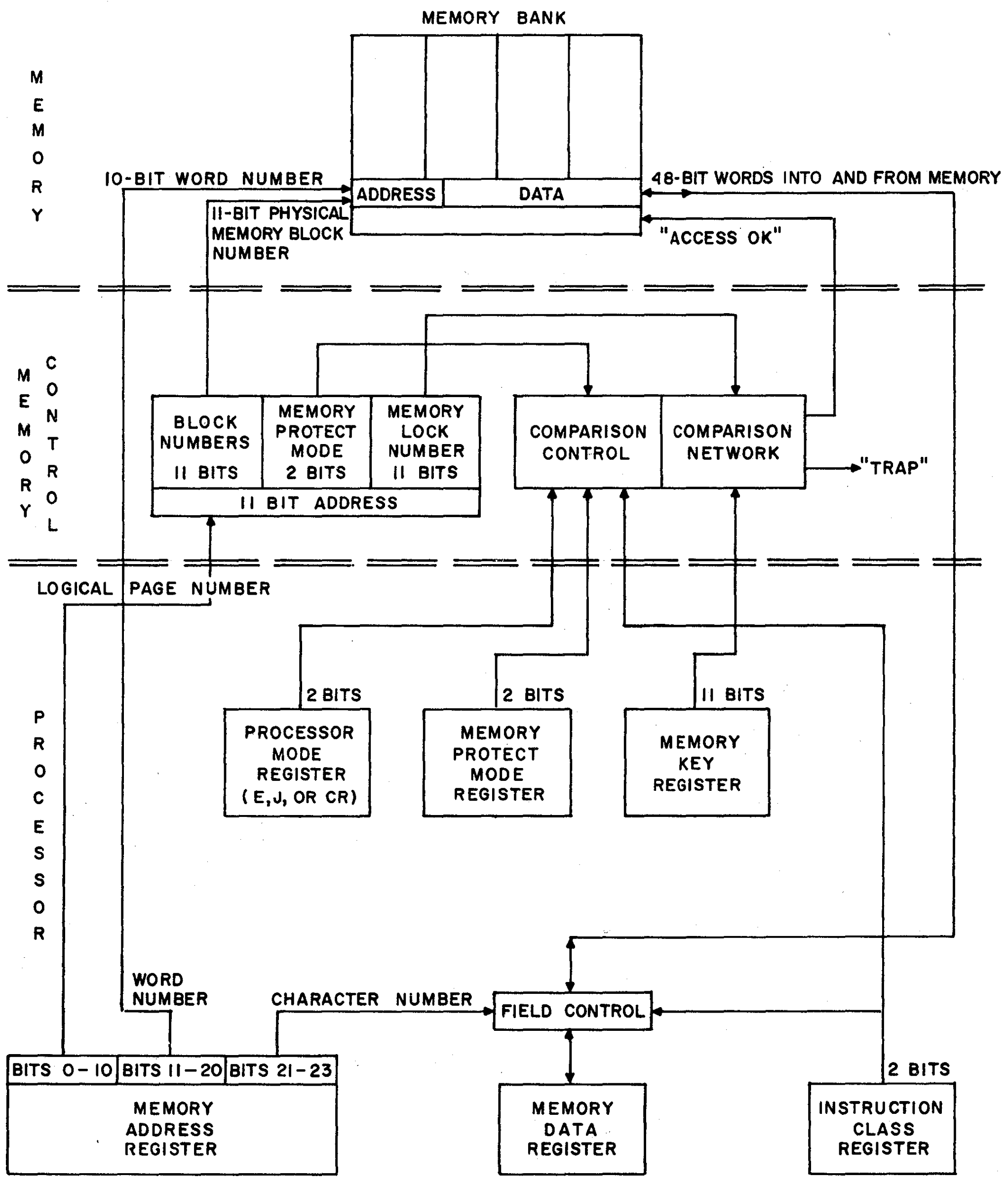
7. MEMORY REMAPPING (PAGING) AND PAGE PROTECT HARDWARE PLANS

This function was introduced in Sections 2.2 and 2.3 and will be discussed in this section.

Figure 10 shows the relationship between some of the hardware elements.*

The 24-bit (non-overlapped) addresses present in all full-word instructions, in all index

* Present tense will be used in this Section for the sake of brevity; it should be understood that future tense is implied.



MEMORY PAGING CONTROL AND PROTECTION

Figure 10

registers, and in all indirect addresses are combined to form an Effective Address in the Memory Address Register of each processor. This 24-bit field is treated as consisting of three parts:

Bits 0-10 identify any one of the 2,048 1024-word logical pages of main memory. Bits 11-20 identify the word number in a page. Bits 21-23 constitute character number within a word for character-string commands; Bit 21 specifies half-word for jumps and indirect address locations; and these bits are irrelevant in instructions that reference full-word operands. The three fields are treated in different ways.

Bits 0-10 select individual 24-bit words in Control Memory, and may be looked upon as being one level of indirection (in addition to indirection that may have been inserted by a processor) in all full word instructions. Bits

11-20 are used directly, adjoined to the physical block number contained in the CM word selected by Bits 0-10. Bits 21-23 are used to delineate character string boundaries.

Individual memory banks (32K in main memory or 262K in bulk memory) receive address information through Memory Control in two fields, which are adjoined to identify single words: In both cases the 10-bit Word Number (within a page) field is transmitted unaltered from a processor through Memory Control; each memory bank receives the Physical Block Number field (bits 6-10 for main memory, Bits 3-10 for bulk memory) from the Page Number section of Control Memory.

For the Advanced System, thin-film main memory is organized in word pairs; Bit 20 is consequently adjoined to Bits 21-23 in selecting the field to be presented to the processor's Memory Data Register.

Added to other control signals to the memory banks, the "Access ok" line signals that the requested access is acceptable to memory-protect, and permits the access to take place.

Memory Control contains, in addition to switching and race-resolution facilities, the Control Memory and circuitry for the memory-protect function.

Control Memory is a 1024-word, 48-bit, half-word-addressable module using the same technology as used for Main Fast Memory; for the 213, this is 1.15 microsecond nominal cycle time magnetic cores; for the Advanced System, 250 nanosecond nominal cycle time thin film. Because of the small size of this unit, actual operating speeds will be somewhat higher; it will be shown below that the speed of Control Memory is not a critical determining factor in system speed.

As pointed out by B. Arden,[12] it is desirable in a remapping (paging) system that all possible logical addresses exist in the hardware, irrespective of the actual size of physical memory. In those 213's that are provided with paging and in all Advanced Machines, all locations from 0 to $2{,}097{,}151_{10}$ will exist as valid logical addresses. The leftmost 11 bits of each logical address select a "logical page number" in Control Memory, which contains, in addition to 13 bits of memory-protect information applicable to the associated 1024_{10}-word block of physical memory addresses, the 11-bit number identifying that particular physical block number as assigned by the Executive.

A typical program being executed operates for significant periods of time with two pages of memory being accessed repetitively; strings of instructions (most of them half-word in space occupied) are executed in address-number sequence until a jump occurs, and many jumps are to other instruction strings within the same thousand-word page; most instructions contain one address and reference one operand location, many successive instances of which either will be in unbroken sequence or will be in other locations within the same thousand-word page. Thus, a page number is likely to be repeated many times in succession, so that most references for either program or data to Control Memory would be redundant with the previous reference of the same type.

In order to avoid repetitious translation of page numbers (i.e., evocation of block numbers and of memory-protect data currently applicable to each), the currently-most-active 16 half-words of Control Memory will be retained in registers in Memory Control. Reference to these registers, when appropriate, is automatic,* as follows:

Each Active Page Register is 35 bits in length, consisting of an 11-bit Logical Page Number and a 24-bit string containing the contents of a half-word of Control Memory.

Every memory access request to Memory Control causes all 16 of the 11-bit fields to be checked simultaneously. An exact match causes the Control Memory reference to be bypassed, since the desired paging information exists in the 24-bit string representing the desired half-word in Control Memory.

If no match is found, a normal Control Memory access is performed. The 24-bit string of information about that page number, in addition to being applied to the processes of block selection and memory protect, is then inserted to replace the previous contents of that Active Page Register that has been least active during a fixed previous time period (6 milli-

* We have been unable to determine the author of this concept; we believe he was a student associated with one of the university computing centers.

seconds in the 213, 1 ms in the Advanced System) as determined by simple digital hardware.

Preliminary examination of samples of compiled coding from various types of programs has convinced us that, at the thousand-word page size and with two processors active in either a 213 or an Advanced System, the probability of a page number currently standing in one of the Active Page Registers is between 0.97 and 0.99; i.e., from 97% to 99% of page translations will not access Control Memory.

Control Memory access times (worst cases will be about 800 nanoseconds in the 213 and 140 nanoseconds in the Advanced System) will be covered by other logical operations in about 30% of the instances in the 213, and about 80% in the Advanced System; Active Page Register access times (about 90 and 12 nanoseconds for the two systems) will be covered in about 60% and 80% of the instances respectively.

Thus, the typical time costs of paging will be as follows:

213: For perhaps one-fiftieth of the instances, an average of 550 nanoseconds; for other instances, an average of 36 nanoseconds; effective increase in instruction execution time, about 45 nanoseconds.

Advanced System: For perhaps one-fiftieth of the instances, 28 nanoseconds; for others, slightly less than 10 nanoseconds; effective instruction time increase, about 10 nanoseconds.

A remark on page size: The dollar cost of paging hardware and the time cost of its use both increase rapidly as page size is reduced. Smaller page sizes permit memory assignments by the Executive to be performed with greater precision (i.e., with less unassignable memory within pages) and hence reduce the probability that a particular job initiation will require repaging of all active jobs. Our earlier thinking tended toward 256 words as the size of a page; development of the Implicit Base Register Selection mechanization shifted the optimum page size to 512 words for the 213 and 1024 words for the Advanced System. The size chosen is, as noted above, 1024 words for both systems.

With regard to the type of Control Memory, our earlier thinking centered on very fast nondestructive-read memory, which is presently designable with about 80 nanosecond read-access time and about 5 microsecond write time. The combination of medium-speed Control Memory plus Active Page Registers is slightly cheaper and considerably faster at the present time, and we do not anticipate a relative change in these factors by the time of manufacture of the first Advanced Systems; hence, the latter scheme was chosen.

For the remainder of this Section we shall, for the sake of brevity, refer to page number accesses as though all were to Control Memory; as explained above, while they may be so considered for program planning purposes, most memory accesses in fact use only the Active Page Registers.

The second Control Memory field we shall discuss is Memory Protect Mode, which contains 2 bits and which is used together with the 2-bit Processor Mode field and the 2-bit Instruction Class field to establish the permit-reject decision prior to checking of the processor's numerical key against the memory page's numerical lock.

Figure 11 displays the results of application of these four items of information. "1" indicates that the action is permitted if the key fits; "0" indicates that an error trap will occur irrespective of key-lock relationship.

We shall refer to processors in Executive Mode, Job Mode, and Common Routine Mode

	PROCESSOR MODE	M.P. MODE	READ	WRITE	I/O	JUMP
EXECUTIVE	00	00	1	1	1	1
	00	01	1	1	1	1
	00	10	1	1	0	1
JOB	01	00	0	0	0	1
	01	01	1	1	0	1
	01	10	1	0	0	0
COMMON ROUTINE	10	00	1	1	1	1
	10	01	1	0	1	1
	10	10	1	0	0	1

MEMORY PROTECT MODES

Figure 11

as E processors, J processors, and CR processors, respectively.

Memory Protect Mode 0 is used only for the executive program and for data reserved to it. All actions—Read, Write, I/O, Jump—are permitted to E and CR processors; none, to a J processor.

Memory Protect Mode 1 is used for the normal job-assigned memory area, for program and data alike. All actions—Read, Write, I/O, Jump—again are permitted to E processors; Read, Write, Jump, to J processors; Read, I/O, Jump, to CR processors.

Memory Protect Mode 2 is used for Common Routines by E and CR processors, both of which can Read and Jump to Mode 2 pages; E processors can Write into them. It is also used for Common Data by J processors, which, as explained in Section 4.2, can Write into them under hardware-controlled protection against writing into a Common Data area when any other processor is either Reading or Writing into that same Common Data area. The latter (separate) protection scheme is not restricted to full-page delineation, but may be assigned under program control to continuous-address strings of any number of full words for each Common Data area, within which two words are required to hold a Read-tally word and a Write-indicator word.

A Memory Lock Number of 11 bits is stored in the Control Memory half-word for each page of memory. A Key Number is placed in the 11-bit Key Number Register in a processor each time it embarks on a task, whether in E, J, or CR Mode, and the same number is assigned to all memory pages permitted to be accessed for that task.

The ability to assign different key numbers for various kinds of executive functions was provided as a debugging aid, primarily for the circumstance in which more than one processor is in E Mode at the time of a trap brought on by a program fault within the Executive program itself.

In this paging scheme, the actual moving of information in connection with memory reallocation or job relocation is replaced by the process of renumbering memory pages, which in the 213 takes about 2.1 microseconds per two pages. This corresponds to a move speed of about 1000 words per microsecond.

It may be remarked that the page-renumbering concept in memory allocation is analogous to the register-renaming process used within many of the algorithms in the 212 and the 213 to save the time that would otherwise be required to move data from register to register.

The paging concept described here is a particularization of the Adjoint Addressing scheme proposed by Cheydleur.[17]

8. CONTROL OF INPUT/OUTPUT DEVICES BY CENTRAL *VS.* AUXILIARY PROCESSORS

Several factors affect the decision as to the kind of processors to be used for I/O device control.

At the present time, both total economy and modularity favor the use of one or more auxiliary processors. This assertion requires defense, since a first look would lead one to the reverse conclusion in each case.

As for economy, the auxiliary processor makes excellent use of automatically-assigned buffer memory that is no faster than it needs to be, and is consequently economical. If the processor itself, which is a very-low-cost part of the system, were omitted, most of the I/O cost would remain: passive switches, unit-record buffers, and the I/O devices themselves. Economy is affected by the speed with which the devices can be operated. As an indication of observed performance in this area, consider the operation of the type 101 Processor when used with our R I N G M A S T E R (multiprogrammed) program to control the flow of data between magnetic tape and an arbitrary (limited by memory size because of buffer space requirements, but not less than eight for a 32,768-character-memory single processor) number of I/O devices. When operating with one 600 cards-per-minute reader, one 200-cards-per-minute punch, and two 900-lines-per-minute printers, the printers operate at 96% of rated speed. Initiation of a third printer function does not interrupt other operations, and each of the three printers operates at about 94% of rated speed. We have concluded that economy alone does not justify elimination of the auxiliary processor.

As for modularity, the auxiliary processor takes its output data from, and sends input data to, the mass memory (jointly-accessible drum), and receives its assignment to individual I/O tasks from the Executive. A change in the number of I/O device channels to more than eight requires addition of another auxiliary processor and buffer memory connected through the existing mass memory controller. There is no question of a housekeeping load beginning to affect the speed of the central processor(s), because all I/O devices are on-line with the auxiliary processor(s), and control communication is required only for few-millisecond periods at the initiation and termination of I/O tasks or in response to error conditions.

For computational - colloquy - by - time - sharing operation, the above comments are even more appropriate than for job-oriented operation. Operation of up to a thousand or so keyboard consoles, or several hundred such consoles plus a few dozen high-data-rate CRT-equipped consoles, requires the full resources of a reasonably high-powered special-purpose switching computer. The Type 170, as used in the 212 Time Sharing System now under development and as used in the second phase of the AUTODIN project, provides such power at a fraction of the cost of similar tasks performed by a general-purpose processor. As an example of the kind of power required, consider that line monitoring and buffer-full checking alone, for a thousand 10-characters-per-second channels, would overtax the full-time capabilities of some machines that until recently would have been considered to be high-powered computers!

Another CCTS consideration is that of remote-service graceful degradation, discussed in Section 1.8. As the number of on-line users goes up, the likelihood grows that it will become utterly unacceptable to shut everyone down because of a system hardware fault. Redundancy of processors, main and auxiliary memories, system switching, and I/O devices required for successful support to remote consoles, together with appropriate system planning and executive programming, will permit remote service to go on, if at reduced-service level, while hardware is being repaired. A reduced-scale version of a single station AUTODIN second-phase system would provide the "foreground" (remote user) assured-continuity capability for a thousand keyboard remotes. Even if the system is not made redundant in central facility as regards foreground users, isolation (by a separate processor) of remote users from direct access to the central facility reduces the amount of central system time that can be lost because of faults in lines or remote units.

The need for over-all system graceful degradation, however, now seems to favor the use of the central system for control of the on-line I/O devices.

For that reason, even though raw system economy is not quite as good as with the special-purpose stored-program I/O controller, we intend, for later 213's and for all Advanced System installations, to provide the option of central processor control of all local I/O devices. Initial 213 installations will use one 101 processor for up to eight I/O devices, with additional 101's as may be required with a larger number of local or remote card/printer stations.

Remote CCTS consoles, also, could be controlled by the central processor(s), but our present plans call for full-time use of one or two communications switching stored-program processors, Philco Type 170, for this purpose. This choice was made as a result of preliminary calculations indicating that use of central processor(s) for control of remotes, would add significant CPU workload if sufficiently-rapid (100 millisecond) response is to be assured to an on-line population at the level of 100 concurrently-active users in a 213 system (400 in an Advanced System). This question will be given further study by means of system simulation.

9. COMMENTS ON AN UNSOLVED PROBLEM—MEMORY SCAVENGING

Certain basic logical problems, which must be solved in at least a crude way if we are to achieve fully automatic job segment control at a performance level approaching the optimum, do not yet seem to have been even discussed constructively, much less solved. As an example we will mention one simple problem, that of determining optimum time to release memory space, that is frequently mentioned among sophisticated users but that, to the best of our knowledge, has not yet been seriously worked upon.

How is the system to determine when main memory space should be released for reassignment?

For lack of a rational approach to this problem, we have been forced to avoid it; hence, in the present 213 Executive, main memory space is released only when:

a) Control information has been supplied by the user that specifies when final use of specified memory areas has been made, or
b) A job segment terminates.

Common Routine (with which, in this section, we include Common Subroutine) capability, in order to be used effectively, must be supported by features in the Executive that provide for automatic loading of Common Routines when and only when they are needed but not already in Main Memory and for the automatic release of space occupied by them as soon as they are no longer needed.

As with any job to be loaded, each job that may make use of Common Routines must identify at load time those routines in its control prescript. The special designator signals that each routine so designated must be treated differently, as outlined below.

The Executive maintains two lists to identify the two way association between presently-loaded Common Routines and the programs that call them.

List A, which is in the Executive area, consists of two-word items: the first word of each item is the name of a presently loaded Common Routine, and the other word is the current count of calling programs requiring that routine to be resident in Main Memory.

List B, which is stored in each calling routine's area, consists of variable-length items: each item's first word is the designation of a calling program, and subsequent words of the item identify all Common Routines that it may require.

Usage of these lists will now be evident: as each calling program having Common Routine requirements is scanned for call information, its Common Routine requirements are used to create an item in List B. List A is then searched to determine whether needed Common Routines are already loaded. As each required Common Routine not yet present is loaded, an item is added to List A: the first word is the Common Routine's name, and the second word contains unity. For each called-for Common Routine that is already loaded, the second word of its item on List A is incremented by unity.

As a job segment terminates, its List B is scanned and the second words of all entries of List A corresponding to its Common Routine requirements are decremented by unity and checked for zero value. Those found to be zero are deleted from List A and the space assigned to each corresponding Common Routine is released for reassignment.

While the Common Routine loading control problem as outlined above is relatively trivial, there is another memory allocation problem that is not: we must provide for growing and shrinking memory allocations during a job. The classic example that demonstrates this need is sorting with magnetic tape as the scratch medium. During the Sort phase, high speed requires large memory allocation; during the Merge phase, data space requirements are minimal. Clearly, it is desirable to release much of the original allocation at the end of the Sort phase. In the case of the ubiquitous compile-and-go-user, memory space requirements after compile may go either up or down, and subsequent executions may have data-dependent run-time space needs. With some restrictions, we shall meet the growing-shrinking space challenge.

These restrictions relate to the levels of source language at which dynamism is permitted in memory allocation and release. Our present procedural higher-language processors, for publicly-recognized versions of FORTRAN and COBOL, are perforce language-limited to memory allocation that is fixed at load time. Having developed hardware with facile provision for dynamic memory allocation, we are attracted by the ALGOL concept of automatic allocation of memory upon entry to a program block and automatic release upon exit from a block. We find unappealing a current trend toward inclusion in higher-language processors themselves the direct user-specified control of memory allocation; we feel that this places a premium upon attention to hardware considerations, which is appropriate only at the hardware-oriented language level.

For "job mode" operation as defined in Section 1.5, release determination based upon either a) or b), i.e., at user-signified option, is probably adequate. The job-oriented user can realistically be expected to pay some attention to memory space-time strategy, even if he works in a language that permits some degree of spiritual detachment from the hardware.

In the computational-colloquy time-sharing (CCTS) mode, however, with continually-increasing emphasis upon the *user goal* (answer to problem) as opposed to our traditional preoccupation with *user means* (procedure for problem solving), the user/hardware detachment becomes quite real. In such an environment, we should have a more realistic strategy for scavenging memory space than our present technique of awaiting its evident abandonment by the user. We shall continue to seek such a strategy.

10. SYSTEM GROWTH AND COMPATIBILITY

10.1 212-213

The 213 logically includes the 212 and utilizes much of the same hardware technology. Consequently, some of the 213's operations occur at about the same speed as those of the 212, although in some (such as multiplication) the 213 is at least twice as fast and others, constituting 213 hardware macro-operations, are carried out at much higher speeds than the same macro-operations carried out as programmed sequences in the 212.

The outstanding differences between these systems, as to both hardware and software, are the increased flexibility and generality of the 213 together with its applicability in large multiprocessor installations.

The 213 will serve as a live test facility —alive in that it will be applied to advanced problems by sophisticated user organizations as well as by Philco Programming Research & Development—for development of solutions to the operational problems outlined in Section 3.

New hardware capabilities will handle efficiently some economically crucial aspects of the address-manipulation and program-linkage problems and will provide significant improvement over earlier machines in such functions as memory protection. General answers to some aspects of multiprogramming problems, however, require further study.

For example, it appears that a completely unrestricted general solution to the Common Routine problem, permitting arbitrary linkages to be established among subroutines, may require, even in a system with paging hardware, manipulation of both logical and physical memory page numbers. Approaches to this problem now under study would require use of either content-addressed or associative memory of medium speed in the memory allocation process.

Initial 213 hardware and software fairly frequently will perform relocation of currently resident routines when job segments are loaded or reloaded, accepting relatively minor inefficiency, and limitations of the extent to which automatic micro-segmentation will be economically practicable, for the sake of generality.

10.2 Advanced System

The Advanced System will contain the 213 and like it, will be capable of executing—unchanged—routines written for earlier 2000 systems. In addition to using advanced hardware technology, including a number of new mass memory and input/output devices, and operating at considerably greater speeds, the Advanced System will differ from the 213 in the following respects:

a) Certain engineering design decisions, such as the optimum organization of the (automatic-in-hardware) look-ahead and look-aside portions of processor control and the extent of overlapping of control memory and main memory access, will be different from those in the 213 because of the different relative speeds of some hardware elements and because of the greater operational flexibility that will be required in the Advanced System. Some design decisions must await detailed analysis of actual running programs in user installations in a multiprogrammed operating mode. These engineering choices will be deferred until the needed data are on hand.

b) The extent of vocabulary extension that is economically justifiable beyond that of the 213 will likewise depend upon 213 experience and dynamic analysis of running programs and, in addition, will be affected by programming language developments over the next two years.

c) Software scope for the Advanced System will be considerably greater than that for the 213, although several major items will carry over unchanged and all 213 software will be usable.

11. MASS (BLOCK ACCESS) MEMORY

11.1 In 213

With the 210 and 211, only magnetic tape (90K ch/s data rate) was provided for on-line-accessible program and data files. With the 212, either 90K ch/s or 240K ch/s tape and a high-data-rate (960K ch/s) but slow access (140-240 milliseconds) disc file of 5.2 million words per unit was provided. With the 213, additional flexibility was required.

The 213's Mass Memory Controller provides for up to eight program-compatible units per controller. The units differ in data rate, access time, and in capacity per unit; they consist of an updated verson of the 212's moving-head disc and two (different-data-rate) versions of the same drum unit:

a) Disc—10,485,760 (5×2^{21}) words capacity, 140-240 milliseconds access time, 960K ch/s data rate, 8,192 words per revolution, limited to one access at a time for each unit, 3 milliseconds maximum access time for 8,192-word transfer if prepositioned.

b) Fast Drum—2,097,152(2^{21}) words per unit, 50 milliseconds worst-case access time, 0.6 milliseconds maximum initiation time for 8,192-word transfer, data rate 1.32 million ch/s, single-channel.

c) Multi-Channel Drum — 1,835,008(7×2^{18}) word per unit, 50 milliseconds worst-case access time, 4 milliseconds maximum initiation time for 2,048-word transfer, data rate 320K ch/s on each of 4 unrestricted channels.

We use the moving-head disc for low-access-rate (less than three accesses per second) large-volume files. It will for some time be the most economical unit when considering capacity and cost alone.

The fast drum will be used for job-swapping when numbers of users in a time-sharing system rises to a point that crowds the access-rate capacity of the moving-head disc; the fast drum is practicable for access rates up to 15 per second, using transfers as large as 8,192 words per message.

The multiple-channel drum is used for system program files and for input/output tanking. When used with short messages (typical: 128 words [8 unit records for I/O]), and with sequence-sorting of messages, access rate can be up to several hundred per second; with messages read or written in first-come, first-served sequence, practicable access rate is up to 60 per second. For 2,048-word fixed-size messages, highest practicable access rate is 75 per second.

One of the drawbacks of high-rate rotating magnetic storage devices has been the large buffer areas needed for efficient data handling. The advent of large capacity Slow Core has changed all that; the fastest device can be buffered by a Slow Core memory installation provided only that at least 2 of the quarter-million-word modules are used, and even 8,192-word buffers pose no problems with directly-addressable main memory that includes at least half-million-word capacity Slow Core in addition to the Fast Core memory.

11.2 Mass Memory in the Advanced System

In the past, magnetic tape has served well as the only mass memory device, and it may continue for a long time as the most economical carry-away medium. It should continue as the indicated choice for file processing medium where data files are to be processed in orderly sequence and where files are limited to the order of 100 million characters so that they do not require multiple reels of tape. In general, however, the several-minute access time and the mandatory requirement for handling of reels or cartridges (because total capacity is too small to justify leaving files on-line) will, we believe, limit tape usage primarily to carry-away and ordered-file-processing applications.

While the 213's rotating machinery is relatively powerful by comparison with other mass-memory equipment that has been made available with general-purpose computing systems, a little arithmetic quickly brings one to the conclusion that the system would be hampered in several ways if some limiting parameters were not changed. The critical variables are capacity, access time (Read and Write), data rate, and whether or not carry-away capability exits.

From the standpoint of data rate, the moving magnetic surface performs well. For the high-data-rate problem, the Fast Drum seems likely to continue as the preferred device. The Advanced System version will have the same capacity as the 213 version (2 million words per unit), and the same ability to initiate a one-revolution order promptly. It will rotate slightly faster (35 milliseconds per revolution) and will store 32,768 words per revolution (vs. 8,192 in the 213). It will be capable of one Read and one Write operation concurrently at a data rate of 7.5 million characters per second for each.

Like other organizations, we are examining the available technologies in search of significantly better characteristics for three mass memory functions: (a) Fast Access, implying up to several thousand accesses per second, for blocks of a few hundred characters, are possible; (b) Large Capacity, providing on-line access in no more than one second to data files as large as present-day shelf libraries (10,000 reels of magnetic tape); and (c) Carry-Away cartridge-type files at least as large in capacity as magnetic tape reels but having access times of less than one second.

For the Fast Access function, modern high-resolution slow-decay magnetic-deflection electrostatic storage tubes under development for military applications look interesting. A single tube can store a few thousand lines of a few thousand bits each, to be written or read by slow servo-controlled sweep techniques in a millisecond. The combination of access time, economy, and "militarizability" offered by such a device now appears promising; and for this reason development is being carried forward.

For the Large Capacity function, the same CRT sweep and encoding technology combined with mechanical page-changing in a photographic storage scheme now looks promising; and this, too, is being studied here.

For the Carry-Away function, cartridge-mounted multiple magnetic tape strip devices meet basic requirements. A proposed tape strip scheme, for example, offers access time of less than one second to any word in a cartridge having the capacity of several reels of magnetic tape.

The characteristics of these three developmental devices are summarized in Table 1.

As a hedge on the development of the

		Electronic Mass Store	*Photo Optical Mass Store*	*Tape Strip Mass Store*
	Basic Advantage	Quick Access	Large Capacity	Carry-Away
Capacity, Millions of Words	Total	4.	16,000.	66.
	Page	.065	1.	.016
Max. Access, milliseconds, Any Word	Read	.1	1,000.	600.
	Write	.2	3.6 million (1 hour)	600.
Max. Access, milliseconds, Within Page	Read	.1	.1	35.
	Write	.2	.3*	35.
			*(Assumes 1 million word EMS Buffer)	
Data Rate, Thousands of Char./second	Read	2500.	100.	120.
	Write	1700.	50.	120.

CHARACTERISTICS OF THREE NEW MASS MEMORY UNITS

Table 1

electrostatic block-storage device, we have designed a drum control hardware scheme for optimizing the sequence of issuance of short-message read and write orders. This will permit fixed-block-size messages (128 words) to be issued at a rate of up to 2,000 per second, to or from any block addresses within a group of our present 1.85 million-word multi-channel drum units.

12. SYSTEM CONFIGURATIONS

12.1 Minimum System Requirements

The standard 213 and Advanced System software will require the following minimum hardware configuration:

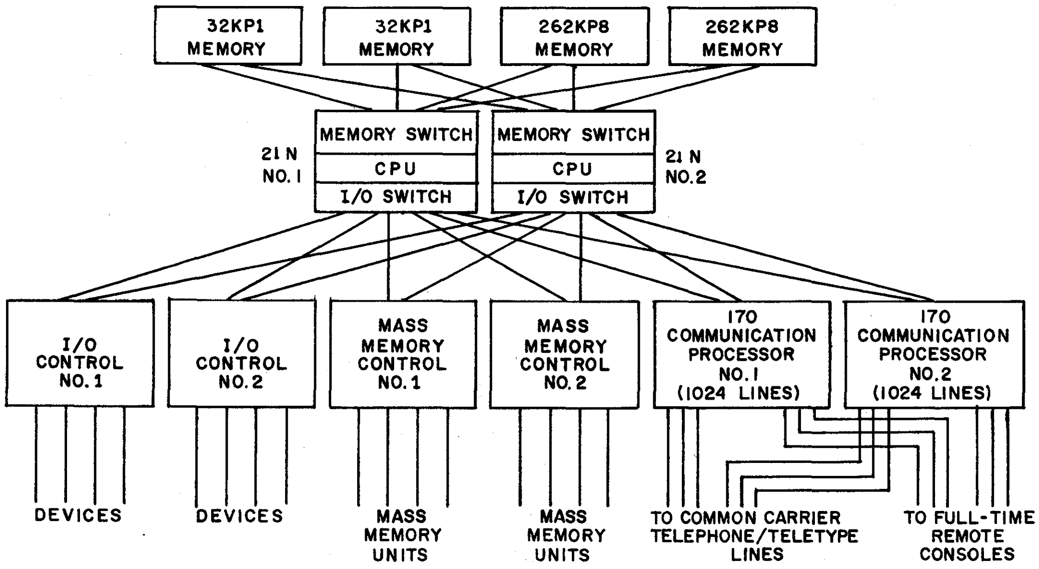
One Central Processing Unit
32K Fast Core
One I/O Switch
One Mass Memory Control
One 1.85M Multi-Channel Drum
One 101 I/O Device Control with
  One Card Reader
  One Card Punch
  One Printer
  One Magnetic Tape Unit
    or Tape Strip Unit

Note that the software is insensitive in particular to the number of processors in a system (up to the design limit of four); that fast and slow directly addressable memory can be mixed in any combination up to the design maximum of 2 million words; and that magnetic tape or TapeStrip is used by the software only for system loading or carry-away purposes.

12.2 Switching Redundancy

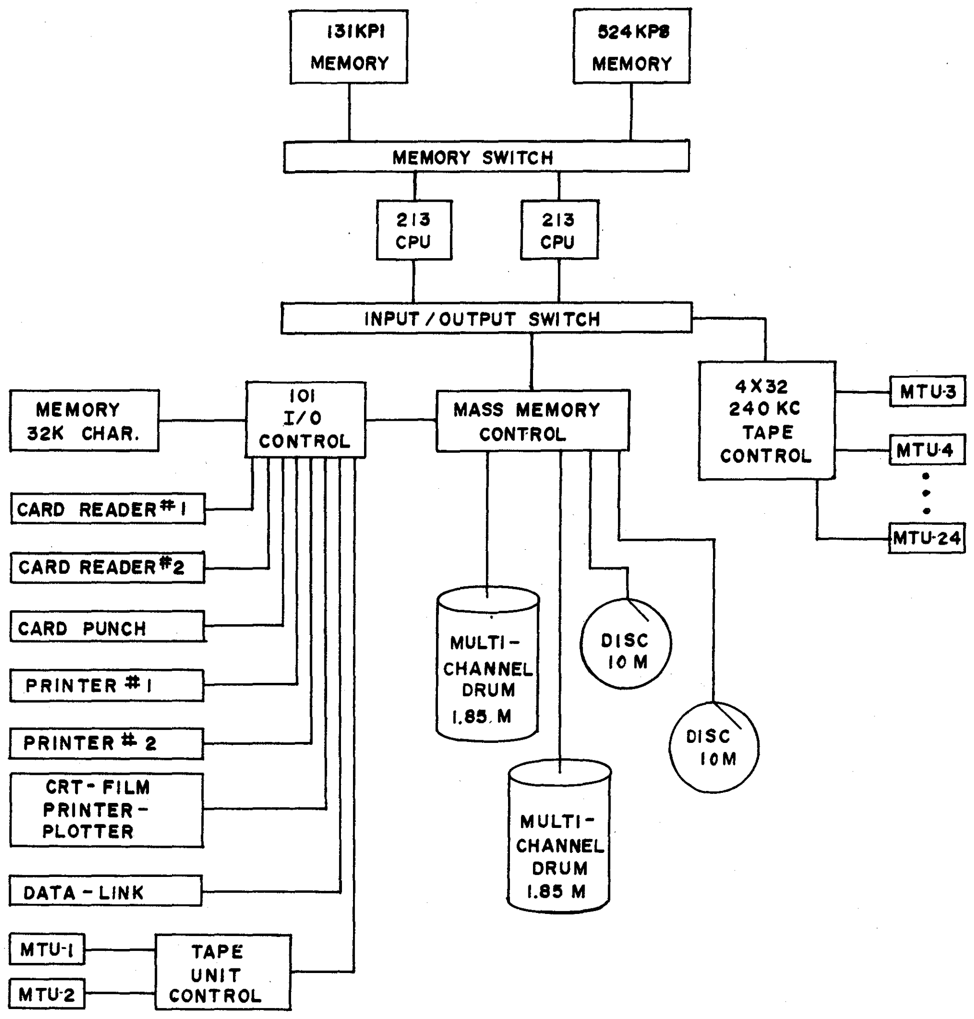
The configuration diagrams below will, for simplicity in discussion, show Memory Switch and Input/Output Switch hardware as though it were centralized in two separate units. In fact, for the sake of Graceful Degradation (Section 1.8) as well as for performance and cost considerations, active electronics of the switching hardware is physically distributed among the Central Processors, as shown in Fig. 12. It may be seen that every major physical unit (32K units of fast memory, 262K units of slow memory, each I/O Control and Memory Control, and each Communications Processor) is physically connected to each Central Processor Unit with its associated (redundant) Memory Switch and I/O Switch hardware. Thus, no single fault, including one in the areas of system switching, can shut down more than half the system.

The arrangement shown in Fig. 12 does not provide within the system for redundancy


32KP1 MEMORY
32KP1 MEMORY
262KP8 MEMORY
262KP8 MEMORY
MEMORY SWITCH
CPU
I/O SWITCH
21 N NO. 1
21 N NO.2
I/O CONTROL NO. 1
I/O CONTROL NO. 2
MASS MEMORY CONTROL NO. 1
MASS MEMORY CONTROL NO. 2
170 COMMUNICATION PROCESSOR NO. 1 (1024 LINES)
170 COMMUNICATION PROCESSOR NO. 2 (1024 LINES)
DEVICES
DEVICES
MASS MEMORY UNITS
MASS MEMORY UNITS
TO COMMON CARRIER TELEPHONE/TELETYPE LINES
TO FULL-TIME REMOTE CONSOLES

SYSTEM SWITCHING REDUNDANCY

Figure 12



213 INSTALLATION INCLUDING ON—LINE TAPE

Figure 13

in access to remote consoles. The dialed-up remotes have protection in that initiation of a remote connection through the common-carrier network will encounter "busy" signals from all lines into a 170 that is down and consequently will be directed through the other 170. Connections that are active when a 170 fails, however, may be vulnerable; and each full-time remote is shown associated with a particular 170 so that half of them will go down if one 170 fails.

Both of these restrictions can, if desired, be removed by providing, as mentioned in Section 12.4, switching redundancy between the 170's and their remote lines.

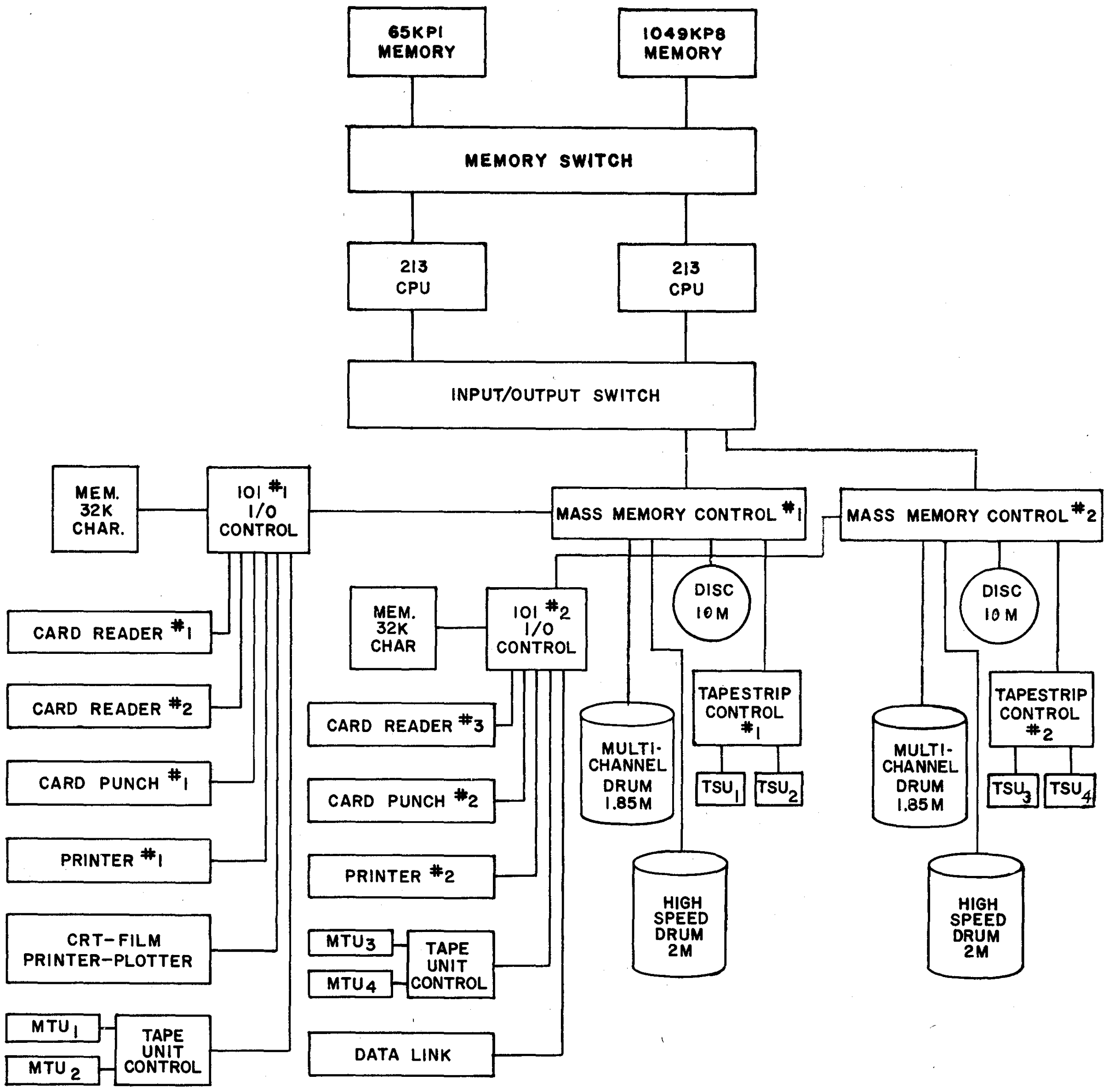
12.3 213 Systems

A typical medium-size 213 installation that will use magnetic tape for file processing is shown in Fig. 13. This particular system uses four 32K Fast Core memory modules and two 262K Slow Core memory modules.

A 213 system that does not use magnetic tape on-line (tape is used only for auxiliary input/output through each 101 I/O Control) is shown in Fig. 14. In this system, the CRT Printer-Plotter is associated only with one 101, and its service continuity, together with that of the other I/O devices associated with that 101, is thus vulnerable. Likewise, the Data Link interface and the remainder of the I/O devices are associated only with the second 101, so that those devices together with all of the remote card/printer stations will go down if the second 101 fails.

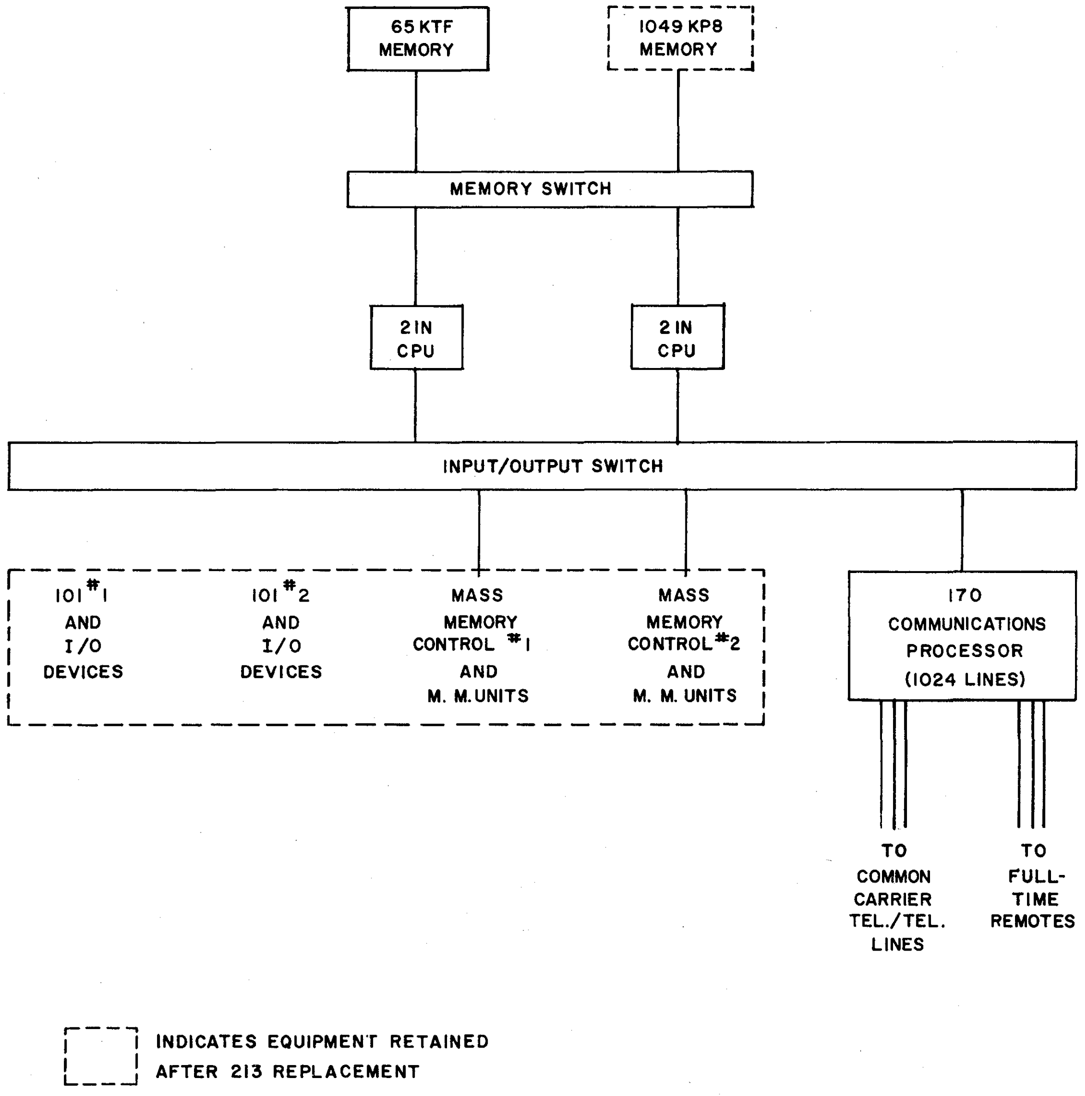
12.4 Advanced Systems

An Advanced System configuration including a single-processor CCTS Time Sharing facility is shown in Fig. 15. The particular system shown is intended as a growth replacement for the system of Fig. 14, with the addition of CCTS. At this stage, the system will provide "graceful degradation" as described in Section 1.8 for only job-oriented work; a major failure in the Type 170 Communications Processor system could shut down the time-sharing capability until repair has been performed. Note that only the central processors, high-speed



213 INSTALLATION WITHOUT ON-LINE TAPE

Figure 14


65 KTF MEMORY
1049 KP8 MEMORY
MEMORY SWITCH
2 IN CPU
2 IN CPU
INPUT/OUTPUT SWITCH
101 #1 AND I/O DEVICES
101 #2 AND I/O DEVICES
MASS MEMORY CONTROL #1 AND M. M. UNITS
MASS MEMORY CONTROL #2 AND M. M. UNITS
170 COMMUNICATIONS PROCESSOR (1024 LINES)
TO COMMON CARRIER TEL./TEL. LINES
TO FULL-TIME REMOTES
INDICATES EQUIPMENT RETAINED AFTER 213 REPLACEMENT

ADVANCED SYSTEM WITH SINGLE 170

Figure 15

memory, and memory switch need be changed when upgrading from the 213; the 213 equipment retained is in the dotted boxes.

A more comprehensive system providing for graceful degradation in both job-oriented and CCTS capabilities is shown in Fig. 16. This duplex-170 installation, with distributed-switching reconfiguration facilities, is modeled on the pattern of the AUTODIN second-phase system as mentioned in Section 1.8. Note also that passive-switch IOCU's have been used for I/O device control, operated under direct control of the central processors and arranged so that loss of a single IOCU will not cause loss of any I/O devices (including Data Links to remote card/printer stations). Also, redundant-switching Mass Memory Controls have been used, so that failure of one control unit does not result in loss of mass memory units.

13. ADVANCED SYSTEM TECHNOLOGY AND DESIGN[20]

13.1 Circuits[20]

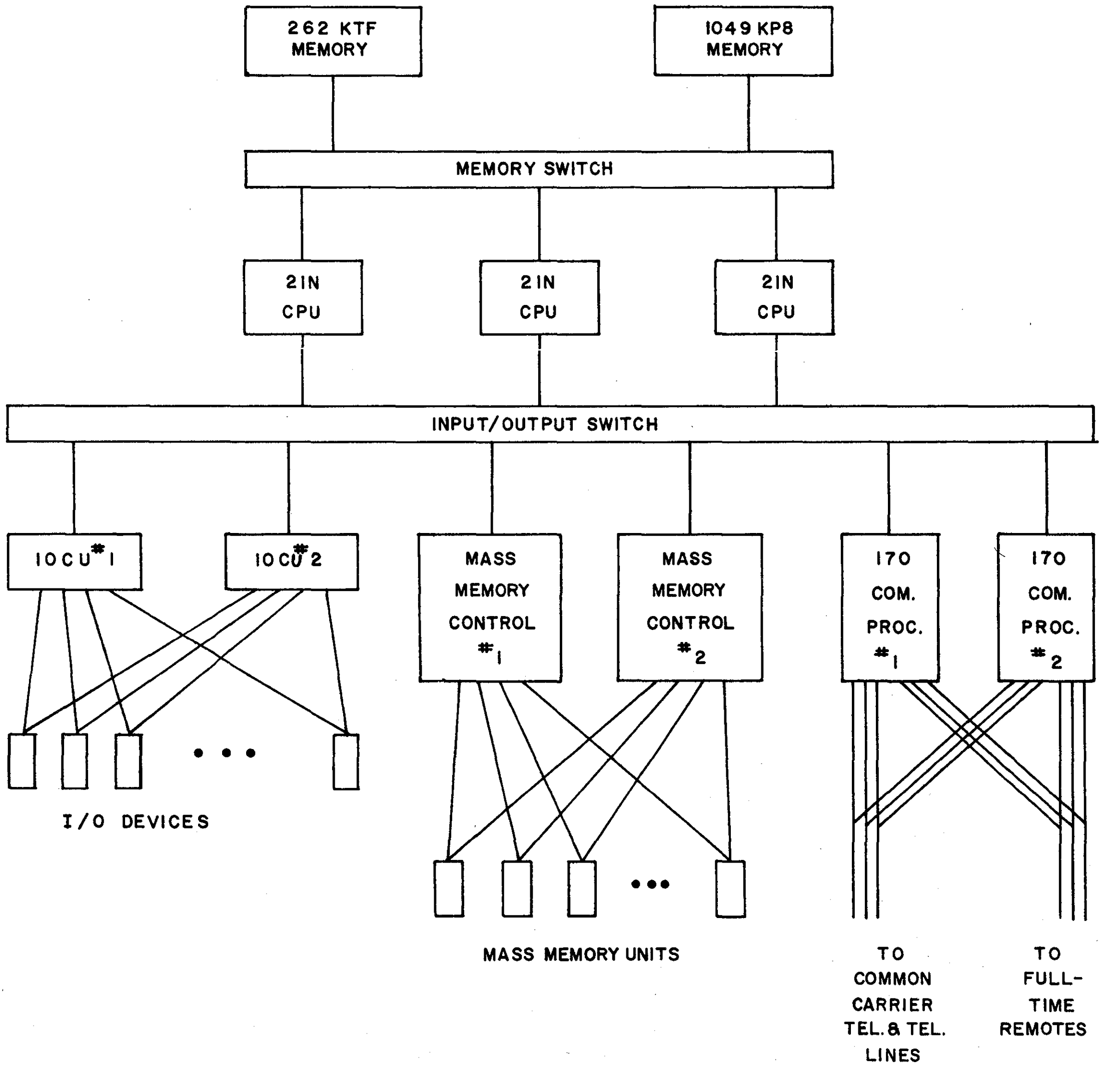
To achieve the performance goals of this advanced system, significant gains are required in the effective circuit speed. The effective circuit speed relates to the speed of the circuitry as a function of inherent delay, method of interconnection, complexity of logic performed, and method of packaging.

The advanced processor, unlike that of previous computers, requires an extensive integration of circuit logic, and packaging technology.

It is also apparent that unless the system performance requirements are recognized at the most fundamental level, namely, the basic circuit, performance requirements will not be achieved. The new machine will use circuits which take full advantage of the advances in technology yet allow utilization of prior experience in producing large-scale reliable equipment.

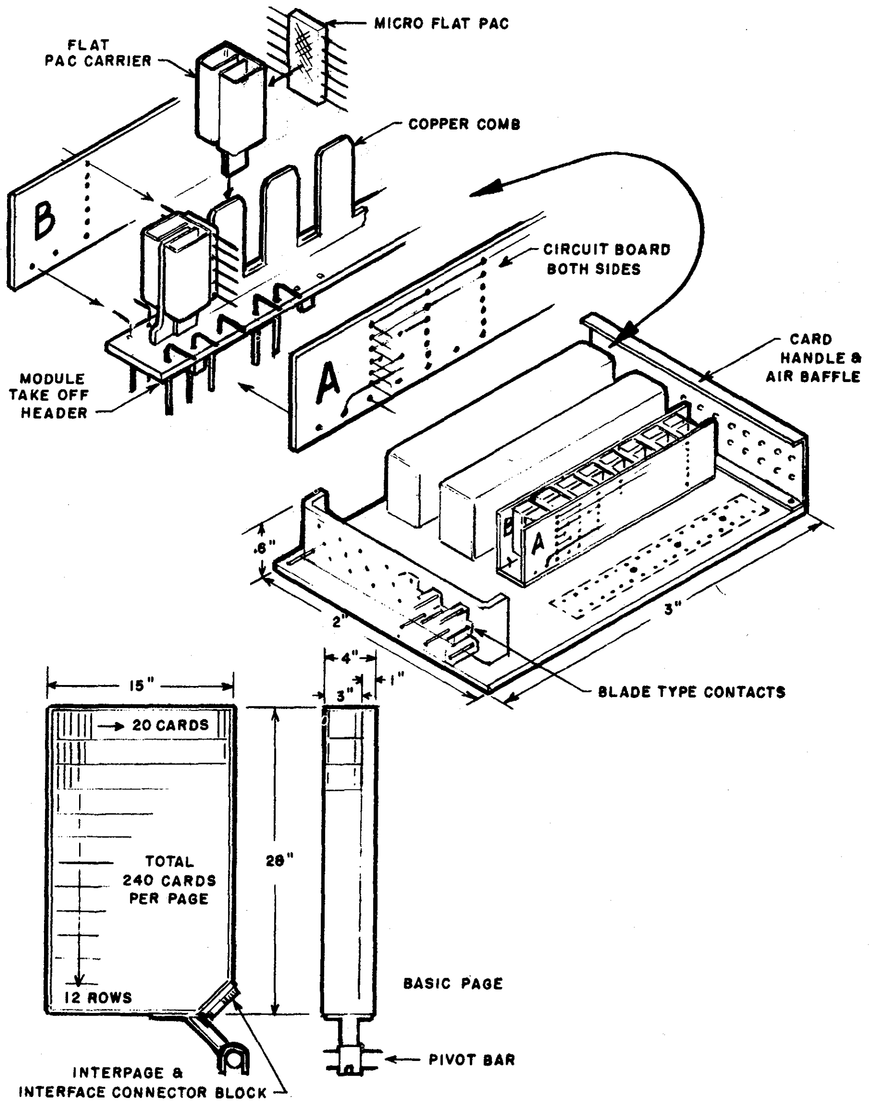
The system requires circuits capable of handling nanosecond pulses with minimal distortion. Anti-saturation circuit techniques, double level logic and smaller geometries are used to achieve this circuit speed. Particular emphasis in development has been placed on those circuit forms which permit total integration.

The advanced system will use a high-speed, anti-saturated, readily integrable diode



ADVANCED SYSTEM WITH FULL REDUNDANCY

Figure 16



THE BASIC CARD

Figure 17

transistor logic element. Typical circuit delays of 4-6 nanoseconds have been demonstrated for fanouts up to 6. This circuit features minimal design variation with loading with emphasis placed on total integration. The object is to achieve the fastest circuit speed within the framework of an integrable machine. Overall system speed is then achieved through the signal path reduction resulting from the use of integrated circuit techniques.

13.2 The Basic Card[20]

The basic pluggable assembly is the functional card, conceived on a generic card scheme whereby multiples of the single card can be arranged to form any major logic block in the machine. The single card contains 4 modules. Each module in turn is composed of eight integrated flat-pack-type circuits. Card connections are made through blade type connectors and are arranged to permit pin densities for combinations of circuits within the basic card. Interconnections within the card are consistent with the high speed attained throughout the system.

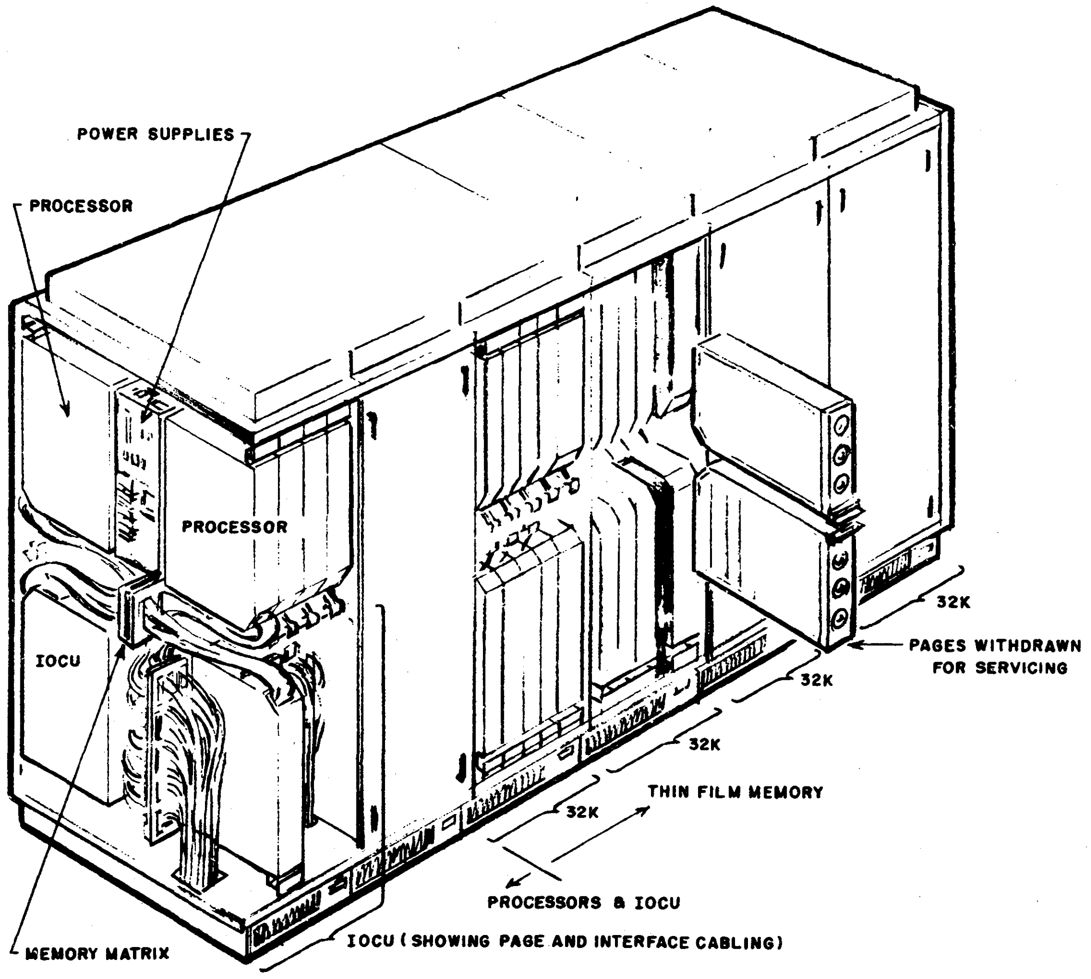
The basic module, Fig. 17, is composed of a copper comb which acts as its main support, heat sink, and ground plane. All microflat packs are attached to spring temper brass carriers which slide over the master comb. The two printer cricuit cards carry all signals and voltages to the lower header assembly. A wave soldering operation completes the module package.

Four of the basic modules are soldered into a printed circuit card which becomes the smallest pluggable assembly. The end of the card contains a plate which acts as a handle, a test point plate, and an air baffle. When all cards are plugged into a frame, the back pan acts as one side of a chimney and the card plates act as the opposite side, thus assuring a positive unidirectional air flow.

The entire package is designed for automated assembly.

13.3 The Page Assembly

To build a complex of basic cards and maintain minimum lead length, a spinal column design is used. The entire processor is based on a horizontal wiring spinal column with branch wiring to all subassemblies. The modular hinged pages have hinge points on the bottom forward edges of the upper pages and on the top forward edges of the lower pages. This design allows easy withdrawal and minimum system interconnection lengths. Connectors mount in the front of each page. Interpage wiring proceeds through plastic cable conduits. To conserve head room because of the diagonal radius from the pivot point, all pages have a 2 to 1 height-to-width ratio and afford fore-and-aft access to the cabinet. This type of con-



SYSTEM PLAN

Figure 18

struction results in a high volumetric efficiency and reduces the number of interpage cables.

The front of each page contains indicators and test points for maintenance operations. In the extracted position, all page panels will be oriented for normal viewing.

High density packaging of high-speed circuits is necessary to reduce interconnection wire lengths. This reduction minimizes wire delay and permits a maximum number of connections without recourse to coaxial or similar transmission line wiring. Since it becomes more difficult to make interconnections as the component density increases, a technique has been devised which preserves the fast wave fronts necessary to small delay switching. Also, this technique provides accessibility to the individual elements of the device.

13.4 The Memory Matrix[20]

The Memory Matrix, or interconnection bus, Fig. 18, is the central spinal column for the main frame. The matrix connects all processors and IOCU's to all thin film memory banks.

Its construction embodies the use of modified strip lines in a compact assembly. All points in the matrix terminate in fork type connectors which are mounted on the matrix top face and interconnect all receiver and sender circuitry.

These circuits mount to matrix printed circuit cards. A complement of cards is cabled to each processor, IOCU, and 8K memory bank. Any card complement for a specific machine configuration has its own built-in circuitry for future expansion to the maximum configuration.

13.5 Processors[20]

The main frame is comprised of individual processor sections and banks of 32K thin film memory. The page design uses the processors to form the left two cabinet sections and thin film memory pages to form the right sections. The entire device is constructed in modules of processors, IOCU's and banks of 32K of thin film memory. The spinal column wiring assembly is a memory matrix of interconnection bus which connects all processors to all memory banks.

Each individual processor, like its 212 and 213 predecessors, will use an overlapped internal organization permitting use of a controlled amount of look-ahead. In addition, true parallel processing within a single sequence will take place in the indexing and operand fetch portions of successive instructions due to the use of multiple identical units within each processor.

Figure 18 shows details of the page installations, interface cabling, processor and memory sizes, power supply locations, and individual page maintenance.

13.6 Thin Film Memory[20]

The magnetic thin film memory to be used in the Advanced Machine is a large capacity, high speed storage unit, packaged on modular pages, and communicates with the processors through the memory switch.

The word size of this memory is 50 bits: 48 information plus 2 parity. The cycle time is 250 nanoseconds, read-to-read, and access time is 100 nanoseconds, measured from the input of the address register to the output of the data register. Each memory bank contains 32,768 words simultaneously addressable in 8,192 word sections. Two adjacent words are read out simultaneously internal to the storage unit. Only the selected word is presented to the system. The unit is a linear select (word organized) unit with destructive read and write restore.

To attain the fast access and cycle time desired, the electrical, mechanical, thermodynamic, production, reliability, and maintainability aspects of the design are considered as an integrated whole.

The electrical, mechanical, and material tolerances require the memory to be linear-select and to operate with destructive read-out with digit write-restore. The storage media is composed of a matrix of magnetic film substrates. Two words are read out simultaneously and write-restores to minimize the decoding and drive hardware required; the desired word is selected in the memory logic section for presentation to the Memory Data Register.

The thin film memory features a special disturb-insensitive film allowing complete matched strip-line construction with constant line impedance, and low impedances to be used throughout the system with very low crosstalk and minimal word to word coupling. In addition to providing a high bit density, the film permits strip-line construction.

The use of matched lines provides no ringing on digit or drive lines, and permits use of fast rise times and short pulses with the resulting larger read-out voltages and shorter cycle times.

A high frequency, balanced, differential sense amplifier constructed in micro-technology in conjunction with the exact geometric construction of the system provides excellent results in read out and lends itself well to the integrated system.

The thin film memory pages (Fig. 14) contain blocks of 2,048 words of memory. Two plates compose a single page. Four pages make up 8K of thin film memory. Associated driver, switch, and sense amplifier circuits are arranged to tie closely into the control and timing circuits located in a logic page servicing 4 memory pages.

13.7 Mechanical and Thermal Considerations

Empty pages are positioned throughout the cabinetry to allow for interface cabling. Servicing and maintenance is accomplished by pulling a top page to its horizontal position where it will lock in, and lifting a lower page to engage the locked out upper page. Locking is accomplished by a quick disconnect at the handles. This design precludes an exotic and costly holdout mechanism for the lower pages. The major Processor/System interface occurs in the IOCU sections. Cabling enters the cabinet through the subfloor and into the IOCU cable matrix.

Thermal densities in the order of .7 watts/cu. in. occur throughout the main frame. With thermal concentrations of these magnitudes, forced convection cooling is a satisfactory technique. The circuits are designed to operate in a range of 15°C to 50°C. An average internal temperature of 40°C is the top allowable limit which has been set in order to offset the possibility of hot air pockets.

An important consideration is the ability to cool a page in the out, or service, position. This is accomplished by the incorporation of air-moving devices in each page. These air movers are of a type which can deliver approximately 80 CFM at .1 inches of water, and not create a loss of head due to own physical size since they must stand in the main cabinet air stream. Two or three of these fans will adequately cool a page in the servicing, or out, position.

An interlock on each page will apply power to the fans while the cabinet is closed for normal operation.

There is a blower chamber associated with each bank of four pages both top and bottom. All chambers are self-contained units, which have their own inlet ducts, filters, and outlets ducts with collapsible ports. The collapsible ports will close when a page is opened for servicing, thus keeping air paths intact. Each blower must deliver approximately 200 CFM against a .5 inch head. The final selection has been made on the basis of CFM and acoustical levels.

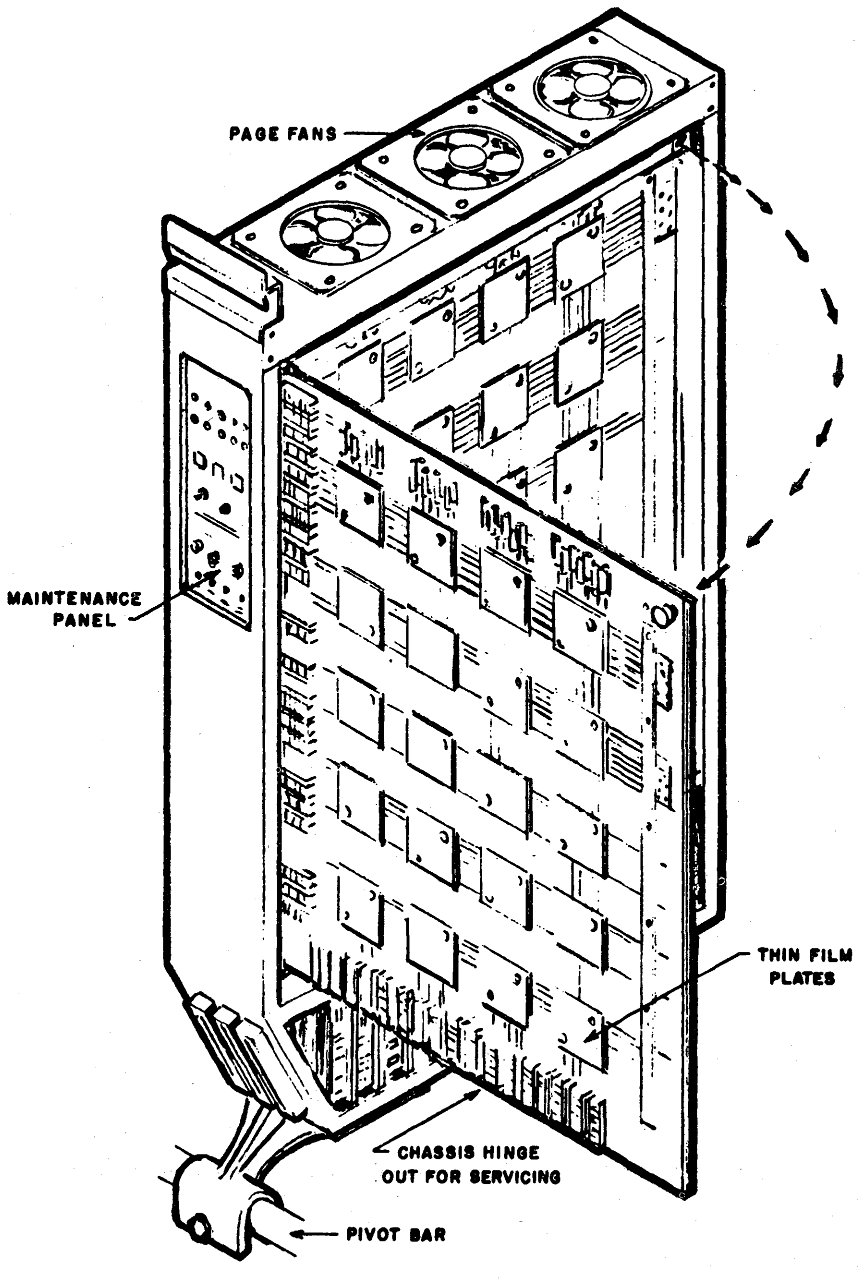
A forced convection system, as described, will adequately maintain the system temperatures between the limits of 20°C to 50°C with an ambient of 20°C ± 5°C. The normal operation with doors closed will effect a vertical cabinet air flow sufficient to remove 10 kw. of heat load. During normal operation, page fans will remain off. Their low surface area will induce very little restriction to air flow. During servicing, a page is withdrawn, an interlock closes power to the saucer fans, and the faulty page thus picks up its own air mover. The bottom blowers keep air moving in the remainder of the lower pages and the top blowers maintain cooling air in all top pages. The scheme is entirely modular. Depending upon the cabinet population, processor, IOCU, memory, etc., blower assemblies can be added or subtracted, page fans can be employed from none to three depending upon the cabinet heat load.

14. SOFTWARE IMPLEMENTATION PLANS

One of the decisive reasons why we have chosen upward compatibility for the 213 and Advanced Systems is that we are convinced the priceless ingredient of really good software is abrasion against challenging users.

The first 213 systems will operate initially with 212 software. A few months later, the following software items will be delivered:

a) 213 SYS, a multiprogramming-multiprocessing executive system



THIN FILM MEMORY PAGE
Figure 19

b) 213 TAC
c) 213 FORTRAN IV } The 212 Assembler and FORTRAN IV Compiler will be generalized to use 24-bit addresses throughout and to make use of mass memory.

d) All 212 software, except 32K SYS, TAC, and the FORTRAN-II and -IV compilers, primarily intended for 32K machine applications, will be integrated to run under the new executive system.

The major products to be integrated are: COBOL-61; SORT II; PERT III, PERT COST; STAT-2000 statistical analysis system; LP-2000 linear programming system (input-compatible with LP-90); XORD self-buffering I/O routine; PIOSGEN I/O generator; PIOS I/O interpreter; APT-III numerical control system; REPORT generator; FILE selection generator; SIMSCRIPT system simulator; and LIBrary mathematical, utility, and service routines and subroutines.

Work on the 212 Time Sharing System will be converted to an extension of the 213 executive system; this extension will be made available for general use about six months later.

Initial Advanced System hardware deliveries will be made about two years after initial customer-site usage of items a) through d).

Thus, both 213 and Advanced Systems will be utilized initially with software that has been used for about two years by customers.

Philco's 1960-released ALTAC compiler, and its successors through the present ALTAC-III compiler which includes FORTRAN-II as a subset, contain these source language features beyond FORTRAN:

1. Symbolic statement labels (begin with alphabetics).
2. Compound statements ("statement; statement; statement").
3. Mixed expressions (integers are floated).
4. Array dimensionality up to 4.
5. Greater subscript generality: variable subscripts attaining zero or negative values are handled properly; subscription of subscripts to arbitrary depth is allowable.

We are pleased that most of these features, among others, are now being considered for adoption in newly proposed procedural languages. 212 users have written compilers for two of the three recognized dialects of ALGOL-58, MAD and JOVIAL. We are studying the applicability of these languages in addition to ALGOL-60, and of other procedural languages not yet publicly defined; with the intent of selecting appropriate languages and language extensions for future compiler development for release as Philco-maintained 213 software products.

15. CONCLUSIONS

The trend toward broadening the application of million-or-more-operations-per-second computing systems seems to us to be unmistakable. At one extreme, more comprehensive solutions of the most complex problem types are being planned, and improvements are being made in the rapidity and efficiency with which service is provided to the requestor of such problem solutions. At the other extreme, the computational-colloquy mode of access to large systems is making them conveniently available to requestors of solutions to relatively small problems, providing not only the superior economy of the super-scale machine, but also its vast power and flexibility together with a continually-broadening range of software. This mode of access to very large machines is also beginning to demonstrate a new order of effectiveness of computation service to research and development organizations whose technical productivity is closely related to the promptness and predictability with which computer solutions to new problems can be prepared and extended.

The effectiveness of multiprogramming in large-system efficiency and the planning flexibility and graceful degradation potential of the multiprocessing system seem to have been accepted as realistic and justifiable goals by many of the most advanced large-system users. We believe that broad application of these concepts in large systems is inevitable.

The Advanced System that is the subject of this paper is intended not only to provide much greater speed and economy but to include in its hardware and software highly efficient and quite general capabilities for facing the challenges outlined in Sections 2 and 4, including that of The Common Routine and Common Data.

16. ACKNOWLEDGEMENTS

On behalf of the people working on the development of this system, I wish to express our gratitude to university staff members who have helped us to understand the nature of certain programming and hardware problems: F. J. Corbato', B. Galler, A. Perlis, and several of their colleagues.

On behalf, in turn, of the rest of Philco management, I wish to express *our* gratitude to the members of the Willow Grove technical staff who have done the hard-core technical work. In particular, R. Brown and J. Spratt of the Engineering Department and G. Cadwallader, L. Ellerson, J. Guernaccini, and A. Shapiro of the Programming Research and Development Department have been enthusiastic, ingenious, and durable.

17. PATENT NOTICE

We believe there is patentable subject matter here; and patent rights are being sought. We emphasize this fact with regard to the hardware-software mechanization of Implicit Selection of Base and Limit Registers and to the consequent procedures for automatic and protected access to Common Routines, which are due to J. Guernaccini; and to those for Common Data, and Exclusive-Occupancy Routines, due to A. Shapiro; in a multiprogramming-multiprocessing computing system.

18. REFERENCES

1. R. Segal, J. Maddox and P. Plano, "Performance Advances in a Transistorized Computer System," *Proc. EJCC*, 1958, p. 168.

2. F. J. Corbatö et al., *The Compatible Time-Sharing System*, Cambridge, Mass., M.I.T. Press, 1963.

3. H. S. Bright and B. F. Cheydleur, "On the Reduction of Turnaround Time," *Proc. FJCC 1962*, Washington, D. C., Spartan Books, 1962, pp. 161-169.

4. M. Kory and P. Berning, "The STL Integrated Operating System," *Proc. ACM 19th Nat. Conf., Aug. 1964*, N. Y., ACM, pp. E2.1.1-E2.1.25.

5. L. R. Lavine, "The Dynamic Rescheduling Operator System 212SYS," *Proc. 13th TUG Meeting*, Washington, D. C., Sept. 1963.

6. R. L. Patrick, "Measuring Performance," *Datamation* 10/7, July, 1964, pp. 24-26; also privately published report, "Time-Sharing, Its Uses and Abuses," 1964.

7. R. M. Fano, "The MAC System: A Progress Report," Technial Report MAC-TR-12, M.I.T., Project MAC, Cambridge, Mass., Oct. 1964.

8. Jules I. Schwartz, "Command Research Laboratory: Introduction to the Time-Sharing System," SDC SP-1722, 14 September 1964.

9. J. B. Dennis and E. L. Glaser, "The Structure of an On-line Information Processing System," to be published as a chapter of a forthcoming book, "Information System Sciences: Proceedings of the Second Congress," Washington, D. C., Spartan Books, 1965.

10. A. B. Shafritz, "The Use of Computers in Message Switching Networks," *Proc. ACM 19th Ann. Conf.*, N. Y., ACM, 1964, pp. N2.3.3-4. (See also remarks in papers by B. R. Jacobellis and F. S. Vigliante, ibid.)

11. A. W. Holt, "Program Organization and Record Keeping for Dynamic Storage Allocation," *Information Processing 62*, Amsterdam, North Holland Publishing Co., 1962.

12. B. Galler, B. Arden and F. Westervelt, private communications.

13. A. Perlis, private communications.

14. Philco Corporation, "Philco 212 Reference Manual TM-29," Willow Grove, Pa., 1963.

15. Work sponsored under contract with the Defense Communication Agency.

16. R. W. Bemer et al., "American Standard Code for Information Interchange," N. Y., *Comm/ACM 6/8 Aug., 1963*, pp. 442-426.

17. B. F. Cheydleur, "The X-3 System—A Design Study," internal report, Philco Computer Div., 1963.

18. H. S. Bright, "A Proposed 64-character Alphabet for Hardware Implementation of the IAL (ALGOL 58) Reference-Level Language," N. Y., *Comm/ACM 2/5 May 1959*, pp. 7-9.

19. These sections were written by A. Shapiro.

20. These sections were written by T. N. Timlin.

21. H. A. Kinslow, "The Time-Sharing Monitor System," *Proc. FJCC, 1964*, Washington, D. C., Spartan Books, 1964, pp. 443-454. (By private communication, Kinslow credited the dynamic memory mapper to his fellow worker, O. W. Johnson.)

22. W. F. Bauer, "Computer Design from the Programmer's Viewpoint," *Proc. EJCC 1958*, pp. 46-51. To quote from Bauer's paper, "The central idea here is that each large metropolitan area would have one or more of these super computers. The computers would handle a number of problems concurrently. Organizations would have input-output equipment installed on their own premises and would buy time on the central computer much the same way that the average householder buys power and water from utility companies. . . . The accounting procedure would reflect the customer's detailed use. . . . The supervisory control units schedule the multiple problems . . . changing priorities as required and distributing work among the processing units. . . . A very important characteristic . . . is that it is modular in construction, and large components can be added or subtracted to make up the configuration currently in demand at the particular installation. Thus the computer can grow and change to meet changing requirements."

Index

ERRATA

AFIPS, VOLUME 26, PART I

ERRATA—AFIPS, VOLUME 26, PART I

The following are corrections to be entered on the designated pages:

Amemiya, Hiroshi: "*A* 10^5 -Bit High-Speed Ferrite Memory System—Design and Operation"

Page 124, Figure 2	Caption should read Reading
142, line 4	Change to $e^{\gamma x}$
, row 3, col. 3	Change to $u_{s2} - \gamma^2$
, row 4, col. 3	Change to u_{m2}
, row 1, col. 1	Remove parens

Weber, J. H.: "UNISM—A Simulation Program for Communications Networks"

Page 243	Place . after $-\ \bar{B}^2$ in equation

Trilling, D. R.: "The Use of a Job Shop Simulator in the Generation of Production Schedules"

Page 277, line 14, right	Change to read—These statistics are well known
279, line 23, right	Change to read—This type of node is a "call-out,"
, line 45, right	Change to read—to assign the necessary binary codes.
282, line 12, right	Change to read—point to hypothesize a schedule analyst and—

Raphael, Bertram: "A Computer Program Which 'Understands' "

582	Add part h (below) to Figure 3

h. NUMBER

```
(•••.     A BOY IS A PERSUN)

(I UNDERSTAND)

(•••.     JOHN IS A BOY)

(I UNDERSTAND)

(•••.     A FINGER IS PART OF A HAND)

(I UNDERSTAND)

(•••.     HOW MANY FINGERS DOES JOHN HAVE Q)

1 (THE ABOVE SENTENCE IS AMBIGUUUS •• BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
  (I DON•T KNOW WHETHER FINGER IS PART OF JOHN)

  (•••.     THERE IS UNE HAND UN EACH ARM)
2 (I UNDERSTAND)

(•••.     THERE ARE TWO ARMS ON A PERSON)

(I UNDERSTAND)

(•••.     HOW MANY FINGERS DOES JOHN HAVE Q)

3 (THE ABOVE SENTENCE IS AMBIGUOUS •• BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
  ((HOW MANY FINGER PER HAND Q))

(•••.     A HAND HAS 5 FINGERS)


(THE ABOVE SENTENCE IS AMBIGUOUS •• BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(I UNDERSTAND)

(•••.     HOW MANY FINGERS DOES JOHN HAVE Q)

(THE ABOVE SENTENCE IS AMBIGUUUS •• BUT I ASSUME (HAS) MEANS (HAS AS PARTS))
(THE ANSWER IS 10)
```

283, line 32, left	Change to read—analysis will begin by looking for any case—
284, line 4, left	Change to read—viewpoint of local management, and the policies—
285, line 19, left	Change to read—man should result in a smaller absolute reduction
286, line 17, left	Change to read—their effects are studied by separate experi—
	Appendices I and II should appear at the end of the paper.

Teitelman, Warren: "Real Time Recognition of Head-Drawn Characters"

Page 559	Correct author's name to "Warren"; Change first word in second paragraph to "In."

Talkin, Albert I.: "The Negative Gradient Method Extended to the Computer Programming of Simultaneous Systems of Differential and Finite Equations"

Page 540	Add derivate dot above x in (1.3); above x in 4th line of Sec. 1.2 above initial x,y,z, and x^2, y^2, z^2, in third line of (2); initial x, y, and x^2, y^2 of second line of (3) and (4); initial y and second x of (5.2)

Yang, C. C. and Tou, J. T.: "Systematic Design of Cyrogenic Logic Circuits"

Page 651, line 7, left	5, 10-13, 15 are references (superscript)
, line 10, left	9, 10, 12 are references (superscript)
653	In each case where pairs X; and X; appear, change to X; and $\overline{X}$;
653, line 41, left	Change to $\overline{X}; = 0$
, line 5, right	Change to $\bar{F}(X_m)$
, line 27	Change second X; to $\overline{X}$;
654, line 7 and rest of left	Change X; to $\overline{X}$;
, lines 25, 26 right	Change X; to $\overline{X}$;
655, last line, left	Change second X; to $\overline{X}$;
, lines 15, 17, 28, right	Change F(X_m) to $\bar{F}(X_m)$
, line 30, 43, right	Change to $\bar{P}_1$ through $\bar{P}_n$
, line 33, right	Change to $\bar{P}_k + P_k\bar{P}_{k+1} = \bar{P}_k + \bar{P}_{k+1}$
656, line 3, left	Change to $\bar{P}_k$
, lines 4, 23, 25, left	Change to $\bar{F}(X_4)$
, lines 15, 16 left	Change to $F(X_4) = \overline{X}_1X_2\overline{X}_3\overline{X}_4 + \overline{X}_1X_2X_3X_4 + X_1\overline{X}_2\overline{X}_3X_4$
, lines 27, 28 left	Change to $\bar{F}(X_4) = X_1(X_2 + X_3) + \overline{X}_2 + X_3)\ \overline{X}_4 + \overline{X}_1\ (\overline{X}_2 + \overline{X}_3X_4)$
, right—	Change to:
, line 1,	$\bar{F}(X_4)$
, line 3, 4	$F(X_4) = (\overline{X}_1 + \overline{X}_2\overline{X}_3)\ (X_2\overline{X}_3 + X_4)\ (X_1 + X_2\ [X_3 + \overline{X}_4])$
, line 8,	$P_1 = \overline{X}_1 + \overline{X}_2\overline{X}_3$, $\bar{P}_1 = X_1\ (X_2 + X_3)$
, line 9,	$P_2 = X_2\overline{X}_3 + X_4$, $\bar{P}_2 = (\overline{X}_2 + X_3)\overline{X}_4$
, line 10,	$P_3 = X_1 + X_2(X_3 + \overline{X}_4)$,
, line 11,	$\bar{P}_3 = \overline{X}_1(\overline{X}_2 + \overline{X}_3X_4)$
, line 17,	and $\bar{P}_i$ $(i = 1,2,$ or $3)$
, line 26,	and $\overline{X}_1$ or X_4 and $\overline{X}_4$ has
, line 32,	and $\overline{X}_i$ possesses

657, line 4, left	function $\bar{F}(X_m)$.
, line 9, left	$\bar{F}(X_m)$ should
, line 2, right	form $\bar{F}(X_m)$,
, line 36, right	$F(X_3) = \bar{X}_2\bar{X}_3 + \bar{X}_1X_2X_3 \qquad (11)$
, lines 39, 40, right	$F(X_3) = \bar{X}_1\bar{X}_2\bar{X}_3 + \bar{X}_1X_2X_3 + X_1\bar{X}_2\bar{X}_3 \qquad (12)$
, lines 43, 44, right	$\bar{F}(X_3) = \bar{X}_1\bar{X}_2X_3 + \bar{X}_1X_2\bar{X}_3 + X_1\bar{X}_2X_3 + X_1X_2\bar{X}_3 + X_1X_2X_3$
658, line 13, left	of X_i is
, line 14, left	and $\bar{X}_1$ is
, line 14, right	form $\bar{F}(X_4)$,
, line 15, right	and $\bar{X}_3$ and
, line 16, right	and $\bar{X}_4$ are
, line 39 right	$S_n(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m)$
659, left line 1	Written $\bar{X}_1$, $\bar{X}_j$, etc.
, line 3,	x_1, $\bar{X}_i$, X_j, $\bar{X}_m$, etc.
, line 5,	parameters $\bar{X}_1$, $\bar{X}_j$, etc.
, line 7,	When $\bar{X}_1$, $\bar{X}_j$,
, line 16,	and $\bar{X}_j$.
, line 19,	Variables $\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j$,
, line 32	'With $\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots$
, line 38–40	$F(X_m) = S_{p,q}(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m)$
, line 43-44	$= S_p(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m)$
	$+ S_q(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m)$
	$\bar{F}(X_m) = \bar{S}_{p,q}(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m$
	$= \sum_{k=0}^{m} S_k(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m)$
	$k \neq p \neq q$
659, right line 12	X_1, $\bar{X}_2$ and $\bar{X}_3$
line 17	$S_p(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m)$ and
line 18	$S_q(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m)$
line 22	$S_k(\bar{X}_1, \ldots, X_i, \ldots, \bar{X}_j, \ldots, X_m$
line 25	$\bar{F}(X_m)$.
660, left lines 3,4	$F(X_3) = X_1X_2\bar{X}_3 + X_1\bar{X}_2X_3 + \bar{X}_1\bar{X}_2\bar{X}_3$
line 8	or $\bar{X}_1$, $\bar{X}_2$
line 9	$\bar{X}_3$. When X_1 and$\bar{X}_2$ or X_1 and $\bar{X}_3$ are
line 11	X_1, $\bar{X}_2$,
line 12	X_3 by
line 15	X_1, $\bar{X}_2$
line 16	and $\bar{X}_3$ become
line 27	$F(X_3) = S_2(X_1, \bar{X}_2, \bar{X}_3)$
line 29	$\bar{F}(X_3) = S_{0,1,3}(X_1, \bar{X}_2, \bar{X}_3)$
line 33	and $\bar{X}_3$ located
661, left lines 33, 34, 35	$\bar{X}_j$ is
line 37	by $\bar{X}_j$
line 39	gate $\bar{X}_j$ in

Avizienis, A.: "Binary-Compatible Signed-Digit Arithmetic"

Page 665, 667, 669, 671	Replace running head with title above.	
667, line 1-8, left	Change column of numbers to;	$$\begin{array}{cccccccc} & .6 & 0 & \bar{7} & \bar{4} & 7 & 5 & \emptyset \\ & .\bar{2} & 7 & 1 & \bar{5} & \emptyset & \emptyset & \emptyset \\ \hline & 4 & 7 & \bar{6} & \overline{11} & 7 & 5 & 0 \\ & 4 & \bar{1} & \bar{6} & \bar{1} & \bar{1} & \bar{5} & 0 \\ 0 & 1 & 0 & \bar{1} & 1 & 0 & 0 & \\ \hline & .5 & \bar{1} & \bar{7} & 0 & \bar{1} & 5 & 0 \\ & .5 & \bar{1} & \bar{7} & 0 & \bar{1} & 5 & \emptyset \\ & .5 & \bar{1} & \bar{7} & 0 & \emptyset & \emptyset & \emptyset \end{array}$$
667, lines 20 & 22 right	Change to $x' = .1\bar{6}$;$x' = .1\overline{77}\ldots\overline{77}$	
667, lines 35-41, right	Change column of numbers to;	$$\begin{array}{cccccccc} & .6 & 0 & \bar{7} & \bar{4} & 7 & 5 & \emptyset \\ & .7 & 7 & 7 & 7 & 7 & 7 & 7 \\ \hline & 15 & 7 & 0 & 3 & 16 & 14 & 7 \\ & 5 & \bar{1} & 0 & 3 & 6 & 4 & \bar{1} \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & \\ \hline & .6 & \bar{1} & 0 & 4 & 7 & 5 & \bar{1} \\ & .6 & \bar{1} & 0 & 4 & 7 & 5 & \emptyset \end{array}$$
668, line 44, left	Change p and x in equation to P X	
line 1, right	Change "recorded" to "recoded"	
line 16, right	Change to "multiples"	
line 44, right	Insert "and" after "form"	
669, line 8, left	Change to P (j-1)	
line 25, right	Change p to q	
line 37, right	Change to "digits"	
line 41, right	Change to R (j-1)	
670, line 24, left	Change to " = 4"	

Fricke, L. H.: "The Use of a Portable Analog Computer for Process Identification, Calculation and Control"

Change running heads to read "Calculation", instead of "Calculating"

686, left	Change words "out" and "in" in the equation to subscripts
691, line 24, right	Change B in equation to a lower case *beta*

Paquette, G. A.: "Progress of Hybrid Computation at United Aircraft Research Laboratories"

699, Eqs (1),(2),(3)	Add derivative dots over final x (double in Eq) (2)
700, Eqs (6),(7),(8)	Add derivative dots over final x in (6); all xs in (7) (double over final x); and double over final x in (8)
701, left, is para	Change to read: "The use of a single argument interval often requires an excessively large data table to adequately describe the function. Some reduction in table size can be accomplished by subdividing the function into regions with different argument intervals. Another procedure is to apply an argument modifying function to the problem argument, i.e.,

Carbrey, R. L.: A Strobed Analog Data

Page 712	Change Fig. 4a to 4b; 4b to 4a

Frederickson, A. A.: Hybrid Simulation of a Lifting Re-entry Vehicle

Page 717, lines 3, 5, 6, right	Add double dots over h in $h \approx 0$, hi in (1) and single over V
718, line 11, left	Add derivative dot over h
719, line 2, right	Add double dot over h
line 4, right	Add single dot over V
728, Eqs (7),(8),(9)	Add derviative dot over q
731, lines 16, 17, 21 left	Add derivative dots over α and β
lines 26, 27, 31, 36, left	Add derivative dot over T, ΔT, and second T_s
731, Eqs (15), (16)	Add derivative dot over T, intial T_s, final α, ΔT, initial β and final T_s
731, Eqs (17), (19), (20)	Add derivative dot over T and ΔT

The following are corrections to AFIPS, Volume 26, Part II:

Credit portrait of Gen. David Sarnoff, p. 2 to "Karsh, Ottawa"